JAVA
Eğitim Notları

JAVA ile Programlama

Giriş

Bu eğitimde iyi bir java programcısı olmak için gerekli olan tüm bilgileri yoğun uygulamalar çerçevesinde inceleyeceğiz. Bu bilgileri öğrenirken aslında bilgisayar programcılığının tüm temellerini de öğrenecek, çalışmamızı tamamladığımızda da java programlama dilinin tüm sözdizimi yapısını, dilin kurallarını ve bu dil içerisinde yer alan temel paketlerin kullanımını rahatlıkla kavramış olacaksınız. Ayrıca temel kavramları öğrendikten sonra ileri düzeyde interaktif bir program yazabilmeniz için gerekli olan kullanıcı arayüz tasarımının Java ile nasıl yapılabileceğini de göreceğiz.

Bu Eğitim Kimler İçin?

Teknik detaylara girmeden önce şunu belirtmek gerekir ki bu çalışma daha önce hiç bilgisayar programcılığı ile ilgili çalışmayanlar için çok iyi bir başlangıç ve az çok bu işi bilenler için de iyi bir arşiv olacak şekilde hazırlanmıştır. Eğitim içerisinde en temel kavramları anlatırken hem mevcut bilgilerinizi ilerletecek hem de önceden öğrendiğimizi sandığımız temel bilgilerinizi tekrarlamış olacaksınız. Ayrıca arasıra değineceğimiz pratik ipuçları sayesinde de, uzun zaman harcayarak elde edeceğimiz bazı bilgilere, hemen erişme şansına sahip olacağız.

Java Platform Bağımsızdır

Doğasında platform bağımsız olarak çalışan Java uygulamalarımızı ve uygulamaları oluşturmak için kullanacağımız araçları hem Windows hem de Linux işletim sistemlerine yönelik hazırladığımızdan herhangi bir kısıtlama içerisine de girmeyeceksiniz.

Bu dersimizde Java programcılığına giriş yapacak ve aşağıda listelenen başlıkları öğrenmiş olacağız:

* Java nedir? Nerelerde kullanılır?
* Nesne yönelimli programcılık nasıl olur?
* Java paketleri
* Kurulum ve ilk ayarlar
* Java ile bir uygulamanın çalıştırılması
* Nesne kavramı ve sınıf tasarımı
* Temel yazım kuralları

Eğitim boyunca öğreneceğimiz tüm bilgileri dikkatlice okumanızı ve sıklıkla tekrar etmenizi öneririm. Ayrıca anlatım sırasında ve pratik amaçlı verilecek örnekleri de mutlaka kendiniz de yazıp denemelisiniz. Unutmayınız ki iyi bir bilgisayar programcısı olmak, bir müzik aletini iyi çalabilmeye benzer. Bunu başarmanın en kolay yolu da sayısız pratik yapmaktır. İlk başlarda anlamsız ve kolay gibi görünen uygulamaları sabırla yazıp denediğinizde ne kadar çok yere takıldığınızı farkedeceksiniz. Ayrıca bu pratiklerin ilerleyen haftalarda nasıl bir yetenek kazandırdığına inanamayacaksınız :)

Nedir Bu Java?

İlk olarak Java programlama dili ve nesne yönelimli programlama tekniği hakkında kısaca bilgi edinmek iyi bir başlangıç olacaktır.

Adı "Green Project" olarak bilinen bir projenin içerisinde yer alan James Gosling, proje için en uygun dilin belirlenmesi işini üstlenmişti. 1984'de Sun Microsystems'de göreve başlayan Gosling, çalışmalarına C++ ile başlamış ancak bu dilin proje için yeterli olmayacağı düşüncesiyle, yine bu dilin bir türevi olan ve ilk adı "Oak" olan yeni bir dili geliştirmeye başlamıştı. Daha sonra yerel bir kafeden çağrışım yaparak bu dilin adını Java olarak değiştirmiştir. O günlerden bu güne Java halen geliştirilmekte olan ve günümüzde pöpülaritesi tartışılmaz olan bir dil olarak kullanılmaktadır.

Java nesne yönelimli bir dildir

Java nesne yönelimli bir programlama dilidir. Bir programlama dilinin nesne yönelimli olması, dilin kendi özelliği itibariyle aynı amaca yönelik görevlerin sınıf (class) denilen yapılar içerisinde toplanmasına ve bu prensibe göre kullanımına izin vermesidir. Nesne yönelimli programcılığın bir çok avantajı vardır. Bunlardan en önemlisi, bir projede birbirinden bağımsız olarak tasarlanan sınıf nesneleri kullanılmasıdır. Böylece projeye belli görevlerin eklenmesi veya çıkarılması projenin bütününe dokunmadan mümkün hale gelebilmektedir.

Bunu bir örnekle daha kolay kavrayabiliriz: Mesela icq benzeri bir program yazdığımızı düşünelim. Böyle bir projede kullanıcının karşısına gelecek uygulamanın görüntüsü ayrı bir mödül, sisteme kaydolan kullanıcıların kayıt işlerinin yürütülmesi ve veri tabanında saklanması işi ayrı bir mödül, ağ bağlantıları ve ağa giriş çıkış kontrollerinin yürütülmesi de ayrı bir mödül olarak tasarlanmakta ve daha sonra birleştirilmektedir. Bu tasarım sırasında herhangi bir modülde meydana gelen aksama diğer mödülleri etkilemeyecektir.

Bundan başka, belli bir görevi üstlenen bir sınıf nesnesi temel alınarak bu göreve yeni eklemeler yapmak (türetme) Java gibi bir nesne yönelimli programlama dili ile oldukça kolaydır. Örneğin; ileriki konularda bahsedeceğimiz şekilde, siz bir pencere uygulaması tasarlamış olabilirsiniz. Daha sonra bir başkası sizin tasarladığınız bu pencereyi hiç bozmadan kullanarak ve üzerine eklemeler yaparak internet explorer'a çok benzeyen başka bir pencere uygulaması haline getirebilir. Eğitimimiz devam ettikçe zaten bu tekniğin aslında ne kadar vazgeçilmez olduğunu daha iyi kavrayacaksınız.

Java Multithread Programlamayı Destekler
Aynı anda birden fazla işi yürütebilecek fonksiyonların çalışması ancak mutlithread desteği sayesinde mümkün olacaktır. Java halihazırda bu yeteneğe sahiptir.

Güvenli ve Sağlamdır
Java'da, C ya C++ da olduğu gibi pointer kullanımı kullanıcıya o kadar açık değildir. Bu nedenle pointer gibi kritik kullanımlar neticesinde doğacak güvenlik problemlerine java da rastlanmaz.

Ayrıca Java'da built-in olarak bulunan birtakım araçlar sayesinde hafıza yönetimi dinamik olarak ele alınmaktadır. Böylece hafızanın bir süre sonra dolması gibi problemler de Java dili içerisinde engel teşkil etmez. Bu araçlardan en önemlisi **Garbage Collector** denilen araçtır. Bu araç hafızada tutulan ve artık kullanılmayan nesneleri otomatik olarak temizler.

Java'nın güvenli olmasının bir diğer nedeni de içerisinde sağladığı çalışma mekanizmasıdır. Bu mekanizmayı şöyle özetleyebiliriz:

Classloader ile gelen güvenlik
Yazılan ve derlenen programlar içerisinde gerekli sınıflar **class loader** denilen bir araçla programa dahil edilirler. Ancak Class Loader ağ üzerinden gelen sınıflarla lokal makinede yer alan sınıfları ayrı yerlerde tutar. Böylece daha önceden güvenilen lokal sınıfların üzerine yazılma ihtimali ortadan kalakacaktır.

Ayrıca JVM tarafından yorumlanan byte-code ların da güvenilirliği test edilmektedir. Bu testi **byte-code verifier** denilen bir araç üstlenir. Bu işlem çalışma zamanı sırasında (run-time) gerçekleştirilmektedir.

Enterprise Uygulamalar İçin Geliştirlmiştir.
Java sağladığı platformlar sayesinde dağıtık (distrubited uygulamaları geliştirmek için oldukça uygundur. Dağıtık uyuglamardan kasıt, sunucu-istemci etkileşimli, veri tabanı kullanan çok katmanlı yapılardır. Bunun için Java Teknolojilerini incelemek gerekmetedir.

Java Paketleri
Paket denilen şeyi aslında daha önce başkalarının yazıp dil içerisinde kullanılacak şekilde hazırladığı program parçaları olarak düşünebilirsiniz. Bu parçalar içerisinde ana programınızı yazmak için bazı özel işleri yapabileceğiniz (dosya işlemleri, ağ işlemleri, ekran işlemleri v.b.) araçlar yer almaktadır. Bu araçlar aslında tüm önemli işlerini temelini oluşturacak görevleri yapabilmektedir. Daha özel uygulamalar bu temel araçların üzerine geliştirilir. Bu nedenle temel araçları en baştan tekrar yazmak yerine paket içerisinden kullanırız. Daha teknik anlamda bu araçlar; sınıflar ve bu sınıflara ait metod ve değişkenlerdir. Bu paketlere aynı zamanda Sınıf Kütüphanleri (Class Libraries) de denilir.

JDK (Java Development Kit)
Bu kütüphanelerden, Pencere Yönetimi (Window Management), Giriş/Çıkış (I/O) ve Ağ İletişimi (Network Communication) uygulamalrına yönelik olan sınıf kütüphaneleri JDK (Java Development Kit) denilen ortam içerisine yerleştirilmiştir. Bunun dışında Java içerisinde bazı işleri kolaylaştıran araçlar da mevcuttur. Bu araçlar debug, deployment ve dökümantasyonu kolaylaştırmak amacıyla kullanılır.

Java Dökümantasyonu
Java programlama dilinin yapısını oluşturan temel paketler dışında birçok farklı alanda program geliştirebileceğiniz yüzlerce paket mevcuttur. Ayrıca bu paketlerde yer alan genel konuya ilişkin yüzlerce sınıf nesnesi yer almaktadır. Bu sınıf nesnelerinin dökümantasyonunu kullanmadan iyi bir Java uygulaması geliştirmenin neredeyse imkansız olduğunu söyleyebilirim.

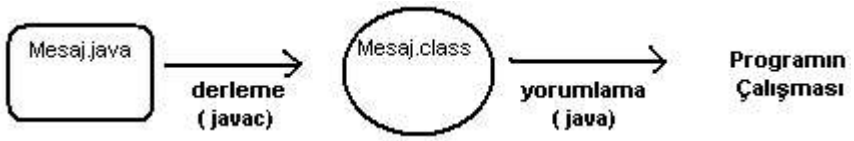
Bunun için, yazılmış kitapların yanında Java ile ilgili herşeyi bulabileceğiniz http://java.sun.com/ adresini incelemenizi tavsiye ederim. En kısa yoldan bu Java paketlerini incelemek istiyorsanız http://java.sun.com/j2se/1.4.1/docs/api/ adresinde aradığınız herhangi bir sınıfa ilişkin tüm method ve değişkenleri bulabilirsiniz.

Java ile geliştirebileceğiniz bir çok alanda uygulama olduğunu söylemiştim. Aslında bu uygulamaları birbirinden kesin bir çizgiyle ayırmak pek doğru olmaz. Ağ üzerinde ortak bir veri tabanını kullanan ve görsel bir kullanıcı grafik arayüzüne sahip olan bir projeyi geliştirmek için veri tabanı, ağ uygulaması ve grafik arayüz tasarımı şeklinde üç ayrı uygulama alanında birden çalışmak gerekir. Ancak güzel olan, Java'nın nesne yönelimli bir programlama dili olması itibariyle, bu uygulamaların ayrı ayrı geliştirilip daha sonra tek bir proje çatısı altında birleştirmenin mümkün olmasıdır.

Java Sanal Makinesi (Java Virtual Machine)
Java'nın platform bağımsız olması demek, herhangi bri makinede yazılmış ve bir işlemci ile derlenmiş bir java uygulamasını, tek bir noktasını bile değiştirmeden başka bir makine ya da işlemci altında çalıştırabilmek anlamına gelmektedir. Java'ya bu özelliği kazandıran mekanizma ise içerisinde barındırdığı JVM (Java Virtual Machine) dir.

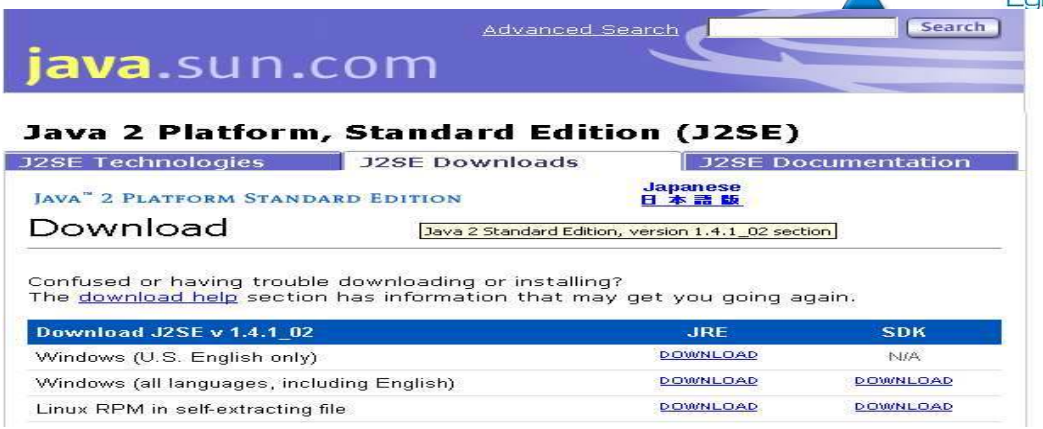
Bilgisayarınızda bir java uygulaması çalıştırabilmeniz için, içerisinde java sanal makinası ve kütüphanelerinin de yer aldığı sdk setini yüklemeniz gerekir. Yazdığınız uygulama, makinada yer alan bu SDK ile gelen, JVM içerisinde çalışacağından, program kodunuzda hiç bir değişiklik yapmadan ister windows altında, ister linux altında, ister Intel, SPARC ya da Alpha işlemcili bir makinada çalıştırabilirsiniz.

Java kaynak kodları .java uzantılı dosyalarda yer almaktadır. Her java dosyası çalıştırılmadan önce derlenerek aynı isimle .class dosyasına dönüştürülür. Bu derleme işlemini yapan program ise javac programıdır. Daha sonra derlenmiş olan bu .class dosyası yine jvm içerisinde yer alan java programı tarafından yorumlanır ve böylece yazdığınız uygulama çalıştırılmış olur. Bunu aşağıdaki resimde görmekteyiz:



Kurulum ve İlk Ayarlar
SDK setinin son versiyonu ve dökümantasyon paketini http://java.sun.com/j2se/downloads.html adresinden indirebilirsiniz. Ancak bu eğitim kapsamında 1.4.2 versiyonu yeterli olacaktır.

Yukarıda seti indirebileceğiniz adresi ziyaret ettiğinizde, indirmek istediğiniz seti belirlediğiniz zaman karşınıza gelen ikinci sayfada yine bu set ile ilgili hangi dilde, hangi platform için ve hangi paketi indireceğiniz konusunda bir seçim yapmanız istenecektir. Ayrıca yine bu sayfada daha önce web adresini verdiğim dökümantasyon paketini de indirebileceğiniz bir bağlantı bulacaksınız.



Burada "Windows (all languages, including English)" yazılı satırda SDK sütunu altındaki bağlantıda yer alan set Windows kullanıcıları için uygundur. Dilerseniz aynı şekilde Linux işletim sistemi için de uygun setleri indirebilirsiniz. İndirme işlemi tamamlandıktan sonra Windows kullanıcıları için .exe uzantılı gelen seti üzerine tıklayarak kurmaya başlayabilirsiniz.

Linux kullanıcıları isterlerse .rpm uzantılı kurulum paketini, isterlerse de .bin uzantılı kurulum dosyasını aynı yerden indirebilirler. İndireceğiniz dosya j2sdk-1_4_1_04-linux-i586.bin ya da j2sdk-1_4_1_04-linux-i586.rpm gibi bir isme sahip olacaktır. Eğer .bin dosyası ile kurulum yapacaksanız, aşağıdaki gibi önce bu dosyaya çalıştırma izni verip daha sonra da çalıştırarak kurulumu tamamlayabilirsiniz:

```
$ chmod a+x   j2sdk-1_4_1_04-linux-i586.bin
$ ./ j2sdk-1_4_1_04-linux-i586.bin
```

.rpm uzantılı paketi kurmak için ilk once root kullanıcısı olmanız gerekmekteir. Daha sonar da yine aşağıdaki komutu kullanarak kurulumu tamamlayabilirisiniz:

```
$ su
# rpm  -ivh  j2sdk-1_4_1_04-linux-i586.rpm
```

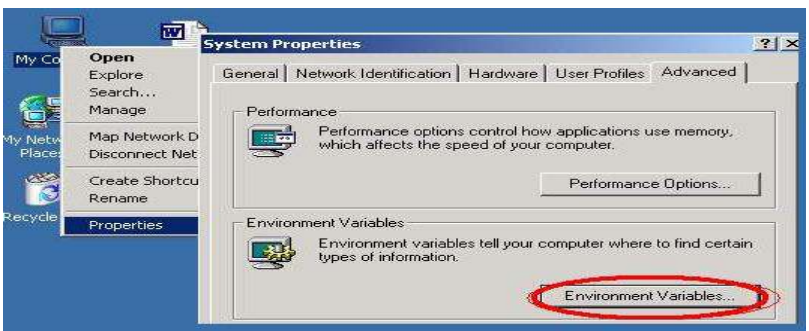
Not: Yukarıda bahsettiğim linux ve windows platformları için kurulum işlemlerine ilişkin açıklamaları, kurulum dosyalarını indirdiğiniz sayfada da bulabilirsiniz.

PATH Ayarı

Kurulum işlemi tamamlandıktan sonra PATH ayarını yapmamız gerekmektedir. İşletim sistemlerinde yer alan ve orjinal adıyla "Environment Variables" denilen bir takım anahtar değerler içerisinde çeşitli bilgiler tutulur. Bu değişkenlerden birtanesi de sistem içerisinde yazılan komutların aranacağı dizinlerin hangi dizinler olacağı bilgisini tutan PATH değişkenidir. Java uygulamalarımızı çalıştırırken kullanacağımız komutların işletim sistemi tarafından her yerden tanınması için bu PATH değişkenine java sanal makinasının komutlarının yer aldığı dizinini de ekleyeceğiz. Burada bu işlemi hem Windows 2000 işletim sistemi hem de Linux işletim sistemi için anlatılacağım. Ancak yukarıda verdiğim bilgiyle birlikte siz de aynı işlemi başka sistemlerde de kolaylıkla yapabilirsiniz. Çünkü neticede yapılacak şey sadece bu PATH değişkenine yeni dizini eklemek olacaktır.

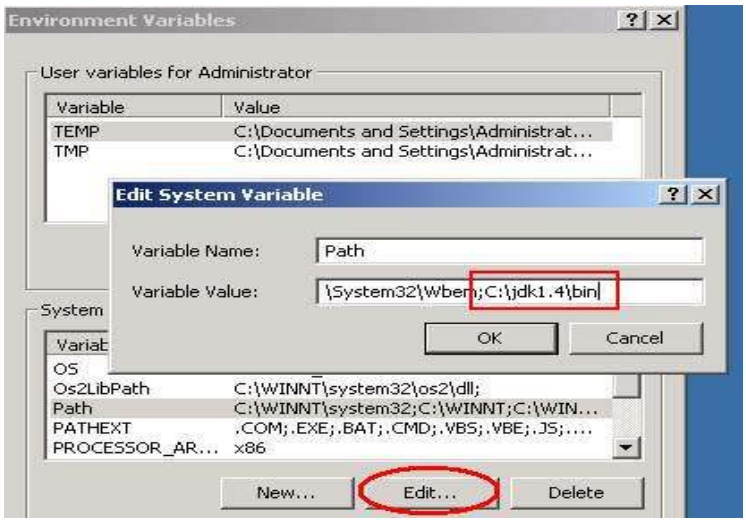
Windows kullanıcılar için:

Masaüstünde yer alan Bilgisayarım (My Computer) simgesine sağ tıklayınız ve Özellikler (Properties) seçeneğini seçiniz. Daha sonra karşınıza çıkan pencerede Gelişmiş (Advanced) sekmesini seçiniz ve burada yer alan Çevresel



Değişkenler (Environment Variables) seçeneğini seçiniz:

Bu sefer karşınıza çıkan pencerede Path isimli değişkeni göreceksiniz. Bu değişkeni seçip Düzenle (Edit) düğmesine basarak eklemek istediğiniz yeni dizini karşınıza çıkan söz diziminin en sonuna önce bir noktalı virgül (;) koyarak yazınız ve değişiklikleri kaydederek sisteminizi yeniden başlatınız. Bundan sonra artık java komutlarını rahatlıkla kullanabiliriz. (Buradaki örnekte sanal makinanın yer aldığı dizin C:\jdk1.4\bin olarak kullanılmaktadır. Bu bilgi sizin makinanızda farklı olabilir)



Linux kullanıcıları için:

Bir terminal içerisinde PATH değişkenine aşağıda gösterildiği gibi java sanal makinasının yer aldığı dizini ekleyiniz ve aşağıdaki gibi export komutu ile bu değişkeni yeni değeri ile tüm kabuklar için kullanıma hazır hale getiriniz (Buradaki örnekte sanal makinanın yer aldığı dizin /usr/local/jdk1.4/bin olarak kullanılmaktadır. Bu bilgi sizin makinanızda farklı olabilir) :

```
$ PATH=$PATH:/usr/local/jdk1.4/bin
[academytech@localhost academytech]$ export PATH
```

Şimdi de değişkenin yeni değerini alıp almadığını kontrol edelim:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/academytech/bin:/usr/local/jdk1.4/bin/
```

Gördüğümüz gibi çıktının en sonunda bizim eklediğimiz yeni dizin de yer almaktadır. Bu çıkış bize işlemi düzgün yaptığımızı anlatmaktadır. Ancak şunu da eklemek gerekir ki linux altında yapılan bu ayar sadece o andaki oturum için geçerlidir. Bu bilginin kalıcı olmasını istiyorsanız home dizininizde yer alan ".bash_profile" dosyası içerisine şu satırı ekleyin:

```
PATH=$PATH:/usr/local/jdk1.4/bin
```

Bu anlatılanları yaptıysanız artık sistem içerisinde hangi dizinde olursanız olun java sanal makinanıza ait komutları doğrudan kullanabileceksiniz.

Java ile Uygulamaya Giriş

İlk ayarları ve kurulumu tamamladığımıza göre artık en temel java uygulamamızı yazalım ve çalıştıralım. Bu uygulama kullanıcıya aşağıdaki gibi bir mesaj penceresi çıkaracaktır:

Resim 4: Ornek 1 ile yazılan uygulama sonunda ekrana gelecek olan mesaj penceresi

Herhangi bir editorde aşağıdaki kodu yazarak "Mesaj.java" adıyla kaydediniz:

```
import javax.swing.*;
import java.awt.*;

public class Mesaj {                              //1. satır
  public Mesaj() {                                //2. satır
    JOptionPane.showMessageDialog(null, "java ile hayat guzel :)");
  }


  public static void main(String[] args) {      //3. satır

    Mesaj A = new Mesaj();
    System.exit(0);
  }
}
```

Şimdi de komut satırında önce bu dosyayı kaydettiğiniz klasöre gidiniz (bu örnekte bu klasör c:\temp olarak yaratılmıştır) ve aşağıdaki gibi önce "javac Mesaj.java" yazıp enter tuşuna basın. Bunu yaptığınızda bilgisayar kısa bir süre yazdığınız kodu jvm'nin anlayacağı şekilde derleyecektir. Daha sonra da yine komut satırında "java Mesaj.java" yazıp enter tuşuna basın. Bu sefer de jvm derlenmiş dosyayı çalıştıracaktır:

C:\temp> javac Mesaj.java
C:\temp> java Mesaj

Not: Linux altında da bu uygulamayı ve bundan sonra göstereceğimiz uygulamaları aynı şekilde çalıştırabilirsiniz.

Bir java uygulaması en temel şekilde böyle çalıştırılmaktadır. Bu işlem linux işletim sistemi altında da değişmeyecektir. Programımıza kısaca bir göz atalım:

İlk satırlarda yer alan "import javax.swing.*" ve "import java.awt.*" bildirimleri kullanacağım nesnelerin tanımlı olduğu paketleri program içerisinde de erişebilir hale getirmektedir. Eğer bu satırlar yazılmasaydı program, gördüğümüz mesaj penceresinin oluşmasını sağlayan JOptionPane adındaki sınıf nesnesini tanımayacak ve bu nesneyi bulamadığına dair bir hata verecekti. Bu yüzden kullandığınız nesnelerin hangi kütühanelerde olduğunuz bilmeniz ve programın en başında import anahtar sözcüğü ile koda dahil etmeniz gerekmektedir.

Yazdığınız java uygulaması da kendi başına bir sınıf nesnesidir ve bu nesne adı ile (bakınız: Mesaj.java - 1. satır) dosya ismi aynı olmak zorundadır. Aksi takdirde programınız düzgün bir şekilde derlenmeyecek ve dolayısı ile çalışmayacaktır.

İkinci satırda "public Mesaj()" şeklinde bir metod tanımlandığını görüyoruz. Nesne ismi ile aynı isme sahip olan bu özel metodlara "başlangıç metodları" (constructor) denilmektedir. Yazılan her java programı bir sınıf nesnesi olduğuna göre bu nesneyi kullanmak için önce bu nesne türünden bir değişken yaratmak gerekir. İşte bu değişken yaratıldığı anda bu özel metodlar otomatik olarak çağrılırlar. Başlangıç metodlarıyla ilgili ayrıntılı bilgiyi daha sonra öğreneceğiz.

Üçüncü satırda yer alan ve "main" adıyla tanımlanmış metod ise yine özel olarak programın çalıştırıldığı ilk metoddur. Burada yukarıda bahsettiğim nesne yaratma işlemi yapılmaktadır. Daha genel söylemek gerekirse; yazdığınız programın çalışabilmesi için, temel programın main metodu içerisinde yazılması gerekir. Biz burada, "Mesaj" sınıf nesnesini eğer main metodu içerisinde kullanmasaydık program yine derlenecekti ancak hiç bir şey çalışmayacaktı. Fakat bu örnekte yazdığımız nesne main metodu içerisinde yaratıldı ve yaratıldığı anda bu nesnenin başlangıç metodu otomatik olarak çağrıldı. Bu çağrım neticesinde de ekrana bir mesaj penceresi çıkmış oldu. Her ne kadar daha önce java ile çalışmamış olan arkadaşlar için main metodunun nesne içindeki bu kullanımı biraz tuhaf gelse de, alışıldığında ne kadar basit ve anlamlı olduğu anlaşılacaktır .

Amacımız şu an için sadece bir giriş yapmak olduğundan main metoduna aktarılan parametre yapısı ve bundan sonraki örneklerde de göreceğiniz "public", "private" gibi anahtar sözcüklerin kullanımı hakkında burada hemen bilgi vermeyeceğim. Şimdilik sadece kullandığım nesnelerin nasıl yazıldığı ve ilerki konularda da göreceğimiz şekilde nasıl ele alındığını ve ayrıca nesneler arasındaki hiyerarşinin nasıl sağlandığını incelemenizin daha faydalı olacağını düşünüyorum.

Örnek 1'de adı messageDialog olan sadece bir tane nesne kullandık. Bu nesneyi göstermek için JOptionPane sınıfında yer alan showMessageDialog metodunu, mesaj penceresinde görülmesini istediğimiz yazıyı parametre olarak girerek çağırdık.

İpucu: Bir java uygulamasını tek bir tıklama işlemi ile çalıştırmak da mümkündür. Bunun için windows altında aşağıdaki gibi bir dosyayı "uygulama.bat" adıyla kaydedersiniz dosya üzerine tıkladığınızda uygulamanız direk olarak çalışacaktır:

```
/*--- uygulama.bat ---*/
   javac Mesaj.java
   java Mesaj
/*--------------------*/
```

Aynı işlemi linux altında yapmak için uygulama.sh adında bir dosya içerisine yukarıdaki satırların aynısını yazarak dosyayı kaydediniz ve konsol üzerinde bu dosyaya aşağıdaki gibi çalıştırma izni vererek dosyayı çalıştırınız:

```
$ chmod 700 uygulama.sh
$ ./uygulama.sh
```

Nesne Kavramı ve Sınıf Nesnelerinin Tasarımı

Java'da her şey bir nesnedir. Nesneler sınıf denilen yapılar ile meydana getirilir. Her nesne kendisine ait bazı bilgileri barındırmaktadır. Bu bilgiler nesneye ilişkin çeşitli değişkenler, nesne ile ilgili bazı işleri yapacak çeşitli fonnksiyonlar ve sabitlerdir. Ayrıca yine bir nesne içerisinde nesnenin kullanacağı başka nesneler de olabilir. Buradan da anlıyoruz ki aslında bir sınıf nesnesi birden fazla veri tipinin bir arada bulunduğu ve belli bir işi yapmaya yönelik hazırlanan birleşik bir yapıdır.
Bir sınıf nesnesi yaratılırken değişmeyen bazı kurallar vardır. Bu kurallar neticesinde siz de kendinize ait sınıflar yaratabilirsiniz. Şimdi temel olarak bir sınıf nesnesinin nasıl yaratıldığına bakalım:

```
public class Kitap {

      String kitap_adi;
      String yazar_adi;
      int sayfa_sayisi;
      float ucret;
      ...

      float ucretiSoyle() {
          ....
      }

       void ucretiDegistir(float yeni_ucret)  {
           ....
```

```
        }
}
```

Bu örnekte de görüldüğü gibi bir sınıf nesnesi yaratırken "class" anahtar sözcüğünü kullanmaktayız. Nesnemizin tamamı, class anahtar sözcüğü ile onu takip eden ve bizim vereceğimiz isimden sonra açılan küme parantezi ile en sonda kapatılan küme parantezi içerisine yazılmaktadır. Nesne yaratılırken ilk kullanılan "public" anahtar sözcüğü nesnenin başka nesneler içerisinde de doğrudan kullanılabileceğini belirtmektedir. Bu anahtar sözcük gibi nesnenin doğrudan kullanılamayacağını belirtebileceğiniz "private" anahtar sözcüğü de mevcuttur. Bunu dışında yeri geldikçe kullanılabilecek başka anahtar sözcüklerin olduğunu da belirtmeliyim. Ancak şu anda yolumuza sadece bu bilgilerle devam etmek daha verimli olacaktır.

Gördüğümüz gibi Kitap isimli sınıf nesnemizin içerisinde "kitap_adi", "yazar_adi" gibi değişkenler yer almaktadır. Bu değişkenlere sınıf değişkenleri denilir ve nesnenin özelliklerini meydana getirirler. Bu özellikler her Kitap nesnesi için değişecektir ancak her Kitap nesnesi, değerleri farklı olsa bile bu özelliklere sahip olacaktır. Ayrıca yine nesnemiz içerisinde bu değişkenlere ek olarak bazı metodların yazılabileceğini de görmekteyiz. Java'da "metod" olarak isimlendirilen kavramlar C ya da C++ programlama dillerindeki "fonksiyon" kavramı ile aynı şeyi anlatmaktadır. Metodlar sayesinde belirli işlerin tek seferde yapılmasını sağlayabiliriz. Mesela bu örnek sınıf nesnesini kullanırken sınıf değişkenleri içerisine sırasıyla kitap adını, yazar adını ve kitabın ücretini yüklediğimizi kabul edelim. Bundan sonra herhangi bir anda kitabın ücretini öğrenmek istediğimizde önce hangi değişkenler olduğunu, hangi değişkende aslında kitap ücretinin yer aldığını öğrenmek yerine bu işi bizim için yapıp bize sadece sonucu üretecek bir metod yazmak daha akıllıca olacaktır. Bu amaçla yazdığımız "ucretiSoyle" isimli metodu çağırısak işimizi kısa yoldan halletmiş oluruz.

Metodların yazımı ve kullanımına ilişkin daha ayrıntılı teknik bilgileri ilerleyen sayfalarda vereceğim. Burada sadece kavram olarak metodları anlamak konuyu kavramak açısından yeterli olacaktır. Sonuç itibariyle yazılan bu sınıf metodları bu sınıf nesnesi ile yapılabilecek operasyonları tanımlar.

Yazım Kuralları Üzerine Bir Not
Bir uygulama yazarken dikkat edilmesi gereken önemli konulardan birtanesi de yazım kurallarıdır. Aslında burada bahsedeceğim bu kuralların bazılarına uymak bir zorunluluk olmasa da ileride iyi bir programcının sahip olması gereken alışkanlıkları edinmenize yardımcı olur.

Parantezlere Dikkat
Bahsedilebilecek ilk kural parantezlere dikkat etmeniz gerektiğidir. Sınıf nesnesini yaratırken açılan parantezi hemen aynı anda kapatıp daha sonra bu iki parantezin arasına yazmaya başlamanız, açılan bir parantezi kapatmayı hatırlama zorunluluğunu ortadan kaldırır ve hata yapma riskinizi azaltır.

İsimlendirme
İkinci kural olarak isimlendirmeden bahsedilebilir. Java'da sınıf nesneleri genellikle büyük harfle isimlendirilir. Ayrıca aslında yazılan her .java dosyasının bir sınıf nesnesi olduğunu da unutmamalıyız. Bir .java dosyasında başka sınıf nesneleri de olabilir ancak bunlardan bir tanesi asıl nesnedir ve diğer nesneler bu ana sınıf nesnesi tarafından kullanılır. Dosya ismi de bu sınıf nesnesi ile aynı isimde olmak zorundadır. Metodların isimlendirilmesi, ilk harf küçük harf, diğer kelimelerin de ilk kelimeye bitişik ve ilk harflerinin büyük harf şeklinde yazılması kuralına uymakta yarar vardır. Örneğimizde yer alan ucretiSoyle ve ucretiDegistir isimli metodlarda bu kurala uyduğumuzu görmekteyiz. Değişkenlerin isimlendirilmesi küçük harflerle yapılmaktadır.

Kodlamada Girintiler (Indentation)
Son olarak bahsedilebilecek kural ise kod yazımı sırasında kullanılan girintilerdir. Dikkat ederseniz yukarıdaki örnekte ilk satırda açılan parantezden sonra diğer bütün bilgileri bir tab içeriden yazdım. Daha sonra metodlara ilişkin parantezleri de açtıktan hemen sonra yazılı kodları temsil eden üç noktayı da (...) yine bir tab içeriden yazdım. Böylece hangi kodların hangi faliyet alanına ait olduklarını daha net bir şekilde görebiliyorum.

Yukarıda bahsedilen bu kurallara uymak, bize yazdığımız kodu daha sonra kolaylıkla anlama şansı sağlar. Ayrıca yardım almak ya da rapor vermek amacıyla kodu inceleyen başka kişiler de bu işi daha rahat yapabilirler. Bu amaçla bu kurallara özen göstermenizi tavsiye ederim.

Şimdi şu ana kadar edinmiş olduğumuz bilgiler ışığında bir örnek daha yazalım. Bu örneğimizde bir tane Kitap isimli sınıf nesnesi yazıp bu nesne türünden iki farklı değişken yaratacağım. Daha sonra da bu değişkenlerin sahip oldukları özellikleri inceleyeceğim.

```
public class Kitap {

  String kitap_adi;
  String yazar_adi;
  int sayfa_sayisi;

  public int sayfaSayisiniVer() {
    return sayfa_sayisi;
  }

  public void kitabiGoruntule() {
    System.out.println("\nRapor");
    System.out.println("********");
    System.out.println("Kitap Adi: " + kitap_adi);
    System.out.println("Yazari: " + yazar_adi);
    System.out.println("Sayfa Sayisi: " + sayfa_sayisi);
    System.out.println("\n");
  }

  public static void main(String[] args) {
    Kitap kitap1 = new Kitap();
    Kitap kitap2 = new Kitap();

    kitap1.kitap_adi = "Puslu Kitalar Atlasi";
    kitap1.sayfa_sayisi = 238;
    kitap1.yazar_adi = "Ihsan Oktay Anar";

    kitap2.kitap_adi = "Vadideki Zambak";
    kitap2.sayfa_sayisi = 307;
    kitap2.yazar_adi = "Balzac";

    kitap1.kitabiGoruntule();
    kitap2.kitabiGoruntule();
  }
}
```

Daha öncede anlatıldığı şekilde kodu yazıp çalıştırdığınızda aşağıdaki gibi bir çıkış alacaksınız:

```
Rapor
********
Kitap Adi: Puslu Kitalar Atlasi
Yazari: Ihsan Oktay Anar
Sayfa Sayisi: 238

Rapor
********
Kitap Adi: Vadideki Zambak
Yazari: Balzac
Sayfa Sayisi: 307
```

Şimdi gelin neler olduğunu inceleyelim: Örnek programımızda daha önce de dediğimiz gibi Kitap isimli sınıf nesnesini çeşitli değişkenleri ve metodlarıyla yarattık. Aynı zamanda nesne içerisine yazdığımız main metodu sayesinde de programımız derlendi ve çalıştırıldı. Şunu hatırlatmakta fayda var: Bir sınıf nesnesi yazılıp derlendiğinde aslında çalışmaz. Sadece o nesne kullanılabilir durumdadır. Bir java programının çalışması demek

aslında sadece main metodunun çalışması demektir. Bu örnekte de Kitap isimli sınıf nesnesini yine Kitap sınıf nesnesi içerisinde yazdığımız main metodu içerisinde kullanıyoruz.

Bu kullanımda Kitap nesnesi türünden iki farklı değişken yaratıyoruz. Bunun anlamı bu iki farklı değişkenin kendine ait olan ama aynı niteliklerde değişken ve metodları olduğudur. Daha farklı bir deyişle; aslında kitap1 değişkeninde de kitap2 değişkeninde de kitap_adi isimli bir String türünden değişken vardır ancak içerdikleri değerler farklı olabilir. Zaten main metodunun son iki satırında da kitap1.kitabiGoruntule() ve kitap2.kitabiGoruntule() metodlarının üretecekleri değerler her iki değişken için farklı olacaktır. Ancak her iki değişken de aslında bir Kitap nesnesi olduğundan kendilerine ait bir kitabiGoruntule() metoduna sahipler demektir.

Buradan çıkaracağımız sonuç sınıf nesneleri aslında birer modeldir ve bu nesneler türünden yaratılan değişkenler de bu modelin birer örneğidir.

Bugünkü çalışmamızı burada noktalıyoruz. Yukarıda anlatılan nesne kavramları, sınıf nesnesi tasarımı ve metodlara ilişkin açıklamaları aslında sadece iyi bir giriş yapabilmeniz amacıyla inceledik. İlerleyen dersler boyunca bu konuları tek tek kendi içlerinde detaylandıracağız.

Değişken Kavramı
Tüm programlama dillerinde olduğu gibi Java'da da değişkenler kullanılmaktadır. Değişken; program yazılması sırasında, içerisinde birtakım değerleri tutan ve kendine ait bir türü olan yapıdır. Bir değişkenin içerisinde tuttuğu değer, program akışı sırasında değiştirilebilir. Ayrıca her değişkenin bir türü vardır ve değişkenlerin içlerinde tuttukları bu değerler de aynı türden olmalıdır.

Kullanılmadan önce değişkenlerin özellikleri hakkında derleyiciye bilgi verilmesi işlemine "bildirim" denir. Bir değişken nesnesi şu özelliklere sahip olur:

değişken ismi
değişken türü
değişken içerisindeki değer
değişkenin faliyet alanı

Değişken İsimlendirme
Değişken ismi, değişken nesneyi temsil eden karakterlerdir. Bu isim belli kurallar dahilinde verilmektedir. Bu kurallar kısaca şöyle açıklanabilir:

değişken isimlerinde boşluk olmamalıdır,
sayfa sayisi -> yanlis
sayfa_sayisi -> dogru
sayfaSayisi -> dogru

değisken isimlendirmede türkçe karakterler kullanılmamalıdır,
ÖzelDeğisken -> yanlis
OzelDegisken -> dogru

Değişkenin türü aslında içerisinde tutacağı değerin özelliğidir. Mesela içerisinde tamsayı değerleri tutan bir değişken ile ondalık sayı değerleri tutan değişken aynı türden değildir.

Bir değişkenin içerisinde değerler saklanarak aktarılır ve kullanılırlar. Değişkenler içerisinde saklanan değerlerin de bir türe ilişkin olduklarını ve içlerinde saklanacak değişkenin türü ile aynı türden olmak zorunda olduklarını unutmayınız. Aşağıdaki örnekte integer türünden bir değişken yaratıp içerisinde de 200 sayisini saklayacağız:

```
public class Degisken {
    public static void main(String arg[]) {
         int a = 200;
```

```
        System.out.println("a degiskeninin sakladigi deger: " + a + " degeridir");
    }
}
```

Bu kodu yazdıktan sonra daha önceki derslerimizde gösterdiğimiz gibi çalıştırınız:

Kodun çalışması sonucunda ekranda şu yazıyı göreceksiniz:

a degiskeninin sakladigi deger: 200 degeridir

Java'da "primitive type" denilen değişken türleri vardır. Bu türlere biz ilkel ya da temel türler diyoruz. Daha sonraki derselerimizde göreceğiz ki temel türlerin dışında bir de nesnel türler vardır.

Kullanılan temel veri tiplerini inceleyelim:

Sayısal Veri Tipi	**Hafızada kapladıgı alan (byte)**	**Sınırları**
int	4	-2.147.483.648, +2.147.483.647
short	2	-32.768, +32.767
long	8	-9.223.372.036.854.775.808L , +8.223.372.036.854.775.807L
byte	1	-128, 127
float	4	-+ 3.40282347E+38F
double	8	-+ 1.79769313486231570E+308

Ön Eklere İlişkin NOT: Yukarıdaki tabloda görüldüğü gibi bazı veri tiplerinin sonuna F ve L gibi ekler getirilmiştir. Bunun nedeni bu veri tiplerinin eklerle birlikte ifade edilmesidir. Bu durum tür bilgilerinin birbirine karışmasını engeller.

Sayısal Olmayan Veri Tipleri
Yukarıdaki tabloda incelemiş olduğumuz veri tipleri sayısal bilgiyi ifade etmek için kullanaılır. Sayısal olmayan bilgiyi ifade etmek için kullanılan veri tipleri de vardır. Şimdi bunları incleyelim:

"char" Veri Tipi:
char veri tipi karakterleri temsil etmek amacıyla kullanılır. Bir karakter klavye ile sisteme tek tuş ya da birkaç tuş kombinasyonu ile ama tek seferde girdiğimiz sembollerdir.

Hafızada 2 byte yer kaplarlar.

Karakter Kümeleri ve Unicode
Aslında tüm dillerdeki sembolleri içine alan en geiş karakter kümesi Unicode karakter kümesidir. Bu küme içerisinde kullanılabilecek tüm karakterler yer alır ve her karakter hexadecimal olarak ifade edilir. 65536 tane karakteri içerecek genişliğe sahiptir. Ancak halihazırda 35000 karakter kullanır.

\u0008 (backspace), \u0009 (tab) gibi...

ASCII karakter kümesi aslında Unicode karakter kümesinin bir alt kümesidir ve ASCII karakterleri 128 tanedir. Hafızada 1 byte yer kaplarlar.

Bunun dışında bildiğimiz ISO8859-9 (latin-5) gibi karakter kümeleri ASCII nin uzantıları olup aslında yine unicode karakter kümesinin bir alt kümesidir.

"boolean Veri Tipi:
Bu veri tipi mantıksal bilgiyi ifade eder ve alacağı değer true ya da false olabilir. Bu değerler ileride göreceğimiz mantıksal operatörler içerisinde ve if, while, for gibi deyim ve döngüler içerisinde kullanılabilir.

Değişken Bildirimi:
Değişken bildiriminde, ilk önce bu değişkenin içerisinde tutacağı değerlerin hangi türden olacağı ve değişkenin hangi isimle anılacağı bildirilir.

Örneğin:

int a;
double benim_degiskenim;
String isim;

System.out.println() metodu üzerine bir not:
Bu metod java tarafından tanınan ve daha önce yazılmış bir metoddur. Bu metodu kullanırken parantezler içerisine yazılan (parametre olarak yazılan) değişken ve sabitler ekranda görüntülenir. Eğer parantez içerisine bir değişken yazılmış ise bu değişkenin içerisindeki değer ekrana yazılacaktır. Ayrıca + operatörü yardımıyla parantez içerisine birden fazla tür yanyana yazilabilir. Örneğin:

Degiskenler.java

```
public class Degiskenler  {

  public static void main(String arg[]) {
     int a;          //satır 1
     a = 200;        //satır 2
     int b, c;       //satır 3
     b = 100;        //satır 4
     c = 400;        //satır 5
     int d = 0;      //satır 6
     System.out.println("a nin degeri: " + a);
     System.out.println("a nin degeri: " + a + " b nin degeri: " + b);
     d = a + b + c;
     System.out.println("Toplam deger: " + d);
  }
}
```

Program çalıştığında şunu göreceksiniz:

a nin degeri: 200
a nin degeri: 200 b nin degeri: 100
Toplam deger: 700

Bu örnekte aslında değişken bildirimine ilişkin farklı yöntemleri görmekteyiz. Bir değişken satır 1'de olduğu gibi ilk önce yaratılıp daha sonra satır 2'de olduğu gibi içerisine değer verilebilir. Ya da satır 6'da olduğu gibi aynı anda yaratılıp ilk değer verilir. Bunun yanı sıra virgül kullanılarak aynı türe ilişkin olmak üzere satır 3'de olduğu gibi tek satırda birden fazla değişken yaratılabilir. Bu örnekte aynı zamanda System.out.println metodunun kullanımını görmekteyiz.

Özel Karakterler İlişkin Bir Not:
C ve C++ programla dillerinde de olduğu gibi Java'da klavyeden girilen bir takım özel karakterlerin karşılığı vardır. Bu özel karakterler bip sesi, enter tuşu, alt satır başı, tab tuşu gibi karakterlerdir. Bu karakterler kullanılırken başlarına \ işareti koyulur ve uygun sembol yerleştirilir. Buna göre:

\n : alt satır başına
\t : tab tuşu
\\ : \ işareti

\" : " işareti
\b : backspace (bir önceki karakteri silme)
\r : kendisinden önce gelen tüm karakterleri satır başına kadar silme

anlamına gelmektedir.

Şimdi kullanıcıdan bir takım bilgileri alan ve daha sonra bu bilgileri ekranda gösteren bir program yazalım. Bu program çalıştığında Resim 6'daki gibi JOptionPane sınıfının showInputDialog isimli metodu yardımıyla meydana gelen bir pencere aracılığıyla bilgiler kullanıcıdan istenecek ve sonra bundan önceki dersimizde de gördüğümüz JOptionPane sınıfının showMessageDialog isimli metodunun meydana getirdiği mesaj penceresi aracılığıyla girilen bilgiler ekranda tekrar gösterilecektir.

Programımızı yazalım:

İsimGir.java

```
import javax.swing.*;  //s1

public class IsimGir {
  public static void main(String arg[]) {
    String isim, yas;  //s2

    isim = JOptionPane.showInputDialog(null, "Lutfen isminizi giriniz: "); //s3
    yas = JOptionPane.showInputDialog(null, "Simdi de yasiniz giriniz: "); //s4

    String rapor = "Adiniz: "+isim+"\nYasiniz: "+yas;  //s5
    JOptionPane.showMessageDialog(null, rapor);        //s6
    //JOptionPane.showMessageDialog(null, "Adiniz:\t"+isim+"\nYasiniz:\t"+yas);s7

    System.exit(0);
  }
}
```

Bu örnekte görüldüğü gibi ilk olarak kullanıcının gireceği bilgilerin tutulacağı değişkenler s2'de yaratılmaktadır. Eğer bu değişkenler olmazsa kullanıcının gireceği bilgiler bir yerde saklanamayacağı için kullanılamazdı. Daha sonra s3 ve s4'de JOptionPane.showInputDialog() metodu yardımıyla ekranda giriş yapılacak olan input dialog penceresi çıkartılmaktadır. Ancak bu metodun özelliği, çağrıldıktan sonra geriye, kendisine girilen yazıyı değer olarak üretir ve bu değer s3 ve s4'de sırasıyla isim ve yas değişkenlerine aktarılmıştır. s5'de ekranda en son bilgilerin gösterileceği pencereye yazılacak olan yazı hazırlanmış ve rapor isimli değişkene aktarılmıştır. Daha sonra String türündeki bu rapor değişkeni JOptionPane.showMessageDialog() metoduna parametre olarak gönderilerek mesaj penceresi hazırlanmıştır. Aynı şekilde ekranda belirecek mesaj, s5'de olduğu gibi önceden hazırlanabileceği gibi hemen o anda s7'deki gibi parametre olarak da hazırlanabilir. Bu örnekte s5 ve s6 yerine sadece s7'de kullanılabilir.

Not: Başında "//" olan satırların Java derleyicisi tarafından es geçildiğini ve programın sanki bu satırlar yokmuş gibi çalışacağını hatırlayınız.

Sabitler:
Değişken nesnelerin içlerindeki değerlerin değişebileceğinden bahsetmiştik. Bazı durumlarda tüm program boyunca yanlışlıkla da olsa değerinin değiştirilmemesi gereken nesnelere ihtiyaç duyabiliriz. Bu talebi karşılamak için sabitler kullanılmaktadır. Sabitlerin bildirimi aynen değişken bildiriminde olduğu gibidir ancak tek fark sabitlerin bildiriminin başına final anahtar sözcüğü getirilmesidir:

Sabitler.java

```
public class Sabitler {
  public static void main(String arg[]) {
```

```
    final double BIRIM_UCRET = 200.5;

    double sure = 2.3;
    //BIRIM_UCRET = 200.8;          //s1
    System.out.println("Bu ay odeme bedeli: " + BIRIM_UCRET * sure + " liradir.");
  }
}
```

Bu örnekte gördüğümüz gibi BIRIM_UCRET adında ve türü double olan bir sabit ile sure adında ve türü double olan bir değişken tanımladık. Bu durumda siz isteseniz bile bu sabiti s1'de olduğu gibi değiştiremezsiniz. Eğer s1'in başındaki // ifadesini kaldırırsanız programın hata verdiğini göreceksiniz. Sonuç itibariyle sabitleri program içerisinde bir değerle yaratıp, içerisinde hep bu değeri tutmasını garanti edebilirsiniz. Sabitin içerisinde tutacağı değer ancak bilinçli olarak ve ilk defa verildiği yerde değiştirilebilir.

Operator Kavramı ve Operatörlerin İncelenmesi:
Operatörler sayısal ya da mantıksal ifadelerin birbirleri ile işleme sokulmasını sağlayan araçlardır. Program yazarken bir çok defa hesap yapmak gerekecektir. Bu sayısal hesapların yapılabilmesi için çeşitli sayısal operatörler kullanılmaktadır. Ayrıca yine operatörler yardımıyla bazen de sayısal olmayan ve kelimelerin karşılaştırılması, eşitliklerin sağlanması ve bunun gibi birbirinden farklı mantıksal işlemler de yapmak gerekmektedir.

Not: Şunu kesinlikle unutmayınız ki; az sonra bazılarını öğreneceğimiz operatörler, kullanıldıkları zaman geriye mutlaka bir değer üretirler. Üretecekleri bu değerler bir sayı, true ya da false gibi boolean bir değer veya bunun gibi operatöre göre değişen değerler olabilir. Ama sonuçta operatörün bir değer ürettiğini ve işlem sonucunda ortada sadece bu değerin kaldığını anlamaya çalışınız. Şimdi bu operatörleri inceleyelim:

= operatoru
Bilinen en genel operatörlerden olan eşitlik operatörü çift taraflı kullanılır. Sağ tarafında yer alan değeri sol taraftaki değişkene atamaya yaramaktadır. Daha önce yaptığımız IsimGir isimli örnekte s3 ve s4'de de olduğu gibi, bazen metodlardan gelen değerleri de değişkenlere aktarmaya yarar. Aslında mantık hep aynıdır. Sağdaki değer soldaki değişkene aktarılır. Bu operatör kullanıldığında, ilk once sağ taraf hesaplanmaktadır.

+, -, *, / operatorleri:
Bu operatörler klasik matematik işlemlerinin hesabında kullanılmaktadır. Ancak istisna olarak + operatorünün daha önce de gördüğümüz IsimGir örneğindeki s5'de olduğu gibi Stringleri birbirleri ile ya da String ve değişkenleri birbirleri ile bağlamak gibi işlemlerde de kullanılması mümkündür.

+=, -= /= ve *= operatörleri:
Bu operatörler birleşik operatörlerdir. Genellikle bir değişkenin değerini eski değeri ve yeni bir sabit artımının toplamı, farkı, bölümü veya çarpımı şeklinde değiştirmek için kullanılır. Örnegin:

```
int x = 10;
x = x  + 4;
```

işleminde x değişkeninin son değeri 14 olacaktır. Bu işlem aynı şekilde += operatoru kullanılarak da yapılabilir. Şöyle ki:

```
int x = 10, sayi = 6;
x +=  4;   //x = x + 4 ile aynı anlamdadır.
sayi = sayi – 6;  //sayi -= 6;
```

++ ve –- operatorleri:
Bu operatorlerin her ikisi de tamamen aynı mantıkta olup verinin değerini 1 artırmak ya da1 eksiltmek amacıyla kullanılır. Ornegin x = x +1; yazmak yerine x++; yazabilirsiniz. Aynı şekilde x = x -1; yazmak yerine x--; yazabilirsiniz.

boolean Operatörleri :

Daha önce de belirttiğimiz gibi eğer bir değişken boolean türünde ise alabileceği sadece iki tane değer vardır. Bu değerler "true" ya da "false" değerleridir. Bu iki değeri bilgisayar mantığında 1 ya da 0 a benzetebiliriz.

Bu iki değeri içerisinde saklamak amacıyla kullanılan değişkenler de boolean türünden olacaktır. Bazı operatörler kullanıldıkları zaman sonuç olarak bu iki değerden birtanesini üretirler. Bu operatörlere mantıksal operatörler denilir.

== ve != operatorleri:
Bu operatör aslında "eşit mi? " sorusunun cevabını üreten mantıksal bir operatördür. Çift taraflı olarak kullanıldığında sağdaki ve soldaki değerlerin eşit olması durumunda "true" sonucunu üretir. Eşitlik yok ise "false" sonucu üretilecektir. Mesela 3 == 5 işlemi false değerini üretir.

== operatorune benzer olan, ancak tam tersini yapan diğer bir operatör de != operatörudur. Bu operatör de yine çift taraflı olarak çalışır ve her iki tarafın birbirine eşit olmaması durumunda "true", eşit olmaları durumunda da "false" değerini üretir. Örneğin 3 != 5 işlemi true değerini üretir.

&& ve || (and ve or) Operatorleri:
Bu iki operator mantık işlemlerinde kullandığımız and ve or operatörleridir. Bunlar da çift taraflı kullanılır.

&& operatoru her iki tarafındaki değer true ise true sonucunu üretecek ama sağ ya da sol taraftan bir tanesi false olması durumunda false değerini üretecektir.

|| operatoru de iki tarafındaki değerlerden en az bir tanesi true olması durumunda true değerini üretecek, her iki tarafın da false olması durumunda false değerini üretecektir.

AnaSinif.java

```
import javax.swing.*;

public class AnaSinif{
    public static void main(String arg[]){
        boolean deger1 = true;
        boolean deger2 = false;
        boolean sonuc1  = deger1 && deger2;

        JOptionPane.showMessageDialog(null, "deger1: " +deger1+"deger2: "+deger2+
                                    "deger1 && deger2 ->"+ sonuc1);

        boolean sonuc2 = deger1 || deger2;
        JOptionPane.showMessageDialog(null, "deger1: " +deger1+"\ndeger2: "+deger2+
                                    "\ndeger1 || deger2 -> "+ sonuc2);

        deger2 = true;
        sonuc1 = deger1 && deger2;
        JOptionPane.showMessageDialog(null, "deger1: " +deger1+"\ndeger2: "+deger2+
                                    "\ndeger1 && deger2 -> "+ sonuc1);

         deger1 = false;
         deger2 = false;
         sonuc1 = deger1 || deger2;
         JOptionPane.showMessageDialog(null, "deger1: " +deger1+"\ndeger2: "+deger2+
                                    "\n\ndeger1 && deger2 -> "+ sonuc1);
    }

}
```

>, <, >=, <= operatöleri:
Bu operatörler de çift taraflı çalışırlar ve eğer sağ sol değerleri gerçeklenirse true aksi taktirde false değerini üretirler. Örneğin:

3 > 5 //false

4 < 10 //true
3 >= 3 //true

() Tür Dönüştürme Operatörü
Şu ana kadar öğrendiğimiz kadarıyla belli bir türe ilişkin değerleri aynı türden değişkenler içerisinde saklıyor ve kullanabiliyorduk. Ancak bazı durumlarda farklı türlerin birbirleriyle işleme girmeleri ve bir türe ilişkin bir değerin aynı olmayan türde bir değişkene atanması gibi işlemler gerekebilir. İşte bu gibi durumlarda bir tür dönüşümü söz konusu olabilir.

Tür dönüşümleri de belirli kurallar dahilinde ve belirli bir hiyerarşi ile olmaktadır. Tür dönüşümleri sırasında bazen bilgi kaybının da olabileceğini unutmamak gerekir. Bunun nedenin her türün ifade ettiği bilginin sınırlarının ve hafızada kapladığı alanın farklı olmasından kaynaklandığını hatırlayınız. Aşağıda bilgi legal olan tür dönüşümleri özetlenmektedir:

byte -> short (bilgi kaybı yok)

short -> int (bilgi kaybı yok)

char -> int (bilgi kaybı yok)

int -> long, double (bilgi kaybı yok), float (bilgi kaybı olabilir)

long -> float, double (bilgi kaybı olabilir)

float -> double (bilgi kaybı yok)

Bilgi kaybı, dönüşümlerde hedef veri tipinin sınırlarının dar olmasından kaynaklanır. Mesela

int n = 123456789;
float f = n; //f in degeri 1.2345692E8 olur.

Yukarıdaki dönüşüler otomatik olarak gerçekleşen dönüşümlerdir. Ancak kasıtlı olarak da tür dönüşümü yapılabilir.Bir değerin türünü dönüştürmek için değerin soluna parantez içerisinde değiştirmek istediğiniz türün adını yazmanız gerekir.

```
public class AnaSinif{
          public static void main(String arg[]){
          double k = 12.96;
          System.out.println("k nin deeri: "+ k);
          int c;
          c = (int)k; //burada double türünden bir deer int türünden bir degiskene
atanyor.
          System.out.println("c nin degeri: "+c);
          }
}
```

Bu örnekte gördüğümüz gibi k değişkeninin içerisindeki değer c değişkenine atandığı için c nin değerinin de 12.96 olmasını beklerdik. Ancak atama sırasında ( ) operatörü ile bir tür dönüşümü yaptık ve bilgi kaybından dolayı c nin değeri 12 oldu. Çünkü int türünden bir değişken virgülden sonra değer alamaz.

Tür Dönüşümlerine İlişkin Özel Metodlar
Tür dönüşümlerini ( ) operatüörü ile yapabileceğimiz gibi bu iş için yazılmış bazı özel metodları da kullanabiliriz. Mesela String olarak verilmiş olan "12" değerini 12 sayısı gibi kullanamayız. Çünkü bu değer çift tırnak içerisinde

verilmiştir ve bir String'dir. Yani siz bu değerle bir sayıyı toplamaya kalkarsanız sonuç da otomatik tür dönüşümünden dolayı bir String olacaktır. Şu örneğe dikkat edelim:

```
public class AnaSinif{
      public static void main(String arg[]){
      String a = "12";
      int b = 13;
      String c = a + b;
           System.out.println("Sonuc: "+c);
           }
}
```

Gördüğünüz gibi java a+b işleminin sonucunu bir String türü olarak belirlemek konusunda bizi zorlamaktadır. Sonucu string olarak hesapladığımzda da bu sefer cevap 25 olması gerekirken 1213 tür. Aslında bu sonuç bir sayı değil bir Stringdir. Yani bir görüntüdür. O halde sayı gibi verilmiş bir Stringi nasıl sayıya çevireceğiz? Bunun için kullanılacak metod Integer isimli bir sınıfın parseInt() isimli metodudur. Mesela kullanıcıdan JOptionPane.showInputDialog() metodu ile yaşını isteylelim ve girdiği yaş ile 3 sayısını çarpıp sonucu bulalım. Ancak biliyoruz ki bu şekilde alınan girişler sayı olarak değil String olarak algılanıyorlar. O halde bu String'i de Integer.parseInt() metodu yardımıyla int türüne dönüştürelim:

```
import javax.swing.*;
public class AnaSinif{
          public static void main(String arg[]){
                    String yas = JOptionPane.showInputDialog(null, "Yasinizi
          giriniz: ");
                    int sonuc = Integer.parseInt(yas) * 3;
                    JOptionPane.showMessageDialog(null, "Hesap sonucu: "+sonuc);
          }
}
```

Buna benzer diğer metodlar:
Integer.toString(12); --> parantez içerisinde verilen int türünü String'e çevirir.
Double.parseDouble(12); --> parantez içerisinde verilen int türünü double türüne çevirir.
Double.toString(12.45); --> parantez içerisinde verilen double türünü String'e çevirir.

Bunlar gibi daha birçok metod vardır. Bunların kullanımı yeri geldikçe dökümantasyon yardımıyla öğrenilebilir.

Stringler
Daha önce sözdizimi olarak da bahsettiğimiz bu tür aslında bir karakter dizisidir. Stringler çift tırnak içerisinde yazılır ve kendilerine ait bir takım özel metodlara sahiptir. Bu bakımdan diğer türlerden biraz daha farklıdır. Burada istenen, Stringlerin metodlara sahip olduğu kavramını vurgulamak ve gerektikçe bu metodları kullanabileceğimizi göstermektir. Bu amaçla bütün metodlar burada anlatılmayacaktır:

"string".length Değişkeni
Bu değişken bir stringin kaç karakterden oluştuğunu yani uzunluğunu verir.

"string".equals("diger string") Metodu
İki String türüne ilişkin değerlerin eşitliğini sayısal değerlerde kulandığımız == operatörü ile kontrol edemeyiz. Bunun yerine String.equals metodunu kullanırız. Bu metod, kendisine parametre olarak verilen "diger string" değeri eğer "string" degerine eşitse true, degilse false degerini üretir.

"string".compareTo("diger string") Metodu:

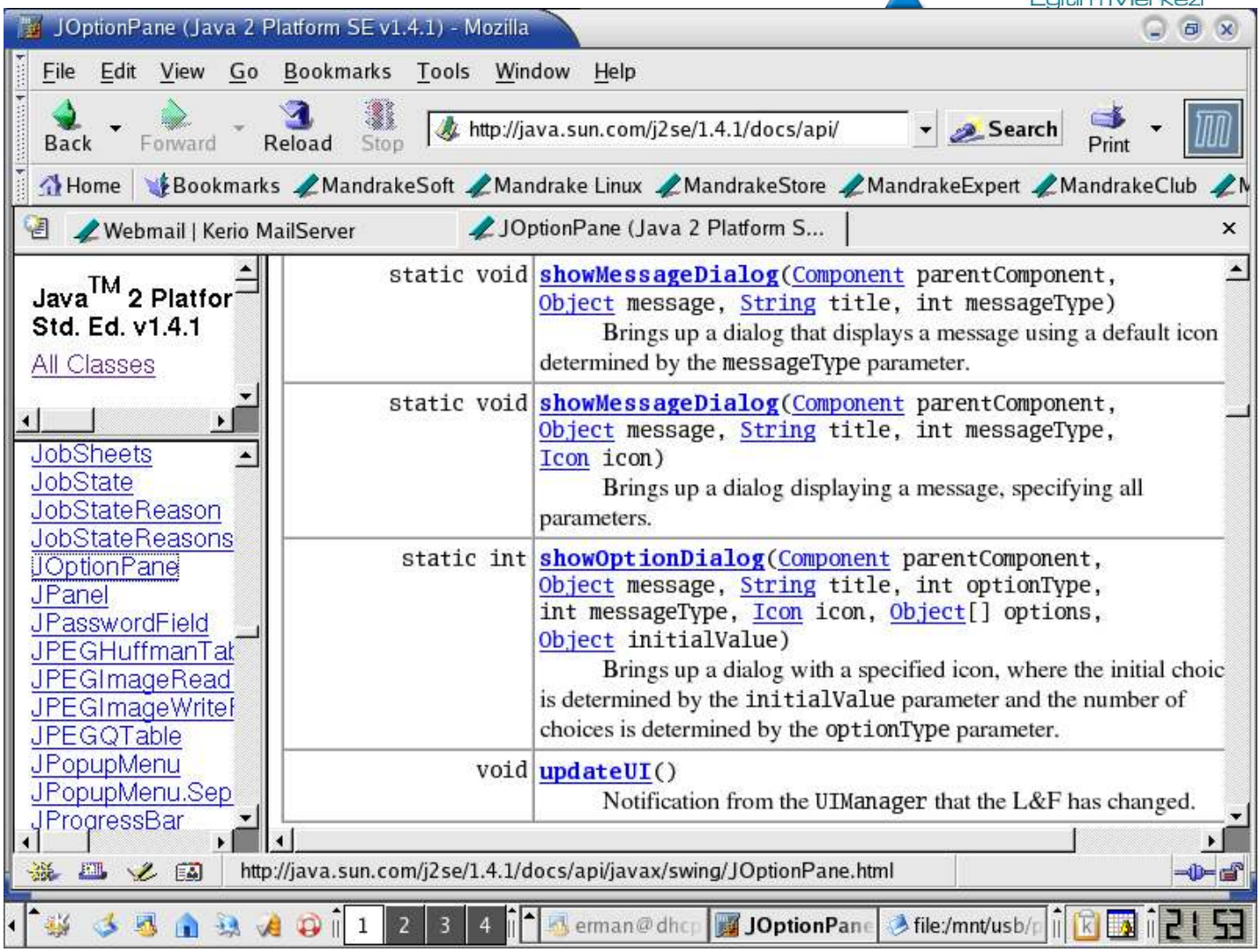
Bu metod kendisine parametre olarak verilen "diger string" ile "string" değerini kıyaslar. Eğer iki string eşitse metod 0 değerini, "string" büyükse 1 değerini, "diğer string" büyükse de -1 değerini üretir.

```
import javax.swing.*;
public class AnaSinif{
          public static void main(String arg[]){
                    String isim = JOptionPane.showInputDialog(null, "Lutfen adnizi
          giriniz");
                    int uzunluk = isim.length;
                    String mesaj = "Isminiz "+uzunluk+" tane karakterden
          olumaktadir";
                    JOptionPane.showMessageDialog(null, mesaj);

                    String gercekSifre = "kkVVuuNN";
                    String girilenSifre = JOptionPane.showInputDialog(null, "Sifreyi
          giriniz: ");
                    boolean sonuc = gercekSifre.equals(girilenSifre);
                    JOptionPane.showMessageDialog(null, "Girdiiniz sifre: "+sonuc);
          }
}
```

Yukarıdaki örnekte, JOptionPane sınıfının showInputDialog() isimli metodu yardımıyla kullanıcının klavyeden girdiği string, isim değişkenine atılmakta ve daha sonra da girilen bu stringin uzunluğu öntanımlı isim.length değişkeni ile alınıp uzunluk değişkenine atanmaktadır. Biz aslında length isminde bir değişken yaratmadığımız halde, yukarıda anlatıldığı gibi isim değişkeni aslında bir String olduğu için, kendi bünyesinde otomatik olarak bu length değişkenini ve diğer String metodlarını barındırmaktadır. Nitekim örnekte de gördüğümüz gibi; gercekSifre isimli String değişkeni ile equals isimli öntanımlı metodu kullanarak, iki stringi birbirleriyle kıyaslayabilmekteyiz.

Vazgeçilmez Kaynağınız: Java Dökümantasyonu!
Daha önce de sık sık Java'da bir çok sınıfın kullanım amacı ile daha önceden yazılmış olduğuna ve bu sınıfların içerisinde de, kullanabileceğimiz bir çok değişken ve metod olduğuna değinmiştik. Hangi sınıfların ne işe yaradıklarını, ve içlerinde hangi metod ve değişken barındırdıklarını ve bunları nasıl kullanabileceğimizi ezbere bilmek imkansızdır. İşte bu yüzden, her java programcısının vazgeçemeyeceği java dökümantasyonunu kullanmak gerekir.

Dökümantasyonu online olarak http://java.sun.com/j2se/1.4.1/docs/api/ adresinde bulacağınızı ilk dersimizde söylemiştik.



Bu sayfayı açtığınızda karşınıza üç çerçeveden oluşmuş bir ekran gelmektedir. Bu ekranda, sol üst kısımda "All Classes" isimli bağlantıyı tıkladığınızda, sol at kısma Java'da öntanımlı tüm sınıfları temsil eden bağlantılar yüklenecektir. Hakkında bilgi almak istediğiniz sınıf ismini bulup üzerine tıkladığınızda da sağdaki kısma bu sınıf hakkındaki tüm metodlar, değişkenler ve bilgiler gelecektir. Resim 7'de çok sık kullandığımız JOptionPane isimli sınıfın showMessageDialog isimli metoduna ilişkin bölümü görmekteyiz.

Bu dökümantasyonu online olarak görüntüleyebileceğiniz gibi, bilgisayarınıza indirip offline olarak da kullanabilirsiniz. Bu işlem için http://java.sun.com/j2se/1.4.1/download.html sayfasından dökümantasyonu indirebilir ve yine aynı sayfada bulacağını kurulum talimatları ile (Installation Instructions) sisteminize yükleyebilirsiniz.

Java ile Programlama
Bölüm 3:

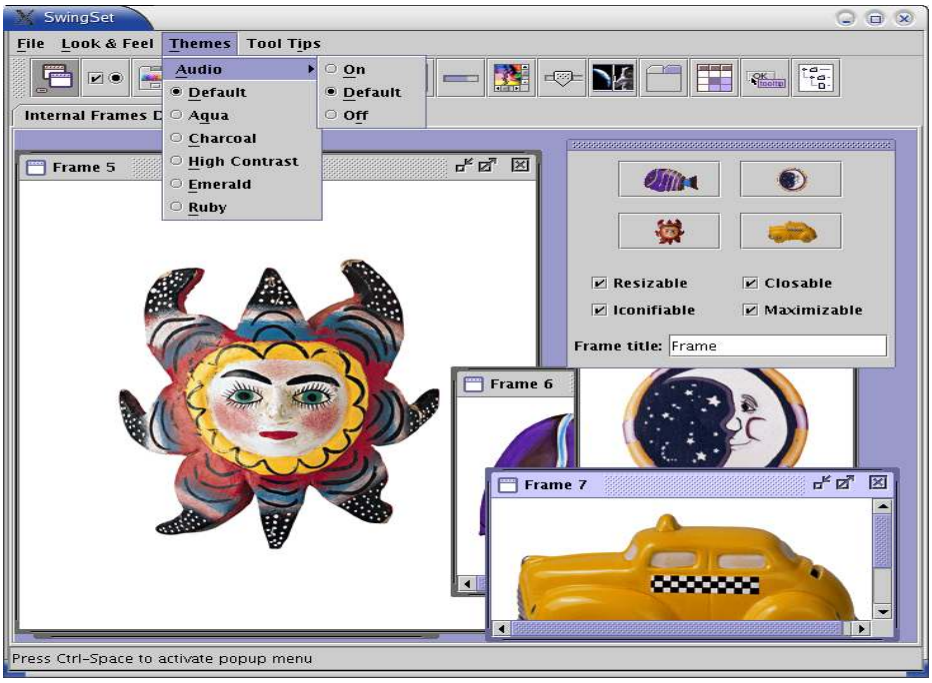
Bundan önceki dersimizde Java programlama dilinde değişkenlerin nasıl kullanıldıklarını operatörleri ve vazgeçilmez kaynağınız olan Java Dökümantasyonu'nu nasıl kullanacağımız öğrenmiştik. Bu sayımızda işi biraz daha ileri götürecek ve program akışını kendi isteğimiz doğrultusunda yönlendirebileceğimiz deyimler ve döngüleri öğreneceğiz.

Yeni bir dil öğrenmenin en zor yanı, dilin temel kurallarını öğrenme aşamasının oldukça yoğu ve sıkıc geçmesidir. Bunun nedeni de öğrenme aşamasında sınırlı bilgiyle henüz ele avuca sığar uygulamalar geliştiremiyor olmamızdır. Sizlerin de şu aşamada aynı sıkıntıyı çektiğinizi tahmin ederek, motivasyonunuzu artırmak amacıyla java içerisinde yer alan demoları göstermek istiyorum. Az sonra nasıl çalıştırabileceğinizi anlatacağım bu uygulamaları inceleyerek, ileride varacağınız noktayı görme şansına sahip olacaksınız:

Okumaktan Sıkılanlar İçin
Demolar sdk ana dizini içerisinde bulunan "demo" isimli dizinde bulunmaktadır. Sdk ana dizininiz sizin java'yı ilk yüklediğiniz yerdir ve ismi de muhtemelen j2sdk.1.3.1 ya da j2dk1.4.2 şeklinde olacaktır. Benim linux sistemimde ana dizinin tam yolu: **/opt/j2sdk_nb/j2sdk1.4.2/** şeklindedir. Buradaki demo isimli dizinin içinde de, jfc ve onunda için de SwingSet2.jar adındaki uygulamayı önce jfc dizini içerisine girerek şu şekilde çalıştırabilirsiniz:

java -jar SwingSet2.jar

Bu uygulamayı dilediğiniz gibi karıştırabilir hatta j2sdk dizini içerisindeki demo dizinin ve altındaki diğer tüm uygulamaları da inceleyebilirsiniz.



Java'da Deyim ve Döngüler:
Program yazarken, yazdığımız kodların yukarıdan aşağıya doğru sırasıyla çalıştırıldığını biliyoruz. Ancak bazen bu kodların bu sırada değil de belirli şartlara bağlı olarak birkaç satırın atlanıp ileriden devam edilmesi ya da aynı satırların birden fazla sayıda çalıştırılması gibi isteklerimiz olabilir. İşte bunun için kullnacağımız bazı deyimler ve döngüler vardır. Bu deyim ve döngüler sayesinde normalde her satırdan bir defa geçen ve yukarıdan başlayarak aşağı satırlara doğru ilerleyen program akışını, bazı satırları belli durumlarda okumadan, ya da bazı satırları birden fazla defa okuyacak şekilde yönlendirmemiz mümkün hale gelir.

Ancak bu deyim ve döngülerin daha iyi anlaşılması açısından öncelikle faliyet alanı kavramına değinmekte yarar var.

Faliyet Alanı Nedir Ne İşe Yarar?
Faliyet alanı, yazdığımız program içerisinde belirli bölgeleri temsil eden ve nesnelerin etkin olduğu sınırları belirleyen alanlardır. Bir nesnenin etkin olduğu bölgeye, o nesnenin "faliyet alanı" denilmektedir.

Faliyet alanı kod içerisinde { ve } ile belirlenmektedir. Bu iki parantez arasında mevcur bir faliyet alanı vardır. Bir değişken eğer { ve } arasında tanımlanmışsa bu değişken bu iki parantez dışında kullanılamaz. Buna göre değişkenin faliyet alanı da bu iki parantez içerisinde ya da yaratıldığı alan içerisinde sınırlıdır.

```
/**** Main.java ****/

import javax.swing.*;

class Main {
  public static void main(String[] args) {
    int k = 10;
    {
      int c = 20;
      JOptionPane.showMessageDialog(null, "k nin degeri: "+k+
                                    "\nc nin degeri: "+c);
    }

    System.exit(0);

  }
}

/******************/
```

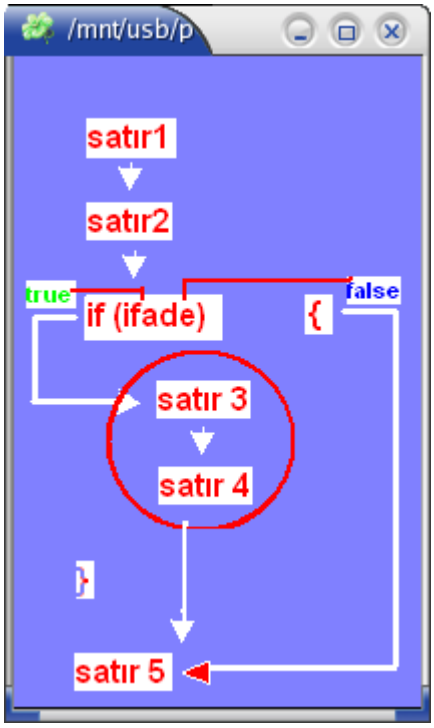
Bu örnekte iki tane faliyet alanı görmekteyiz. Birinci faliyet alanı sınıfımızın kendisidir ve sınıf adı Main yazıldıktan hemen sonra { ile açılmış ve tüm satırlardan sonra da } ile kapatılmıştır. Bu faliyet alanının içinde de k değişkeni tanımlandıktan sonra yine { ile başka bir faliyet alanı açılmış ve bu yeni faliyet alanında da sadece c değişkeni yaratılmıştır. Ancak dikkat ederseniz bu örnekte, sınıfın faliyet alanı, içeride açılan faliyet alanını kapsamaktadır. Bu yüzden k değişkeni tanımlanmamış olduğu içerideki faliyet alanı içerisinde de kullanılabilir. Ancak tersi mümkün değildir. Yani c değişkenini `k'nin tanımlı olduğu faliyet alanında kullanmaya kalksaydık hata mesajı alacaktık.`

```
/**** Main2.java ****/

import javax.swing.*;

class Main2 {
  public static void main(String[] args) {
    int k = 10;
    {
      int c = 20;
    }
    JOptionPane.showMessageDialog(null, "k nin degeri: "+k+
                                  "\nc nin degeri: "+c);

    /*c degiskeni bu faliyet alanında tanımlı degildir*/

         System.exit(0);
  }
}

//bu program çalışmayacaktır

/******************/
```

Faliyet alanına ilişkin unutulmaması gereken bir şey daha vardır: Aynı faliyet alanı içerisinde, aynı isimli birden fazla değişken tanımlanamaz. Bu örnekte k nın faliyet alanı, c nin faliyet alanını da kapsadığından, c nin tanımlandığı yerde k adında başka bir değişken daha tanımlayamazsınız. Çünkü başka faliyet alanı olsa da, içerideki faliyet alanında yukarıda tanımlanan k değişkeni hala geçerlidir. Buna dikkat ediniz.

Faliyet alanına ilişkin bu bilgilerle artık deyim ve döngüleri inceleyebiliriz.

if Deyimi:
if deyimi belli bir şart sağlanması durumunda program akışını kendi faliyet alanı içerisine yönlendirir. Kendi faliyet alanının en son satırına kadar kodları çalıştırır ve daha sonra da program akışı if deyiminin faliyet alanının bittiği yerden devam eder.




```
/***** Deyimler1.java ********/

import javax.swing.*;

class Deyimler1 {
  public static void main(String[] args) {

    String notStr;
    int not;

    notStr = JOptionPane.showInputDialog(null, "Notunuzu giriniz: ");
    not = Integer.parseInt(notStr);

    if(not > 50) {
      JOptionPane.showMessageDialog(null, "Tebrikler. Bu dersten gectiniz :)");
```

```
    }

    System.exit(0);
  }
}
/**********************/
```

Daha teknik söylemek gerekirse, şartın sağlanması demek, aslında if deyiminin kontrol parantezleri içerisindeki ifadenin true olması demektir. (Bundan önceki sayıda anlattığımız, "boolean türü ve boolean operatörleri" konusunu hatırlayınız!). Aynı şekilde bu şartın sağlanmaması ise ifadenin false değerini üretmesi anlamına gelir. Yukarıdaki örnekte not > 50 ifadesi eğer false olursa ekrana hiç bir mesaj gelmeyecektir. Ancak true olursa "Tebrikler. Bu dersten geçtiniz" mesajını alırsınız. Burada girdiğiniz notun aslında bir String türünde olduğunu ve bu değerin Integer sınıfının parseInt metoduyla sonradan integer türüne dönüştürlüp kullanıldığına dikkat ediniz. (Bundan önceki sayıda yer alan "Tür Dönüşümlerine İlişkin Özel Metodlar" konusunu hatırlayınız.)

Peki ya eğer kullanıcı notu 50'den küçük olsa da bir mesaj almak istiyorsa ne yapacak. İşte bu durumda

```
.....
if(ifade) {
      ....
}

else {
      ....
}
```

kalıbını kullanmak gerekecektir. Bu kalıba göre program akışı, eğer if içerisindeki ifade true ise if deyiminin, değilse else deyiminin faliyet alanı içerisine yönlencektir. Az önceki örneği genelleştirelim:

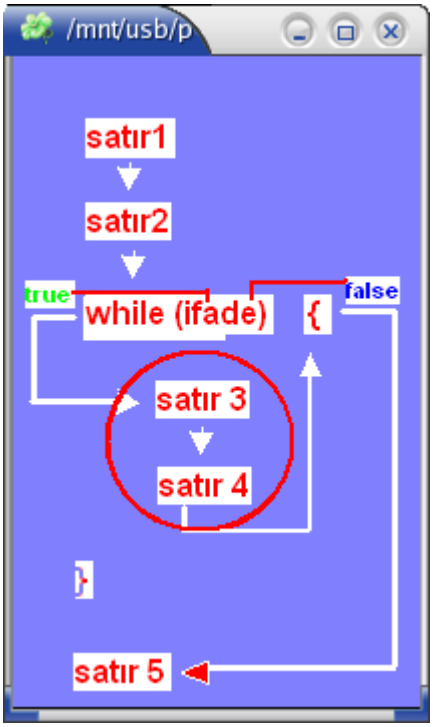
```
/***** Deyimler2.java ********/

import javax.swing.*;

class Deyimler2 {
  public static void main(String[] args) {

    String gectiMesaji = "Tebrikler! Bu dersten gectiniz!";
    String kaldiMesaji = "Malesef bu dersten kaldiniz.";
    String notStr;
    int not;

    notStr = JOptionPane.showInputDialog(null, "Notunuzu giriniz: ");
    not = Integer.parseInt(notStr);

    if(not > 50) {
      JOptionPane.showMessageDialog(null, gectiMesaji);
    }

    else {
      JOptionPane.showMessageDialog(null, kaldiMesaji);
    }

    System.exit(0);
  }
}
/**********************/
```

Aktif olarak projelerde çalıştığınız zaman kullanıcı isteklerinin bitmediğini göreceksiniz :) Diyelim ki bu sefer de kullanıcı programa girdiği nota göre bir harf ile derecesini öğrenmek istiyor olsun. Bu durumda girilen not belli aralıklara göre belli dereceler alacaktır ve bu dereceler 2 taneden fazla olabilir. O zaman da if else kalıbı işe yaramayacaktır. Bu durumlarda if, else if, else if, ... , else kalıbını kullanmak gerekir. Örneği inceleyiniz:

```
/****** Deyimler3.java ******/
import javax.swing.*;

class Deyimler3 {

  public static void main(String[] args) {

    String kaldiMesaji = "Dereceniz F: Bu dersten kaldiniz.";
    String yanlisNot = "Yanlis not girisi! Tekrar deneyiniz.";
    String notStr;
    int not;

    notStr = JOptionPane.showInputDialog(null, "Notunuzu giriniz: ");
    not = Integer.parseInt(notStr);

    if(not > 100 || not < 0) {
      JOptionPane.showMessageDialog(null, yanlisNot );
    }

    else if(not <= 100 && not > 86) {
      JOptionPane.showMessageDialog(null, "Dereceniz: A");
    }

    else if(not <= 85 && not > 71) {
      JOptionPane.showMessageDialog(null, "Dereceniz: B");
    }

    else if(not <= 70 && not > 66 ) {
      JOptionPane.showMessageDialog(null, "Dereceniz: C");
    }

    else if(not <= 65 && not > 50) {
      JOptionPane.showMessageDialog(null, "Dereceniz: D");
    }

    else {
      JOptionPane.showMessageDialog(null, kaldiMesaji);
    }

    System.exit(0);
  }
}
```

while Döngüsü:

Kullanıcı istekleri hala bitmiyor :) Şimdi de kullanıcımız bizden program çalıştığında sürekli notu sormasını ve buna karşılık düşen dereceyi bize vermesini istiyor. Ne zaman ki giriş "Q" karakteri olur, o zaman program sonlansın diyor. Bu durumda not girişi bir döngü içerisine sokulmalıdır. Bu döngünün en başında önce girişin Q olup olmadığına bakılmalı sonra ya döngü tekrar dönmeli ya da sonlandırılmalıdır. Bunun için en uygun döngü while döngüsü olacaktır. while, eğer içerisindeki ifade true ise program akışını kendi içine yönlendirir. Program while faliyet alanının en sonuna geldiğinde tekrar while döngüsünün ilk satırına döner ve ifadeyi yine kontrol eder. Bu işlem, while içerisindeki ifade false olana kadar devam eder.



Not: Bu tür döngülerde, eğer döngünün faliyet alanı içerisindeyken, while içerisideki ifadeyi değiştirecek bir komut yazmazsanız, ilk durumda bu ifade hep true olduğu için ve döngü içinde de hiç değişmediği için sonsuz bir döngüye girersiniz.

```
while(ifade) {
 satir 1
 satir 2
 ...
 satir n

 /*şSimdi akisş  tekrar while(ifade) satirina
   yönlenecek ve kontrol yapılacaktır. eğer
   ifade false olmussa program akışı
   satir n+1 den devam etmektedir.*/
}

satir n+1
```

Bu durumda kullanıcımızın yeni isteğine göre örneği tekrar düzenleyelim:

```
/**** Deyimler4.java ****/

import javax.swing.*;

class Deyimler4 {
  public static void main(String[] args) {

     String yanlisNot = "Yanlis not girisi! Tekrar deneyiniz.";
     String giris = "Bir not giriniz (0 - 100)\nÇikis için notu -1 giriniz";
     String notStr;
     int not;

     notStr = JOptionPane.showInputDialog(null, giris);
     not = Integer.parseInt(notStr);

     while(not != -1) {
```

```
      if (not > 100) {
        JOptionPane.showMessageDialog(null, yanlisNot);
      }
      else if (not <= 100 && not > 86) {
        JOptionPane.showMessageDialog(null, "Not: " +
                                    not + " Derece: A");
      }
      else if (not <= 85 && not > 71) {
        JOptionPane.showMessageDialog(null, "Not: " +
                                    not + " Derece: B");
      }
      else if (not <= 70 && not > 66) {
        JOptionPane.showMessageDialog(null, "Not: " +
                                    not + " Derece: C");
      }
      else if (not <= 65 && not > 50) {
        JOptionPane.showMessageDialog(null, "Not: " +
                                    not + " Derece: D");
      }
      else {
        JOptionPane.showMessageDialog(null,  "Not: " +
                                    not + " Derece: F");
      }
      notStr = JOptionPane.showInputDialog(null, giris );
      not = Integer.parseInt(notStr);
    }

    JOptionPane.showMessageDialog(null, "Program Sonlandi.");
    System.exit(0);
  }
}

/*************************/
```

break Anahtar Sözcüğü:

Bazen bir döngü içerisindeyken, aslında döngünün bitmesine neden olacak şart sağlanmadığı halde özel olarak döngüyü sonlandırmak istediğimiz durumlar doğacaktır. Bu durumların sağlanması neticesinde program break komutunu gördüğü yerde içerisinde bulunduğu ilk döngüyü (çünkü iç içe döngüler de olabilir) sonlandıracak ve döngüden sonraki satırdan itibaren çalışmaya başlayacaktır.

```
/**** Sifre.java ****/

import javax.swing.*;

public class Sifre {
  public static void main(String arg[]) {

    String kontrol_isim = "elvantasbas";
    String k_adi = JOptionPane.showInputDialog(null, "Lutfen kullanici adini
giriniz:");
    boolean x_flag = false;

    while(!k_adi.equals(kontrol_isim)) {

      if(k_adi.equals("x")) {
        JOptionPane.showMessageDialog(null, "Islem sonland?r?l?yor..");
        x_flag = true;
        break;
      }

      k_adi = JOptionPane.showInputDialog(null, "Girdiginiz isim yanlis. Tekrar
```

```
giriniz:");
     }

     if(x_flag) {
          JOptionPane.showMessageDialog(null, "Hoscakalin");
     }

     else {
          JOptionPane.showMessageDialog(null, "Merhaba "+k_adi);
     }

     System.exit(0);
   }
}

/*******************************/
```

Programda while parantezi içerisinde yazılı olan **while(!k_adi.equals(kontrol_isim)** komutuna göre eğer **k_adi** kontrol_isim'e eşit olursa **k_adi.equals(kontrol_isim)** metodu **true** değerini üretecektir. Ancak bu metodun başında yazılı olan **! o**peratörü bu değerin değilini alacak ve **false** değeri üretecektir. Demek ki kullanıcı adı doğru girilmediği sürece bu while parantezi içerisinde **true** değeri olacak, doğru girldiği zaman ise false değeri olacak ve dolayısı ile while bloğuna girilmeyecektir.

Bu örnekte kullanıcı, kullanıcı adını yanlış girdiği ama x girmediği sürece while döngüsü bloklarında kullanıcıdan tekrar isim girmesi istenecektir. İsim yanlış girildiğinde ve x girildiğinde program while içerisindeki if bloğuna girecek ve burada işlem sonlandırılıyor mesajı ile döngüden **break** sayesinde çıkacaktır. Döngüden çıkmadan hemen önce **x_flag** değişkeninin değeri de **true** yapıacaktır. Bu değişken döngüden x ile çıkılıp çıkılmadığı bilgisini tutmak için kullanılan bir ara değişkendir. Döngüden çıkıldıktan sonra eğer x değişkeninin değeri **true** ise while blokları bitimindeki if bloklarına girilecek ve ekrana hoşçakalın mesajı yazılacaktır. Eğer döngüden x yazıldığı için değil de kullanıcı adı doğru girildiği için çıkılmış ise x değişkenin değeri **false** kaldıdığı için if bloğuna girilmeyecek onun yerine else bloğuna girilecektir. Buradaki mesaj ekrana geldikten sonra else bloğundan da çıkılacak ve **System.exit(0)** metodu çalıştırılarak program sonlandırılacaktır.

for Döngüleri:
Şu ana kadar bir koşul sağlandığı sürece otomatik olarak döngüye girilmesini while deyimi ile yapabileceğimizi gördük. Ancak bu döngü koşul sağlandığı sürece, belli bir tekrar sayısı kadar değil her zaman devam ediyordu. Koşul sağlansa bile belirli bir tekrar sayısı da belirlemek istiyorsak o zaman for döngüsünü kullanmak gerekir. for döngüsü şöyle çalışmaktadır:

```
for( <ilk deger> ; <kosul> ; <ifade> ) {
      …
}
```

Buna göre ilk değer ile başlayan döngü ile program for bloğuna girdikten sonra bloğun sonuna geldiğinde koşulun sağlanıp sağlanmadığını kontrol edecek ve koşul sağlanıyorsa ifade yi gerçekleyip tekrar döngüye girecektir. Örneğin:

```
/**** ForDongusu.java ****/

public class ForDongusu {
  public static void main(String arg[]){

      for(int i = 0; i < 10; i++) {

        System.out.println("i nin degeri: "+i);

      }
  }
}
/***********************/
```

```
Bu program çalıştığında ekrana 10 satır boyunca

i nin degeri: 0
i nin degeri: 1
...
i nin degeri: 9
```

yazacaktır. Program akışı her döngüye girişinde i nin değeri ekrana yazılmakta ve blok sonunda i++ ifadesinden dolayı bu değer bir artırılmakta ve koşulun sağlanıp sağlanmadığına bakılmaktadır.

switch Deyimi:
switch deyimi kullanımı da, birden fazla else if kullanımına alternatiftir. Mesela yazdığınız programda kullanıcıdan bir giriş isteyebilirsiziniz. Kullanıcının gireceği bu değer 4 tane değerden birtanesi olabilir ve siz olası her değer içöin bir işlem yapmak istiyor olabilirsiniz. Bunun için

```
if(deger 1) {

}

else if(deger 2) {

}

...

else {

}
```

yapısını kullanabilirsiniz. Ancak böyle durumlarda switch daha iyi bir alternatif olacaktır. switch deyimi de şöyle çalışmaktadır:

```
switch( <degisken> ) {
      case deger1:
            ifade ya da islemler;
            break;

      case deger2:
            ifade ya da islemler;
            break;


      ...
      ...
      case deger n:
            ifade ya da islemler;
            break;

      default:
            ifade ya da işlemler;
            break;
}
```

Bu yapıya göre program akışı, degisken içindeki değere göre case anahtar sözcüğü ile belirlenmiş uygun satıra atlar. O satudaki ifade ya da işlemler gerçekleştirilikten sonra switch deyiminden çıkılır. Aşağıdaki örnek programımızda kullanıcıya bir menü sunalım ve bu menüden bir seçeneği seçmesini isteyelim. Bu seçim neticesinde de seçime uygun bir mesajın ekrana gelmesini sağlayalım. Örneğimizde ilk başta yaratılan secim_flag adındaki boolean değişken ilk değer olarak true olduğu için, program akışının ilk seferde while döngüsüne gireceğini garanti etmiş oluyoruz. Eğer kullanıcı listede olmayan bir seçim girecek olursa, secim_flag değişkeni tekrar true değerini alacak ve kullanıcıdan tekrar bir giriş yapması istenecektir. Kullanıcı listede olan bir giriş yaparsa program sonlanacaktır. Bu durumda listede olmayan girşler girldiği sürece program çalışmaya devam edecektir.

```
/**** Menu.java ****/

import javax.swing.*;

public class Menu {

  public static void main(String arg[]){

    boolean secim_flag = true;

    while(secim_flag) {

      String secim = JOptionPane.showInputDialog(null, "Bir sayi giriniz: (1, 2, 3,
4)");
      int sayi = Integer.parseInt(secim);
      secim_flag = false;

      switch(sayi) {
         case 1:
           JOptionPane.showMessageDialog(null, "Sectiginiz sayi: bir");
           break;
         case 2:
           JOptionPane.showMessageDialog(null, "Sectiginiz sayi: iki");
           break;
         case 3:
           JOptionPane.showMessageDialog(null, "Sectiginiz sayi: uc");
           break;
         case 4:
           JOptionPane.showMessageDialog(null, "Sectiginiz sayi: dort");
           break;
         default:
         secim_flag = true;
           JOptionPane.showMessageDialog(null, "Beklenenden farkli bir sayi
girdiniz");
           break;
       }
     }

     System.exit(0);
  }
}

/*****************************/
```

continue Anahtar Sözcüğü

Herhangi bir döngü içerisine girildğinde, daha döngü bloğunun sonuna gelinmeden o andaki adım için işlemleri sonlandırıp bir sonraki döngü adımına hemen geçilmek isteniyorsa continue anahtar sözcüğünü kullanmak gerekir. Mesela yazdığımız programda 1'den 10'a kadar olan sayılar toplamak istiyoruz ama bu toplam içerisine 3 ve ün katlarını eklemek istemiyoruz. Bunun için bir for döngüsü kullanmak gerekir. Ancak bu döngüyü 3 ve 3'ün katlarına gelindiğinde durdurup bir sonraki değerle döngüye devam etmek gerekmektedir:

```
/**** Toplama.java ****/

public class Toplama  {
  public static void main(String arg[]){
    int toplam = 0;
    for(int i = 0; i < 10; i++) {

       if(i % 3 == 0) {
         continue;
       }
```

```
        toplam += i;
      }

      JOptionPane.showMessageDialog(null, "Toplam: " + toplam);
      System.exit(0);
   }
}
```

Bu örnekte % operatörünü kullandık. Bu operatör soldaki sayının sağdaki sayıya bölümünden kalan değeri üretir. Yani i sayısı o anda hangi değere sahipse, bu değerin 3'e bölümünden kalan sayının 0'a eşit olup olmadığı kontrol edilmektedir. Eğer bu kalan sayı 0'a eşit ise demek ki sayımız ^ya da 'ün bir katıdır. Bu yüzden if parantezi içerisindeki == kontrolü sağlandığından dolayı true değeri üretilecek ve dolayısı ile if bloğu içerisine girilecektir. If bloğu içerisinde continue anahtar sözcüğü sayesinde o andaki for adımı sonlanacak ve bir sonraki i değeri ile for bloğunun en başına dönülecektir.

Diziler

Bu ana kadar programla içerisinde kullanacağımız değişkenler olduğunu ve bu değişkenlerin de belirli türelere sahip olduklarını görmüştük. Ancak biliyoruz ki her değişken sadece bir değer tutmaktadır. İçerisinde değer tutan bu değişkenler de bellekte farklı yerlerde saklanmaktadır.

Dizi denilen değişkenler ise içlerinde birden fazla değer tutarlar ve diğer değişkenler gibi belirli bir türe sahiplerdir. Ayrıca bu değerler de bellekte ardışıl biçimde saklanırlar ve hepsi aynı türe ilişkin olmak zorundadır. Buna göre bir diziyi aslında arka arkaya sıralanmış bir değişken paketi olarak düşünebiliriz.

Ancak şunu unutmamalıyız: Dizilerin içlerinde tutacakları eleman sayısı sahiptir. Bazı elemanlara siz değer vermesseniz bile bu elemanlar için bellekte yer ayrılır. Ayrıca dizinin içerisinde saklanan değerlerin türü ile dizinin türü aynı olmak zorundadır.

Dizilerin Yaratılması:

Normalde değişkenleri yaratırken değişkenin türünü ve sonra da adını yazıyorduk. (int b; ya da double a; gibi) Dizilerin yaratılması ise biraz daha farklıdır. Bir dizinin yaratılması iki adımdan oluşur. Birinci adım "bildirim" dediğimiz ve dizinin türünü ve dizi değişkeninin adını belirlediğimiz adımdır:

```
int []  sayilar;
```

Burada "sayilar" adinda ve içerisinde integer değerler tutacak olan bir dizi bildirimi yapmış olduk. Ancak bu dizi bildiriminin değişken bildiriminden farkının [ ] karakterlerinin eklenmesi şeklinde olduğuna dikkat ediniz.

İkinci adım ise tahsisat denilen ve dizi değişkenin kaç tane değer tutacağını belirlediğimiz adımdır:

```
sayilar = new int[10];
```

Bu örnekte, daha önce bildirimini yaptığımız sayilar dizisine 10 tane integer sayı tutabilecek kadar alanı tahsis etmiş olduk. Bu işlem aslında **new** anahtar sözcüğü sayesinde yapılmış oldu. Ancak, bu adımdan sonra dizi değişkenimiz belirlenmiş ve değerleri saklayabileceği alan tahsis edilmiş olsa bile, dizimizin içerisinde hala hiç bir değer saklanmamaktadır. Dizilerin içerisinde elemanlara erişmek ve yeni elemanlar eklemek için indeks numarası denilen bir kavram kullanılmaktadır. Indeks operatörü [ ] ile gösterilir ve dizinin adının yanına ekleneek kullanılır. Bir dizinin ilk elemanın indeks numarası 0 dır. Buna göre dizimize değer atamak için:

```
sayilar[9] = 12;
sayilar[0] = 3;
```

şeklinde [ ] ve bir indeks numarası kullanılarak atama yapılabilir.

NOT: Burada dikkat edilmesi gereken şey, dizimizin uzunluğundan daha fazla eleman ataması yapmamaktır. Bu örneğimize göre sayilar dizisinin son elemanına **sayilar[9]** ile erişebiliriz. **Sayilar[10] = 23;** hatalı bir atama olacaktır çünkü 10 nolu indekste 11. eleman olur. Ancak dizimiz sadece 10 eleman tutacak alan tahsis edilmişti!

Şimdi kullanıcıdan gelecek olan isimleri bir dizide saklayan bir program yazalım:

```
/**** Kayit.java ****/

1 import javax.swing.*;
2
3 public class Kayit {
4   public static void main(String arg[]){
5
6   String kayit_str = JOptionPane.showInputDialog(null, "Kac tane kayit
gireceksiniz:");
7    String cikisMesaji = "Hernagi bir anda cikmak icin "
8    int kayit = Integer.parseInt(kayit_str);
9
10    String isimler[] = new String[kayit];
11    for(int i = 0; i < isimler.length; i++) {
12        int sayi = i+1;
13        String giris =  JOptionPane.showInputDialog(null, sayi + ". kaydi
giriniz:");
14        if(giris.equals("X"))
15      break;
16        else
17      isimler[i] = giris;
18    }
19
20    String mesaj = "isim Listesi\n********";
21
22    for(int i = 0; i < isimler.length; i++) {
23        if(isimler[i] != null)
24          mesaj += "\n"+i+" --> "+isimler[i];
25    }
26
27    JOptionPane.showMessageDialog(null, mesaj);
28    System.exit(0);
29  }
30 }

/******************/
```

Bu programda önce kullanıcının kaç kayıt gireceği sorulmaktadır. Çünkü dizilerin uzunluğu sabittir ve önceden bir defa yer ayrıldığından bu uzunluk bilinmelidir. Uzunluğu değişen dizileri için başka algoritmik çözümler vardır ancak değişmeyen şer bir dizinin uzunluğu sabittir ve sonradan uzatılıp kısaltılamaz.

Bu örnekte kullanıcının gireceği kayıt kadar yer tahsisati yapıldıktan sonra for döngüsü ile her adımda girilecek olan kayıt istenmekte ve diziye atılmaktadır. Eğer kullanıcı başta belirlediği kadar kayıt girmekten vazgeçip o anda işlemi sonlandırmak isterse isim yerine X girişi yapabilecektir. Programda her giriş yapıldıktan sonra bu değerin X olup olmadığına bakılmaktadır (14. satır). Eğer giriş X olursa döngü break ile sonladırılır (15. satır). Değilse giriş diziye yazılır ve döngü devam eder (16. satır).

NOT: String türünde diziler için tahsis edilen yerlerin içine birşey doldurulmaz ise, bu yerler otomatik olarak null denilen özel bir değerle doldurulur. **null** değeri , " " gibi boş bir String ya da 0 değildir. Kendi başına anlamlıdır.

Bu döngü bittikten sonra yeni bir for döngüsü ile de diziden her kayıt tek tek okunmakta (22. satır) ve "mesaj" adındaki String değişkeni içerisine atılmaktadır (24. satır). Daha sonra bu mesaj isimli değişken ekrana yazılacaktır

(27. satır). Eğer kullanıcı diziye ayrılan yer kadar kayıt girmeyip arada X ile çıkış yapmışsa, bu durumda diziye girilmemiş olan elemanlar yerine **null** değeri olacaktır. Bu değerler varsa mesaj içerisine eklenmemektedir. Bu yüzden o andaki adımda, diziden gelen değerin **null** olup olmadığı kontrol edilmekte, değer null değilse mesaj değişkeninine eklenmektedir (23. satır).

Dizilere ilişkin temel bilgileri öğrenmeyi burada sonandırıyoruz. Siz de kendi yeteneklerinizi sınamak icin diğer dersimize kadar tüm öğrendiklerinizi kullanarak aşağıdaki uygulamayı yapmaya çalışınız. Bu uygulamanın cevabını diğer dersimiz içerisinde bulabileceksiniz :

Dizi Yaratma Şekilleri

```
1)    int a[];
          a = new int[3];
          a[0] = 1,  a[1] = 2, a[2] = 3;

2)    int a[] = {12, 3, 6, 7}

3)    new int[] {1,4,5,6}
```

Java ile Programlama:
Bölüm 4:

Dersimize son dersimizde verdiğimiz uygulamanın çözümü ile başlayacağız. Uygulamamızı tekrar hatırlatalım:

Uygulama: Kendi başınıza içerisinde isimler saklayan bir dizi yaratınız ve daha sonra bu dizide kullanıcıdan aldığınız bir ismi aratınız. Eğer isim dizinin içerisinde bulunursa bu ismin indeks numarasını ekrana yazdırınız. Bulunamazsa ismin bulunamadığına dair bir mesaj yazdırınız.

```
import java.awt.*;
import javax.swing.*;

public class Ara {

  public static void main(String arg[]) {

    String []isimler = {"Hakan Aykac", "Murat Akar", "Cenk Gündüz",
                                                    "Cem Daggeçen"};
    String aranacak = JOptionPane.showInputDialog(null,
                                            "Aradiginiz ismi giriniz: ");
    int index = -1;

    for(int i = 0; i < isimler.length; i++) {
      if(isimler[i].equals(aranacak)) {
        index = i;
        break;
      }
    }

    if(index == -1)
      JOptionPane.showMessageDialog(null, aranacak+" ismi dizide mevcut degildir.");
    else
      JOptionPane.showMessageDialog(null, aranacak+" isminin index numarasi:
"+index);

    System.exit(0);

  }
}
```

Metod Kavramı ve Java'da Metodlar
Daha önce bazı noktalarda metodlara değinmiştik. Metod denilen kavramın, program içerisinde yazılmış küçük program parçaları olduğunu söyleyebiliriz. Metodlar aslında önceden tanımlanmış birtakım işlem şekilleridir. Bu şekilde görevleri metodlar attında oluşturarak gerektiğinde çağırabiliriz. Bunu şöyle bir örnekle hayalimizde canlandıralım:

Yazmış olduğumuz programımızı bir robota benzetelim. Bu robota çeşitli görevler yükleyelim. Bunlardan bir tanesi de "su getir" görevi olsun. Bu görev şu şekilde tanımlansın:

Mutfağa git
Bardağa su doldur
Su dolu bardağı sahibine getir

İşte siz her su isteme işlemi için bu üç komutu robota vermek yerine, sadece 1 defa "su getir" görevini tanımlayabilirsiniz. Daha sonra da su istediğinizde, sadece bu görevi robota vermeniz, bu üç komutun çalıştırılması için yeterli olacaktır. Sonuçta, Java'da da metodlar bu şekilde birden fazla komutu tek seferde ve istenildiği zaman çalıştırabilmek amacıyla tasarlanır.

Aşağıdaki örnekte (Metod.java), ekrana sadece Merhaba yazan bir metod gösterilmektedir:

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class Mesaj {
5    public static void main(String arg[]) {
6      System.exit(0);
7    }
8
9    public void MesajGoster() {
10     System.out.println("Merhaba");
11   }
12 }
```

"**Mesaj.java"** örneğinde de görüldüğü gibi "**MesajGoster"** isimli metod, main metodunun bittiği yerde yaratılmıştır. Ancak bu metodun da tıpkı main metodu gibi sınıfın içerisinde yaratıldığına dikkat ediniz. Metodun yaratılması sırasında ilk satırda

public static void MesajGoster()

şeklinde metodun adı ve daha sonra da
{

}
parantezleri içerisinde bu metodun gerçekleştireceği komutlar yazılmaktadır.
Metodun adını yazdığımız bu ilk satıra "metod bildirimi" denilir. Metod bildirimine ilişkin ayrıntıları bilgilerimiz arttıkça tamamlayacağız.

Not: Metodları isimlendirirken boşluk bırakılmaz. (Mesaj Goster geçersiz ancak Mesaj_Goster gecerli bir metod ismidir)

Not: Main metodu ve sınıf kavramlarına sonraki derslerimizde değineceğiz.

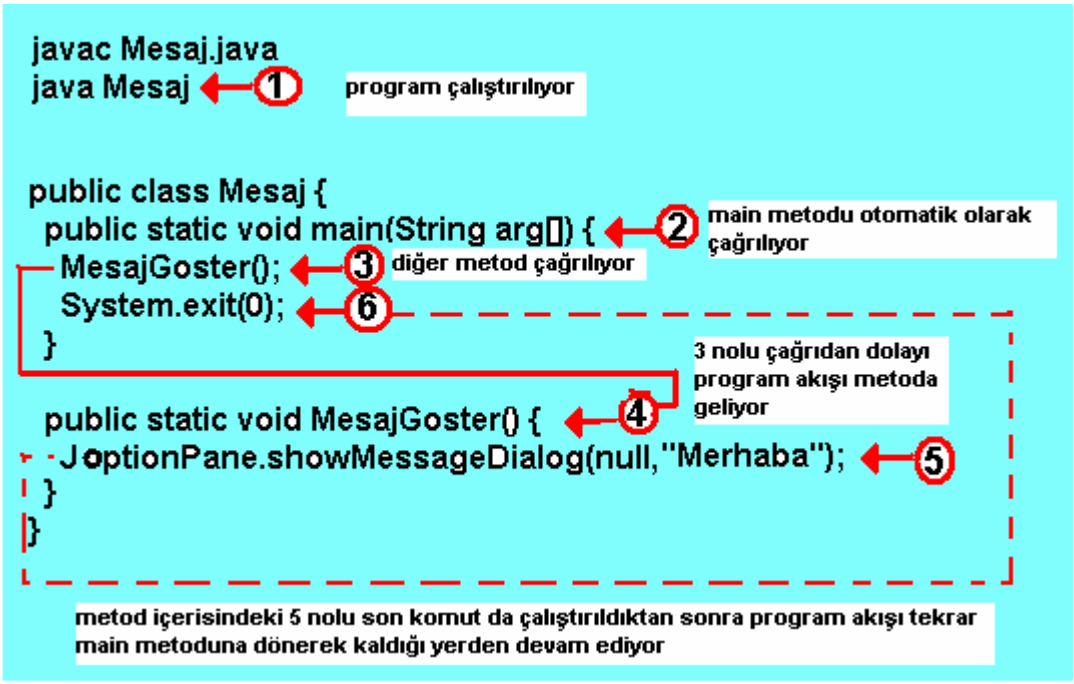
Metodların Çağrılması
Şimdi bu Mesaj.java dosyasını çalıştırmaya kalktığımızda, dosyanın düzgün bir şekilde çalıştığını ancak ekranda hiç bir mesaj yazılmadığını göreceksiniz. Sakın şaşırmayın! Böyle olması çok normal. Bir metod yazıldığı zaman bu onun aynı zamanda çalışacağı anlamına gelmez. Metodun çalışması için sizin onu çağrmanız gerekmektedir. Bunu robota "su getir" emrini vermenize benzetebiliriz. Siz bu emri daha önce tanıtmış olsanız bile, emri vermeden robot size su getirmez. Aynı şekilde yazılmış bir metodun çalışması için de metodun "program akışının olduğu yerde" çağrılmış olması gerekir. Bir metodu çağırmak için metodun yaratıldığı şekilde adını ve adın yanına da ( ) parantezlerini eklememiz gerekir:

```
. . .
MesajGoster();
. . .
```

gibi... O halde örneğimizde de metodun çalışmasını görmek için main metodu içerisinde MesajGoster isimli metodumuzu çağıralım:

```java
import java.awt.*;
import javax.swing.*;

public class Mesaj {
  public static void main(String arg[]) {
    MesajGoster();
    System.exit(0);
  }

  public static void MesajGoster() {
    JOptionPane.showMessageDialog(null, "Merhaba");
  }
}
```

Programımız ilk çalıştırıldığında her zaman main metodu "otomatik olarak çağrılır" Daha sonra main metodu içerisinde yazılan her satırdaki komutlar sırasıyla çalıştırılır. Program akışı MesajGoster(); şeklinde yazılmış olan metod çağırma komutuna geldiğinde MesajGoster metodunun yazıldığı yere atlar ve buradaki komutları sırasıyla çalıştırırarak tekrar main metodunda kaldığı yerden çalışmaya devam eder. Tabi ki metod ismi MesajGoster den başka bir isim de olabilir. Sadece buradaki örnekte bir metod yazılmış ve adı öyle konulmuştur. Bu akış prensibi her metod için geçerlidir.



Şimdi işlerimizi daha da genelleştirelim ve ekrana sadece Merhaba değil de, istediğimiz bir mesajı gönderen bir metod oluşturalım. Aslında bu işi zaten **JOptionPane** sınıfının **showMessageDialog** isimli metodu ile yapabiliyorduk. Ancak bu metodun adı çok uzun olmakla beraber program içerisinde birden fazla yerde çağırdığımızda çok fazla yer kapladığını görmekteyiz. Bu yüzden aynı işi yapan ve adı sadece "**M"** olan bir metod yazalım. Ancak bu metoda her seferinde hangi mesajı yazacağını da göndermemiz gerekecektir. Çünkü her seferinde ekrana başka bir mesaj yazılmasını isteyebiliriz. İşte bu noktada karşımıza parametre kavramı çıkar.

Metodların Parametreler ile Kullanımı

Parametreler yardımıyla metodlara birtakım değişkenler göndermemiz mümkündür. Önceki örneklerimizde MesajGoster isimli örnek metodumuzu parametresiz olarak kullandık. Aşağıdaki örnekte M isimli metod parametre ile çalışan bir metod olup, ekrana, kendisine parametre olarak gönderilen mesajı basmaktadır.

```
1  import java.awt.*;
2  import javax.swing.*;
3
4  public class Mesaj3 {
5    public static void main(String arg[]) {
6      M("Merhaba Chuvy!");
7      System.exit(0);
8    }
9
10   public static void M(String mesaj) {
11     JOptionPane.showMessageDialog(null, mesaj);
12   }
13 }
```

Burada dikkat edilirse 6. satırda çağrılan M metodu 10. satırda parametre ile bildirilmiştir. Metod bildiriminde parametreler metodun ( ) parantezleri arasında önce tür adı ve daha sonra da parametre adı yazılarak ve birden fazla parametre arasında da virgul kullanılarak tanıtılabilir. Yani:

public static void Metod_Adi(<tür> parmaetre1, <tür> parametre2) {
...
}

şeklinde bildirimi genellemek mümkündür. Örneğin:

public static void M(String mesaj): Sadece String türünde bir parametre alan ve bu parametreyi metod içerisinde "mesaj" ile temsil eden bir metod bildirimi.

public static void Metod2(int x, double y): Adı x olan ve int türünde bir parametre ile adı y olan ve double türünde bir parametre alan ve kendi adı da Metod2 olan bir metodun bildirimi.

Bu şekilde bildirimi yapılan bir metodu çağırıken de metodun adı ve daha sonra da ( ) parantezleri içerisine bildirimde verilen sırada ve türlerde değerle göndermemiz gerekir. Mesela M isimli örnek metodumuza her seferinde String türünde farklı mesajlar gönderebiliriz. Gönderilen b her String değeri, metod bildiriminde "mesaj" isimli değişkene kopyalanır ve metodun içerisinde de bu değişken aracılığıyla ele alınır. Mesela bildirimi

public static void Islem(int sayi, String mesaj)

şeklinde yapılmış olan bir metodu şu şekillerde çağırabiliriz:

Islem(6, "Bugun hava guzel");
Islem(12, "Çok çalışmak lazım");
Islem(4, "Java Öğrenmek Lazım");
...

Burada yapılan şey aslında metodun çağrılması neticesinde gönderilen değerler, metod bildirimi ile tanımlanan değişkenlere sırasıyla kopyalanır (ya da atanır.) Öyle ki;

Islem(6, "Bugun hava guzel");

şeklinde metodu çağırdığımızda 6 değeri sayi değikenine ve "Bugün hava güzel" değeri de mesaj değişkenine atanır ve metod içinde bu şekilde kullanılır.

Ancak Islem("Selam", 12); gibi bir çağrım geçerli değildir. İşlem isimli metodun bildiriminde önce int türünde sonra da String türünde parametre yazılmıştır. Metodu çağırırken de aynı sıraya uymamız gerektiğini unutmayalım.

ÖNEMLİ NOT: Özetle, metodları bildirirken metodun alacağı parametreleri tür ve parametre isimlerini yazarak, metodları çağırıken de parametreler yerine değer göndererek işlem yapmaktayız. Bu iki işlemi birbirine karıştırmamalıyız.

Metodlarda Geri Dönüş Değeri Kavramı

Yukarıda da anlattığım gibi, metodlara birtakım parametreler göndererek bazı işlemler yaptırmalarını sağlarız. Bu işlemler kendi içerisinde uygulanıp tamamlanan işlemler olabileceği gibi, sonunda bir sonuç üretilen işlemler de olabilir. İşte sonuçta elde edilen bu değerleri kullanabilmemiz için, metodun bu değerleri bize geriye göndermesi gerekir. Metodların işlemlerinin tamamlaması neticesinde geriye gönderdikleri bu değere "geri dönüş değeri" denilmektedir. Aşağıdaki örnekte parametre olarak bir çemberin yarıçap değerini alan "alanHesapla" isimli metod, çemberin alanını hesaplayarak geriye bu değeri bize göndermektedir:

```
1  import javax.swing.*;
2  import java.text.NumberFormat;
3
4  class Alan {
5    public static void main(String[] args) {
6      double sonuc;
7
8      String yrcp = JOptionPane.showInputDialog(null, "Cemberim yaricapini giriniz");
9      sonuc = alanHesapla( Double.parseDouble(yrcp) );
10     String mesaj = yrcp + " yaricapli cemberin alani: "+sonuc;
11     JOptionPane.showMessageDialog(null, mesaj);
12
13     System.exit(0);
14   }
15
16   public static double alanHesapla(double yaricap) {
17     double alan = Math.pow(yaricap, 2) * 3.14;
18     return alan;
19   }
20 }
```

Örnekte de görüldüğü gibi, 9. satırda kullanıcıdan istenen yarıçap değeri 10. satırda metoda parametre olarak gönderilmektedir. Ama aslında parametre olarak gönderilmeden hemen önce "Double.parseDouble()" metodu ile bu String türündeki yarıçap değeri double türüne dönüştürülmekte ve dönüşüm sonucu gelen değer "alanHesapla" isimli metoda parametre olarak gönderilmektedir. Gönderilen bu parametre, metod bildiriminde belirtilen "yaricap" isimli değişkene aktarılmaktadır.

NOT: Bu aktarım neticesinde, gelen parametre artık metod içerisinde yaricap ismiyle kullanılacaktır. Eğer metod, 1'den fazla parametreli bir metod olsaydı, bu durumda gönderilen parametre değerleri aynı sıra ile, metod bildiriminde tanımlanan parametre değişkenlerine aktarılacaktı.

Daha sonra "alanHesapla" metodu kendisine gelen bu parametre'nin, "Math" sınıfının "power" metodu yardımıyla karesini alarak 3.14 (pi sayısı) ile çarpmakta ve üretilen sonucu da 17. satırda olduğu gibi alan değişkenine atamaktadır.

NOT: "Math.power" metodu 1. parametre ile kuvveti alıncak sayıyı, 2. parametre ile de bu sayının kaçıncı kuvvetini alacağını belirler. Geri dönüş değeri olarak da hesapladığı sonucu gönderir.

return Anahtar Sözcüğü

Alan.java örneğinde, 18. satırda kullanılan "return" ifadesi, önündeki değişken değerini metodun çağrıldığı yere göndermektedir. Bu durumda hesaplanan alan değeri, return ifadesi ile metodun çağrıldığı 9. satıra gönderilmektedir. Buraya geriye gönderilen değer de "sonuc" isimli değişkene atanarak tutulmuş olur.

Eğer 9. satırda

sonuc = alanHesapla(Double.parseDouble(yrcp));

ifadesi yerine sadece

alanHesapla(Double.parseDouble(yrcp));

şeklinde metodu çağırsaydık, metod içerisinden alan değeri gönderilecek, ancak bu değeri tutacak "sonuc" gibi bir değişken olmadığı için geriye gelen değer boşa gidecekti ve ele alınamayacaktı.

Bu örnekte dikkat edilmesi gereken bir diğer konu da; geri dönüş değeri olan metodların bildirimlerinde void yerine bir tür adı yazılması gerektiğidir. Örneğimizde geriye gönderilen değerin türü "double" türünde bir değer olduğu için olduğu için, 16. satırda yapılan metod bildiriminde metod isiminden hemen önce geri dönüş değerinin türünün yazıldığını görmekteyiz. Fakat önceki örneklerimizde geri dönüş değeri olmayan metodlarda bu kısma "void" yazmıştık.

Dizilerin Metodlara Parametre Olarak Gönderilmesi
Daha önce dizilerin ne amaçla yaratıldıklarını ve nasıl kullanıldıklarını görmüştük. Dersimizin bu kısmında hem metod kullanımını biraz daha pekiştirmek, hem de diziler üzerinde pratik yapmak amacıyla dizilerin metodlara aktarılmasını inceleyeceğiz.

Aşağıdaki örnekte çeşitli isimlerden oluşan bir diziyi "**alfabetikSirala"** isimli metoda gönderiyoruz. Bu metod kendisine parametre olarak karışık sırada isimlerden oluşan bu diziyi alfabetik sırada yeni bir dizi içerisinde oluşturarak geriye de bu yeni diziyi gönderiyor.

```
import javax.swing.*;
import java.util.Arrays;
import java.lang.System;

class DiziSirala {
  public static void main(String[] args) {

    String liste[] = {"Evren Akar", "Ahmet Aykaç", "Fatih Demir", "Ender Uçkun"};

    String gelenDizi[] =  alfabetikSirala(liste);
    diziGoruntule(liste, "Orjinal Dizi");
    diziGoruntule(gelenDizi, "Sirali Dizi");

    System.exit(0);
  }

  public static String[] alfabetikSirala(String [] dizi) {

    String sirali_dizi[] = new String[dizi.length];
    System.arraycopy(dizi, 0, sirali_dizi, 0, dizi.length);
    Arrays.sort(sirali_dizi);

    return sirali_dizi;
  }

  public static void diziGoruntule(String []dizi, String dizi_adi) {

    String mesaj = dizi_adi+" Içerigi\n***************\n\n";

    for(int i = 0; i < dizi.length; i++)
      mesaj += "dizinin "+i+". elemani: "+dizi[i]+"\n";

    JOptionPane.showMessageDialog(null, mesaj);
  }
```

```
}
```

Diziler metodlara parametre olarak aktarılırken sadece isimlerinin aktarılması yeterlidir. Örneğimizde 5. satirda yaratilan "**liste"** isimli dizimiz, 6. satırda metod çağrılması sırasında, parantez içerisinde sadece ismi yazılarak "**alfabetikSirala"** isimli metoda aktarılmaktadır.

Diziyi parametre olarak alan metod bildiriminde ise, parametre kısmında gelecek olan dizinin türü, [ ] sembolü ve bir parametre adı yazılması gerekir. Örneğimizde yarattığımız "**alfabetikSirala"** isimli metodun 12. satırdaki bildiriminde de, gelecek olan dizi parametresi "**dizi"** isimli değişkene aktarılacaktır. Ancak bu değişkenin de bir "**String"** dizisi değişkeni olarak tanımlandığına dikkat ediniz.

"alfabetikSirala" isimli metoda gelen bu dizi, parametre aktarımından dolayı artık metod içerisinde "**dizi"** ismiyle kullanılacaktır.

"**System"** sınıfının "**arrayCopy"** isimli metodu, 1. parametre ile aldığı bir dizinin, 2. parametre ile aldığı indeksinden itibaren, 3. parametre ile aldığı dizinin, 4. parametre ile aldığı indeksine, 5. parametre ile aldığı sayıda elemanı kopyalar. Bu metod yardımıyla, 14. satırda olduğu gibi metod içerisinde "**dizi"** ismiyle tutulan orjinal dizimizin tüm elemanları, "**sirali_dizi"** ismiyle yarattığımız diziye kopyalanmaktadır.

"**Arrays"** isimli sınıfın "**sort"** isimli metodu, kendisine parametre olarak verilen bir diziyi sıralamaktadır. Biz de örneğimizde bu metodu kullanarak "**sirali_dizi"** içerisinde bir kopyasını aldığımız dizi isimli dizimizi sıralamaktayız.

Dizilerin Geri Dönüş Değeri Olarak Kullanılması

Bir diziyi geri dönüş değeri olarak gönderirken benzer şekilde "**return"** anahtar sözcüğü önünde dizinin isminin yazılması yeterlidir. Ancak dikkat edilmesi gereken önemli nokta geriye gelen diziyi tutacak değişkenin de bir dizi değişkeni olması gerektiğidir. Örneğimizde 16. saturda yazılan "**return"** ifadesi ile yeni sıralı dizimizi 6 satırda metodu çağırdığımız yere geri dönüş değeri olarak göndermekteyiz. 6. satıra gelen bu yeni dizi "**gelenDizi"** isimli dizi değikenine aktarılmaktadır.

Örnekte 19. satırda yazmış olduğumuz "**diziGörüntüle"** isimli metod kendisine 1. parametre ile gönderilen bir dizinin 2. parametre ile gönderilen başlığı kullanarak içeriğini görüntülemektedir. Bu metod yardımıyla orjinal ve sıralanmış dizilerimizin içeriklerini de görüntülemkteyiz.

Bu örnekler sayesinde metod ve dizi kullanımına ilişkin teknikleri daha iyi anlayacağınızı umuyorum. Ders içerisinde verilen örnekleri mutlaka satır satır yazmanızı öneririm. Böylece hiç farkında olmadan Java'da birçok şeyin pratik hale geldiğini göreceksiniz.

Şimdi dizilere ilişkin son bir örnekle metod ve dizilerin birlikte kullanımını toparlayalım.

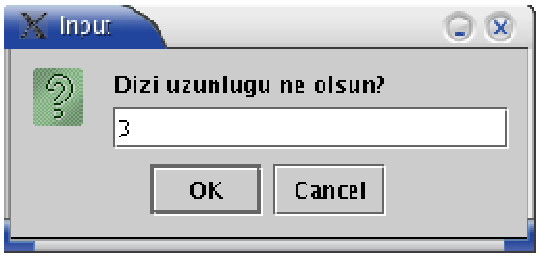
```
import javax.swing.*;
import java.util.Arrays;
import java.lang.System;

class HistogramSinifi {

  public static void main(String[] args) {

    int uzunluk = Integer.parseInt(JOptionPane.showInputDialog(null, "Dizi uzunlugu
ne olsun?"));

    String []liste = diziYarat(uzunluk);
    Histogram(liste);

    System.exit(0);
```

```
  }

  public static String[] diziYarat(int boyut) {

    String isimler[] = new String[boyut];

    for(int i=0; i < isimler.length; i++)
      isimler[i] = JOptionPane.showInputDialog(null, i+". ismi giriniz");

    Arrays.sort(isimler);
    return isimler;
  }

  public static void Histogram(String dizi[]) {

    JTextArea cikisEkrani = new JTextArea(15, 15);
    String cikis = "isim \t uzunluk \n -------------------\n";

    for(int i = 0; i < dizi.length; i++) {
      cikis += dizi[i] + "\t";
      for(int k = 0; k < dizi[i].length(); k++)
        cikis += "*";
      cikis += "\n";
    }

    cikisEkrani.setText(cikis);
    JOptionPane.showMessageDialog(null, cikisEkrani);
  }
}
```

Örneğimizde 5. satırda "**JOptionPane"** isimli sınıfın "**showInputDialog"** isimli metodu, geri dönüş değeri olarak kullanıcının gireceği **String** bir değer göndermektedir ve bu değer de "**Integer"** sınıfının "**parseInt"** isimli metodu içerisine parametre olarak verilerek integer bir sayıya dönüştürülür. Dönüştürülen bu sayı da "**uzunluk"** isimli değişkene atanır. Daha sonra 7. satırda **"liste"** ismiyle yaratılan dizimiz de, bu uzunluk değeri kadar olacaktır. Ama dikkat ederseniz bu sefer dizi yaratma işlemi bizim yazdığımız "**diziYarat"** isimli metod içerisinde yapılmaktadır.

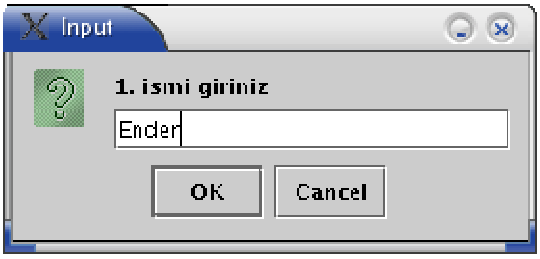
13. satırda tanımlanan "**diziYarat"** isimli metod kendisine parametre olarak verilen uzunlukta bir String dizisi yaratır. Daha sonra 16. satırda yazılan for döngüsü ile bu dizinin elemanlarını kullanıcıdan alarak diziye yerleştirir. Bu işlemi 17. satırda olduğu gibi, for döngüsünün her i adımında, kullanıcıdan istediği giriş değerini dizinin i. elemanına atayarak yapmaktadır. Daha sonra 19. satırda yaratılan ve içerisine isimler yerleştirilen dizimiz "**Arrays"** isimli sınıfın "**sort"** isimli metodu ile sıralanmaktadır. En son olarak da 20. satırda, yaratılan bu dizi "**return"** ifadesi ile metodun çağrıldığı 7. satıra gönderilir. Bu gönderme neticesinde de yaratılan ve sıralan bu dizi, liste isimli dizi değişkenine atanmış olur.

8. saturda, "**Histogram"** isimli metod çalıştırılmaktadır. Bu metod kendisine parametre olarak verilen bir dizinin her elemanının kendisini ve bu elemanın karakter uzunluğu kadar * simgesini ekrana basar. Metod bu işlemi yapmadan önce ilk olarak **"JTextArea"** denilen özel bir java nesnesi yaratmaktadır. Bu nesne 25. satırda yaratılmaktadır. Java nesneleri ve nesnelerin yaratılması konusunu sonraki dersimizde ayrıntılı olarak göreceğiz. 26. satırda "**cikis"** isimli String değişkeni de en son ekranda belirecek mesajı tutmak amacıyla oluşturulmaktadır. Ancak bu değişken ilk yaratıldığında içi boş bir şekilde oluşturulmaktadır. 28. satırdaki for döngüsü ile dizinin her elemanı tek tek ele alınacaktır. 29. satırda, i. adımdaki eleman ve bir tab boşluk karakteri "**cikis"** değişkenine yazılır. 30. satırdaki for döngüsü, i. adımda ele alınan dizi elemanının her karakteri için adım adım * işaretini yine cikis isimli değişkene yazar. En son olarak da 33. satırda "**cikis"** isimli değişkenin sonuna **"\n"** şeklide alt satıra geçme karakteri eklenerek 28. satırdaki döngünün bir sonraki adımına geçer ve aynı işlemler diğer dizi elemanları için de tekrarlanır. Tüm dizi elemanları bu şekilde ele alındıktan sonra, 35. satırda olduğu gibi "**cikisEkrani"** isimli nesnenin içerisinde yer alacak yazı, yine bu nesneye ait öntanımlı "**setText"** isimli metod yardımıyla "**cikis"** isimli değişkenin içerisinde saklanan değere set edilir. (Java'da nesnelerin öntanımlı metodlarını sonraki derslerimizde inceleyeceğiz.) En son olarak da metod 36. satırda olduğu şekilde bu nesneyi showMessageDialog isimli metoda vererek ekranda görünmesini sağlar.

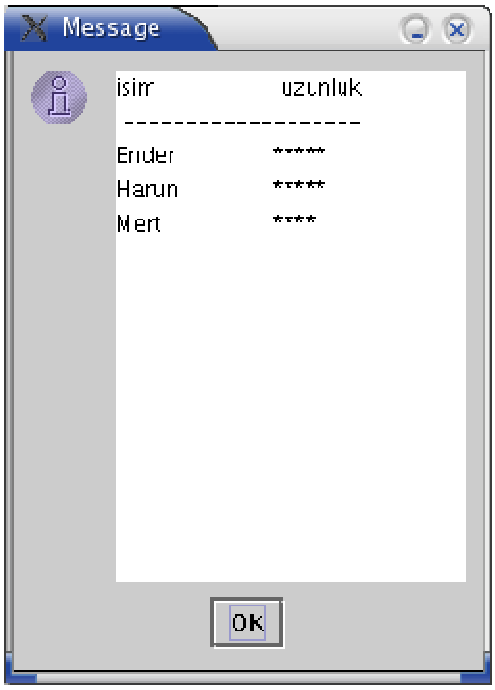


1. ekran

2. ekran



3. ekran



Dikkat ederseniz daha önceki örneklerimizde JOptionPane.showMessageDialog isimli metoda hep bir mesaj yazıyorduk. Ama bu sefer metoda, " " içerisinde yazılmış bir String mesajı yerine, bir nesne gönderdik ve tüm mesajımızı da bu nesne içerisinde yazdık.

Uygulama: Bir dizinin içerisindeki elemanları sıraya dizen static bir metod yaziniz ve main metodu icerisinde yazdiginiz bu metoda diziyi gondererek siralayiniz.

UYGULAMA: Kendi başınıza içerisinde isimler saklayan bir dizi yaratınız ve daha sonra bu dizide kullanıcıdan aldığınız bir ismi aratınız. Eğer isim dizinin içerisinde bulunursa bu ismin indeks numarasını ekrana yazdırınız. Bulunamazsa ismin bulunamadığına dair bir mesaj yazdırınız.

Dizilerin Kopyalanmasına İlişkin NOT:
Bir dizi yaratıldıktan ve içeriği güncellendikten sonra bunu aynen başka bir diziye kopyalamak için System.arraycopy metodu kullanılır. Bu metodun kullanımı:

System.arraycopy(kaynak_dizi, baslama_noktasi, hedef_dizi, hedef_baslama_noktasi, kopyalancak_eleman_sayisi);

şeklindedir:

Java ile Programlama:
Bölüm 5:

Önceki dersimizde hatırlarsanız Java'da metodların nasıl kullanıldığını ve yaratıldığını ayrıntılı olarak inceleyerek, devamında da dizilerin metodlara parametre olarak gönderilmesine ilişkin birtakım örnekler üzerinde çalışmıştık. Önceki derslerimizi iyice tekrar ettiğinizi ve verilmiş örnekleri de teker teker yazarak denediğinizi umuyorum. Bilinen örnekleri bile sabırla yazmak inanın çok şey öğretecektir.

Bugünkü dersimizde artık nesne yönelimli programcılığın en önemli kavramı olan sınıfları inceleyeceğiz. Önümüzdeki derslerin birçoğu artık bu sınıfları kullanmak ve geliştirmek üzerine olacak. Bu yüzden burada basitleştirerek anlatmaya çalıştığım bu temel kavramları iyice anlamaya ve irdelemeye çalışmanızı öneririm.

Sınıf Kavramı:
Biliyoruz ki genel anlamda program yazabilmek için çeşitli değişkenler yaratıp kullanıyoruz. Yaratılan her değişkenin bir türü olduğunu ve yaratıldıktan sonra bu değişkenlerin de artık birer nesne olduğunu görmekteyiz.

Örneğin

"**int a;"** dediğimizde a isimli değişkeni içerisinde tamsayılar tutacak biçimde,

"**double b;** " dediğimizde b isimli değişkeni içerisinde virgüllü sayılar tutabilecek şekilde ya da

"**String c[]; "** dediğimizde c isimli değişkeni içerisinde String türünde değerler tutan bir dizi şeklinde yaratmış oluyoruz. Yani değişkenleri bilinen nesne türlerinden birtanesi cinsinden yaratıyoruz.

İşte bugün öğreneceğimiz şekilde, sınıfları yazarak önce **yeni bir nesne türü yaratıyor** ve daha sonra da yaratmış olduğumuz bu "**nesne türü cinsinden" sınıf değişkenleri yaratarak** bu sınıfları kullanabiliyoruz. Ya da bir başkasının yaratmış olduğu bir sınıf türünden yeni bir değişken yaratabiliyor ve yine bu değişkeni kullanabiliyoruz.

Aslında bundan önceki derslerde, önceden yaratılmış bazı sınıfları zaten kullanmıştık. Örneğin "**JOptionPane"** isimli sınıfa ait "s**howMessageDialog"** ya da "**showInputDialog"** gibi metodları kullanırken aslında **JOptionPane** isimli sınıfı kullanmış olduk.

Sınıf Nedir?
Şimdi biraz daha genel olarak sınıfların ne olduğundan bahsetmekte fayda var. Sınıf denilen şey aslında birden fazla metod ve değişkeni birarada tutan bir yapıdır. Ancak bu değişken ve metodların da aynı göreve ilişkin olması beklenir. Örneğin icq benzeri bir uygulamada en kaba şekilde görsel kısım ve ağ bağlantıları olmak üzere iki kısım olduğunu kabul edelim. Bu uygulamanın pencere şekli, rengi, düğmelerin yeri v.b. işleri üstlenecek birçok metod ve değişkeni olsa da hepsinin amacı aslında görselliği sağlamaktır. Yine benzer şekilde arka planda bağlantıları ayarlayan, kontrol eden bir çok metod ve değişken de ağ ayarları için kullanılır.

Bu durumda aynı projeye ilişkin ama ayrı görevleri üstlenen tüm metod ve değişkenleri bu iki ana sınıf altında toplamak hem organizasyonda hem de tasarımda kolaylık sağlar. Bu iki kısım birbirinden bağımsız tasarlanarak bir araya getiriliebilir. Bu da projeye esneklik sağlar.

Sınıfların Yaratılması
Bundan önce yazdığımız örneklerde aslında biz hep bir sınıf ve bu sınıfın içerisinde de birtakım metod ve değişkenler yaratıyorduk. Şimdi yine örnek bir sınıf yaratalım. Bu sınıf kendi içerisinde sadece bir değişken tutsun ve bu değişkenin değerini elde edebilecek ya da değiştirebilecek bazı metodları olsun:

```
class DegiskenTutan {

  int x;

  public void setX(int c) {
    this.x = c;
```

```
  }

  public int getX() {
    return this.x;
  }
}
```

DegiskenTutan.java dosyasındaki örnekte, bahsettiğimiz bu sınıfı yine sınıf ile aynı isimde olan "**DegiskenTutan.java"** dosyası içerisinde yarattık ve kaydettik. Biz aslında burada "**DegiskenTutan"** isimli yeni bir nesne türü yaratmış oluyoruz.

DegiskenTutan sınıfının 2. satırında yaratılan "**x"** isimli bir global değişkeni bulunmaktadır. Bu değişkenin global olmasının nedeni yaratıldığı yerin herhangi bir metodun içinde değil de en dış faliyet alanında olmasıdır. Bu nedenle sınıf içerisinde tanımlı diğer tüm metodlar bu değişkene erişebilir ve değişkeni kullanabilirler.

this Anahtar Sözcüğü
Sınıf içerisinde yazılmış olan "**getX"** isimli metod, bu "**x"** değişkeninin içerisindeki değeri geri döndürmektedir. Yalnız dikkat edilirse 9. satırda gerçekleşen bu geri döndürme işlemi sırasında sadece değişkenin adı değil, onun yerine "**this.x**" şeklinde bir kullanım görmekteyiz. "**this**" anahtar sözcüğü özel bir anahtar sözcük olup noktadan sonra gelen değişken isimlerinin sınıfa ait olduğunu ifade etmektedir. Yani 9. satırda yazılan

r**eturn this.x;**

ile kastedilen şey, ilgili sınıfa ait olan "x" isimli değişkeninin geri döndürüleceğidir. Yine sınıf içerisinde yazılmış olan "**setX"** isimli metod da kendisine parametre olarak gelen ve c değişkeninde tutlan değeri "**this.x"** değikenine, yani sınıfa ait olan "**x"** değişkenine atamaktadır.

Gördüığümüz gibi yazmış olduğumuz bu yeni sınıf içerisinde başka değişken ya da metod bulunmamaktadır. Peki ya madem artık bu sınıf yazıldı ve kullanıma hazır, sınıfı nasıl kullanabiliriz? Şimdi bu sorunun cevabını verelim:

Yazılmış Sınıfların Kullanılması
Bazı özel durumların dışında, en genel biçimiyle bir sınıf nesnesi ya da bildiğimiz şekliyle bir sınıf türünden değişken, şu şekilde yaratılır:

SinifAdi degisken = new SinifAdi();

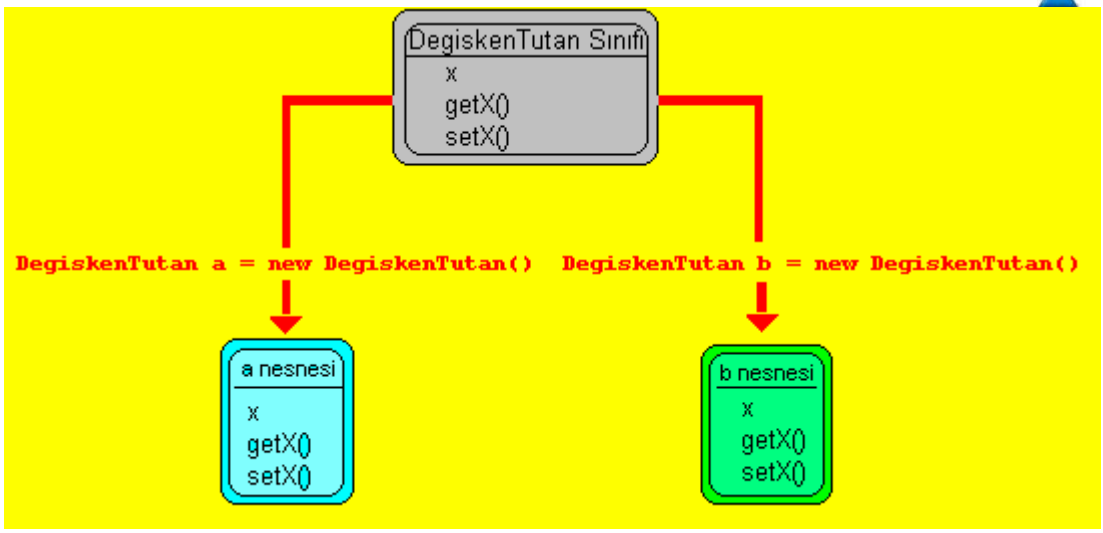
Mesela, az önce yazmış olduğumuz "DegiskenTutan" isimli sınıf türünden bir değişken de:

DegiskenTutan a = new DegiskenTutan();

şeklinde yaratılmaktadır. Bu yaratma işlemini başka bir sınıf içerisinden yapabilirsiniz. Burada vurgulanması gereken en önemli nokta, yaratılan her sınıf değişkeninin (a'nın) aslında gerçek sınıfın bir kopyası olduğudur (instance). Yani biz yukarıdaki gibi a isimli ve DegiskenTutan türünde bir sınıf nesnesi ve yine aynı şekilde

DegiskenTutan b = new DegiskenTutan();

b isimli başka bir DegiskenTutan isimli sınıf nesnesi yarattığımızda, bu iki nesne de artık kendilerine ait birer "setX" ve "getX" metodları ile bir "x" değişkeni barındırmaktadır. A nesnesinin "x" değişkeni ile b nesnesinin "x" değişkeni artık farklıdır. Ama ikisinin de bir "x" değişkeni vardır. Yine aynı şekilde "a" nesnesi ile "b" nesnesinin "setX" ve "getX" metodları vardır ama her metod kendi nesnesine ait olan "x" değişkeni üzerinde işlem yapabilir.



Şimdi "DegiskenTutan.java" dosyası ile aynı klasör içerisinde yer alan "GenelSinif" isimli yeni bir sınıf daha yaratalım ve bu yeni sınıf içerisinde de "DegiskenTutan" isimli sınıfı kullanalım:

```
import javax.swing.*;

public class GenelSinif {

  public static void main(String arg[]) {
    DegiskenTutan a = new DegiskenTutan();
    DegiskenTutan b = new DegiskenTutan();
    a.setX(10);
    b.setX(20);

    String message = "a daki x degeri: "+a.getX();
    message += "\nb deki x degeri: "+b.getX();

    JOptionPane.showMessageDialog(null, message);

    System.exit(0);
  }
}
```

Şimdi yazmış olduğumuz bu sınıfları kaydettikten sonra çalıştırabilmek için şu işlemleri yapmamız gerekmektedir.:

Öncelikle komut satırında bu sınıf dosyalarını yazmış olduğunuz dizin içerisine cd komutu yardımıyla giriniz. Örneğin bu dizin linux kullanıcıları için "/home/academytech/kodlar" ya da windows kullanıcıları için "C:\kodlar" ise, dizin içerisine girmek için:

```
$cd /home/academytech/kodlar

ya da

C:\> cd kodlar

yazmanız gerekmektedir.
```

2 Şimdi sırasıyla "DegiskenTutan.java" dosyasını ve "GenelSinif.java" dosyasını derleyelim:

```
javac DegiskenTutan.java
javac GenelSinif.java
```

3 Son olarak da GenelSinif.java dosyasını çalıştıralım:

```
java GenelSinif
```

Not: Burada görüyoruz ki "DegiskenTutan.java" dosyası sadece derleniyor ancak GenelSinif.java gibi sonra da çalıştırılmıyor. Bunun nedeni asıl çalışan programın "GenelSinif" olması "DegiskenTutan" sınıfının ise sadece kullanılıyor olmasıdır. Çünkü "GenelSinif" sınıfı içerisinde "main" metodu vardır. Bununla ilgili ayrıntılı açıklamayı yazının ilerleyen kısımlarında zaten bulabileceksiniz.

Örneğimizde 7. ve 8. satırlarda DegiskenTutan sınıfı türünden "a" ve "b" isimli 2 değişken ya da iki "DegiskenTutan" sınıfı nesnesi yaratılmaktadır. Daha sonra 9. satırda a nesnesinin "setX" metodu "a" nesnesinin "x" değişkeni değerini 10, "b" nesnesinin "setX" metodu da "b" nesnesinin "x" değişkeninin değerini 20 yapmaktadır.

Dikkat ederseniz "a" ve "b" nesneleri de "setX" ve "getX" metodlarına sahiptir ve bu metodları kullanmak için nesne adından sonra "." işareti ve sonra da metod adını yazmak gerekir. "a" ve "b" nesnelerinin tıpkı "DegiskenTutan" sınıfı gibi metodlara ve değişkenlere sahip olmalarının nedeni, bu nesnelerin de yaratılırken "DegiskenTutan" sınıfı türünden yaratılmış olmalarıdır.

12. ve 13. satırlarda hazırlanan "message" isimli String değişkeni içerisine ekranda görülecek yazı ve "getX" metodları yardımıyla "a" ve "b" nesnelerinin "x" değişkenlerinin içerisindeki değerler hazırlanmaktadır. En son olarak da "message" isimli değişken içerisinde hazırlanan bu son görüntü "JoptionPane" isimli sınıfın "showMessageDialog" isimli metodu yardımıyla ekrana gönderilmektedir.

Bu örnekten çıkaracağımız sonuç şudur; Genel olarak yazılmış sınıfların kullanılabilmesi için öncelikle bu sınıflar türünden değişkenler yaratılarak bu sınıfların birer kopyası ya da nesnesi çıkarılmalı ve sınıfa ait değişken ve metodlar bu nesneler aracılığıyla kullanılmalıdır. Ayrıca gördüğümüz gibi bir sınıfa ait metodlar kullanılırken sınıf nesnesinden sonra "." işareti ve daha sonra da metodun adı yazılmaktadır.

Bu örneğimizde "a" ve "b" isimli sınıf nesneleri yaratıldıktan sonra, sahip oldukları "x" değişkenlerine "setX" metodu ile değerler verilmektedir. Peki eğer biz daha nesneyi yaratorken hemen o anda "x" değişkenine değer vermek istersek ne olacak? Bunun için "başlangıç metodu" kavramını bilmemiz gerekir.

Sınıfların Başlangıç Metodları (Constructors):
Sınıfların başlangıç metodları bizim bildiğimiz anlamda metodlardan biraz daha farklıdır. Bu metodların en önemli özelliği, bir sınıf nesnesi yaratılırken otomatik olarak çağrılmalarıdır. Mesela "DegiskenTutan" sınıfı türünden bir nesne yaratılırken aslında "new" anahtar sözcüğünden sonra bir metod çağrıldığını görmekteyiz:

```
DegiskenTutan a = new DegiskenTutan();
```

İşte bu "DegiskenTutan()" isimli metod, "DegiskenTutan" isimli sınıfın başlangıç metodudur. Peki o zaman başlangıç metodlarının normal metodlardan farkı nedir?

1 Başlangıç metodları kesinlikle ve kesinlikle sınıf ismiyle aynı olmak zorundadır. Eğer başlangıç metodunun ismini verirken sınıf isminden farklı bir isim kullanırsanız bu metod artık başka bir metod olarak kabul edilir. Örneğin:

DegiskenTut -> Yanlış. Sınıf ismi DegiskenTutan o halde başlangıç metodunun ismi bu olamaz

Degiskentutan -> Yanlış. Sınıf isminde T harfi de büyük. İsim, büyük küçük harf bazınbda da aynı olmak zorundad?r.

2 Başlangıç metodları geri dönüş değerine sahip değildir. Yalnız burada geri dönüş değeri void demek istemiyorum. Başlangıç metodlarının geri dönüş değeri gibi bir kavramları yoktur. void ile yok başka anlamlara gelir.

3 Bir sınıfın birden fazla başlangıç metodu olabilir. Bütün başlangıç metodları da sınıf ismiyle aynı olmak zorundadır ancak her birinin parametreleri birbirlerinden farklı olmak zorundadır.

Biz başlangıç metodlarını genelde bir sınıf türünden bir nesne yaratırken, bu nesnenin değişkenlerine birtakım ilk değerler vermek amacıyla yazarız. Aslında tasarladığımız bir sınıfa başlangıç metodu yazmak zorunda değiliz. Ancak başlangıç metodu yazmamışsak, nesne yaratılırken Java **otomatik olarak** bizim yerimize boş bir hayali başlangıç metodu yazar ve onu çağırır. Eğer yazmışsak bizim yazdığımız başlangıç metodunu çağırmaktadır.

Şimdi tüm bu anlattıklarımızı "DegiskenTutan" sınıfımıza başlangıç metodları ekleyerek irdeleyelim. Ama bu sefer "DegiskenTutan" ve "GenelSinif" isimli sınıfları "DegiskenTutan2" ve "GenelSinif2" olarak değiştirelim. Tabi aynı değişikliği dosya isimlerinde de yapmayı unutmayalım:

```
/*** DegiskenTutan2.java ***/
class DegiskenTutan2 {
      int x;

      public DegiskenTutan2() {
            this.x = 0;
      }

      public DegiskenTutan2(int c) {
            this.x = c;

      }

      public void setX(int c) {
            this.x = c;
      }

      public int getX() {
            return this.x;
      }
}
```

```
/*** GenelSinif2.java ***/
import javax.swing.*;

public class GenelSinif2 {

  public static void main(String arg[]) {

    DegiskenTutan2 a = new DegiskenTutan2();
    DegiskenTutan2 b = new DegiskenTutan2(2);

    String message = "a daki x degeri: "+a.getX();
    message += "\nb deki x degeri: "+b.getX();

    JOptionPane.showMessageDialog(null, message);

    System.exit(0);
  }
}
```

Bu örnekte görüyoruz ki; "GenelSinif2" dosyasının 7. satırında, "a" isimli "DegiskenTutan2" türünden sınıf değişkeni, "DegiskenYutan2" sınıfının 5. satır?nda tanımlanan boş olan başlangıç metodu ile yaratılıyor ve bu metodun içerisinde de "a" nesnesinin "x" değişkenine 0 değeri veriliyor. Ancak "GenelSinif2" çerisinde 8. satırda yaratılan "b" isimli ve "DegiskenTutan2" türündeki nesne, "DegiskenTutan2" isimli sınıfın 9. satırda tanımlanmış ve int parametre alan başlangıç metoduyla yaratıldığı için bu "b" nesnesine ait olan "x" değişkeni nesne yaratılırken verilen 2 değerini alıyor. Bu örnekte, bir sınıf türünden nesne yaratılırken yapılmasını istediğimiz birtakım ilk işlemleri başlangıç metodları yardımıyla yaptırabildiğimizi görüyoruz.

NOT: Başlangıç metodları sadece nesne yaratılırken çağrılır. Başka bir zamanda bu metodların çağrılmasına izin verilmez.

main Metodu
Şu ana kadar dikkat ederseniz "main" metodu ile ilgili çok az ayrıntıdan bahsettim. Aslında bu metod da birtakım özelliklere sahip özel bir metoddur. Şimdi yukarıda yapmış olduğumuz örneklere bir göz atmanızı istiyorum. Bu örnekler de "GenelSinif" isimli sınıf içerisinde "main" metodu varken "DegiskenTutan" isimli sınıfın içerisinde böyle bir metod yok. Ama her iki örnek de düzgün bir şekilde çalışabiliyor. Bunun nedenini şöyle açıklayalım:

Bu ana kadar gördük ki bir sınıf nesnesi ilk önce yazılmakta ve daha sonra da kullanılmaktadır. Ancak kullanılmadan önce tüm nesneler öncelikle "javac" komutu ile derlenmelidir. İşte bir java dosyasının "javac" ile derlenmesi işleminden sonra bir de "java" komutu ile çalıştırılabilmesi için, o java dosyası (ya da sınıfı) içerisinde "main" metodu olması gerekir. Yani çalıştırılabilir sınıfların main metodunu sahip olmaları gerekir. Aksi takdirde

java <Sinif_adi>

komutundan sonra ekranda aşağıdaki gibi "main" metodunun bulunamadığına dair bir mesaj ile karşılarsınız:

"java.lang.NoSuchMethodError: main"

Ancak "DegiskenTutan" gibi sadece başka bir sınıf içerisinde kullanılmak üzere tasarlanmış sınıflar doğrudan çalıştırılmayacağı için, onların içerisinde main metodu olmasına gerek yoktur. Bu tür sınıfların yukarıdaki örnekte de görüldüğü gibi sadece derlenmesi, kullanılabilmeleri için yeterli olacaktır.

Özetle; yazdığımız programlarda birbirinden farklı görevleri üstlenmiş bir çok sınıfı barındıran java dosyaları ve hepsini en son olarak tek çatı altında toplayacak ve asıl programın çalışmasını sağlayacak genel bir sınıfımız olacaktır. Ve tüm bu sınıflar içerisinde sadece ana programı çalıştıracak olan bu genel sınıf içerisinde "main" metodu olmalıdır. Diğer sınıfları kullanabilmeniz için, genel sınıf da dahil olmak üzere hepsini java ile derlemelisiniz. Ancak programı çalıştırmak için sadece genel sınıfı java komutu ile çalıştırmanız yeterlidir.

Sonuç olarak; bir sınıf java ile çalıştırıldığı zaman, otomatik olarak içerisindeki main metodu çağrılacaktır. Eğer sınıf içerisinde main metodu yoksa, böyle bir sınıf "javac" ile derlenebilir ancak "java" ile çalıştırılamaz!

main Metodu parametreleri:
"main" metodunun şu ana kadar hiç değinmediğimiz bir özelliği de aldığı parametrelerdir. Bu metod yazılırken görmekteyiz ki

public static void main(String arg[])

şeklinde bir String dizisini parametre olarak almaktadır. "İyi de bu parametre ne işe yarıyor?" diye merak ettiğinizi tahmin ediyorum. Artık bunu öğrenme zamanım?z geldi. Öncelikle aşağıda verilen "MainParametreleri.java" isimli sınıfı yazıp kaydediniz:

```
class MainParametreleri {

	public static void main(String arg[]) {

		System.out.println("1. parametre:"+arg[0]);
		System.out.println("2. parametre: "+arg[1]);

		System.exit(0);
	}

}
```

Şimdi aşağıda gösterildiği gibi bu sınıfı önce javac komutu ile derleyip daha sonra java komutunun bu özel kullanımı ile çalıştırınız:

javac MainParametreleri.java
java MainParametreleri bir iki

Bu şekilde sınıfı çalıştırdığınızda ekranda şöyle bir çıkış göreceksiniz.

1. parametre:bir
2. parametre: iki

İşte burada, java komutu ile "MainParametreleri" sınfını çalıştırırken sırasıyla verdiğiniz "bir" ve "iki" isimli bu iki String değeri, main metodunun pototipinde tanımlanmış olan "arg" isimli String dizisine atulmıştır. "main" metodunun paramtresi olan bu "String arg[]" dizisi bu örnekte olduğu gibi bir sınıf çalıştırılırken kullanıcının vermek istediği parametreleri alır ve daha sonra program içerisinde kullanılabilmesi için saklar. Örnek programımızın 5. ve 6. satırlarında "arg[0]" ve "arg[1]" ile bu girilen parametreleri kullanabildiğimzi görmekteyiz. Buradaki kullanma şeklimiz bu parametreleri sadece ekrana yazmak gibi anlamsız bir işlem gibi görünse de daha sonra işe yarar şekillerde kullanabileceğimizi düşünebiliriz.

1. parametre: bir
2. parametre: iki

Sınıfların public ve private Metod ve Değişkenleri
Şu ana kadar belirli işleri yapabilmek amacıyla metod ve değişkenleri birarada tutan nesneleri yaratmayı ve kullanmayı, yani sınıfların genel olarak kullanımını görmüş olduk. Bir sınıf içerisinde çeşitli metod ve değişkenler tutuyor ve daha sonra biz de bu sınıf türünden nesneler yaratarak bu metod değişkenleri kullanabiliyoruz.

Bazen bir sınıf içerisinde yazılan metod ya da değişkenleri sadece sınıfın içerisinde kullanmak isteyebiliriz. Yani başka bir yerde bu sınıf türünden yaratılan bir değişken ile kullanmayacağımız ancak sınıfın kendi içinde kullanılması gereken metod ve değişkenler olabilir. Bunu anlamak için "DegiskenTutan" sınıfına tekrar dönelim. Biz "DegiskenTutan" sınıfı türünden bir "a" nesnesi yarattığımızda "a" nesnesi kendisine ait olan "x" değişkenini:

```
a.x = 10;
```

şeklinde değiştirebilir. Ancak biz bu şekilde bir müdaheleye izin vermek istemiyor olabiliriz. Eğer "a" nesnesinin "x" değişkeni değiştirilecekse, böyle bir işlemin "setX" gibi bir metodla yapılmasını, değişkene "a.x" diyerek doğrudan erişilememesini istiyor olabiliriz. İşte bu durumlarda sınıf içerisine "x" değişkeninin başına bir "private" anahtar sözcüğü eklememiz gerekir. Yani "DegiskenTutan" sınıfında "x" değişkeninin bildirimi:

```
class DegiskenTutan {
      private int x;

      public Degiskentutan() {

      ...
```

şeklinde olursa, artık "DegiskenTutan" türünden "a" gibi bir nesne yaratılsa bile, "a.x" gibi bir erişim yasaklanmış olur. Ancak sınıf içerisindeki bir metod "x" değişkenine yine de ulaşabilir.

Eğer genel olarak özetlemek gerekirse; sınıflar içerisinde bildirilen "private" türündeki değişkenlere dışarıdan erişilemez, ancak sınıfın diğer metodları ile erişilebilir.

Bu durum sınıfın private metodları için de geçerlidir. Eğer sınıf yazılırken bir metodun başına "private" anahtar sözcüğü eklenirse, bu metod da artık dışarıdan erişilemez, sadece sınıf içerisindeki diğer metodlar tarafından kullanılabilir bir hale gelir. Mesela "DegiskenTutan" isimli sınıfa aşağıdaki gibi bir "private" metod ekleyelim:

```
class DegiskenTutan {
      private int x;
      ...
      private void resetX() {
            this.x = 0;
      }
      ...
```

```
	public setX(int c) {
		this.resetX();
		this.x = c;
	}
}
```

Bu şekilde "private" olarak bildirilmiş "resetX" metodu artık bu sınıf türünden yaratılmış nesneler ile dışarıdan kullanılamaz. Yani aşağıdaki gibi bir kullanım geçersizdir:

```
class BaskaSinif {
	public static void main(String arg[]) {
		DegiskenTutan a = new DegiskenTutan(2);
		a.resetX();
/*	YANLIŞ! resetX private bir metoddur
Bu şekilde dışarıdan kullanılamaz.*/
		...
	}
}
```

En son örnekte gördüğünüz gibi "a.resetX()" gibi bir çağrım geçerli değildir. Çünkü "DegiskenTutan" sınıfının "resetX" isimli metodu "private" bir metoddur. Bu nedenle, bu sınıf türünden yaratılmış olan "a" nesnesi aracılığıyla bu metoda dışarıdan erişilemez. Ancak önceki örnekteki gibi "resetX" metodunu sınıfın diğer bir metodu olan "setX" metodu kullanabilmektedir. Değişken ya da metodların "private" ya da "public" olması, bu metod ya da değişkenlerin sadece dışarıdan kullanımı etkileyen bir durumdur. Sınıf içerisindeki diğer metodlar bu "private" değişken ya da metodlara erişebilirler.

Tüm bunların bir sonucu olarak da , sınıflar yazılırken yaratılan "public" metod ve değişkenler, sınıf nesneleri aracılığıyla dışarıdan erişilebilirler.

Statik Metod ve Değişkenler

Şu ana kadar öğrendiklerimizi ele aldığımızda neden **JOptionPane** sınıfının **showMessage** isimli metodunu kullanmak için JOptionPane sınıfı türünden bir nesne yaratmıyoruz? diye sormamız gerekir.

Bazı metodlar yazıldıkları sınıflar içerisinde statik olarak yazılırlar. Bu türden statik olarak yazılmış metodları kullanmak için sınıf nesnesi yaratmak gerekmez. Bu metodları kullanmak için yazıldıkları sınıf isminden sonra "." işareti koyarak doğrudan metodu çağırabilirsiniz.

Ancak tahmin edersiniz ki bir metod statik olarak yazılmışsa genel amaçlı bir metod olmalıdır. Mesela JOptionPane isimli sınıfın showMessageDialog ya da showInputDialog isimli metodları genel amaçlı olup her zaman kullanılabilmektedir. Ekrana bir mesaj penceresi çıkarmak ya da kullanıcıdan bir giriş değeri almak için bir nesne yaratmaya gerek yoktur. Siz de kendi tasarladığınız sınıflar içerisinde bu şekilde genel amaçlı metodlar yazmak isterseniz bunları statik olarak yaratabilirsiniz.
Bir metodun statik olarak yaratılabilmesi için, metodun prototipinin başına "static" anahtar sözcüğü eklenir. Aşağıdaki iki örneği yazıp kaydediniz:

```
import javax.swing.*;
class StatikTest {

	String mesaj;

	public StatikTest() {
		mesaj = "";
	}

	public StatikTest(String m) {
		this.mesaj = m;
	}

	public static void ozelMesaj(String m) {
		JOptionPane.showMessageDialog(null, m);
```

```
	}

	public void kayitliMesaj() {
		JOptionPane.showMessageDialog(null, this.mesaj);
	}
}
class GenelSinif3 {

	public static void main(String arg[]) {

		StatikTest.ozelMesaj("Bu mesaji dogrudan gosterdim");

		StatikTest a = new StatikTest("Bu mesaj icin nesne yarattim");
		a.kayitliMesaj();

	}
}
```

Bu örneği çalıştırdığınızda ekrana iki tane mesajın geldiğini göreceksiniz. Bu mesajlardan ilki, "GenelSinif3" isimli sınıfın 5. satırında, "StatikTest" sınıfının statik metodu doğrudan çağrılarak ekrana gelmiştir. Gördüğümüz gibi, **bir sınıfın statik metodunu kullanmak için** ilk önce **o sınıf türünden bir nesne yaratmaya gerek yoktur**. Çünkü statik metodlar sınıf ismi ve sonra ". " işareti yazıldıktan sonra doğrudan çağrılabilirler. Ancak ikinci mesaj "StatikTest" sınıfının "kayitliMesaj" isimli ve statik olmayan metodu sayesinde ekrana geldiğinden, bu metodu kullanabilmek için ilk önce "StatikTest" sınıfı türünden bir nesne yaratılması ve metodun da bu nesne aracılığıyla çağrılması gerekir.

Uygulama Sorusu: İçerisinde tarih tutan ve tarihlerle ilgili işlemler yapabilen metodları barındıran bir sınıf tasarlayınız. Yapılması gereken işlemler:

a. iki tarihin birbiriyle kıyaslanması,
b. tarih üzerine belirli bir gün sayısı eklenebilmesi,
c. tutulan tarihin gösterilmesi,
d. tutulan tarihin değiştirilmesi şeklinde olmalıdır.

Bu işlemlerin her biri için sınıf içerisinde bir metod tasarlamanız gerekmektedir. Böyle bir sınıfın günü, ayı ve yılı tutan değişkenleri olmalıdır. Sınıfı tasarladıktan sonra, bu sınıfı kullanan ve bu sınıf türünden örnek nesneler yaratarak bunları kullanan başka bir genel sınıf yazınız ve tasarımınızı bu nesneler üzerinde test ediniz.

Nihayet bu dersimizin de sonuna gelmiş bulunuyoruz. Nesne yönelimli programlama tekniğinin özünü oluşturan sınıfları ve sınıfların kullanımını öğrendiğimiz bu dersimizi iyice sindirmenizi ve örnekleri de tekrar yazarak kendi başınıza denemenizi ısrarla öneririm.

Uygulama Sorusu2: Icerisinde array tutan (int turunden) bir sinif yaratiniz (IntArray). Bu sinif icerisinde tuttugu bu array i dinamik olarak genisletebilmelidir. Ozetle bu sinifta asagidaki metdolar yer almalidir.

```
void add(int toAdd);
void add(int []toAddArray);
int findMax();
int findMin();
int remove(int index);
int display();
int reverseDisplay();
public IntArray(int a);
public IntArray(int []a);
public IntArray();
```

Java ile Programlama
Bölüm 6:

Sort yontemleri (bubble sort ve selection sort) anlatılacak.

ODEV: String diziyi sort eden bir metod ornegi (Bir sonraki derste inceleyecegiz)

Uygulama Sorusu (Derste yapilacak):
Sınıfların dizilere parametre olarak gönderilmesine iliskin ornekler
- Person turunden bir sinif yarat (isim, yas)
- Bu sinif turunden elemanlari olan bir array yarat
- Bu arrayi bir metoda gonder ve ekranda goruntule
- Sonra compareTo metodunu Person sinifina ekle ve sort bilgisiyle bu sefer bu listeyi sirala

```
public class Person {

	private String name;
	private int age;

	public Person() {

		this.setName("No person");
		this.setAge(0);
	}

	public Person(String name, int age) {
		this.setName(name);
		this.setAge(age);
	}

	public String toString() {
		return "Person Name: " + this.getName() + "\nPerson Age: " +
this.getAge();
	}

	public static void sortPersonArray(Person p_array[]) {

		/*for(int i = 0; i < p_array.length; i++)
			for(int j = 0; j < p_array.length; j++)
				if(p_array[i].compareTo(p_array[j]) > 0)
					swap(p_array[i], p_array[j]);*/

		for(int i = 0; i < p_array.length; i++)
			for(int j = i+1; j < p_array.length; j++)
				if(p_array[i].compareTo(p_array[j]) > 0)
					swap(p_array[i], p_array[j]);
	}

	private static void swap(Person p1, Person p2) {

		Person temp = new Person();

		temp.setAge(p1.getAge());
		temp.setName(p1.getName());

		p1.setAge(p2.getAge());
		p1.setName(p2.getName());

		p2.setAge(temp.getAge());
		p2.setName(temp.getName());
	}
```

```
	public int compareTo(Person p) {

		return this.getName().compareTo(p.getName());

	}



	/**
	 * @return Returns the age.
	 */
	public int getAge() {
		return age;
	}
	/**
	 * @param age The age to set.
	 */
	public void setAge(int age) {
		while(age < 0)
			age = Integer.parseInt(
					JOptionPane.showInputDialog(null, "Yas bilgisi 0 ya da
daha kucuk olamaz!\nTekrar giriniz"));

		this.age = age;
	}
	/**
	 * @return Returns the name.
	 */
	public String getName() {
		return name;
	}
	/**
	 * @param name The name to set.
	 */
	public void setName(String name) {
		this.name = name;
	}

	public static void main(String[] args) {

		Person array[] = {new Person("ali", 12), new Person("ahmet", 24), new
Person("zeren", 13)};
		Person.sortPersonArray(array);

		for(int i = 0; i < array.length; i++)
			System.out.println(array[i]);

	}
}
```

Bu derse kadarki derslerimizde artık nesne yönelimli programlama tekniğinin merkezinin oluşturan sınıfların ne anlama geldiklerini ve nasıl tasarlandıklarını öğrenmiştik.

Bugün tasarımını ayrıntılı olarak öğrendiğimiz sınıfların kullanımına ilişkin bazı örnekler üzerinde duracağız. Ayrıca Java'da öntanımlı olarak bulunan ve kendi özeli işlerimiz için kullanabileceğimiz sınıfların bazılarını da görmüş olacağız. Size tavsiyem burada anlatılan ekleri mutlaka kendi başınıza da yazıp denemenizdir.

Sınıfların Kendi Kendilerini Kullanmaları:
Bundan önceki örneklerimizde bir sınıfın yazıldıktan sonra, başka bir sınıf içerisinde ya da örneklerde yaptığımız gibi main metodu içerisinde kullanılabildiğini görmüştük. Bazı sınıflar otomatik olarak çalıştırılabilmek amacıyla da

yazılabilirler. Bir sınıfın otomatik olarak çalışabilmesi için içerisinde main metodu olması gerektiğini zaten biliyoruz. İşte bu şekilde main metodu içerisinde bir sınıf, kendisi türünden nesne yaratıp bunu kullanabilir.
İşin özü aslında şudur: Bir sınıf yazıldıktan sonra, bu sınıf türünden nesneleri istediğimiz her metod ya da her sınıf içerisinde artık yaratıp kullanabiliriz. Bu genel kuralı bilirsek, zaten sınıf nesnelerinin ne zaman ve nerede yaratılacağına dair bir problemimiz de kalmaz.
Aşağıdaki örneğimizde, matematikte kullanılan karmaşık sayılara ilişkin değer ve metodları barındıran bir sınıf yazacağız. Bu sınıf aynı zamanda "main" metodu içerecek ve kendisi türünden nesneler yaratarak bu nesneler üzerinde işlemler yapacak:

```
1  import javax.swing.*;
2
3  class KarmasikSayi {
4    double gercel, imajinel;
5
6    public KarmasikSayi() {
7      this.gercel = 0;
8      this.imajinel = 0;
9    }
10
11   public KarmasikSayi(double g, double i) {
12     this.gercel = g;
13     this.imajinel = i;
14   }
15
16   public KarmasikSayi topla(KarmasikSayi digerSayi) {
17     KarmasikSayi sonuc = new KarmasikSayi(this.gercel+digerSayi.gercel,
18                                           this.imajinel+digerSayi.imajinel);
19     return sonuc;
20   }
21
22   public KarmasikSayi cikart(KarmasikSayi digerSayi) {
23     return new KarmasikSayi(this.gercel-digerSayi.gercel,
24                             this.imajinel-digerSayi.imajinel);
25   }
26
27   public void sayiyiGoruntule() {
28     mesaj(this.gercel+" + "+this.imajinel+"i");
29   }
30
31   public void mesaj(String mes) {
32     JOptionPane.showMessageDialog(null, mes);
33   }
34
35   public String toString() {
36     return this.gercel+" + "+this.imajinel+"i";
37   }
38
39   public static void main(String[] args) {
40     KarmasikSavi savi1 = new KarmasikSavi(3.0,5.0);
41     KarmasikSayi sayi2 = new KarmasikSayi(4.0,-3.2);
42     KarmasikSayi sonuc;
43
44     sonuc = sayi1.topla(sayi2);
45     sonuc.sayiyiGoruntule();
46
47     System.out.println("sayi 1: "+sayi1);
48    }
49
50 }
```

Şimdi bu örneğimizi biraz inceleyelim. Sınıfımızın iki adet başlangıç metodu olduğunu görmekteyiz. Bu metodlardan birtanesi, 6. satırda tanımlanmakta ve hiçbir parametre almadan yaratılan KarmasikSayi nesnesinin "gercel" ve "imajinel" isimli değişkenlerini 0.0 değerine eşitlemektedir. Diğer başlangıç metodu ise 11. satırda tanımlanmaktadır

ve nesne yaratılırken kendisine verilen parametreleri kullanarak "gercel" ve "imajinel" değişkenlerinin değerlerini atamaktadır.

NOT: Sınıfın kendisine ait olan değişken ve metodlarını sınıf içerisinde kullanırken, bu değişken ve metodların başına "this." eklemiş olduğuma dikkat ediniz. Daha önce de söylediğim gibi this anahtar sözcüğü sınıfın kendisini temsil eder. Örneğin 12. satırda "this.gercel" ifadesi, "o anda içerisinde bulunduğumuz sınıfa ait olan gercel isimli degiskeni" temsil etmektedir.

16. satırda tanımlanan "topla" isimli metod parametre olarak başka bir "KarmasikSayi" nesnesi alarak, bu sayının "gercel" ve "imajinel" değişkenleri ile sınıfın kendi "gercel" ve "imajinel" değerleri toplamını elde eder ve bu değerlerle, "sonuc" adında yeni bir "KarmasikSayi" nesnesi yaratarak bu değişkeni 19. satırda geri döndürür. Buna benzer şekilde 22. satırda yazılmış olan "cikart" metodu da aynı işlemi çıkartma için yapar. Ancak bu metod biraz daha ilginç bir yöntemle değer döndürmektedir. Bu sefer sonucu tutacak ayrı bir değişken yaratmak yerine, 23. satırda olduğu gibi "return" anahtar sözcüğünün önünde yeni bir "KarmasikSayi" nesnesi yaratılarak hemen o anda döndürülür. Bu yöntemde arada "sonuc" gibi başka bir değişken içerisinde "KarmasikSayi" nesnesi yaratma ihtiyacı ortadan kalkmaktadır.

Yazdığımız Sınıfların toString metodu:
Dikkat ederseniz 35. satırda "toString" adında ilginç bir metod yer almaktadır. Bu metod aslında main metodu içerisinde hiç çağrılmamıştır. Ancak ilginçtir ki, 47. satırdaki "sayi1" isimli sınıf değişkeni ekrana yazılmak istendiği zaman sanki bu metod çağrılmaktadır. Çünkü ekranda toString metodunun döndürdüğü String ifadesi yazılmaktadır. Aslında gerçekten de 47. satırda sayi1 değişkeninin ekrana basıldığı noktada otomatik olarak bu metod çağrılır.

Yazılan tüm sınıflara aslında toString metodu da eklenebilir. Eğer

"String toString()"

prototipini bozmadan sınıfınız içerisinde bir toString metodu yazarsanız, bu sınıfa ilişkin bir sınıf değişkenini ekrana basmak istediğinizde bu metod otomatik olarak çağrılmaktadır. Ancak metodun içini istediğiniz gibi yazabilirsiniz. Eğer yazdığınız sınıf içerisinde bu metodu eklemez ve bu sınıf türünden bir değişkeni ekrana basmak isterseniz, o zaman Java otomatik olarak bir toString metodunu kendisi yazacak ve buna ilişkin bir String değeri de dödürecektir. Ancak bu değer oldukça anlaşılmaz bilgiler içermektedir. Özetle sınıfınızı nasıl ifade etmek istiyorsanız, buna ilişkin String toString() prototipli bir metod yazabilirsiniz.

Öntanımlı Java Sınıflarının Kullanımı:
Dersimizin bundan sonraki kısmında yeni bir konuya geçmeden önce, Java'da öntanımlı bazı sınıflara ilişkin örnekler yapmakta yarar görüyorum. Göreceğimiz bu sınıflar, yazacağımız programlarda birçok yükü üzerimizden almakta ve bize bazı yararlı hazır metodlar sunmaktadır. Bu nedenle işimizi görecek ve daha önceden yazılmış bu hazır sınıf ve metodları kullanarak zaman kaybı ve zahmetten kurtulabiliriz.

Bir konuda sizleri uyarmak isterim: Burada bu öntanımlı java sınıflarının tüm metodlarını detaylı bir şekilde göstermek yerine, bu sınıflar hakkında bilgi sahibi olmanızı ve sadece birkaç metod kullanımıyla sınıfı sizlere tanıtmayı hedefledim. Kendi uygulamalarınız için burada genel olarak öğrenmiş olduğunuz bu sınıfların size uygun metodlarını ve burada gösterilmeyen başka öntanımlı sınıfları java dökümantasyonu ve internette yapacağını aramalar ile bulabilirsiniz. Java dökümantasyonunun online halini:
"http://java.sun.com/j2se/1.4.2/docs/api/index.html" adresinde bulabilirsiniz.

Özetle buradaki örnekleri sizlere sadece neyi nerede arayacağınızı bilmeniz açısından veriyorum. Şimdi bu sınıfları incelemeye başlayalım.

Math Sınıfı:
Bu sınıf çeşitli matematiksel işlemleri yapabileceğimiz metodları sunmaktadır. Bu metodlar içerisinde sin, cos, tan, kuvvet alma, kök alma v.b. metodlar bulunmaktadır. Bu sınıfı kullanabilmeniz için uygulamalarınızın başında herhangi bir kütüphaneyi "import" etmenize gerek yoktur. Aşağıda Math sınıfınının bazı metodlarının kullanımına ilişkin bir örnek verilmektedir. Bu örnek, daha önce yapmış olduğumuz karmaşık sayı sınıfının biraz daha geliştirilmiş hali ve Math sınıfının fonksiyonlarının kullanıldığı bir örnektir. Ayrıca bu sınıfa "aciHesapla" adında yeni bir metod daha eklenmiş ve sınıfın içerisinde saklanan karmaşık sayının açı değerleri bu metod yardımıyla hesaplanmıştır.

```
1
2
3  import javax.swing.*;
4
5  class KarmasikSayi2 {
6
7    double gercel, imajinel;
8
9    public KarmasikSayi2(double g, double i) {
10     this.gercel = g;
11     this.imajinel = i;
12   }
13   public double imjineliVer() {
14     return this.imajinel;
15   }
16   public double gerceliVer() {
17     return this.gercel;
18   }
19   public String sayiyiVer() {
20     return this.gerceliVer()+" + "+this.imjineliVer()+"i";
21   }
22   public double normuVer() {
23    return Math.sqrt((Math.pow(this.gerceliVer(), 2)+
24                      Math.pow(this.imjineliVer(), 2)));
25   }
26   public double aciyiRadyanVer() {
27     return Math.atan2(this.imjineliVer(),this.gerceliVer());
28   }
29   public double aciyiDereceVer() {
30     return  this.aciyiRadyanVer() * 180 / Math.PI;
31   }
32   public String sayiyiAciVeNormIleVer() {
33     return "|"+this.normuVer()+"| * Q["+this.aciyiDereceVer()+"]";
34   }
35   public void sayiyiGoruntule() {
36     mesaj(this.gercel+" + "+this.imajinel+"i");
37   }
38   public void mesaj(String mes) {
39     JOptionPane.showMessageDialog(null, mes);
40   }
41   public void sayininTumBilgileri() {
42     String m = "z = "+this.sayiyiVer() +
43                                " sayisina iliskin bilgiler:\n**********\n";
44     m += "\nSayinin gercel kismi: Re(z) =  "+this.gerceliVer();
45     m += "\nSayinin imajinel kismi: Im(z) = "+this.imjineliVer();
46     m += "\nSayinin normu: |z| = "+this.normuVer();
47     m += "\nSayinin acisi (radyan olarak): Q(z) = "+this.aciyiRadyanVer()+"rad";
48     m += "\nSayinin acisi (derece olarak): Q(z) = "+this.aciyiDereceVer()+
49                                                          " derece";
50     m += "\nSayinin diger ifadesi: z = "+this.sayiyiAciVeNormIleVer();
51     mesaj(m);
52   }
53   public static void main(String[] args) {
54     KarmasikSayi2 sayi = new KarmasikSayi2(3.0, 4.0);
55     sayi.sayininTumBilgileri();
56
57     System.exit(0);
58   }
59 }
```

Şimdi örneğimizi daha detaylı inceleyelim. Herşeyden önce genel bir noktayı vurgulamak istiyorum. Dikkat ederseniz "KarmasikSayi2" sınıfının 23, 24, 27 satırlarında, "gercel" ve "imajinel" değerlerini "this.gercel" diyerek doğrudan almak yerine bu değerleri 13 ve 16. satırlarda yazılmış olan "gerceliVer" ve "imajineliVer" metodları yardımıyla alıyoruz. Bunun nedeni sınıfın veri elemanlarını korumak ve böylece sınıf hiyerarşisini düzgün bir şekilde kurmak içindir.

Bu teknik bizlere oldukça önemli ve sağlam bir mekanizma sunar. Zaten dikkat ederseniz "gercel" ve "imajinel" isimli veri elemanları da dışarıdan doğrudan erişimin engellenmesi amacı ile 7. satırda "private" olarak tanımlanmıştır. Veri elemanlarına erişimin bu şekilde kontrollü olarak yapılması gerekir. Eğer bu elemanlara erişilmek gerekiyorsa, bu işi doğrudan değil de set ve get metodları gibi metodlar yardımıyla yapmak gerekir. Kullanıcıya sadece bu metodlar sunulmalı, sınıfın veri elemanlarına erişim ise metodlar içerisinde ve gerektiği gibi yapılmalıdır.

23. satırda kullanılan "Math.pow" isimli metod 1. parametre ile aldığı sayının, 2. parametre ile aldığı sayı kadar kuvvetini hesaplar ve sonucu geri döndürür. Bu metodu "gercel" ve "imajinel" kısımların karelerini hesaplamak için kullandık. Bu karelerin toplamının da karekökünü, tahmin ettiğiniz gibi Math.sqrt metodu ile elde ettik. Yani özetle;

Math.pow değerleri ile elde edilen "gercel" ve "imajinel" kısımların karelerinin toplamını, Math.sqrt metoduna paramtere olarak doğrudan göndererek sonucu da doğrudan geri döndürdük. Sizler de bol bol program yazdıkça bu tip hızlı yazım tekniklerinizi geliştireceğinizden emin olabilirsiniz. Ancak pratik olması amacıyla kısa kodlar yazarken bile kodunuzun okunabilir olmasına dikkat ediniz.

Örneğimizin 27. satırında kullandığımız "Math.atan2" isimli metod kendisine parametre olarak verilen "y" ve "x" gibi iki sayının polar koordinatlarda sahip olduğu açı değerini hesaplayan bir metoddur. Yani aslında yaptığı işlem y/x değerinin arktanjant değerini hesaplamaktır. Fakat geri dönen bu değer radyan cinsindendir. Bu nedenle, radyan olarak elde edilen açı değeri 29. satırda yazılan "aciyiDereceVer" metodu ile derece olarak da elde edilebilir.

Hızlı bir şekilde listelemek gerekirse Math sınıfının bazı diğer metodları şunlardır:

int Math.abs(int a)	a sayısının mutlak değerini geri döndürür. Aynı metodun float, double ve long türler için de versiyonları vardır.
double Math.acos(double a)	a sayısnın ters kosinüs değerini döndürür. Benzer şekilde asin ve atan metodları da vardır.
double Math.cos(double a)	a sayısnın cosinüz değerini geri döndürür. Benzer şekilde sin metodu da vardır.
double Math.exp(double a)	a sayısnın e sayısını kuvveti olarak hesaplar ve bu değeri döndürür.
int Math.max(int a, int b)	a ve b sayılarından büyük olanın değerini geri döndürür. Benzer şekilde bu metodun double, int, float ve long versiyonları da vardır.
double Math.random()	0 ile 1 arasında rastgele pozitif bir sayı üretir ve bu değeri döndürür.

DecimalFormat Sınıfı:
Java ile yazdığımız programlarda sayısal değerler çoğu zaman anlaşılması zor ve gereksiz uzunlukta üretilir. Ancak Java bizim için sayının ne kadar uzunlukta olması gerektiğini otomatik olarak bilemez ve bu nedenle en doğru ve orjinal şekilleriyle bu değerleri üretmektedir. Eğer yukarıdaki örneği yazıp çalıştırdıysanız, elde edilen sonuçların (açı değerleri gibi) virgülden sonra oldukça fazla miktarda basmak içerdiğini görmüşsünüzdür.

DecimalFormat sınıfı, sayıları istediğimiz gibi ifade edebilmemize olanak sağlar. Bu sınıfın kullanımına ilişkin bir örnek yazalım:

```
1
2
3  import javax.swing.*;
4  import java.text.DecimalFormat;
5
6  class OndalikKisim {
7
8    public static void main(String[] args) {
9      DecimalFormat ikiHane = new DecimalFormat("00.00");
10     double a = 1.45768943;
11
12     String str = "Sayinin ilk hali: " + a;
13     str += "\n\nSayinin yeni hali: " + ikiHane.format(a);
14     JOptionPane.showMessageDialog(null, str);
15
16   }
17 }
```

Örneğimize dikkat edersek, "DecimalFormat" sınıfını kullanmadan önce 4. satırda olduğu gibi "java.text.DecimalFormat" kütüphanesini programa dahil etmeliyiz. 9. satırda yaratılan bir DecimalFormat nesnesi yapmak istediğimiz ifade biçimini String bir değer olarak tutmaktadır. Bu nesne yaratıldıktan sonra, nesne

aracılığıyla istenilen bir sayı "format" metodu kullanılarak 13. satırda olduğu gibi biçimlendirme yapılabilir. "format" metodunun geri dönüş değeri String bir değerdir.

Sargı Sınıfları:
Java'da yaygın olarak kullanılan sınıflar arasında sargı sınıfları da vardır. Bu sınıflar aslında genel olarak aynı prensiple kullanılır. Bı sınıfları kullanmaktaki amaç bilinen temel türleri (integer, float, double gibi) String türüne ya da tersine, String türündeki bir ifadeyi temel türlere dönüştürmektir. Aslında bu tür sınıflara olan ihtiyacı örnek üzerinde açıklarsak daha anlaşılır olacaktır.

Mesela en son yaptığımız "OndalikKisim" sınıfı örneğinde, 10. satırda tanımlanan "a" değişkeni içerisindeki değerin biçimlendirilmiş hali String türünde elde edilmektedir. Ancak biz bu biçimlendirilmiş hal ile başka matematiksel işlemler yapmak istediğimizde String türündeki bu sonuç işimize yaramayacaktır. Bu sonucun tekrar double türünde elde edilmesi gerekir. Bunun için String türünde elde edilen bu değer "Double.parseDouble" metodu ile tekrar double türüne dönüştürülebilir. Bu metodun örnek kullanımı:

```
...
  double yeniBicim = Double.parseDouble(ikiHane.format(a));
   ..
```

şeklinde olacaktır. Bu dönüşümleri yapan diğer bazı sargı sınıfları ve metodları da şunlardır:

int Integer.parseInt(String str)	str ile gelen String biçimindeki sayıyı integer biçiminde geri döndürür.
String Integer.toString(int a)	integer türündeki a sayısının String türünde değerini geri döndürür.
double Double.parseDouble(String str)	str ile gelen String biçimindeki sayıyı double biçiminde geri döndürür.
String Double.toString(double a)	double türündeki a sayısının String türünde değerini geri döndürür.
float Float.parseFloat(String str)	str ile gelen String biçimindeki sayıyı float biçiminde geri döndürür.
String Float.toString(float a)	float türündeki a sayısının String türünde değerini geri döndürür.
String String.valueOf(int a)	integer türündeki a sayısnın String değerini geri döndürür. Aynı metodun double, float, ve long türünde parametre alan versiyonları da vardır

ArrayList Sınıfı:
Daha önce öğrenmiş olduğumuz diziler konusundan bildiğimiz gibi bir dizi yaratıldıktan sonra artık otomatik olarak boyutu değişmiyordu. Siz diziyi belirli bir uzunlukta açtıktan sonra dizinin tüm alanları doluysa, yeni bir eleman eklemek için birtakım algoritmik yöntemler uygulamanız gerekir. Ya da diziden bir eleman çıkardığınız zaman, diğer elemanları birer basmak geri kaydırmak gibi birtakım işleri de yapmanız gerekir.

"ArrayList" sınıfı bu tür görevleri otomatik olarak yapan ve bize dinamik olarak kullanabileceğimiz bir dizi mekanizması sunan bir sınıftır. Eğer böyle bir diziye sahip olmak istiyorsanız aşağıdaki şekilde bir "ArrayList" nesnesi tanımlamanız gerekir:

ArrayList dinamikDizi = new ArrayList();

Daha sonra bu nesneyi artık kendi dizinişmiş gibi düşünerek "ArrayList" sınıfının sunduğu metodlar yardımıyla eleman ekleme, çıkartma ve bulma gibi işlemler yapabilirsiniz. Mesela "dinamikDizi" nesnesine eleman eklemek için:

dinamikDizi.add("merhaba");

ya da

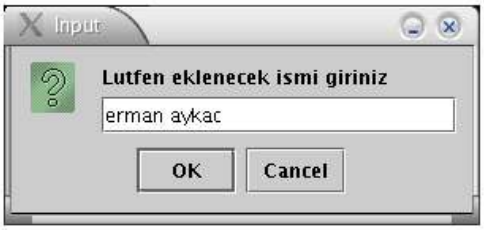
int a = 3;
dinamikDizi.add(a);

ArrayList sınıfının yukarıda gördüğümüz gibi "add" isimli metodunun yanısıra diğer metodlarından bazıları da şunlardır:

void remove(int index)	Dizinin verilen indeks numarasındaki elemanını çıkartır ve diziyi buna göre düzenler. Örnek kullanım: int a = dinamikDizi.remove(2); //2.eleman siliniyor
int size()	Dizinin içerisindeki elelman sayısını verir. Örnek kullanım: int a = dinamikDizi.size();
Object get(int index)	Dizinin index nolu elemanını geri döndürür. Örnek kullanım: String str = dinamikDizi.get(0);
int indexOf(Object o)	o ile temsil edilen nesnenin dizi içerisindeki indeks nuaarasını geri döndürür. Eğer eleman dizide yoksa -1 geri döndürür.

Aşağıdaki "KayitListesi.java" uygulamamızda hem böyle bir dinamik dizi kullanımını, hem de bugüne kadar öğrendiklerimizle gerçeğe oldukça yakın bir java uygulamasının yazımını görmekteyiz. Bu uygulama çalıştığında, ekrana resimdeki gibi bir menü sunulmakta ve kullanıcının yaptığı seçimlerle kayıt ekleme, silme, kayıtları listeleme gibi işlemler yapıldıktan sonra tekrar aynı menü ekrana gelmektedir. Kullanıcıya istediği zaman programdan çıkabileceği bir seçenek de sunulmaktadır.

Ekran 1: Program calistiginda ekrana gelen menu



Ekran 2: Kullanici ana menuden 2. secenegi seciyor

Ekran 3: Kullanici ana menuden 2. secenegi seciyor
Ancak listede olmayan bir kayit giriyor

Ekran 4: Kullanici ana menuden 3. secenegi seciyor

Ekran 5: Kullanici ana menuden 0. secenegi seciyor

```java
import javax.swing.*;
import java.util.*;

class KayitListesi {

  ArrayList liste = new ArrayList();

  private void menuEkrani() {

    String str="Lutfen seciminizi yapiniz:\n**********\n";
    str+="\n[1] Listeye Yeni Isim Ekle";
    str+="\n[2] Listeden Isim Sil";
    str+="\n[3] Listeyi Goster";
    str+="\n[0 ya da Cancel] Programi sonlandir";

    String secimStr = this.girisAl(str);
    if(secimStr==null)
      secimStr="0";
    else if(secimStr.equals("")||secimStr.matches("[^0-9]+"))
      this.menuEkrani();

    int secim = Integer.parseInt(secimStr);
    switch(secim) {
      case 0: this.cikis();break;
      case 1: this.kayitEkle(); break;
      case 2: this.kayitSil(); break;
      case 3: this.listeyiGoster(); break;
      default:JOptionPane.showMessageDialog(null, "Yanlis bir secim yaptiniz!");
      this.menuEkrani();break;
    }
  }


  private void kayitEkle() {

    String eklenecek= this.girisAl("Lutfen eklenecek ismi giriniz");
    if(eklenecek!=null && !eklenecek.equals("")) {
      if (liste.indexOf(eklenecek.toUpperCase()) != -1)
        JOptionPane.showMessageDialog(null, "Bu kayit zaten listeye eklenmis");
      else {
        liste.add(eklenecek.toUpperCase());
        JOptionPane.showMessageDialog(null, "Kayit Listeye Eklendi");
      }
    }
    this.menuEkrani();
  }


  private void kayitSil() {

    String silinecek= this.girisAl("Lutfen silinecek ismi giriniz");
    if(silinecek!=null && !silinecek.equals("")) {
      if(liste.indexOf(silinecek.toUpperCase()) != -1) {
        liste.remove(liste.indexOf(silinecek.toUpperCase()));
        JOptionPane.showMessageDialog(null, "Kayit listeden silindi");
      }
      else
        JOptionPane.showMessageDialog(null, "Listede zaten böyle bir kayit yok!");
    }
    this.menuEkrani();
  }


  private void listeyiGoster() {

    String str = "Liste\n**********\n";
    for(int i=0; i < liste.size(); i++)
      str += "["+i+"] "+liste.get(i)+"\n";

    str += "\n**********\n";
    str += "Toplam --> "+liste.size()+" kayit";
    mesaj(str);

    this.menuEkrani();
  }


  private void cikis() {

    int secim=JOptionPane.showConfirmDialog(null, "Cikmak istediginizden emin misiniz?");
    if(secim==0)
      System.exit(0);
    else
      this.menuEkrani();
  }


  public void uygulamaBaslat() {

    this.menuEkrani();
  }


  public void mesaj(String str) {

    JOptionPane.showMessageDialog(null, str);
  }


  public String girisAl(String str) {

    return JOptionPane.showInputDialog(null, str);
  }


  public static void main(String[] args) {

    KayitListesi a = new KayitListesi();
    a.uygulamaBaslat();
  }
}
```

KayitListesi.java Uygulamasındaki Nesne Yönelimli Programcılık İzleri
Uygulamamız içerisinde , her seçime ilişkin bir metod uygun işi üstlenmektedir. Ayrıca tüm alt görevleriyle ve değişkenleriyle birlikte bu uygulama tek bir sınıf altında ve tek bir amaç için oluşturulmuştur. Programlarınızda bu şekilde iş bölümü yapma işlemi nesne yönelimli programcılığın başlıca presiplerinden birisidir.

Yazmış olduğumuz bu uygulama tek bir sınıf nesnesi olarak tasarlanmış ve 106. satırdaki main metodu içerisinde de bir değişken aracılığıyla çalıştırılmıştır. Ayrıca dikkat ederseniz uygulamamızın bazı metodları "private" olarak tasarlanmıştır. Bunun nedeni; bu metodların iç mekanizma işleyişi için kullanılıyor olmasıdır. Yani 109. satırda olduğu gibi kullanıcı, bu sınıfın "uygulamaBaslat" metodunu çağırabilmeli ancak diğer metodlarını çağıramamalıdır. Bu metodların hangi sırada ve ne zaman çağrılacağı program içerisinde sınıfın kendisi tarafından ve uygun bir hiyerarşi ile belirlenmektedir. Bu nedenle, bir değişken üzerinden erişilmemesi istenen metodlar private olarak tasarlanmıştır. Böylece bu metodlara, sadece sınıfın diğer metodlarının erişebilmesine izin verilmektedir. Kullanıcının hizmetine ise sadece "public" metodlar sunulmaktadır.

StringBuffer Sınıfı:
Daha önceki derslerimizde öğrendiğimiz üzere String sınıfı bazı yararlı metodlara sahiptir. Hatta bu metodlardan "equals" ve "matches" isimli olanlarını da, "KayitListesi.java" uygulamamızda 19.satırda ve "toUpperCase" isimli olanı da 37. satırda kullanmaktayız.

Bu metodlar sayesinde çeşitli kolaylıkları olan String türünde değişkenlerin içerikleri sabit uzunluktadır. Bu nedenle içeriği de dinamik olarak değiştirilebilen bir sınıf türüne ihtiyaç duyulmaktadır. Bu amaçla StringBuffer isimli sınıf kullanılabilir.

StringBuffer sınıfı kullanılmadan önce bu sınıf türünden bir nesne yaratılmalıdır:

StringBuffer strbuffer = new StringBuffer();

Bu şekilde, içerisinde hiç karakter tutmayan ve 16 karakter kapasiteli bir StringBuffer nesnesi yaratılabilir. Bu sınıfın başka başlangıç metodları da vardır. Mesela 20 karakter kapasiteli bir nesne yaratmak için:

StringBuffer strBuf = new StringBuffer(20);

Ya da hazır bir String değişkenini StringBuffer halinde yaratmak için:

StringBuffer str = new StringBuffer("Merhaba");

yazmak gerekir. Bu sınıf içerisinde tutulan String değerinin String olarak kullanılabilmesi için, sınıfa ait olan toString metodu kullanılabilir. Mesela az önce "Merhaba" String değeri ile yaratmış olduğumuz "str" isimli StringBuffer nesnesinin String değeri:

String a = str.toString();

şeklinde elde edilebilir.

Bu sınıfın sahip olduğu bazı metodlar şunlardır:

StringBuffer append(String)	StringBuffer nesnesi içerisinde tutulan String sonuna parametre ile verilen String değerini ekler ve bu yeni hali StringBuffer nesnesi olarak geri döndürür. Bu metodun int, float, double, char [] parametreli versiyonları da vardır.
int capacity()	Nesnenin sahip olduğu karakter kapasitesini verir.
void ensureCapacity(int minCapacity)	Nesnenin kapasitesini minCapacity parametresi ile verilen değer haline getirir.
char charAt(int index)	Nesnenin tuttuğu String değeri içerisinde verilen indekste yar alan karakteri geri döndürür.
StringBuffer reverse()	Nesnenin içerisinde tutulan String değerininin ters çevrilmiş halini geri döndürür.

Ancak yine de hatırlatmak isterim ki StringBuffer sınıfının daha birçok metodu vardır. Diğer metodları her zaman söylediğim gibi Java dökümantasyonu içerisinde bulabilirsiniz.

NumberFormat ve DecimalFormat Sınıfı: Çıkışın Biçimlendirilmesi

Aşağıdaki gibi bir numerik hesap yaptıktan sonra
x = 10000.0/3;

System.out.println(x) metodu yardımıyla bu sonucu ekranda görüntüleyebiliriz.

Ancak ekrana gelecek olan bu görüntü 3333.333333335 şeklinde olacaktır. Oysa biz çıkışın daha anlamlı olmasını isteyebiliriz. Bunun için DecimalFormat sınıfından yararlanıyoruz.

```
import java.text.DecimalFormat;

public class Main {

	public static void main(String[] args) {
		DecimalFormat formatter = new DecimalFormat("0.0000 YTL");
		double x = 100.0/3.0;
		System.out.println(x);
		System.out.println(formatter.format(x));
	}
}
```

Ayrıca sistemde tanımlı olan yerel sayı, para birimi (currency) gibi değerleri kullanmak için de NumberFormat sınıfından yararlanabiliyoruz.

```
import java.text.NumberFormat;
import java.util.*;

public class Main {

	public static void main(String[] args) {

		NumberFormat formatter = NumberFormat.getCurrencyInstance();
		formatter.setMaximumFractionDigits(4);
		formatter.setMaximumIntegerDigits(6);
		formatter.setMinimumIntegerDigits(6);



		double y = 1000.0/3.0;
		String s = formatter.format(y);

		System.out.print(s);
	}
}
```

GregorianCalendar Sınıfı
Java'da tarihlerle ilgili işlemler yapmak için kullanılabilecek 2 tane sınfı vardır.

Tarihte herhangi bir noktayi ifade etmek için "Date" sınıfı kullanılabilir.

```
import java.util.Date;

public class Main {

	public static void main(String[] args) {
```

```
            Date d1 = new Date();
            d1.setYear(1999-1900);
            d1.setDate(12);
            d1.setMonth(0);

            Date d2 = new Date();
            d2.setTime(System.currentTimeMillis()); //1,1,1970 den bugune kadar gecen
milisaniye

            if(d1.before(d2))
                  System.out.println(d1 + " tarihi,  " + d2 + " tarihinden daha once
geliyor");


            System.out.print(d1 + "\n" + d2);

      }
}
```

GregorianCalendar sınıfı, Date sınıfından daha fazla metoda sahiptir.
GregorianCalendar ile kullanabileceğiniz başlangıç metodları:

```
new GregorianCalendar()
new GregorianCalendar(1999, 11, 31);
new GregorianCalendar(1999, Calendar.DECEMBER, 31, 23, 59, 59)
```

GregorianCalnedar kullanarak zamanı elde etme ve istediğimiz gibi değiştirmeye ilişkin temel metodlar şunlardır:

```
GregorianCalendar simdi = new GregorianCalendar();
int month = simdi.get(Calendar.MONTH);

simdi.setTime(Calendar.MONTH, 3);

simdi.add(Calendar.MONTH, 3);
```

```
import java.util.Calendar;
import java.util.GregorianCalendar;

public class Main {

      public static void main(String[] args) {


            GregorianCalendar simdi = new GregorianCalendar();

            int ay = simdi.get(Calendar.MONTH);
            int gun = simdi.get(Calendar.DAY_OF_WEEK);
            int yil = simdi.get(Calendar.YEAR);
            int tarih = simdi.get(Calendar.DATE);
            int saat = simdi.get(Calendar.HOUR);
            int dakika = simdi.get(Calendar.MINUTE);
            int saniye = simdi.get(Calendar.SECOND);

            System.out.println("Tarih: " + tarih + " " +  hangiAy(ay)  + " " + hangiGun(gun)
+ " " + yil);
            System.out.println("Saat: " + saat + ":" +  dakika  + ":" + saniye);
      }

      public static String hangiAy(int n) {
```

```
		switch (n) {
			case 0:
				return "Ocak";
			case 1:
				return "Subat";
			case 2:
				return "Mart";
			case 3:
				return "Nisan";
			case 4:
				return "Mayis";
			case 5:
				return "Haziran";
			case 6:
				return "Temmuz";
			case 7:
				return "Agustos";
			case 8:
				return "Eylul";
			case 9:
				return "Ekim";
			case 10:
				return "Kasım";
			case 11:
				return "Aralık";
		}

		return "Jupiter :)";

	}

	public static String hangiGun(int n) {

		switch (n) {
			case 1:
				return "Pazar";
			case 2:
				return "Pazartesi";
			case 3:
				return "Sali";
			case 4:
				return "Carsamba";
			case 5:
				return "Persembe";
			case 6:
				return "Cuma";
			case 7:
				return "Cumartesi";
		}

		return "Pazoncesi :)";

	}
}
```

ODEV:
Dinamik buyuyen int array dizisini double, String ve int turleri icin genellestirip gerekli tum metodlari da barindiran bir DinamicArray nesnesi yaratiniz.

Önümüzdeki derste Sınıfların Türetilmesi konusunu detaylı bir şekilde inceleyeceğiz.

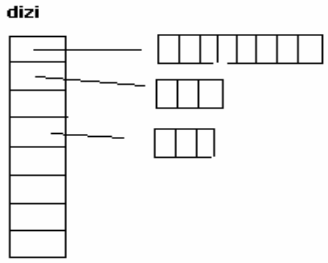
Java ile Programlama
Bölüm 9 - 10

Stringler
Daha önce de sıkça kullanıdığımız String türünde değişkenler asılnda ilkel değişkenlerden biraz daha farklıdır. Burada ilkel kelimesinden kasıt her dilde tanımlı olan ve java'da da yer alan en temel değişkenlerdir. Java'da String'ler bu temel olma durumunun biraz daha üstünde nesnelerdir. Zaten String nesnelerinin kendilerine ait bazı metodların olması da bu durumu açıkça göstermektedir. Teknik olarak Stringler karakter dizileridir.

Bir String nesnesi ile kullanılabilecek özel metodları şöyle listelyebiliriz:

a) **int length()**: Nesnenin içerisindeki karakter uzunluğunu verir.
b) char charAt(int index): Nesne içerisinde istenen indexteki karakteri verir.
c) String substring(int start, int end): start ve end indeksleri arasındaki küçük string parçasını verir.
d) String toUpperCase(): String nesnesinin tüm karakterlerini büyük harfe çevirir.
e) String toLowerCase(): String nesnesinin tüm karakterlerini küçük harfe çevirir.
f) String trim(): String nesnesi içersindeki boşlukları ortadan kaldırır ve yeni bvir string üretir.
g) int indexOf(String str): String nesnesi içerisinde "str" string parçasını arar ve bulduğu ilk indexi gönderir.
h) boolean equals(String str): String nesnesini parametre olarak verilen str ile karşılaştırır. Eşitlik durumunda true, diğer durumda false değerinin üretir.
i) boolean equalsIgnoreCase(String str): equals ile aynı işi yapar ama büyük küçük harf durumuna dikkat etmez.
j) String valueOf(int a): Verilen ilkel tipi String'e çevirir. Parametresi diğer türleri de alabilir.

Çok Boyutlu Diziler
Çok boyutlu diziler aslında her elemanı başka bir dizi olan dizilere çok boyutlu diziler denilir.



ODEV

1) Bir çift boyutlu dizinin tüm elemanlarinin toplamini bulan bir metod yaziniz:
   a. İnt findTotal(int arr[][]);

2) Verilen bir çift boyutlu diziyi dortgn haline getiren bir metod yaziniz. Verilen dizinin her elemani ayni uzunlukta olmayabilir.
   a. İnt [][] makeSquare(int arr[][]);

Odev:
Magic Square bulma

Bir cift boyutlu dizi eger her satirinin toplami, her sutunun toplami ve her iki ana diagonalin toplami esitse ve dizi kare ise (satir ve sutun sayisi esit) bu dizi magic'tir.

Bu islemi yapan ve prototipi

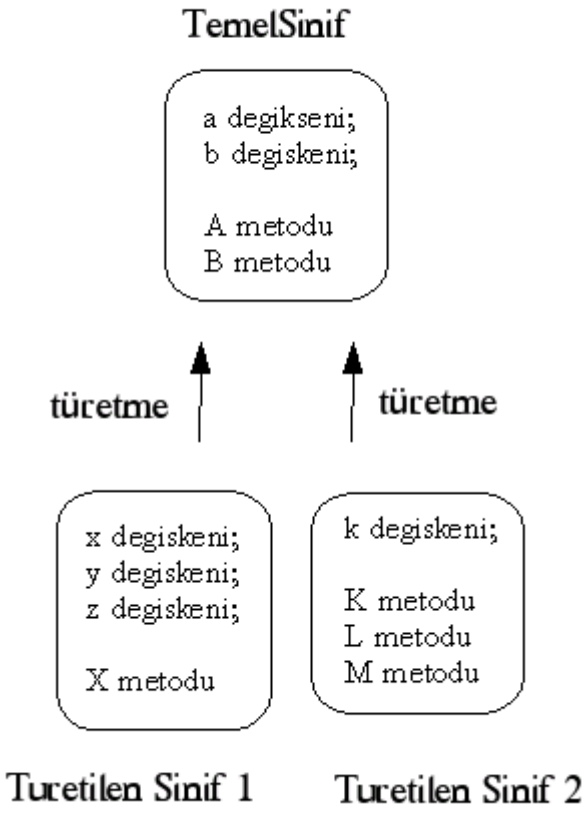
public static boolean isMagic(int array[][])

bir metod yaziniz.

İpucu: Programi parcalara bolunuz. Mesela sadece tek boyutlu bir dizinin toplamini bulan ya da bir cift boyutlu dizinin sadece i. Sutunun toplamini bulan ya da bir cift boyutlu diznizn sadece 1. ya da sadece 2. ana diagonalinin toplamini bulan metdolari ayri ayri tasarlayiniz ve daha sonra bunlari ana programda kullaniniz.

Sınıfların Türetilmesi
Sınıfların türetilmesi kavramını şöyle bir örnekle açıklamaya çalışalım: Diyelim ki çok büyük bir programlama projesinin bir kısmında görev aldınız. Öyle ki sizin yapmanız gereken kısım bir ekran üzerine yerleştirilecek olan seçenek düğmelerinin tasarımı olsun. Ancak bu işlemi yapabilmeniz için elinizde herşeyden önce bir ekran görüntüsünün olması lazım. O halde öncelikle bir ekran görüntüsü oluşturan program parçası, ya da artık yeni öğrendiğimiz kavramlara dayanarak, temel ekran işlemlerini barındıran sınıf olması gerekmektedir. Ancak sizin asıl işiniz olmadığından, ekran işlemlerine ilişkin bir sınıf yazmanız ve daha sonra da kendi düğmelerinizi de bu sınıf içerisine eklemeniz gereksiz bir iş olacaktır. Ayrıca bu ekran işlemleri gibi temel niteliğe sahip ve sizin dışınızda diğer kişilerin de başka işlemler için ihtiyaç duyabileceği bir sınıf kendi başına ayrı bir şekilde tasarlanmalıdır. Eğer bu sınıfın temel özelliklerini koruyarak, üzerine herkesin ihtiyaç duyacağı yeni özelliklerle birlikte yeni bir sınıf haline getirileceği bir mekanizma olursa, önceden harcanan emek de yeni işlemler için temel teşkil edecek nitelikte olup, gereksiz iş gücü kaybı engellenmiş olacaktır.

İşte Java'da bu mekanizmaya sınıfların türetilmesi denilir. Şekilde türetme iilişkisine ilişkin bir örnek yer almaktadır.



Buna göre şekildeki "TemelSinif"tan türetilen "Sinif 1" ve "Sinif 2" isimli sınıfların işlevleri "Temel Sınıf"ın işlevlerini de içermektedir. Buna göre "Sinif 1"; kendi içerisinde yer alan "x", "y" ve "z" değişkenleri ile "X" metodunun yanı sıra, "TemeSinif" dan türediği için "a","b" değişkenleri ile "A", "B" metodlarına da sahip olmaktadır. Benzer şekilde "Sinif 2", kendisinde yer alan "k" değişkeni ile "K", "L" ve metodları ve türetme ilişkisi yüzünden "a", "b" değişkenleri ile "A" ve "B" metodlarına da sahip olmaktadır.

Şekilde bu türetme ilişkisinin "Sinif 1" ve "Sinif 2" den "TemelSinif"a doğru giden oklarla gösterilmesi bir yanlışlık değil, kasten yapılmaktadır. Nesne yönelimli programlamada türetme ilişkisi genellikle bu şekilde, yani, türetilmiş sınıftan türetilen sınıfa doğru gösterilmektedir.

Örnek Uygulama 1
O halde kavramsal olarak incelediğimiz türetme işlemini bir de uygulamalı olarak görelim. Kisi.java, Ogrenci.java ve KisiDeneme.java dosyalarından oluşan klasik örnekte bir kişi kaydını temsil eden bir sınıftan, bir öğrenci kaydını temsil eden farklı bir sınıf türetilmektedir.

```
class Kisi {

        String isim;
    String adres;
    char cinsiyet;

  public Kisi(String gelenIsim, char gelenCinsiyet, String gelenAdres) {
                this.isimSetEt(gelenIsim);
                      this.adresSetEt(gelenAdres);
                      this.cinsiyetSetEt(gelenCinsiyet);
    }

    public void isimSetEt(String yeniIsim) {
            this.isim = yeniIsim.toUpperCase();
    }
```

```
    public void adresSetEt(String yeniAdres) {
            this.adres = yeniAdres.toUpperCase();
    }

    public void cinsiyetSetEt(char gCinsiyet) {
            if(gCinsiyet != 'e' && gCinsiyet != 'k') {
                        System.out.println("Cinsiyet bilgisi \"e\" ya da \"k\"
olmali.");
                                    System.out.println("Yanlis bir deger girdiniz...");
                                    System.out.println("Program sonlaniyor");
                                    System.exit(0);
                }

                else
                        this.cinsiyet = gCinsiyet;
    }

    public String ismiGetir() {
            return this.isim;
    }

    public String adresiGetir() {
            return this.adres;
    }

    public char cinsiyetiGetir() {
            return this.cinsiyet;
    }

    public String toString() {
            String str="";

        str += "Kisi Bilgileri\n";
        str += "Isim: "+this.ismiGetir()+"\n";
        str += "Cinsiyet :"+this.cinsiyetiGetir()+"\n";
        str += "Adres: "+this.adresiGetir()+"\n";

        return str;
    }
}
```

Örnekte, türetme işlemine maruz kalan sınıfın "Ogrenci" isimli sınıf olduğunu görmekteyiz. Bu nedenle "Ogrenci" isimli sınıf yaratılırken sınıf bildiriminin devamına "extends" anahtar sözcüğü ve daha sonra da bu sınıfın türetildiği temel sınıf olan "Kisi" sınıfının adı yazılmaktadır.

```
class Ogrenci extends Kisi {

        String bolum;
   int donem;

   public Ogrenci(String ad, char cns, String adr, String blm, int dnm) {
            super(ad, cns, adr);
      this.bolumSetEt(blm);
      this.donemiSetEt(dnm);
   }

   public String bolumuGetir() {
            return this.bolum;
   }
```

```
    public void bolumSetEt(String yBolum) {
                this.bolum = yBolum.toUpperCase();
    }

    public int donemiGetir() {
                return this.donem;
    }

    public void donemiSetEt(int yDonem) {
                this.donem = yDonem;
    }

    public String toString() {
                String str = super.toString();
        str += "Bolum: "+this.bolumuGetir()+"\n";
        str += "Donem: "+this.donemiGetir()+"\n";
        return str;
    }
}
```

Böylece Ogrenci sınıfı artık aynı zamanda bir "Kisi" sınıfı haline gelmiş ve kendi değişken ve metodlarının yanı sıra, "Kisi" sınıfının değişken ve metodlarına da sahip olmuştur. Bu nedenle "Ogrenci" sınıfını yazarken bir daha isim, cinsiyet ve adres değişkenleri ve bu değişkenler ile ilgilenen gerekli metodları yazmaya gerek yoktur.

```
class KisiDeneme {

        public static void main(String arg[]) {

        Ogrenci o1 = new Ogrenci("Dogac Berkun", 'e', "Kurtulus Cd. No 24",
                                                "Bilgisayar Bilimleri", 2);

      System.out.println(o1);
                o1.donemiSetEt(3);
      System.out.println(o1);
    }

}
```

KisiDeneme.java dosyasındaki "main" metodu içerisinde en son oluşturduğumuz "Ogrenci" sınıfını kullanarak bir nesne yaratmaktayız. Daha sonra da önceden öğrendiğimiz "toString" metodu işlevinden yararlanarak (bkz. Bölüm 6) oluşturduğumuz bu "Ogrenci" türünden nesneyi ekrana basmaktayız.

Sınıflarda super Metodu:
En son yazdığımız "Ogrenci" sınıfının başlangıç metodu içerisinde "super" isimli ilginç bir metod kullanılmaktadır. Oysa "Ogrenci" isimli sınıfın böyle bir metodu yoktur. O halde hiç yazılmamış olan bu metod nedir?

Başka bir sınıftan türetilmiş olan sınıflarda, "super" ile temsil edilen metod, türetmenin yapıldığı sınıfın başlangıç metodunu temsil etmektedir. O halde bizim örneğimizde "Ogrenci.java" dosyasının 7. satırındaki

```
super(ad, cns, adr);
```

kullanımı ile "Ogrenci" sınıfının türediği "Kisi" sınıfının uygun parametreli başlangıç metodunun çağrıldığını söyleyebiliriz. Böylece "KisiDeneme.java" dosyasında kullanıcı "Ogrenci" nesnesini yaratırken, aslında arka planda, bu "Ogrenci" nesnesinin içerisinde yer alan "Kisi" nesnesine ait kısımlar, "Kisi" nesnesinin başlangıç metodu ile oluşturulmakta ve daha sonra da "Ogrenci" nesnesinin kendisine ait olan kısımlar oluşturulmaktadır. Fakat bu işlemlerin hepsi neticede "Ogrenci" nesnesinin başlangıç metodu içerisinde yapılmaktadır. "Ogrenci" nesnesi "Kisi" nesnesinden türediği için, bünyesinde "Kisi" nesnesine ait kısımları da barındırmakta ve dolayısı ile "super" metodu yardımıyla gerektiğinde "Kisi" nesnesinin başlangıç metodlarına ya da "Ogrenci.java" dosyasının 29. satırında olduğu gibi "Kisi" nesnesini başka metod ve değişkenlerine erişme hakkına sahip olmaktadır.

Örnek Uygulama 2
Şimdi örneğimizi biraz daha genelleştirelim ve bu sefer içerisinde "Öğrenci2" sınıfı türünden nesneleri tutan ve bu nesneler ile ilgili çeşitli işlemler yapan bir "Kayitlar" isimli bir sınıf tasarlayalım.

```
class Kisi2 {

      String isim;
    char cinsiyet;

   public Kisi2(String gelenIsim, char gelenCinsiyet) {
            this.isimSetEt(gelenIsim);
        this.cinsiyetSetEt(gelenCinsiyet);
    }

    public void isimSetEt(String yeniIsim) {
            this.isim = yeniIsim.toUpperCase();
    }

    public void cinsiyetSetEt(char gCinsiyet) {
            if(gCinsiyet != 'e' && gCinsiyet != 'k') {
            System.out.println("Cinsiyet bilgisi \"e\" ya da \"k\" olmali.");
             System.out.println("Yanlis bir deger girdiniz...");
             System.out.println("Program sonlaniyor");
             System.exit(0);
        }

        else
            this.cinsiyet = gCinsiyet;
    }

    public String ismiGetir() {
            return this.isim;
    }

    public char cinsiyetiGetir() {
            return this.cinsiyet;
  }

  public String toString() {
            String str="";

    str += "Kisi Bilgileri\n";
    str += "Isim: "+this.ismiGetir()+"\n";
    str += "Cinsiyet :"+this.cinsiyetiGetir()+"\n";

    return str;
  }

}


class Ogrenci2 extends Kisi2 {

      String bolum;
    int donem;
    int ogrenci_no;
    String ortalama;
```

```
    public Ogrenci2(String ad, char cns, String blm, int dnm, int ogrNo, double ort)
      {
            super(ad, cns);
        this.bolumSetEt(blm);
        this.donemiSetEt(dnm);
        this.numaraSetEt(ogrNo);
        this.ortalamaSetEt(ort);
    }

    public String bolumuGetir() {
            return this.bolum;
    }

    public void bolumSetEt(String yBolum) {
            this.bolum = yBolum.toUpperCase();
    }

    public int donemiGetir() {
            return this.donem;
    }

    public void donemiSetEt(int yDonem) {
            this.donem = yDonem;
    }

    public int numarayiGetir() {
            return this.ogrenci_no;
    }

    public void numaraSetEt(int ogrNo) {
            this.ogrenci_no = ogrNo;
    }

    public String ortalamayiGetir() {
            return this.ortalama;
    }

    public void ortalamaSetEt(double yeniOrt) {
      if(yeniOrt < 1.0)
      this.ortalama = "F";
    else if(yeniOrt >= 1.0 && yeniOrt < 1.5)
      this.ortalama = "D";
    else if(yeniOrt >= 1.5 && yeniOrt<2.0)
      this.ortalama = "C";
    else if(yeniOrt >= 2.0 && yeniOrt<3.6)
      this.ortalama = "B";
    else if(yeniOrt >= 3.6 && yeniOrt <= 4.0)
      this.ortalama = "A";
  }

  public String toString() {
            String str = super.toString();
    str += "Bolum: "+this.bolumuGetir()+"\n";
    str += "Donem: "+this.donemiGetir()+"\n";
    str += "Ogrenci No: "+this.numarayiGetir()+"\n";
    str += "Ortalamasi: "+this.ortalamayiGetir()+"\n";
    return str;
  }
}
```

```
class Kayitlar {

        Ogrenci2 dizi[];

        public Kayitlar() {

        }

    public Kayitlar(Ogrenci2 yeniEleman) {
                this.dizi = new Ogrenci2[1];
       this.dizi[0] = yeniEleman;
        }

    public void ekle(Ogrenci2 eklenecekOgr) {

                if(this.dizi == null) {
        dizi = new Ogrenci2[1];
         this.dizi[0] = eklenecekOgr;
       }

                else {
                        Ogrenci2 tmp[] = new Ogrenci2[dizi.length+1];

                        for(int i=0; i < dizi.length; i++)
                tmp[i] = dizi[i];

                        tmp[tmp.length-1] = eklenecekOgr;
        this.dizi = tmp;
       }
    }

    public void cikart(int id) {

        Ogrenci2 tmp[] = new Ogrenci2[dizi.length-1];
       int indeks=-1;

                for(int i=0; i < dizi.length; i++)
        if(dizi[i].numarayiGetir() == id)
                                indeks = i;

       for(int i=0; i < indeks; i++)
        tmp[i] = dizi[i];

       for(int i = indeks; i < dizi.length-1; i++)
        tmp[i] = dizi[i+1];

       dizi = tmp;

    }

    public void listele() {

        System.out.println("Kayitlar listeleniyor:");
       System.out.println("--------------------");

                for(int i=0; i < dizi.length; i++)
```

```
                        System.out.println(""+dizi[i]);
    }

    public Ogrenci2 enBuyukOrtalamaliOgrenci() {

                Ogrenci2 max=dizi[0];

        for(int i=0; i < dizi.length; i++) {
                        if(dizi[i].ortalamayiGetir().compareTo(max.ortalamayiGetir()) == -1)
                max = dizi[i];
        }

        return max;
    }
}

class KayitlarDeneme {

        public static void main(String arg[]) {

                Kayitlar x = new Kayitlar();

        x.ekle(new Ogrenci2("Evren Banger", 'e',"Bilgisayar", 2, 100, 2.54));
        x.ekle(new Ogrenci2("Aykut Toygar", 'e',"Bilgisayar", 2, 101, 2.4));
        x.ekle(new Ogrenci2("Hande Kumek", 'k', "Bilgisayar", 3, 102, 3.5));
        x.ekle(new Ogrenci2("Ilker Kara", 'e', "Bilgisayar", 2, 103, 1.55));
        x.ekle(new Ogrenci2("Cemile Caglar", 'k',"Isletme", 4, 201, 3.7));
        x.ekle(new Ogrenci2("Erdem Aykac", 'e', "Isletme", 4, 202, 3.58));

        x.listele();

        Ogrenci2 sonuc = x.enBuyukOrtalamaliOgrenci();
        System.out.println("En buyuk ortalamali ogrenci: \n"+sonuc);
    }
}
```

Bu sınıf daha önceki derslerimizde yazmış olduğumuz "KayitListesi" sınıfını andırmakla beraber, bu sefer sınıfın veri elemanı bu kayıtları tutacak "Ogrenci2" türünden bir dizi olacaktır. Bu arada , "Ogrenci2" sınıfı da bir önceki örnekte kullandığımız "Ogrenci" sınıfına çok benzer olup sadece bazı metodları değiştirilerek yeniden yazılmıştır. Özetle bu örnek "Kisi2.java", "Ogrenci2.java" ve "Kayitlar.java" dosyalarından oluşmakta ve en son olarak "KayitlarDeneme.java" dosyasında çalıştırılmaktadır.

Şimdi de gelelim bu uygulamanın detaylı bir incelemesine: "KayitlarDeneme.java" dosyasındaki main metodu içerisinde "Kayitlar" sınıfı türünden bir nesne yaratılmış ve bu nesneninin "ekle" isimli metodu yardımıyla birtakım "Ogrenci2" sınıfı türünden nesneler hemen o anda parametre olarak yaratılarak listeye eklenmiştir. Bu "Ogrenci2" türünden nesneleri önce değişken olarak yaratıp daha sonra "ekle" metdouna parametre olarak gönderebilirdik. Örneğin:

```
 ...
 Ogrenci2 o2 = new Ogrenci2("Evren Banger", 'e',"Bilgisayar", 2, 100, 2.54)
  x.ekle(o2);
  ...
```

Ancak bizim kullanıdığımız yöntem hem daha pratik hem de gereksiz değişkenlerden arınmış bir program yazmamızı sağlamaktadır.

"Kayitlar.java" sınıfının içerisindeki

```
        public Kayitlar(Ogrenci2 yeniEleman)
```

prototipli başlangıç metodu daha önce hiç kayıt eklenmemiş durum için tek elemanlı yeni bir "Ogrenci2" dizisi yaratır ve "yeniEleman" isimli "Ogrenci2" türünden parametreyi de bu diziye ekler. Yine "Kayitlar" sınıfının metodlarından birisi olan ekle isimli metod da durumu iki türlü ele alır. Eğer daha önce hiç kayıt eklenmemişse önce tek elemanlı bir "Ogrenci2" dizisi açılarak bu kayıt diziye eklenir (satır 16-18). Ama zaten daha önceden eklenmiş kayıtlar varsa o zaman ilk olarak eski diziden 1 eleman fazla uzunlukta "tmp" adında yeni bir geçici dizi yaratılır (satır 22), daha sonra eski dizinin tüm elemanları bu diziye de kopylanır (satır 24-25) ve en son olarak da bu tmp dizisinin fazla olan sondaki boş eleman yerinde de metoda parametre olarak gönderilen "eklenecekOgr" isimli "Ogrenci2" nesnesi yerleştirilir (satır 27). Tüm bu işlemlerin sonunda da artık kayıtların son halini tutan "tmp" dizisi, sınıfın kendi dizisi olan "dizi" isimli "Ogrenci2" dizisine atanır (satır 28).
Buna benzer olarak; sınıfın cıkart isimli metodu da çıkarılacak olan elemanının dizideki indeks numarasını saptadıktan sonra (satır 37-39) bu elemana kadar olan ve bu elemandan sonraki tüm elemanları "tmp" dizisine kopyalar (satır 41-42 ve 44-45) ve "tmp" dizisini yine sınıfın asıl dizisi haline getirir (satır 47).

"Kayitlar" sınıfının "public Ogrenci2 enBuyukOrtalamaliOgrenci()" isimli metodu ise "Ogrenci2" nesnelerinin ortalamarına bakarak en yüksek ortlamalı "Ogrenci2" nesnesini geri dönüş değeri olarak gönderir. Geriye dönen bu değer "KayitlarDeneme.java" dosyasında bir "Ogrenci2" değişkeninde tutularak (satır 16) ekrana basılır (satır 17). Burada tekrar hatırlatmak isterim ki; "sonuc" adındaki "Ogrenci2" sınıf değişkeni 17. satırda olduğu gibi ekrana basılmak istendiğinde, aslında sonuc değişkeni aracılığıyla ve "Ogrenci2" sınıfına ait olan "toString" isimli metod otomatik olarak çağrılmaktadır ve bu metodun hazırlayıp gönderdiği String türünden ifade ekrana basılmaktadır. Tabi ki bu ifadeyi hazırlayan da sınıfı yazan kişidir.

Umuyorum ki detaylı olarak hazılayıp anlatmaya çalıştığım bu örnek ile birlikte, sınıfların tasarlanması, başka sınıf ve metodlar içerisinde kullanılması ve türetilmesine ilişkin kavramları daha rahat kavramış olursunuz.

Uygulama Sorusu
Yukarıda verilen "Kayitlar" sınıfı örneğini biraz daha geliştirerek bu sınıfa öğrencileri not ortalamalarına göre, isim sırlarına göre listeleyen ve sadece belirli bölümlerdeki ya da belirli dönemlerdeki öğrencileri görüntüleyen metodlar ekleyiniz ve bu yeni sınıfın adını da "Kayitlar2" olarak değiştiriniz. İpucu olarak; yazmanız gereken bu metodların prototiplerini aşağıdaki şekilde verebiliriz:

public void notOrtalamasinaGoreListele()
public void isimSirasinaGoreListele()
public void bolumdekiOgrenciler(String bolumAdi)
public void donemdekiOgrenciler(int donem)

Şunu da belitmeliyim ki, "Kayitlar" sınıfına ekleyeceğiniz bu metodların yanı sıra, eğer gerekeceğini düşünüyorsanız mevcut metodların işleyişlerini ya da bu sınıfın içerisinde kullanılan "Ogrenci2" sınıfının da yine bazı metodlarının işleyişini değiştirebilirsiniz.

Polymorphism ve Final Metodlar genel açıklama

```
Ornek:
class TemelSinif {

	public void method() {
		System.out.println("Temel sinifin metodu");
	}

	public final void method2() {
		System.out.println("Temel sinifin final metodu");
	}
}
```

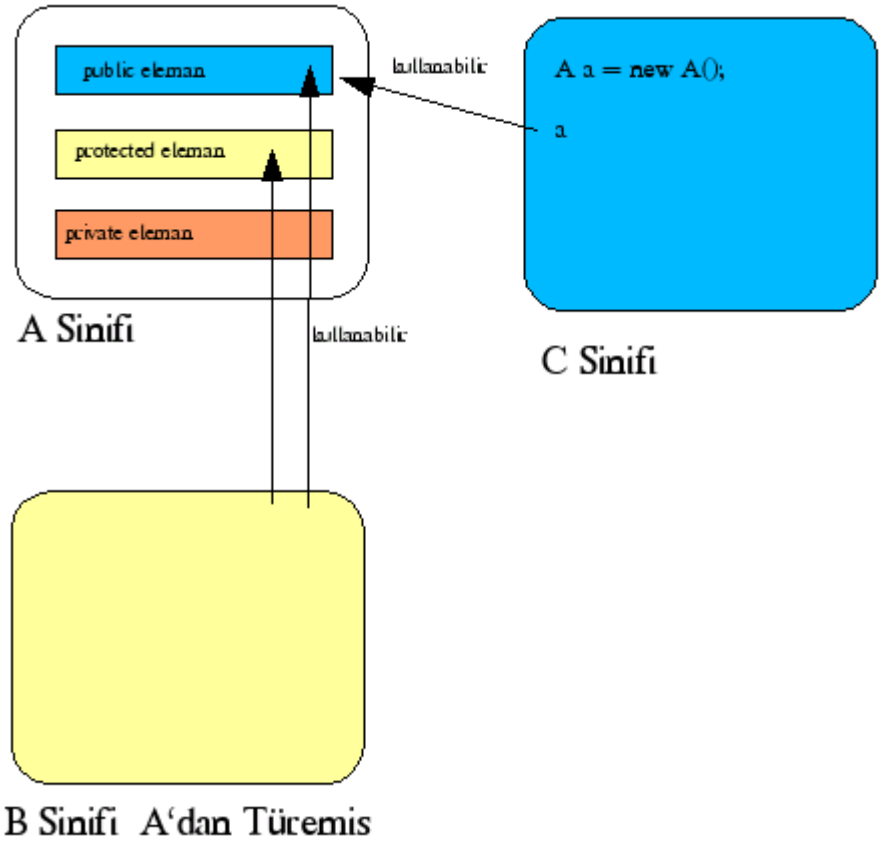
```
class TuremisSinif1 extends TemelSinif {


	/*Temel sinifin metodunu override etmis*/
	public void method() {
		System.out.println("TuremisSsinif_1 sinifin metodu");
	}

	/*Temel sinifin metodunu override etmis*/
	public void method2() {
		System.out.println("TuremisSsinif_1 sinifin final metodu");
	}
}

class TuremisSinif2 extends TemelSinif {


}

class Main {

	public static void main(String arg[]) {

		TuremisSinif1 t1 = new TuremisSinif1();
		TuremisSinif2 t2 = new TuremisSinif2();
		TemelSinif tt = new TemelSinif();

		t1.method();
		t2.method();

		tt = t1;

		tt.method();

		tt = (TemelSinif)t1;
		tt.method();


	}
}
```

Instanceof Ornek

```
public static void main(String[] args) {
		Object ar[] = {new Student("A", 12, 1232, 2), new Sayi(13)};

		for(int i = 0; i < ar.length; i++) {
			if(ar[i] instanceof Student)
				((Student)ar[i]).m_student();
			else if(ar[i] instanceof Sayi)
				((Sayi)ar[i]).m_sayi();

		}

	}
```

Sınıfların protected veri elemanları:
Daha önceden biliyoruz ki sınıflarda private olan bir değişken ya da metod sadece kendi sınıfının diğer metodları tarafından erişilebilir haldedir. Bununla beraber public sınıf elemanları, hem bu sınıfın diğer metodları, hem de bu sınıf türünden yaratılmış olan nesneler tarafından, yani dışarıdan erişilebilir haldedir.

Eğer bir sınıfın veri elemanı sadece bu sınıfın kendi metodları ya da bu sınıftan türetilmiş sınıfların metodları ile erişilebilir ancak dışarıdan erişilemez olsun isteniyorsa, böyle veri elemanlarının protected veri elemanı olarak bildirilmesi gerekir. Bu durumda protected veri elemanlarının erişim bakımından public ve private veri elemanları arasında bir seviyede olduğunu söyleyebiliriz.



Protected Metodlar
Bir sınfıfın bazı metodları protected olarak tasarlanabilir. Böyle metodlar da sadece sınıfı türeten alt sınıflar içerisinde kullanılabilir ancak dışarıdan kullanılamazlar. Böylece sadece türetme yapan sınıflara güveniliyorsa ve dışarıya kapatılmak isteniyora bir metod protected olarak tasarlanabillir. Bunun bir örneğini ileriki konularda göreceğimiz Object sınıfının protected metdu olan "clone" metodunu incelreken göreceğiz.

Object Sınıfı
Şu ana kadar sınıfların türetilmesine ilişkin birçok şey öğrendik. O halde artık Java da temel bir sınıf olan Object sınıfını da inceleyebiliriz. Java'da her şey bir nesnedir. Aslında bu tanım hem kavramsal hem de teknik anlamda doğrudur.

Kavramsal olarak Java'da herşeyin bir nesne olduğunu zaten bu noktaya kadar tartıştık. Teknik olarak da Java'da herşeyin bir nesne olması, Java'daki tüm tanımlı sınıfların aslında "Object" adı verilen özel bir sınıftan türemesi anlamına gelmektedir.

Öyle ki sizin yazdığınız bir sınıf da aslında öntanımlı olarak Object isimli sınıftan türemiştir.

Object Sınıfının Metodlarının Yeniden Yazılması
Ayrıca daha önce öğrenmiş olduğumuz "function overriding" kendi yarattığımız sınıflarda Object sınıfının bazı metodlarını kendi isteğimize göre sınıfımız içerisinde değiştirerek yazabiliriz. Aslında toString isimli metodun böyle bir metod olduğunu bu noktada belirtmek gerekir.

Object sınıfının toString Metodu
toString metodu Object sınıfında yazılmış olan bir metoddur. Ancak daha önce de söylediğimiz gibi Java'da yazılan her sınıf, hatta bizim kendi yazdığımız sınıflar da dahil, birer Object sınıfı kabul edildiğinden (yani otomatik olarak bu sınıftan türediğinden), Object sınıfına ait olan bu metodları kendi başımıza yazabiliriz.

Object sınıfının finalize Metodu
finalize() isimli metod da Object sınıfına ait bir metoddur. Bu metodu da kendi sınıfımıza göre yazabiliriz. Ancak daha önce bu metodun ne işer yaradığını bilmekte fayda var.

Önceden öğrendiğimiz gibi bir sınıf yaratıldığı zaman otomatik olarak uygun olan başlangıç metodu (constructor) çağrılmaktadır. Ayrıca sistemde kullanılmayan nesnlerin de garbage collector yapısı ile otomatik olarak ortadan aldırıldığını da söylemiştik.

İşte sınıflarda benzer şekilde bir sınıf ortadan kaldırılırken, yani işlevi sonlandırılırken de otomatik olarak çalğrılan bir metod vardır. Bu metod finalize() isimli bir metoddur. O halde nasıl başlangıç metodu içerisine sınıfımızın birtakım ilk işlemlerini yazıyorsak, finalize metodu içerisine de bir sınıf sonlanırken yapılması gereken son işlemleri tanımlayabiliriz. Bu son işlemlere örnek olarak:

a) **Daha önce açılmış bir dosyanın kapatılması**
b) **Daha önce yapılmış bir veri tabanının sonlandırılması**
c) **Daha önce açılmış bir session'ın sonlandırılması**
d) **Daha önceden kullanılan birtakım geçici değişkenlerin sonlandırılması**

verilebilir.

Ancak şunu da belirtmek gerekir ki Java'da finzlize metodu Garbage Collector devreye girdiği zaman çağrıldığı için bu metodu tam olarak ne zaman çağrılacağı bilinemez. Bu nedenle bazen öngörülemeyen sonuçlar ortaya çıkabilir.

O halde sonuç olarak kullanıcı aslında kendi istediği zaman çağırabileceği birtakım son işlemleri yapacak bir metodu yine kendisi başka bir isimle yazabilir. Böylece son işlemler doğru sırada ve kullanıcının isteğine bağlı olarak yapılacaktır. Bu da finalize metoduna olan gerekliliği ortadan kaldırmaktadır.

Bunun dışında Object sınıfında başka metodlar da vardır. Ancak bu metodlar ilerde öğreneceğimiz bazı kavramları gerektirdiğinden şimdilik Object sınıfına bu kadar değineceğiz.

Object Sınıfına Her Nesne Atanabilir
Java'da herşeyin bir Object olduğunu tekrar hatırlayalım. Aslında Object isimli sınıf türünden bir değişkenin bu nedenle her türden nesneyi tutabileceğini de söyleyebiliriz.

```
public class ObjectOrnek {

	public static void main(String[] args) {

		Object dizi[] = new Object[3];

		dizi[0] = "string nesnesi";
		dizi[1] = new Integer(13);
		dizi[2] = new Deneme();

	}
```

```
}

class Deneme {


}
```

Yukarıdaki örnekte Object türünden bir dizi yarattık. Bir dizi içerisinde tüm elemanların aynı türden olması gerektiği halde birbirinden farklı türleri nasıl oluyor da bir dizide saklayabiliyoruz? İşte bunun nedeni az önce söyledğim gibi aslında her türün aynı zaman bir Object olmasından kaynaklanmaktadır.

```
class ObjectDeneme {

	public static void main(String arg[]) {


		Object ar[] = {new Deneme(), "String ekledik", new Integer(12), new BaskaBirSinif()};

		BaskaBirSinif temp = (BaskaBirSinif)ar[3];
		System.out.println(temp.methodX());

		/*Olmaz!*/
		//ar[3].methodX();


	}
}


class Deneme {

	public static void methodY() {
		System.out.println("Deneme sinifi");
	}


}


class BaskaBirSinif {

	public static String methodX() {
		return "12";
	}

}
```

Soyut sınıflar

Soyut sınıf kavramı aslında tamamen türetme ile ilişkilidir. Bu nedenle soyut sınıfların anlamlı kullanılması aslında bir türetme işleminin yapılmasını gerektirir. Türetme işlemini hatırlayacağımız gibi bir sınıf başka bir sınıfının değişken ve metodlarını alabilmektedir. Böylece önceden yazılmış birtakım metod ve değişkenlerin yeraldığı bir sınıf varsa ve biz de bu metod ve değişkenlere ihtiyaç duyuyor ve aynı zamanda yeni metod ya da değişkenler de eklemek istiyorsak, mevcut sınıftan bir türetme yaparak sadece yeni eklemek istediğimiz metod ve değişkenleri yazmamız yeterli olacaktır.

Bir sınıf içerisinde daha önce görmüş olduğumuz metod tanımlamalarından farklı olarak "soyut metod" denilen bir metod şekli daha vardır. Bir metod soyut olarak yazılacaksa metod prototipinde public, private ya da protected

anahtar sözcüğünden sonra "abstract" anahtar sözcüğü yazılmadır. Ayrıca sınıf içerisinde bu şekilde prototipi yazılan soyut metodun içerisine de hiçbirşey yazılmaz. Örneğin:

public abstract String ismiVer();

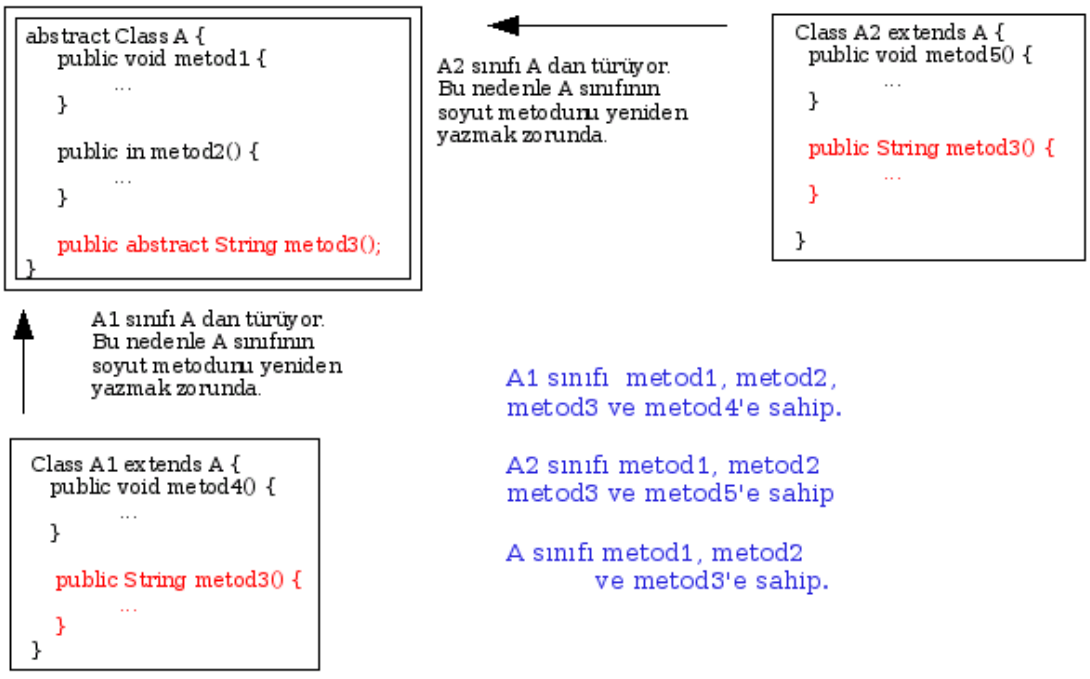
Yukarıdaki örnekte bir soyut metod bildirimi yapılmış ve içerisi yazılmadan noktalı virgül ile bildirim tamalanmıştır. İkinci kural olarak da içerisinde soyut metod barındıran bir sınıftan türetilen yeni bir sınıf, bu soyut metodun içerisini kendi içinde yazmak zorundadır. Aksi takdirde türetme işlemi hatalı sayılır. Sonuç olarak:

Soyut bir metodun temel sınıf içerisinde sadece prototipi yazılır.
Bu temel sınıftan türeyen bir sınıf sadece prototipi yazımış olan bu soyut metodu kendi içerisinde tamamen yazmak zorundadır.
İçerisinde soyut metod barındıran sınıflara soyut sınıf denilir ve böyle bir sınıfın bildiriminin başına Class anahtar sözcüğünden önce "abstract" yazılmak zorundadır.

Peki ama soyut metodların bu kullanımı ne işe yarar?

Sınıf bildirimi içerisinde yazılmış olan soyut metodların nasıl çalışacağı, ancak bu sınıftan türetilen sınıflar içerisinde yazılmaktadır. Temel sınıf içerisinde bu metodların nasıl işleyeceklerinden ziyade geri dönüş değerlerinin ve parametrelerinin ne oldukları yazılır. O halde içerisinde soyut metodlar barındıran bir sınıftan türetme yapıldığında, ondan türeyen her sınıf içerisinde bu metodun yeniden yazılması gerekecek ve dolayısıyla bu sınıftan türeyen tüm sınıfların böyle bir metod barındıracağı garanti edilmiş olacaktır.

Böylece birçok kişinin görev aldığı büyük bir projede yazılması kesinlikle gerekli olan ancak sadece işleyişi değişebilen metodları temel bir sınıf içerisinde soyut olarak tanımlarsak ve yazılacak tüm yeni sınıfların da bu temel sınıftan türetilerek yazılmasını şart koşarsak, herhangi bir kontrole gerek kalmaksızın yazılması zorunlu olan bu metodların birbirinden bağımsız kişilerin tasarlayacağı tüm sınıflarda kesinlikle yer alacağından emin olabiliriz. Bu durum aşağıdaki şekilde de özetlenmiştir:



Böylece kavramsal olarak soyut metod ve soyut sınıfın ne olduğunu yeterince incelediğimizi düşünüyorum. Şimdi yukarıda okuduklarımız pekiştiren bir örnek yapalım. Bu örnekte soyut metod kullanımını içeridiği gibi aslında türetme işleminin de güzel bir kullanımını göstermektedir.

Örneğimizde en temel olarak "Sekil" adında bir sınıf yeralmaktadır. Bu sınıf tüm geometrik şekillerin temelini teşkil ettiğinden içerisinde yer alacak olan alanHesapla, cevreHesapla, kenarSayisiVer gibi metodların yazılması anlamsızdır. Ancak bir geometrik şekil bu metodlara sahip olmalıdır. O halde biz "Sekil" sınıfınının bu metodlarını soyut olarak tasarlayıp gerekli olan geometrik şekile ilişkin bir sınıfı da bu sınıftan türetirsek, soyut olan bu metodları

ilgili şeklin özelliklerine göre yeniden yazabiliriz. Bu düşünceyle de örneğimizde "Sekil" sınıfından türeyen "Kare", "Ucgen" ve "Daire" sınıflarını da yazmaktayız:

Arayüzler

Arayüzler aslında teknik olarak bir sınfın ne yapması gerektiğini anlatmaktadır. Yani aynen soyut sınıflarda olduğu gibi, temel bir sınıfın operasyonlarını tanımlamak amacıyla tüm metodlarının soyut olması soncunda arayüzler meydana gelir.

Bu bakış açısıyla ir arayüzün aslında tam anlamıyla bir sınıf olmaktan ziyade "yapılması gerkenler listesi" olarak tanımlanması da mümkündür.

Nesneler arasındaki büyüklük küçüklük kriterinin önemi
Bunu şu temel örnekler açıklayabiliriz. Daha önceki derslerimizde de öğrendiğimiz gibi belirli nesneler üzerinde bir sıraya dizme işlemi yapılabilmesi için bu nesnlerim birbirleri arasında bir büyüklük-küçüklük ilişkisinin tanımlı olması gerekir. İlkel veri tipleri için bu ilişki zaten ">" ya da "<" operatörü ile kolaylıkla anlaşılabilmektedir. Ancak bizim yarattığımız nesneler üzerinde (mesela Student ya da Dortgen sınıfları gibi) doğal bir büyüklük küçüklük kriteri yoktur. Çünkü java açısından bu nesnelerin mantıksal olarak ne işe yaradıkları bilinemez. O halde biz büyklük kriterini de içercek compareTo ismli bazı metodlar yazarız. Bu metodlar yardımıyla da bu kriteri kullanıur ve böylece nesneleri sıralayabiliriz.

Comparable Arayüzü

İşte bu işlem için java programlama dili içerisinde "Comparable" adında bir arayüz tanımlanmıştır.

```
public interface Comparable  {
        int compareTo(Object other);
}
```

Bu arayüz bize kendisinde türetilen bir sınıfın içerisinde bu compareTo metodunun zorunlu olarak yazılması gerektiğini anlatır. Ayrıca bir nesne eğer bu arayüzden türetilirse, yine java içerisinde zaten öntanımlı olarak kullanılan "Arrays" isimli sınıfın sort metodu yardımıyla bu sınıf türünden nesneler de otomatik olarak sıralanabilecektir.

NOT: Arrays isimli sınıf genel olarak içerisinde birçok statik metod barındıran ve dizi işlemleri yapmayı kolaylaştıran bir sınıftır.

Comparable'dan türetme neden gerekli?

Peki ama biz zaten kendimiz istediğimiz bir sınıf içerisine compareTo metodu ekleyebileceksek o halde neden sınfımızı Comparable nesnesinden türetelim ki??

Bunun nedeni "Arrays.sort" metodu içerisinde bir yerlerde a[i].compareTo(a[j]) gibi bir kod yazmaktadır. O halde bu metoda gonderilen bir nesne dizisinin içerindeki her nesnenin comparTo metodu olduğu garanti edilmelidir. Bunun da tek yolu sizin gonderdiginiz ve sıralanmasını istediğiniz bu nesnelerin Comparable arayüzünden türetilmiş olmalarıdır.

Arayüzlerin Özellikleri

1) Arayüzler ile instance yaratılamaz
Teknik olarak aryüzleri içerisinde hiçbir metod yazılmadığından bir aryüz nesnesini doğrudan yaratmak hataya neden olur. Yani aşağıdaki gibi bir kod geçersizdir:

```
Comparable x = new Comparable();
```

2) Arayüzler içerisndeki değişkenler sabit olmalıdır
Arayüzlerde yaratılan metodların içerinde yerleştirilen değişkenler her zman public static final kabul edilirler.

3) Birden fazla arayüzden türetme

Nir ınıf normalde en fazla bir sınıf türetebilir. Ancak birde fazla sınıftan türetme yapmamın tek yolu kullanılan teme sınıflardan sadece birtanesinin sınıf, diğerlerinin ise arayüz olması gerekir. Örneğin Comparable gibi bir de Movable isimli bir arayüzün varlığını kabul edersek, Student isimli bir sınıf Person isimli bir sınıftan türetme ve Comparable ile Movable isimli arayzlerden de implement tyaparak aslında birden fazla sınıftan türetme yapabilir:

class Student extends Person implements Comparable, Moveable {

```
	public int compareTo(Object obj2) {

	}

	public void move(double x, double y) {

	}
}
```

Nesne Klonlama

Java da yaratılan bir sınıf nesnesinin bir başka sınıf değişkenine kopyalanildiğini ve hatta temel sınf türünden bir değişkene türemiş sınıfların da atnabildiğini ve buna ilişkin detayları önceki derslerimizde görmüştük.

Ancak bir sınıf nesnesi bir başka sınıf değişkenine kopyalandığında aslında iki değişkenin de hafızada aynı nesneye işaret ettiklerine ve bu nedenle de birinci değişken üzerinde değişiklik yapıldığında aslında ikiniciyi de değiştirdiğimizi belirtmek gerekir. Aşağıdaki örneği inceleyiniz:

= operatörü ile aslında nesneyi değil, adresini kopyalarız.

```
class Person {

	String name;

	public Person(String name) {
		this.name = name;
	}

	public String toString() {
		return this.name;
	}

	public void setName(String new_name) {
		this.name = new_name;
	}

	public static void main(String[] args) {
		Person p1 = new Person("Hakan Gozutok");
		Person p2 = p1;

		System.out.println("Birinci nesne icersindeki isim: " + p1);
		System.out.println("Birinci nesne icersindeki isim: " + p2);

		p2.setName("Caglar Aytek");

		//p1 içersindeki isim degistirilmedigi halde Caglar Aytek
		//olacaktir
		System.out.println("Birinci nesne icersindeki isim: " + p1);
		System.out.println("Birinci nesne icersindeki isim: " + p2);
	}
}
```

Bu şekilde meydana gelen değişimden kurtulmak ve gerçek anlamda nesnenin tam bir kopyasını almak için nesnenin klonlanması gerekir.

Yarattığımız her nesne aslında bir Object tir. Bunu daha önce öğrenmiştik. İşte Object sınıfı içerisinde yer alan "clone()" isimli bir metod bir nesnenin tam bir kopyasının başka bir nesne şeklinde klonlanmasını sağlar. Ancak bu işlem sanıldığı kadar basit değğildir. Bunu aşağıdaki örnek ile açıklayalım:

Bir sınıfın örneklerini, doğrudan "clone" metodu ile sadece kendi içerinde kopyalayabiliriz

```
class Person {

	String name;

	public Person(String name) {
		this.name = name;
	}

	public String toString() {
		return this.name;
	}

	public void setName(String new_name) {
		this.name = new_name;
	}

	public static void main(String[] args) {
		Person p1 = new Person("Hakan Gozutok");
		Person p2 = p1;

		System.out.println("Birinci nesne icersindeki isim: " + p1);
		System.out.println("İkinci nesne icersindeki isim: " + p2);

		p2.setName("Caglar Aytek");

		//p1 içersindeki isim degistirilmedigi halde Caglar Aytek
		//olacaktir
		System.out.println("Birinci nesne icersindeki isim: " + p1);
		System.out.println("İkinci nesne icersindeki isim: " + p2);

		Person p3 = new Person("");
		try {
			p3 = (Person)p1.clone();
		} catch (Exception e) {};

		p3.setName("Ahmet Aykac");
		System.out.println("Birinci nesne icersindeki isim: " + p1);
		System.out.println("Ucuncu nesne icersindeki isim: " + p3);

	}
}
```

Nesneleri, kendileri dışında başka metodlar içerisinde doğrudan klonlamak mümkün değildir.

Yukarıda anlattığımız klonlama işleminde aslında clone metodu ile nesneyi kopyalrken kendi sınıfı içerisinde bu işlemi yaptık. Oysa tamamen sınıfın dışında daha önceden yaratılmış bir sınıf türünden örneği kolnlamaya kalktığımızı düşünelim. Bu sefer işlem yapılamayacaktır. Çünkü klonlanmak istenen bu sınıfın Cloneable arayüzünden implement edilmesi şarttır. Ayrıca bu işlemde bir checked exception meydana gelir (Checked Exception ileriki koularda ele alınacaktır. Şimdilik sadece ele alınması zorunlu olan exceptionlar olarak tanımlayabilriiz.)

Bir nesnenin heryerde klonlanabilir olmasını sağlamak
Böyle durumlarda İlgili nesne Cloneable arayüzünden implement edilir ve clone metodu polymorphism yöntemiyle yeniden yazılabilir. Bu yeniden yazımda clone metodunun prototipi

public Object clone();

şeklinde yazılır.

Ancak şunu da belirtmek gerkir ki aslında Cloneable arayüzü içerisinde hiçbir metod tanımlı değildir.

Cloneable içerisinde hiçbir metıd yoksa neden bu sınıftan implementasyon yapıyoruz?
clone metodu aslında Cloneable sınıfı içerisinde tasnımlı bir metd değildir. Bu metod Object sınıfı içerisinde tanımlıdır ve protected anahtar sözcüğü ile tanımlanmıştır. Ancak yine de kullanıcıya bilgi vermek amacıyla Cloneable dan implement yapılmaktadır.

Etiketleme Arayüzleri
NOT: Burada gördüğümüz gibi java'da bazı arayüzler bu şkilde "etiketleme" amacıyla kullanılır. Bu arayüzlerde hiç metod bulunmaz ve sadece gerektiği durumlarda "instanceof" operatörüyle kullanabilmek için implement edilirler.

NOT: Aslında clone metodu Object sınıfı içerisinde protected olarak tanımlanmıştır. Bütün sınıflar Object sınıfının bir alt sınıfı ise ve bir alt sınıf türediği sınıfın protected metodlarına erişebiliyorsa, o halde neden herhangi bir nesne başka nesnenin clone metoduna doğrudan erişemez? Protected metod konusundan da hatırlayacağımız gibi bir sınıf kendisinin atası olan sınıfın protected metoduna türetme sırasında, kendi içerisinde erişebilir. Ancak bu sınıfın atasında yer alan protected metoda başka bir sınıf içerisinde erişilemez. Bu nedenle bir sınıfın başka bir sınıf içerisinde clone işlmeine tabi tutulması için bu sınıfın clone metodunu public olakrak yeniden bildirmesi gerekir.

```
Şimdi özet olarak bir nesnenin herhangi bir yerde clone işlemine tabi tutulabilmesi
için:

class A implemenrs Cloneable {


      public Object clone() {
            try {
                  return super.clone();

            }

      }

      ...
```

şeklinde clone metodunu yeniden yazması gerekir. Böylece bu nesnenin clone metodunu çağırmak isteyen kiş aslında public olan clone metodunu çağrabilir. Ancak nesne içerisinde bu public clone meodu, kendi ata sınıfının protected olan clone metodunu çağırmaktadır. Ata sınıf içerisindeki bu clone metodu da nesnenin veri elemanlarını alan alan kopyalar ve bir nesne olarak geri döndürür.

İçerisinde başka sınıfları veri elemanı olarak kullanan sınıfların klonlanması
Böyle durumlarda yazılan public clone metodu bu veri elemanlarını da klonlayacak şekilde dikkatlice yazılmalıdır. Diyelim ki A bir sınıf olsun içinde B adında başka bir sınıfı veri elemanı olarak kullansın:

```
class B  {
      double y;
}

class A implements Cloneable{
      int a;
      String str;
      B b;

      ...

      public Object clone() {
            try {
                  /*A nesnesi klonlarak geçisi bir nesnede
                  tutluyor. Ama b değerinin adresi kopylanadı.
                  Çünkü b aslında B türünden ve alan alan kopylamada b nin kendisi
                  değil adresi kopylanadı.*/
                  A cloned_a = (A)super.clone();

                  /*nesnenin kendi kopyasını tutan geçici
                  değişkenin b alanının da klonu alınıyor. Artık geçici kopya da
                  adres kopyalam işlemi düzeltildi.*/
                  cloned_a.b = (B) b.clone();

                  /*Geçici nesne geri gönderiliyor*/
                  return cloned_a;
            }
      }
}
```

Ödev: (2 haftalik)

Çok kapsamlı bir Student nesnesi yaratınız. Bir Student nesnesinin veri elemanları:

a- String student_name
b- Vector courses (Ogrencinin aldigi dersler, dinamik bir array olmali, her elemani Course olmali)
c- Adress address (Ogrencinin adresi)
d- Date b_date (Ogrencinin dogum tarihi)
e- int term; (Ogrencinin donem bilgisi)

Bununla birlikte gerekli metodlari ile birlikte bu nesneyi tasarlayiniz.

Bu nesnenin tasarlanabilmesi icin tabi ki Date, Course ve Address nesnelerinin de gerekli metodlariyla birlikte tasarlanmasi gerekmektedir.

Date sinifi, icerisinde bir tarih ve zaman bilgisi tutan bir siniftir. Bu sinifin veri elemanlari day, month, year, hour, minute, second olmalidir. Ve tabi ki bu degiskenleri isleyebilecek uygun metodlari barindirmalidir. Bu sinifin metodlarindan bazilari asagida verilmistir:

public void addDay(int toAdd) - tariha toAdd kadar gun ekler.
public void addMonth(int toAdd) -tarihe toAdd kadar ay ekler
public void addYear(int toAdd) -tarihe toAdd kadar yil ekler
public int diff(Date d) - iki tarih arasindaki gun farkini veren bir metod
public void addCourse(Course new_course) - yeni bir course alip vector nesnesine ekleyecek
/*secime bagli*/
public int getAge() - Ogrencinin yasini retrun edecek. (Sistem zamanini almaniz gerekir)

Bunun disnda da gerekli olan diger metodlar eklenecektir.

Address sinifi sadece address bilgisini tutmak ve kullanabilmek icin yaratilacaktir. Bu sinifin tum degisken ve metdolari tasarimciya birakilmistir.

Course sinifi sadece 1 dersin bilgilerini tutmak icin kullanilacaktir. Bu sinifin tum degisken ve metdolari tasarimciya birakilmistir.

Tum metodlarin icerigi tasarimciya birakilmistir. Ancak yaratilan nesnelere iliskin dokumantasyon hazirlanacaktir .Bu dokumantasyonda bir metodun sadece adi, aldigi parametre ve geri donus degeri anlatilacaktir. Metodun icerideki isleyisi anlatilmayacaktir.

Bir Student nesnesi Comparable arayuzu ve Person turunde bir arayuzden'den turemelidir. Person interface'i asagida verilmistir.

```
public interface Person {

        public String getName();
        public String getAddress();
        public String getFamilyName();
        public String listCourses();
        public Date getBirthday();

}
```

Hata Yakalama (Exception Handling)
Java'da error adı verilen hatalar meydana geldiğinde program akışı doğrudan sonlandırılmaktadır. Ancak bazen, aslında derleme sırasında hata olduğu belli olmayan ancak çalışma zamanı sırasında yapılan hatalı işlemler neticesinde ortaya çıkan hatalar olabilir.

Çalışma zamanı sırasında meydana gelen hatalar
Mesela program içerisinde "x/a" gibi bir ifade oldugunu varsayalım. Bu durumda program derlenirken bu ifade hatalı olarak algılanmaz. Ancak program çalışırken a değişkenine 0 değeri atanırsa bu ifade matematiksel olarak sonsuz anlamına gelir ve bu da başka bir hata üretilmesine neden olur.

İşte bu tur hatalara error değil "exception" adı verilir.

Exception ele alınabilir ama error alınamaz
Exception ile error kavramları saece tanım olarak değil, nitelik olarak da farklıdır. Çünkü java da excepton'lar runtime sırasında yakalanabilir. Ancak error olduğu zaman bu durum kesinlikle programın çalışmasını sonlandırır.

Exception ların ele alınması
Exception, programda bazı özel bloklar içerisinde ele alınır. Programın çalışması sırasında bir exception olursa hemen o anda bir "Exception" nesnesi yaratılır. Bu nesne, aslında Exception nesnesinden türemiş bir başka exception da olabilir. Ama neticede en temel anlamda bir "Exception" nesnesi yaratılır.

Çalışma zamanı sırasında üretilen exception'lar ele alınabilir
Daha sonra eğer programcı tercih etmişse bu exception nesnesi özel bir blok tarafından yakalanır ve yine aynı bloklar içerisinde programcının istediği kodları çalıştırır.

```
try {
	openFile();
	readFile();
	closeFile();
}

catch(Exception e) {
	doSomething();
}
```

Yukarıda da görüldüğü gibi aslında exception ların yakalanma prensibi, exception olma ihtimali olan satırların bir try blogu içerisinde yazılması ve meydana gelen exception'ı yakalayacak bir catch blgunun da hemen bu try bloklarının sonuna yerleştirilmesi şeklindedir.

Exception'lar Es Geçilemez
Exceptionlar uretildigi anda programcı tarafından ele alınmazsa program exception ile birlikte sonlanır. Bu nedenle bir exception ele alınmadan program akışına devam edilemez.

Kontrol edilen ve Kontrol Edilmeyen Exception lar
Aslında Exception lar kontrol edilen ve kontrol edilmeyen olmak uzere 2'ye ayrılır.

Çalışma zamanı sırasında meydana gelen bazı exceptionların kontrol edilmesi pogramcının isteğine bağlıdır. Bu tip exception lar "RunTimeException" nesnesinden türemişlerdir. Programcı bu exceptionları ele almazsa program bir hata mesajı ile sonlanır. Ya da programcının ele aldığı şekilde akışına devam ederi. (dizi indeksinin geçilmesi, sıfıra bölme hatası)

Kontrol edilen exceptionların ele alınması prgoramcının isteğine bağlı değildir. Bu exceptionların ele alınması zorunludur. Mesela bir sodyanın açılması okunması ya da kapanması işlemleri kontrol edien exceptionlardır. Bu tür exceptionlar programcının hatasından kaynaklanmayan exception lardır (Bozuk bir dosya ya da URL)

NOT: Aslında bütün exception'lar Throwable nesnesinden türemiştir.

Exception ların Yakalanması
Aşağıdaki örnekte kulanıcıdan bir sayı girmesi istenecektir.

```
import javax.swing.*;

public class HataYakalama {

	public static void main(String[] args) {
		String str = JOptionPane.showInputDialog(null, "Bir sayı
								giriniz");
		try {
			int n = Integer.parseInt(str);
			System.out.println("Girilen sayi: " + n);
		}

		catch(NumberFormatException e) {
			e.printStackTrace();
			System.exit(0);
		}
	}
}
```

Eğer kullanıcı bir sayı yerine yanlısklıkla bir metin girerse, o zaman bu metinin Integer.parseInt metodu yardımıyla sayıya donusturulmesi bir runtime exception uretilmesine neden olur. Ve program akısı hemen RunTimeExeption ureterek catch bloguna atlar. Boylece hata yakalanmıs olur. Eger hata yaklalanmaz ise JVM u hatayı kendisi yakalar ve ekrana asağıdaki gibi bir exception mesajı gondererk program akışını sonlandırır:

```
java.lang.NumberFormatException: For input string: "gdg"
 at java.lang.NumberFormatException.forInputString(Unknown Source)
 at java.lang.Integer.parseInt(Unknown Source)
 at java.lang.Integer.parseInt(Unknown Source)
 at HataYakalama.main(HataYakalama.java:9)
```

```
import javax.swing.*;

public class HataYakalama {

	public static void main(String[] args) {
		int n;
		String str = JOptionPane.showInputDialog(null, "Bir sayı giriniz");

		try {
			n = Integer.parseInt(str);

		}

		catch(NumberFormatException e) {
			JOptionPane.showMessageDialog(null, "Exception yakalandı");
			n = Integer.parseInt(JOptionPane.showInputDialog(null, "Giris
yanlis tekrar deneyiniz:"));

		}

		System.out.println("Girilen sayi: " + n);
	}
}
```

KISACA REKURSIF METODLAR

Uygulama:
Kullanıcıdan doğru giriş alana kadar çalışan bir sayı isteme mekanizması yazınız.

Cevap:
```
import javax.swing.*;

public class HataYakalama {


	public static int sayiAl(String onEk) {
		JOptionPane.showMessageDialog(null, "DEBUG: metoddayiz");
		int n = -1;
		String str = JOptionPane.showInputDialog(null, onEk + " Bir sayı
giriniz");

		try {
			JOptionPane.showMessageDialog(null, "DEBUG: try blogundayiz");
			n = Integer.parseInt(str);
		}

		catch(NumberFormatException e) {
```

```
                JOptionPane.showMessageDialog(null, "Exception yakalandı");
                onEk = "Giris yanlis tekrar tekar deneyiniz.\n";
                return sayiAl(onEk);

            }

            return n;

        }

        public static void main(String[] args) {

            System.out.println("Girilen sayi: " + sayiAl(""));
            System.exit(0);
        }
}
```

Exception Ureten Metdolar
Su ana kadar ogrendiklerimizden görüyoruz ki Java'da bazı satırlar exception oluşmasına neden olabilir. O halde bazı metodlar da exception uretilmesine neden olabilirler.

Bir metod eger exception uretiyorsa bu metodu kullandığımız satırında try blogu içerisinde yazılması gerekir. O halde Java'da bazı hazır metodları incelerken bunların exception uretip uretmediklerine de bakılmalıdır.

FileInputStream
read
Integer.parseInt metodlarının dokumantasyondan incelenmesi.

Eğer biz de yazdığımız metodların exception uetmelerini istiyorsak metod prototipinin sonuna throws anahtar sözcüğü ve devamında da üretilecek exception'ın tipini yazarız.

```
public String getValue(int index) throws IndexOutOfBoundsException {

        İf(index == null || index >= values.length) {
                throw new  IndexOutOfBoundsException();
        }
}
```

Checked ya da Unchecked Exception Nasıl Yaratılır
Eger exception gonderen method "Exception" ya da ondan tureyen bir exception'ı throw ediyorsa bu exceptionlar checked sayilir. O halde bu metodu kullanan kisi metodu try bloklari icerisinde yazman zorudadir.

Ancak gonderilen exception "RunTimeException" ya da ondan turemis bir exception ise metodumuz unchecked exception uretiyor demektir. O halde bu metodu try bloklari icerisinde yazmak zorunda degiliz.

finally Blok Yapısı
finally blok yapısı catch bloklarının sonuna eklenir Bir exception yakalansa da yakalanmasa da program akışı finally blokları içerisine yönlenecektir. Aşağıdaki örneği incelyiniz:

```
try {
        code a;
}
```

```
catch(IOException e) {
	code b;
}
finally {
	code c;
}
```

1. Burada code a; hiçbir exception üretmeyebilir. Bu durumda "code a" çalıştıktan sonra "code c" çalışacaktır. Program akışı daha sonra finally bloğu altındaki kodları çalıştırmaya devam eder.

2. Bu durumda "code a" IOException üretir ve bu nedenle "code b" çlaışır. Daha sonra yine "code c" çalışacaktır. Eğer catch bloğu herhangi bir exception üretmez ise program akışı yine try bloklarının bittiği yerden devam edecektir.

3. try blokları içerisinde catch içerisinde yakalanmayan bir exception üretilirse o zaman "code a" ve "code c" çalıştıktan sonra exception kullanıcıya gönderilir.

Aşağıdaki örneği değişik senaryolar için deneyiniz:

```
package derstekiornekler;


public class ExceptionTest {


	public static void main(String[] args) {


		try {
			System.out.println("Satir 1");
			throw new Exception();
		}
		catch(Exception e) {
			System.out.println("Catch blogu icindeyiz");
		}
		finally {
			System.out.println("Finally blogu icindeyiz.");
		}


		System.out.println("Catch blogu disindayiz");
	}
}
```

NOT: finally bloklarını açılan streamleri, exception olsa bile kapatabilmek amacıyla kullanabiliriz.

Java ile Programlama
Bölüm 8:

Java'da Dosya İşlemlerine Giriş

Programcılığın vazgeçilmez işlerinden birisi de disk üzerindeki dosyalarla çalışmaktır. Buraya kadar konuları öğrenmek amacıyla kullanmış olduğumuz örneklerde aslında veriler sadece o anda çalışan program içerisinde ele alınabiliyordu.

Profosyonel anlamda yazılan programların çoğu kullanıcının ihtiyaç duyacağı verileri dosyalar üzerinde saklamakta ve gerektiğinde bu dosyalardan verileri okuyarak ya da yeni verileri dosyalara yazarak işleyişlerini sürdürmektedir. Bu nedenle programcılıkta amaç ne olursa olsun dosyalar üzerinde işlem yapma ihtiyacı mutlaka karşınıza çıkacaktır.

File Sınıfı

Java'da dosya işlemlerine ilişkin ilk olarak "File" isimli sınıfın kullanımını inceleyerek başlayalım. Bu sınıf disk üzerinde yer alan bir dosyayı program içerisinde de bir nesne haline getirerek programcının dosya üzerinde işlem yapabilmesini sağlamaktadır. Sınıfın başlangıç fonksiyonu temel olarak bir dosyanın yol (path) bilgisini alarak File nesnesini yaratır. Örneğin:

```
...
File dosya1 = new File("Dosya.txt");
File dosya2 = new File("c:\ornekler\dosya.txt");
File dosya3 = new File("/home/Academytech/Dosyam.txt")
...
```

Tabi yukarıda verilen bu örneklerde, "File" sınıfının başlangıç fonksiyonuna parametre olarak verilen bu yolların diskte de düzgün bir şekilde mevcut olması gerekmektedir. Yani bulunduğumuz dizinde Dosya.txt ya da "c:\ornekler\" dizini altında bir "dosya.txt" ya da "/home/Academytech/" dizini içerisinde bir "Dosyam.tx"t dosyası bulunmalıdır. Aksi takdirde "dosya1", "dosya2" ya da "dosya3" ismi ile temsil edilen "File" nesneleri düzgün bir şekilde yaratılamayacaktır.

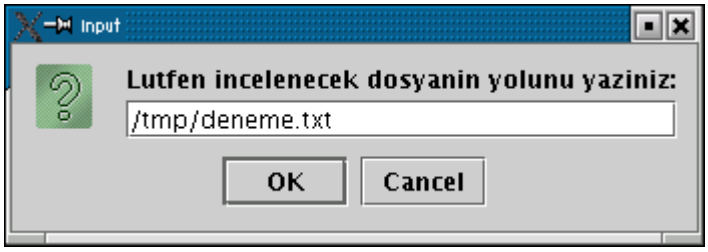
Not: Java'da dosya işlemlerinin yapılabilmesi için io.* kütüphanelerinin programın en üst satırında "import" anahtar sözcüğü ile programa dahil edilmesi gerekmektedir.

Aşağıdaki örnekte temel olarak "File" sınıfı ile dosya işlemlerinin ele alınmasına ilişkin bir program yazılmaktadır.

```
1  import java.io.*;
2  import javax.swing.*;
3
4  class Dosyalar {
5
6    public static void main(String arg[]) {
7
8          String dosyaYolu = JOptionPane.showInputDialog(null, "Lutfen "+
9                        "incelenecek dosyanin yolunu yaziniz:");
10
11         File dosya = new File(dosyaYolu);
12
13         String str = "Dosya ile ilgili genel bilgiler\n";
14         str += "Dosyanin adi:  " + dosya.getName() + "\n";
15         str += "Dosyanin konumu:  " + dosya.getPath() + "\n";
16         str += "Dosyanin boyutu:  " + dosya.length() + " bayt\n";
17
18         JOptionPane.showMessageDialog(null, str);
19
20   }
21 }
```

Örneğimizde temel bir File nesnesi yaratarak aslında diskte yer alan bir dosya bilgilerinin nasıl ele alındığını görmekteyiz. Bu örnekte benim girmiş olduğum "/tmp/dosya.txt" yolu aslında kendi sistemimdeki bir dosyanın

yerini temsil ettiğinden siz kendi örneğinizi yazarken kendi diskinizdeki yolu yazmaya dikkat ediniz. Program çalıştırıldığında "InputDialog" nesnesi ile aşağıdaki resimde gösterildiği gibi sizden bir dosya yolu istenecektir:



Girdiğiniz bu yol bilgisi programımızın 8. satırında yazıldığı şekilde "dosyaYolu" değişkenine aktarılmakta ve daha sonra da 11. satırda olduğu gibi bu değişken yardımı ile nesne yaratılmaktadır. Bu nesne yardımıyla kullanılabilecek birkaç metodu 14, 15 ve 16. satırlarda görmekteyiz.

System.out.println metodu

Bu metodu daha önceki derslerimizden gayet iyi biliyoruz. Aslında bu metodun System paketi içerisindeki out sınıfının println metodu olarak düşlünebilirsiniz. Ancak gerçekte durum bu değildir. Buradaki durumun anlaşılması, java'da dosyalar konusunu kavramak için önemli bir başlangıç olacaktır. O halde ilk önce bu durumu inceleyelim.

out System sınıfı içerisinde statik bir nesnedir (instance)

Aslında "out", System sınıfı içerisinde yer alan static bir instance'dır. Bu yüzden programcı bu instance'a doğrudan erişebilir. Ayrıca bu nesne bir "PrintStream" nesnesidir.

Stream Kavramı:

Programcılık açısından dosya işlemlerini anlamanın en iyi yolu Stream kavramını bilmekle başlar. Stream denilen nesne aslında bir şeyin okunması ya da yazılması için bir soyutlamadır (abstraction). Bu soyutlama gereklidir, çünkü okunacak ya da yazılacak bilginin tipi her zaman değişebilir. Yani okunan ya da yazılan bilgi bir uygulamada manyetik karttan yapılabileceği gibi başka bir uygulamada da diskten yapılabilir.

Burada Stream aslında kelime anlamı olarak akış anlamına gelir ve programcılık anlamında "bilgi akışını" ifade eder. Bu yaklaşıma göre okuacak bilgi stream den okunur ve yazılacak bilgi de stream e yazılır. Böylece bilginin tipi ne olursa olsun okunma ve yazılma aşamasında stream üzerinde ortak bir bilgi akışı sağlanacaktır. Bu ifadeyi kullandığımız örneklerle daha iyi pekiştireceğimizi düşünüyorum.

Stream kullanımı ile bilgi ne olursa olsun okuma ya da yazma şekli hep aynıdır:

```
stream aç
daha fazla bilgi oldukça (ya da yazdıkça)
        bilgi oku (ya da yaz)
stream kapat
```

○ halde out nesnesine geri dönelim. out nesnesi, bu şekilde adı OutputStream denilen bir arayüzden türetilmiş bir sınıf olan PrintStream sınıfı türündendir. Aslında bizim ekrana bilgi gönderme şeklimiz her işletim sisteminde farklıdır. Ancak JVM içerisinde platforma göre implement edilmiş olan bu PrintStream nesnesi bize bu soyutlamayı sağlar ve böylece içerisinde yer alan println metodunu platform bağımsız olarak kullanabiliriz.

flush Kavramı

Aslında yazılması ya da okunması istenen bilgiler yukarıda nalatılan stream'ler içerisine ve oradan da yazılması ya da okunması istenen yere gönderilir. Ancak bir stream'in kendi içini ne zaman boşaltacağı (yani istenen bilgiyi gerekli yere ne zaman ulaştıracağı) belli değildir. Bu işleme "flush" denilir. Programcı isterse bunu kendisi de yapabilir.

InputStream Nesnesi

Yine System sınıfı içerisinde yer alan ve out nesnesine benzeyen bir nesne daha vardır. Bu nesnenin adı "in" ve türü de InputStream sınıfı türündendir. Bu nesne de konsoldan bir bayte lık bilgi okuma yetenegine sahiptir. Ancak okuma yeteneği bu kadarla sınırlıdır. Bu nesneyi "debug" işlemleri için kullanabiliriz.

```
public class InputStreamOrnek {

	public static void main(String[] args) {
		for(int i = 0; i < 10; i++) {
			System.out.println("i: " + i);
			try {
				System.in.read();
			}
			catch(Exception e) {

			}
		}
	}
}
```

Standart Giriş ve Çıkış

Java'da yukarıda bahsettiğimiz bu stream kavramı ile standart giriş ve standart çıkışa yazma ve okuma yapmak mümkün olacaktır. Aslında standart giriş-çıkışı komut satırı ortamı olarak düşünebilirsiniz.

Fazla detaya inmeden söylemek gerekirse standart çıkışa yazma yapma işlemi aslında daha önceden bildiğimiz Sytem.out.print("yazi") metodu ile yapılmaktadır. Arka planda aslında "System" isimli sınıfın içerisinde tanımlı "PrintStream" türünden yaratılmış ve adı "out" olan static bir değişkeni kullanmaktayız.

BufferedReader <-- InputStreamReader <-- InputStream

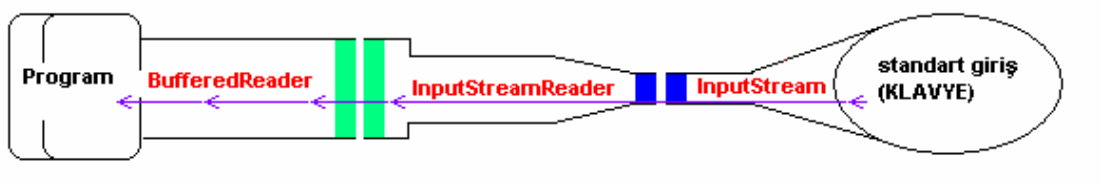
Aynı şekilde standart girişten program içerisine okuma yapmak da mümkündür. Bu işlem için yetenekleri sınırlı olan System.in nesnesine InputStream nesnesi ve BufferedReader nesnesi yardımıyla yetenek kazandıracak ve en son BufferedReader nesnesinin readLine metodu ile okuma yapacağız:

```
 1 import java.io.*;
 2
 3 class Dosyalar2 {
 4
 5   public static void main(String arg[]) {
 6
 7         BufferedReader in = new BufferedReader(
 8                 new InputStreamReader(System.in));
 9
10         try {
11
12                 System.out.println("\n\n");
13                 System.out.print("Lutfen okunacak bilgiyi giriniz: ");
14                 String line = in.readLine();
15                 System.out.print("\n\n\n");
16                 System.out.println("Okunan satir: " + line);
17
18         } catch(IOException e) { e.printStackTrace(); }
19   }
20 }
```

Örneğimizde 7. satırda BufferedReader nesnesi yaratılmıştır. Bu nesnenin başlangıç fonksiyonuna parametre olarak bir "InputStream" nesnesi hemen orada yaratılmakta ve gönderilmektedir. "InputStream" nesnesi okuma yapmak amacıyla kullanılır. Bu nesnenin de başlangıç fonksiyonuna, okumanın yapılacağı yer olan standart giriş parametre olarak verildiğinden bu stream standart girişe yönlendirilmiş, yani standart girişten okuma yapacak şekilde hazırlanmıştır.

Program çalıştırılırken aşağıdaki resimde görüldüğü gibi komut satırı ortamında bizden birşeyler girmemizi isteyecek ve bu bilgileri daha sonra ekrana yazacaktır. Bilginin standart girişten alınmasını sağlayan komut da 14. satırda görülmektedir. Aslında bu komut soyutlanmış "BufferedReader" nesnesinin "readLine" metodur ve bu nesne yukarıda da anlatmış olduğum gibi standart girişe yönlendirlmiştir.

```
$ javac Dosyalar2.java
$ java Dosyalar2
Lutfen okunacak bilgiyi giriniz: bu yazi standart girsten yaziliyor
Okunan satir: bu yazi standart girsten yaziliyor
```



Text Dosyalarının Okunması ve Yazılması:
(FileReader ve FileWriter sınıfları)

Yukarıda bahsettiğimiz stream kavramı sayesinde standart girişten yapılan okuma aynı şekilde bir text dosyasına yönlendirilirse, o zaman dosyadan okuma yapmak da mümkün olacaktır. Aynı şekilde standart çıkışa gönderilen bir bilgi text dosyasına yönlendirilmiş bir stream e gönderilirse o zaman da text dosyasına yazma işlemi yapmış oluruz. İşte burada da stream soyutlamasının kolaylığını görmekteyiz. Aşağıdaki örnekte, bahsettiğimiz bu dosyadan okuma işlemini görmekteyiz.

```
import java.io.*;

public class DosyaIslemleri {

      public static void main(String[] args) {
            String str;
            String fileName = "deneme.txt";

            try {
            BufferedWriter out = new BufferedWriter(
                              new FileWriter(fileName));

            out.write("Deneme Satir 1 ");
            out.write("Deneme Satir 2\n");
            out.write("Deneme Satir 3");
            out.close();


            BufferedReader in = new BufferedReader(
                              new FileReader(fileName));

            while((str = in.readLine()) != null) {
                  System.out.println(str);
            }
            in.close();
```

```
            } catch(IOException e) {}
        }
}
```

NOT: Aslında BufferedReader ve BufferedWriter sınıfları olmadan da doğrudan FileWriter ve FileReader sınıfları yardımıyla dosyaya text yazmak ya da dosyadan text okumak mümkündür. Yani bu soyutlama zaten yapılmıştır. Ancak Buffered sınıfları yardımıyla bu okuma ve yazma işlemi daha büyük verier için daha verimli bir şekilde yapılmaktadır.

Veri Dosyalarını Okunması ve Yazılması:
(DataOutputStream, DataInputStream)

Veri dosyaları önceki örnekte kullanmış olduğumuz text dosyalarından biraz daha farklıdır. Bildiğimiz gibi program yazarken değişkenlerin yaratılması sırasında onların türlerini de belirtmek zorundayız. Daha sonra da bu değişkenlere verilen değerleri de aynı türden vermek zorundayız.

İşte veri dosyalarında saklanan bilgiler de bu şekilde tür bilgisi ile birlikte sanki hafızada(ram'de) saklanıyormuş gibi saklanmaktadır. Yani text dosyalarında olduğu gibi verilerin tamamı string şeklinde saklanmaz. Bu nedenle de veri dosyalarını bir text editörü ile açıp okumaya kalktığınızda bilgilerin anlaşılır olmadığını görürüz. Aşağıdaki örnekte bir veri dosyasına birtakım bilgiler yazılmakta ve daha sonra da aynı dosyadan bu bilgiler okunmaktadır.

Aslında buradaki mantık yine yukarıdaki anlattıklarımıza benzer. Yukarıda bir dosyaya text yazmadan önce bir FileWriter stream açarak dosyaya giden bir yol açmış olduk. Ancak bu yola verileri dha verimli yüklemek için yolun ucuna bir BuffereWriter tüneli ekleyerek onun üzerinden dosyaya gönderdik. Ancak bu şekilde ancak text verileri dosyaya yazabiliyoruz.

Dosyaya verileri tür bilgisi ile gönderebilmek için önce dosyaya ulaşan bir FileOutputStream tüneli açıyoruz. Daha sonra bu tünelin ucuna BufferedOutputStream ve onun da ucuna DataOutputStream ekleyerek verileri tür bilgileriyle birlikte dosyaya gönderebiliyoruz.

```
 1 import java.io.*;
 2 import javax.swing.*;
 3
 4 class Dosyalar4 {
 5
 6   public static void main(String arg[]) throws IOException {
 7
 8         //Once dosyaya yaziyoruz
 9         DataOutputStream out = new DataOutputStream(
10                 new BufferedOutputStream(
11                         new FileOutputStream("CikisDosyasi.data")));
12         out.writeDouble(2.65);
13         out.writeInt(12);
14         out.close();
15
16         DataInputStream in = new DataInputStream(
17                 new BufferedInputStream(
18                         new FileInputStream("CikisDosyasi.data")));
19         double r1 = in.readDouble();
20         int r2 = in.readInt();
21         System.out.println("\nOkunan veriler: \n" + r1+"\n"+r2);
22         in.close();
23   }
24 }
```

Yukarıdaki örneğimizde bu sefer veri yazma işlemi yapmaktayız. Bu nedenle stream nesnemizi 9. satırda olduğu gibi veri yazabilecek ve 16. satırda olduğu gibi veri okuyabilecek şekilde hazırlamaktayız. 12. ve 13. satırlarda sırayla

yazılan double ve int veriler okunurken de yazıldıkları sıra ile okunurlar. Veri yazma işlemini program içerisindeki bir değişkenin hafızada değil de dosyada saklanması gibi düşüebiliriz. Çünkü bazen verileri bu şekilde tür bilgileriyle saklamak ve daha sonra da ele almak gerekmektedir. Bu işlemi bir text dosyasıyla yapmaya kalkarsak verilerin sadece görüntülerini saklamış oluruz. Bu durumda verilerin daha sonra tekrar okunması sırasında tür bilgileri kaybolacağından örneğimizdeki 19. 20. satırdakilere benzer atamalar mümkün olmayacaktır.

throws Kullanımının getirdiği kolaylık

Ayrıca dikkat edersek bu örneğimizde "try" ve "catch" bloklarını kullanmak yerine üretilecek "Exception" bilgisini daha farklı ele aldık. Eğer bir metodun içerisinde "Exception" üretilme ihtimali varsa örneğimizde olduğu gibi metodun prototipine "throws" anahtar sözcüğünden sonra üretilecek Exception yazılırsa, metod içerisinde bu Exception'ı üretebilecek satırları "try" bloklarında ele almaya gerek kalmaz. Buna ilişkin detaylı bilgileri bir önceki dersimizde bulabilirsiniz.

Özetleyecek Olursak

Text Veri ---> BufferedWriter ---> FileWriter ---> DOSYA

Program <--- BufferedReader <--- FileReader <--- (TextVeri) DOSYA

Ham Veri--->DataOutputStream-->BufferedOutputStream--->FileOutputStream--->DOSYA

Ham Veri<---DataInputStream<--BufferedInputStream<---FileInputStream<---
(HamVeri)DOSYA

Hep Sırayla Okuduk, Sırayla Yazdık Ama..

Dersimizin buraya kadar olan kısmında stream yaratarak dosyalara yazma ve okuma işlemlerini gördük. Ancak bu yazma-okuma işlemlerini hep sırayla yaptık. Yani bir veriyi okumadan, ondan daha sonra gelen veriyi okumak mümkün değildi. Yine aynı şekilde ilk veriyi yazmadan ikinci veriyi yazmak yukarıda gösterdiğimiz stream ler ile olanaksızdı. Böyle bir işlem yapmak için daha farklı bir stream kullanmamız gerekmektedir. Dosyalara bu şekilde erişim isteğine teknik olarak "rastgele erişim" denilir. Şimdi bunu inceleyelim.

Dosyalara Rastgele Erişim İşlemi

Bir dosyaya rastgele erişim yapılabilmesi için öncelikle "RandomAccessFile" sınıfı türünden bir değişken yaratmak gerekir. Bu değişken yaratılırken 1. parametre olarak erişimin yapılacağı dosya ve ikinci parametre olarak da erişim hangi modda yapılacağı bilgisi verilmelidir.

Örneğin "CikisDosyasi.dat" dosyamıza okunabilir ve yazılabilir modda erişmek istiyorsak nesneyi şu şekilde yaratmamız gerekir:

RandomAccessFile myFile = new RandomAccessFile("CikisDosyasi.dat", "rw");

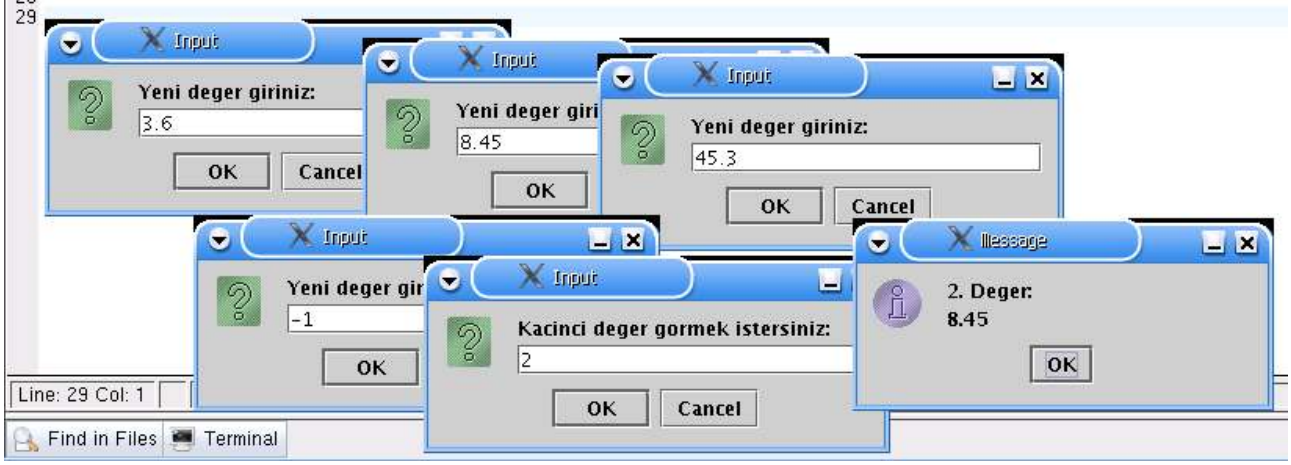
Dosyaya rasgele erişim için kullanılabilecek en genel modlar şu şekilde özetlenebilir:

r: Dosya sadece okuma için açılır. Herhangi bir yazma işlemi yapılırsa IOException üretilir.

rw: Dosya yazma-okuma için açılır. Eğer dosya hali hazırda mevcut değilse yeni bir tane yaratılır.

Aşağıdaki örnekte bir dosya içerisine veriler yazıldıktan sonra rastgele erişilerek okunmaktadır. Burada okuma işlemi sırayla yapılmamaktadır.

```
1 import java.io.*;
2 import javax.swing.*;
3
4 class Dosyalar5 {
5
6   public static void main(String arg[]) throws IOException {
7
8         RandomAccessFile rf = new RandomAccessFile("CikisDosyasi.dat", "rw");
9
10        double yeniSayi = Double.parseDouble(
11                          JOptionPane.showInputDialog(null, "Yeni deger giriniz: "));
12        System.out.println(""+yeniSayi);
13        while(yeniSayi != -1) {
14                rf.writeDouble(yeniSayi);
15                yeniSayi = Double.parseDouble(
16                     JOptionPane.showInputDialog(null, "Yeni deger giriniz: "));
17        }
18
19        int pozisyon = Integer.parseInt(
20                          JOptionPane.showInputDialog(null, "Kacinci deger gormek istersiniz:"));
21        rf.seek(pozisyon * 8);
22        JOptionPane.showMessageDialog(null, pozisyon+". Deger:\n"+rf.readDouble());
23
24        System.exit(0);
25
26  }
27 }
28
29
```



Örneğimizde 8. satırda okuma-yazma modunda açılan dosyamıza bazı double değerler ekliyoruz. Bu ekleme işlemini bir döngü ile yapıyoruz. Kullanıcı 13. satırdaki sorgu gereği "-1" girmediği sürece girilen bilgiler önce double türüne dönüştürülüyor ve daha sonra da dosyaya yazılıyor. Bu dönüştürmenin nedeninin, girilen bilgini hep string olmasından dolayı olduğunu önceden hatırlayınız.

Daha sonra 21. satırda kullanılan "seek()" metodu sayesinde dosya içerisinde rastgele bir pozisyona gidiyoruz. Bu pozisyon, seek metoduna asıl pozisyon * 8 olarak gönderiliyor. İlk pozisyon "0" olduğu için kullanıcının istediği pozisyondan 1 çıkartarak seek() metoduna gönderiyoruz. Böylece kullanıcı gerçekten istediği sayıyı elde edebiliyor.

NOT: seek() metodunun içerisine yazılan 8 * cursor ifadesindeki 8, double veriler icin. Her veri icin bu dger degisir. Mesela int veriler icin bu deger 4, byte veriler icin ise 1 dir. Bu bilgiler dokumantasyonda yer alan write metodlarinda mevcuttur.

Object Serialization Hakkında

```
import java.awt.*;
import java.io.*;

public class ObjectReaderWriter {
  String filePath;

  public static void main( String args[] ) {
    ObjectReaderWriter orw = new ObjectReaderWriter();
  }

  ObjectReaderWriter() {
    try {
```

```
      //  Allow the user to specify an output file.
      FileDialog fd = new FileDialog( new Frame(), "Nesnelerin yazilacagi dosya..",
FileDialog.SAVE );
      fd.show();
      filePath = new String( fd.getDirectory() + fd.getFile() );

      //  Next, create an object that can write to that file.
      ObjectOutputStream outStream =
        new ObjectOutputStream( new FileOutputStream( filePath ));


//    Create instances of each data class to be serialized.
      MySerialObject serialObject1 = new MySerialObject(12, "Academytech");
      MySerialObject serialObject2 = new MySerialObject(24, "Caglar");

      //save each object
      outStream.writeObject( serialObject1 );
      outStream.writeObject( serialObject2 );
      outStream.flush();
      outStream.close();
      //  Allow the user to specify an input file.
      fd = new FileDialog( new Frame(), "Nesnelerin cekilecegi dosya",FileDialog.LOAD
);
      fd.show();
      filePath = new String( fd.getDirectory() + fd.getFile() );

      ObjectInputStream inStream = new ObjectInputStream( new FileInputStream(
filePath ));

      // Retrieve the Serializable object.
      MySerialObject serialObject = ( MySerialObject )inStream.readObject();
      //MySerialObject serialObject2 = ( MySerialObject )inStream.readObject();

      //  Display what we retrieved:
      System.out.println( serialObject );
      inStream.close();
    }
    catch(Exception e) {}
  }
}
```

File Sınıfının İncelenmesi:
Java'da dosya işlemlerini ele alabileceğimiz "File" isimli bir sınıf nesnesi daha vardır. Bu nesne yardımıyla aslında yine bir soyutlama yapılmkata ve böylece disk üzerindeki dosya ya da dizinler platform bağımsız olarak kullanıbilmektedir.

Önceki örneklerimizde dosyaları hep birer String ile ifade etmiştik. Burada bir dosyanın "File" nesnesi haline getirildikten sonra da bazı metdolar yardımıyla nasıl kullanılabileceğine dair bir örnek yapacağız. Örneğimizi incelemeden önce bir File nesnesine ait bazı metodları inceleyelim:

boolean canRead(): Nesnenin ifade ettiği dosyanın okunabilirliğini kontrol eder.
boolean canWrite(): Nesnenin ifade ettği dosyanın yazılabilirliğini kontrol eder.
boolean createNewFile(): Nesneye verilen path içerisinde boş bir dosya yaratır. Ancak bu dosyanın mevcut dizin içerisinde önceden yaratılmamış olması gerekir.
boolean delete(): Nesnenin işaret ettği path içerisindeki dosyayı siler.
boolean exists(): Verilen path içerisinde yer alan dosya ya da dizinin var olup olmadığına bakar.
String getAbsolutePath(): Nesnenin ifade ettiği dosya ya da dizinin tam yol bilgisini string olarak verir.

String [] list(): Nesnenin işaret ettiği dizin içerisindeki dosya ya da dizinlerin bir listesini verir. Bu liste de String türünden bir dizi olarak oluşturulur.

boolean mkdir(): Verilen path içerisinde yeni bir dizin yaratır.
boolean renameTo(File hedef): Nesnenin işaret ettiği dizin ya da dosyayının ismini parametre ile verilen yeni isme dönüştürür.

Tabi ki aslında "File" nesnesi yukarıda özetlenenden daha fazla hazır metoda sahiptir. Ancak burada genel bilgi vermek amacıyla bu kadarından bahsetmeyi yeterli görüyorum. Daha önce de söylediğim gibi herhangi bir java sınıfının içerdiği değişken ya da metodları bulabileceğiniz dökümantasyon sayfasına (1.4.2 için) http://java.sun.com/j2se/1.4.2/docs/api/index.html adresinden erişebilirsiniz.

Şimdi File sınıfını kullanarak yazacağımız bir örnekle konuyu daha iyi pekiştirelim:

Odev:

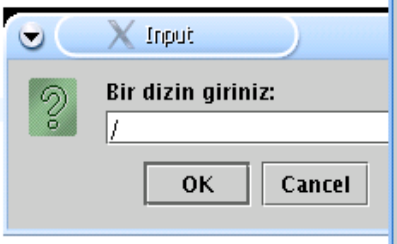
İçerisinde

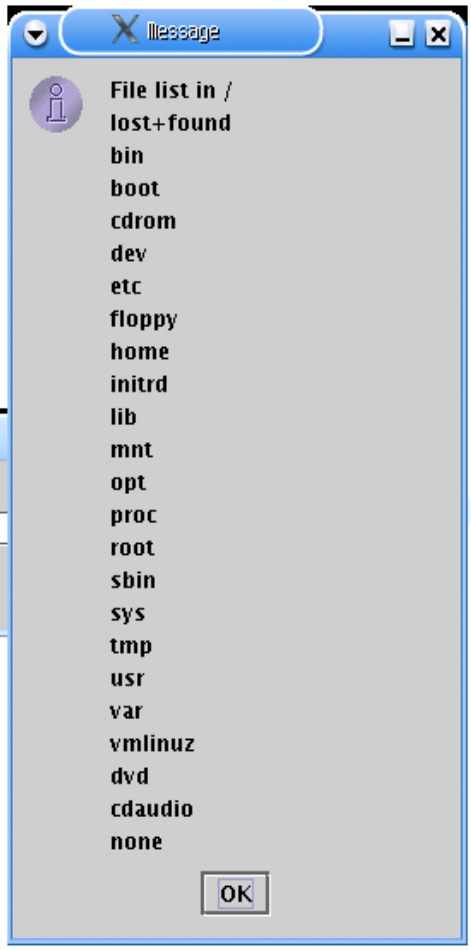
Name:Surname:Address:ID
Academy:tech:26:Address:1302

seklinde bilgiler olan bir dosyadan bu bilgileri cekerek Person nesnesi haline getiren ve bu personlari bir dizide saklayan bir uygulama yaziniz. Daha sonra bu diziyi sort ederek (Person nesnesinin compareTo metodu olmali) ekrana basiniz. Ve en son olarak bu bilgileri nesneler halinde Object Serialization yontemi yardimiyla baksa bir dosyada saklayiniz.

Arastirma konusu: Dosyadan Turkce karakter okuma (ISO8859-9 formati, dosyalarin iso8859-9 formatinda acilmasi)

```
1  import java.io.*;
2  import javax.swing.*;
3
4  class Dosyalar6 {
5
6    public static void main(String arg[]) throws IOException {
7
8          String dir = JOptionPane.showInputDialog(null, "Bir dizin giriniz: ");
9
10         File file = new File(dir);
11         String dirList[] = file.list();
12
13         String result = "File list in " + dir + "\n";
14         for(int i = 0; i < dirList.length; i++)
15                 result += dirList[i] + "\n";
16
17         JOptionPane.showMessageDialog(null, result);
18         System.exit(0);
19   }
20 }
21
22
23
24
```





Örneğimizde kullanıcıdan 10. satırda alınan bir dizin bilgisi "File" nesnesi halinde yaratılmaktadır. Böylece program içerisinde File sınıfının sunacağı hazır metodlardan yararlanılabilir. Zaten 11. satırda bu metodlardan birisi olan list metodu yardımıyla verilen dizin içerisindeki dosyalar String dizisi halinde elde edilir.

Bu noktaya kadar dosya işlemlerini içeren başlıkların çoğunu basit ve anlaşılır örneklerle incelemiş olduk. Bunu ötesinde daha büyük uygulamalarda bu küçük parçalar kullanılarak konu daha iyi pekiştirilecektir.

Dersimizin bundan sonraki kısmında Java programlama dilinin en güçlü yanlarından birisi olan grafik arayüz geliştirmeyi öğreneceğiz.

Java ile Programlama
Bölüm 10:

Java ile Grafik Arayüz Geliştirme:
Grafik arayüz denilen şey aslında arka planda çalışan programların daha anlaşılır olabilmesi amacıyla, ön tarafta kullanıcıya sunulan ortamdır. Aslında bu tür ortamlarla zaten günlük bilgisayar kullanımında sıklıkla karşılaşırsınız. Örneğin internette gezinebilmek amacıyla kullandığımız Netscape ya da Mozilla gibi web tarayıcılar birer grafik araryüzdür. Gerçekte bilgisayarımız web siteleriyle, arka tarafta çeşitli protokoller ve özel komutlar yardımıyla iletişim kurar ve sayfaları alır ya da gönderir. Ancak ön planda çeşitli text alanları, düğmeler ve menüler yardımıyla bu işlemleri daha anlaşılır şekilde yapabiliriz.

Artık masaüstü kullanılan işletim sistemlerinin tamamı grafik arayüzleri ile donatılmıştır. Bir programın giriş çıkış bilgilerine ilişkin teknik detayların kullanıcıdan soyutlanması ve ona daha anlaşılır bir etkileşim ortamının sunulması gerekir. Bu nedenle, yazılan programlar en son aşamada bir grafik arayüzü ile desteklenir ve böylece daha kullanışlı hale gelir.

Java'da grafik arayüz geliştirmek önceden yazılmış birtakım sınıfların kullanılması ile sağlanır. Bu sınıflar en genel anlamda javax.swing ve java.awt paketleri içerisinde toplanmıştır.

Öncelikli olarak bilinmesi gereken birtakım temel kavramlar incelendikten sonra sayısız grafik arayüz nesnesine ilişkin örnekler göreceğiz. Bu örneklerin tamamında temel kavramları kullanacağız.

JFrame Kavramı:
Geliştireceğimiz tüm grafik arayüz uygulamaları (GUI) aslında JFrame sınıfından türetilmiş uygulamalar olacaktır. JFrame sınıfı aslında işletim sisteminden en temel pencere fonksiyonlarını alacak şekilde oluşturulmuştur. Kullandığınız işletim sistemine göre bu fonksiyonlar ve bunların ekrana getirilmesi işlemleri farklılık göstereceğinden siz sadece JFrame sınıfını türeterek kendinizi bu farklılıktan soyutlarsınız. Zaten ilk derslerimizde de bahsettiğimiz bu teknik nesne yönelimli program geliştirmenin avantajlarından birisi olarak karşımıza çıkmaktadır. Aşağıdaki örnekte çok basit bir JFrame nesnesi ile grafik uygulamalarına başlayabiliriz.

```
import javax.swing.*;
import java.awt.*;

public class Cerceve extends JFrame {


    public Cerceve() {

        super("Benim ilk uygulamam");

        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.setSize(300, 400);
        this.setResizable(false);
        this.show();
    }

    public void paint(Graphics g) {

        g.setColor(new Color(200, 20, 208));
        g.drawString("Merhaba ", 150, 200);
        g.drawLine(0, 0, 300, 300);
        g.setColor(Color.BLACK);
        g.drawRect(149, 188, 50, 20);
        g.setColor(Color.CYAN);
        g.drawOval(30, 30, 100, 150);
        g.fillOval(30, 30, 80, 120);

    }
```

```
    public static void main(String[] args) {

        new Cerceve();
    }
}
```

Yukarıdaki örnekte oldukça basit bir GUI uygulaması görmekteyiz. Daha önce de söylediğimiz gibi bu uygulama önceden hazır olan JFrame sınıfının türetilmesi ile elde edilmektedir. Şimdi bu uygulamayı detaylı olarak ele alalım ve örnek üzerinde temel kavramlara değinelim.

Türetilen sınıfımızın başlangıç fonksiyonu içerisinde kullandığımız "super" metodu önceden hatırlayacağımız gibi temel sınıfın başlangıç metodunu çağırmaktadır. Bu metod kendisine gönderilen bir string ifadeyi ekranda çalışan frame in başlığı haline getirmektedir.

JFrame Metodları:
JFrame nesnesinden türeyen bir sınıf JFrame'e ait olan hazır metodları kullanabilir. Yine uygulamamızın başlangıç metodunda kullandığımız "setSize", "setResizable" ve "show" metodları bu tür metodlardandır. "setSize" metodu ekrana gelecek olan arayüzün satır ve sütün uzunluğunu oluşturur. Bu bilgiler piksel olarak metoda verilmektedir. "setResizable" metodu ekrana gelen arayüzün fare yardımıyla genişletilip genişletilemeyeceğini belirlemek için kullanılır. Parametre olarak verilen "false" değeri arayüzün sınırlarını sabit hale getirmektedir. En son satırda kullanılan "show" metodu arayüzümüzün çalıştıktan sonra ekranda görünür olmasını sağlar.

32 satırda uygulamamız türünden bir nesne yarattığımızda başlangıç fonksiyonu otomatik olarak çağrılacak ve gerekli boyut, listener ve diğer ayarlar yapıldıktan sonra show metodu sayesinde çerçevemiz ekranda görünür halde olacaktır.

24. satırda tanıtılmış olan paint metodu çerçevemiz oluşturulurken otomatik olarak çağrılan bir metoddur. Bu metod her zaman bir "Graphics" nesnesini parametre olarak alır. Graphics nesnesine ait olan metodlar sayesinde uygulamamız üzerine birtakım çizimler ve yazılar yerleştirebiliriz. 26. satırdaki metod sayesinde çizilecek ya da yazılacak bilgilerin hangi renkte olacağı belirlenir. Bu renk belirleme işlemi aslında setColor isimli metoda gönderilen bir "Color" nesnesi ile belirlenir. Hemen o anda yarattığımız Color nesnesinin başlangıç metodunda da 3 tane tam sayı girlmektedir. Bu sayılar sırasıyla kırmızı yeşil ve mavi bileşenlerini temsil eder. Her bileşen 0-255 arasında bir değer alır ve istedğimiz renk bu 3 bileşenin karışımından elde edilir. 27. satırdaki "drawString" metodu ile uygulamamız üzerine istenilen bir yazıyı yazabiliriz. drawString metoduna verilen diğer 2 parametre ise bu yazının hangi piksellerden başlayarak yazılacağını belirler.

Swing ve AWT
Java'da GUI nesnelerinin geliştirebilmek için kullanılan paketler swing ve awt paketleridir. Aslında swing paketi awt'den daha sonra geliştirilmiş ve nesneler swing paketi içerisinde hem artmış hem de daha küçülmüştür Yani awt paketinin geliştirilmiş hali swing paketididir. Ancak yine de event handling mekanizmasi için awt paketi hala kullanılmaktadır.

GUI Uygulamalarına Detaylı Bir Bakış
Önceki dersimizde hazırlamış olduğumuz "JavaGui" örneğinde, GUI üzerinde yapılan bir takım hareketlerin listener nesneleri tarafından yakalandığından bahsetmiştik. Hatta GUI uygulamamızda pencerenin "çarpı" düğmesine bastığımızda uygulamanın gerçek anlamda sonlanması, bu hareketin yakalanması ve bizim istediğimiz şekilde ele alınması sonucunda mümkün olmuştu.

Listener Kavramı
GUI uygulamalarında "listener" kavramı çok önemli bir yer tutar. Bu önemi GUI uygulamalarının yazılış nedenine bakarak rahatlıkla anlayabiliriz. Öyle ki; bir GUI uygulaması yazılmış bir programın kullanıcı tarafından daha rahat

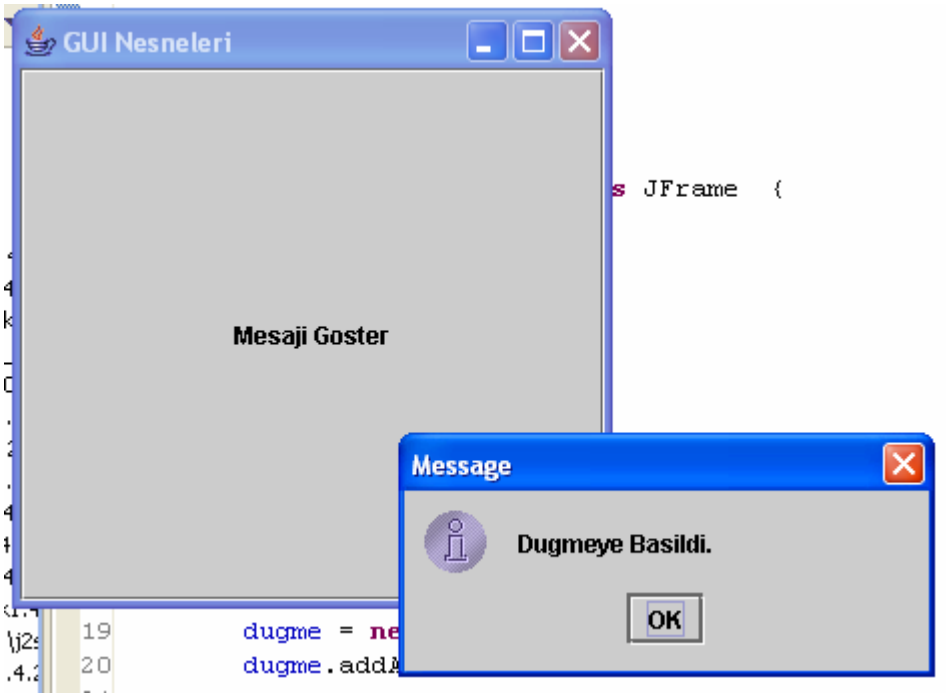
kullanılabilmesi amacı ile yazılmaktadır. O halde kullanıcının hizmetine sunulan bir GUI uygulaması kullanıcının isteklerini de algıyabilmelidir.

GUI uygulamaları, genel olarak bir takım hazır nesnelerin (Düğme, ScrollBar, Text Alanları v.b.) uygulama üzerine yerleştirilmesi şeklinde hazırlanır. İşte bu uygulmaların ikinci kısmını da bu nesneler üzerinde yapılan hareketlerin algılanması oluşturur.

Aşağıdaki örneğimizde basit bir GUI çerçevesi üzerinde bir Jbutton nesnesi eklenmekte ve bu nesnenin harektleri takip edilecek şekilde uygulama tasarlanmaktadır:

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.event.*;

public class SwingOrnek1 extends JFrame  {

	Container c;
	JButton dugme;
	Dinleyici dinleyici;

	public SwingOrnek1() {

		super("GUI Nesneleri");

		c = this.getContentPane();
		dinleyici = new Dinleyici();

		dugme = new JButton("Mesaji Goster");
		dugme.addActionListener(dinleyici);

		c.add(dugme);

		this.addWindowListener(new WindowAdapter() {
			public void WindowClosing(WindowEvent e) {
				System.exit(0);
			}
		});

		this.setSize(300, 300);
		this.show();
	}
}

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Dinleyici implements ActionListener {
	public void actionPerformed(ActionEvent e) {

		if(e.getActionCommand() == "Mesaji Goster") {
			JOptionPane.showMessageDialog(null, "Dugmeye Basildi.");
		}
	}
}

public class Monitor {

	public static void main(String[] args) {
		new SwingOrnek1();
	}
```

}



Örneğimiz 3 dosyadan oluşmaktadır. Bu dosyalardan "Monitor.java" dosyası aslında kendi başına bir işlevi olmayan ve genel olarak tüm uygulamayı çalıştırmaya yönelik bir sınıf olarak yazılmıştır.

"Dinleyici.java" sınıfı aslında orjinal adıyla "listener" olarak isimlendirilen ve ActionListener arayüzünü implement eden bir sınıftır. Bu sınıf kendisine atanan nesneleri sürekli dinleyen ve ilgili nesneye yapılan her hareketi takip eden bir sınıftır. Bu sınıf içerisinde yer alan "actionPerformed" isimli metod, bu sınıfın dinlediği herhangi bir nesneye müdahele olduğu zaman otomatik olarak çağrılmaktadır.

Aslında daha teknik olarak ifade etmek gerekirse, herhangi bir nesneye yapılan müdahele sonucunda Java'da "ActionEvent" yaratılmaktadır. Yaratılan bu "ActionEvent" ise ilgili nesneyi dinleyen listener sınıfının "actionPerformed" metoduna parametre olarak gönderilmektedir. O halde bizim örneğimizde "SwingOrnek1.java" dosyasının 20. satırında dinlenmek üzere eklenen "JButton" nesnesine dokunulduğunda bir action yaratılmakta ve bu nesneyi dinelyen "dinleyici" isimli sınıfa gönderilmektedir. Bu metoda gelen eventin hangi nesneye ait oldugu "ActionEvent" nesnesinin "getActionCommand" metoduyla anlaşılmaktadır. Bu metodun geri dönüş değeri ActionEvent halinde gelen nesnenin üzerinde yazan etikettir. O halde şunu da belirtmek gerekir ki listener tipindeki bir sınıf aynı anda birden fazla nesneyi dinleyebilir. Ya da bir başka deyişle, birbirinden farklı birden fazla nesne, aynı listener sınıfının dinlenme listesine eklenebilir. Bizim örneğimizdeki "dugme" isimli JButton nesnesi, SwingOrnek1.java dosyasında 20. satırda dinleyici sınıfının dinlenecek nesnelerinden birisi olarak eklenmektedir. Dinleyici degişkeni de yine aynı dosyadaki 10. satırda "Dinleyici" sınıfı türünden yaratılmıştır. Dinleyici sınıfı da "ActionListener" arayüzünden türetildiğine göre, bu sınıfın dinleme yeteneği vardır.

Action Nesneleri Hakkında

Genelde belirli bir menu nesnesi tarafından tetiklenen bazı komutlar bazı başka nesneler tarfından da tetiklenebilir. Mesela bir menu nesnesinde tanmlı olan bir komut bir toolbar nesnesi içerisinde de tetiklenebilir.

Kullanıcı belirli bir nesne için belirli bir fonksiyon atayabilir. Bu nesne bir menu nesnesi, bir dugme ya da bir toolbar nesnesi olabilir.

Mesela adı blueAction olan bir nesne ColorAction nesnesinde türemiş bir nesne olsun. ColorAction nesnesi de ActionListener'ı implement eden bir nesne olsun. O halde yaratılan bir düğmenin dinleyicisi olarak blueAction nesnesi kullanılabilir ve bu nesne de sadece arka plan rengini maviye dönüştürecek şekilde yazılmış olabilir.

O halde artık bu nesne birden fazla nesne tarafında listener olarak kullanılabilir.

Swing paketleri içerisinde Action denilen bir arayüz sayesinde yukarıda bahsettiğimiz bu mekanizma çok verimli bir şekilde ele alınabilir. Bu arayüz birçok listener arayüzünde tetiklenen metoduları içerir:

```
interface Action  {
        void actionPerformed(ActionEvent e);
        void setEnabled(boolean b);
        boolean isEnabled();
        void putValue(String key, Object value);
        Object getValue(Sting key);
        void addPropertyChangeListener(PropertyChangeListener listener);
        void removePropertyChangeListener(PropertyChangeListener listener);
}
```

putValue ve getValue metodları action nesnesi içerisinde anahtar değer ikililerinin tutmaya olanak verir. Bunun için Action sınıfı içerisinde tanımlı bazı sabitlerden yararlanırız.

NAME, SMALL_ICON, SHORT_DESCRIPTION, LONG_DESCRIPTION, MNEMONIC_KEY, ACCELERATOR_KEY, ACTION_COMMAND_KEY gibi...

Eğer action bir menu ya da toolbar nesnesine eklendiği zaman NAME ve SMALL_ICON otomatik olarak nesne bu değerler için ayarlanır. SHORT_DESCRIPTION değeri de nesne de tooltip olarak set edilir.

Action, bir arayüz olduğu için daha kolay bir kullanımı sağlayan AbstractAction sınıfı kullnılabilir. Bu sınıf içerisinde tüm metodlar gerektiği şekilde implement edilmiştir. Ancak sadece actionPerformed metodu soyut olarak bırakılmıştır.
Mesela yeni bir action tanımlamlayalım:

```
public class ColorAction extends AbstractAction {

      public ColorAction(String name, Icon icon, Color c) {

            putValue(Action.NAME, name);
            putValue(Action.SMALL_ICON, icon);
            putValue("color", c);
            putValue(Action.SHORT_DESCRIPTION, "Set panel color to " +
name.toLowerCase);
       }

      public void actionPerformed(ActionEvent e) {
            Color c = (Color)getValue("color");
            setBackground(c);
      }
}
```

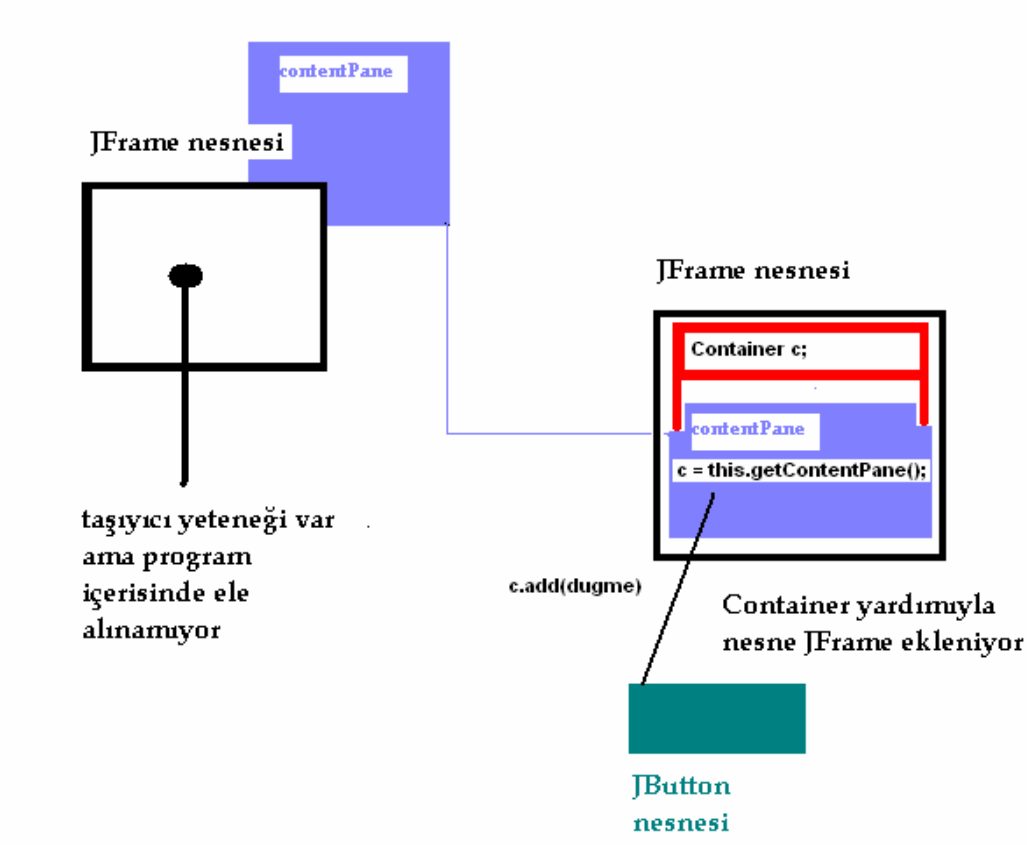
Bundan sonra bu action istenilen nesneler için kullanılabilir:

```
Action blueAction = new ColorAction("Blue", new ImageIcon("blue.gif"), Color.BLUE);
JButton blueButton = new JButton(blueAction);
```

Container Kavramı

Şu ana kadar aslında bir GUI nesnesinin nasıl dinlendiğini öğrendik. Peki ama bir GUI nesnesi nasıl çerçeve içerisine eklenebilir? Bu işlem örneğimizin 16. satırında gerçekleşmektedir. GUI nesnelerinin bazıları diğer nesneleri üzerinde taşıma yeteneğine sahiptir. Bu yeteneğe sahip olan nesnelerden birtanesinin JFrame olduğunu görmekteyiz. Öyle ki bizim örneğimiz temelde bir JFrame nesnesidir. Bu sayede üzerine bir JButton nesnesi ekleyebiliyoruz. Ancak JFrame nesnesine bu eklemeyi yapmadan önce 8. satırda olduğu gibi "Container" sınıfı türünden bir değişken yaratmaktayız. Bu değişken nesneleri tutabilen bir yapı haline gelmektedir. 16. satırdaki "getContentPane" isimli metod yazmış olduğumuz JFrame nesnesinin panelini bu taşıyıcıya atadığından, container nesnesine eklenen her

nesne dolayısı ile JFrame nesnesi üzerine eklenmiş olacaktır. Bu da bizim uygulamamızın (SwingOrnek1 sınıfı) kendisidir.
Container ve JFrame arasındaki ilişkiyi aşağıdaki şekilde irdeleyebiliriz.



NOT: Bizim uygulamamız JFrame nesnesidir. Çünkü uygulamamızı JFrame nesnesinden türetmekteyiz. Önceki dersleri hatırlayınız.

Layout Kavramı
Genel olarak bakıldığında uygulamamızda bir sorun görmekteyiz. Öyle ki "dugme" adıyla eklemiş olduğumuz JButton nesnesi bütün çerçeveyi kaplamaktadır. Bu görüntü alışılmış bir düğme görüntüsünden farklıdır. Bunun sebebi bu tür nesneleri taşımak üzere yarattığımız Container nesnesinin "layout" modelini belirtmemiş olmamızdır.

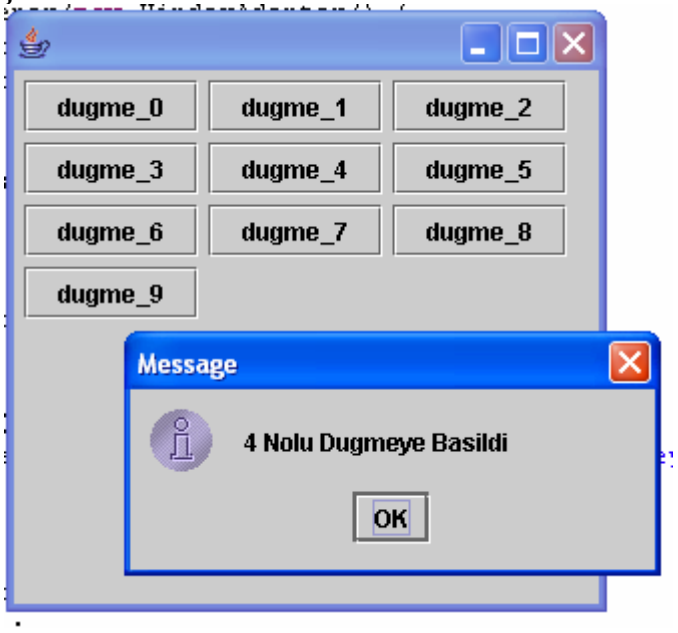
Layout denilen kavram taşıyıcı görevi üstlenen bir nesnenin kendisine eklenen diğer nesneleri hangi düzende yerleştireceğinin belirleyen bir kavramdır.

Layout modellerinden ilki olarak FlowLayout modelini inceleyebiliriz. Bu taşıyıcı nesneye (yani container nesnesine) eklenen her nesne (JButton ve daha sonra öğreneceğimiz diğer Swing nesneleri gibi) arka arkaya ve satır satır eklenmektedir. Çerçeve üzerinde bir satır bittikten sonra eklemeye alt satırdan devam edilir.

Aşağıdaki örnekte FlowLayout modeli kullanılarak nesneler eklenmektedir. Ayrıca bu örnekte listener sınıfı için yeni ve ayrı bir sınıf yaratmak yerine çerçevenin kendisi aynı zamanda Listener olarak kullanılmaktadır. Bu nedenle addActionListener metoduna eklenecek sınıf olarak "this" yani sınıfın kendisi verilmektedir.

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.event.*;

public class LayoutDeneme1 extends JFrame implements ActionListener {
```

```
    Container c;
    JButton dugmeler[] = new JButton[10];

    public LayoutDeneme1() {

        c = 0this.getContentPane();
        c.setLayout(new FlowLayout(FlowLayout.LEFT));

        for(int i = 0; i < dugmeler.length; i++) {
            dugmeler[i] = new JButton("dugme_"+i);
            dugmeler[i].setSize(10,15);
            dugmeler[i].addActionListener(this);
            c.add(dugmeler[i]);
        }

        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        this.show();
    }

    public void actionPerformed(ActionEvent e) {

        for(int i = 0; i < dugmeler.length; i++)
            if(e.getSource() == dugmeler[i]) {
                JOptionPane.showMessageDialog(this, i + " Nolu Dugmeye Basildi");
            }
    }

    public static void main(String arg[]) {
        new LayoutDeneme1();

  }
}
```



"LayoutDeneme1" sınıfı, bildiriminden de anlaşıldığı gibi hem bir JFrame nesnesi, hem bir listener hem de kendi başına çalışabilen bir uygulama olarak tasarlanmıştır. Uygulama üzerine yerleştirilecek JButton nesneleri bir JButton dizisi içerisinde saklanmaktadır. Bu dizi 9. satırda yaratılmaktadır. 17 satırda ise for döngüsünün her adımında dizinin ilgili elemanı JButton nesnesi olarak yaratılmakta ve yaratılırken de otomatik bir isim verilmektedir.

18. satırda JButton nesnesine ait olan "setSize" isimli bir metod sayesinde her JButton nesnesi sabit bir büyüklükte yaratılmakta ve böylece daha doğal bir görüntü sağlanmaktadır. Aynı döngü içerisinde ilgili nesne yaratılır

yaratılmaz ActionListener görevini üstlenen sınıfa, yani bu örnekte sınıfın kendisine eklenmektedir. Sınıfın kendisi ActionListener olduğu için de sınıf içerisine ActionListener arayüzüne sahip olan "actionPerformed" metodunun mutlaka yazılması gerekir

(Arayüzlerde belirtilen metod prototiplerinin, bu arayüzleri implement eden sınıflarda yazılmak zorunda olduğunu hatırlayınız).
Bu işlem 32. satırda yapılmaktadır.

20. satırda yaratılan her JButton nesnesi Container olarak yaratılan c nesnesine eklenmektedir. Ancak 14. satırda görüldüğü gibi c nesnesinin layout yapısı "FlowLayout" olduğundan eklenen her JButton nesnesi sırayla ve arka arkaya eklenmekltedir. Bu sonuç, program çalıştırıldığında yukarıda şekilde olduğu gibi açıkça görülmektedir. Ayrıca FlowLayout bildirimi yapılırken kullanılan LEFT sabit değişkeni ekleme sırasının her satır için soldan sağa doğru olacağını belirler. Bunun dışında RIGHT ya da CENTER da kullanılabilir.

Bu örnekte birden fazla aynı türe ilişkin nesnelerin bir dizi içerisinde otomatik olarak nasıl yaratıldığını 16. satırda görmekteyiz. Genel olarak bu sıra şu şekilde listelenebilir:
Dizinin her adımı için:

a) Nesneyi yarat
b) Nesneyi özelleştir (Boyut, renk vb.)
c) Nesneyi listener sınıfına kaydet
d) Nesneyi conatiner nesnesine ekle.

32. satırda da bu şekilde dizide saklanan nesnelerin nasıl kullanıldığını görmekteyiz. Döngünün her adımında ilgili nesnenin otomatik olarak verilmiş adı yardımıyla mesaj penceresi hazırlanmaktadır. Ancak bu sefer üzerinde işlem yapılan nesne, nesnenin üzerindeki isme bakılarak değil, gerçekten kendisine bakılarak tespit edilmektedir. Bunu sağlayan metod ActionEvent sınıfının "getSource" isimli metodudur. Demek ki bir ActionEvent yaratıldığında bu nesnenin hangi nesne olduğunu ya "getActionCommand" yardımıyla nesne üzerindeki isime bakarak, ya da "getSource" metodu yardımıyla nesnenin kendisine bakarak anlayabiliriz.

Bir başka layout modeli de BorderLayout'dur. Bu modele göre taşıyıcı nesne üzerinde kuzey, güney, doğu ve batı olarak 5 tane alan vardır ve nesneler bu alanlardan birisine yerleştirilir. Aslında contenPane içerisindeki öntanımlı layout modeli de zaten budur. Eğer yukarıdaki örnekte layout modeli BorderLayout olarak değiştirilirse nesneler buna göre istenen bölgeye yerleştirilebilir. Gerekli değişiklerle birlikte az önceki örneğin yeni hali aşağıdadır:

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import javax.swing.event.*;

public class LayoutDeneme2 extends JFrame implements ActionListener {

      Container c;
      JButton dugmeler[] = {new JButton("dugme_0"), new JButton("dugme_1"),
                                    new JButton("dugme_2"), new JButton("dugme_3"),
                                    new JButton("dugme_4")};

      public LayoutDeneme2() {

            c = getContentPane();

            c.add(dugmeler[0], BorderLayout.WEST);
            c.add(dugmeler[1], BorderLayout.EAST);
            c.add(dugmeler[2], BorderLayout.SOUTH);
            c.add(dugmeler[3], BorderLayout.NORTH);
            c.add(dugmeler[4], BorderLayout.CENTER);

            for(int i = 0; i < dugmeler.length; i++)
                  dugmeler[i].addActionListener(this);
```

```
            this.addWindowListener(new WindowAdapter() {
                public void WindowClosing(WindowEvent e) {
                    System.exit(0);
                }
            });
            this.setSize(300, 300);
            this.show();
        }

        public void actionPerformed(ActionEvent e) {

            for(int i = 0; i < dugmeler.length; i++)
                if(e.getSource() == dugmeler[i]) {
                    JOptionPane.showMessageDialog(this, i + " Nolu Dugmeye Basildi");
                }
        }

        public static void main(String arg[]) {
            new LayoutDeneme2();
        }
}
```

Bu örnekte görüldüğü gibi ekleme işleminde kullandığımız add metoduna bir de eklemenin hangi bölüme yapılacağı bilgisi de verilmelidir. Buraya kadar sadece en temel kavramları vermek amacıyla Container, Layout ve Swing nesnelerini en az özellikleriyle inceledik.

Şu anda amacımız genel yapıyı ve hiyerarşiyi kavramak olduğundan bu nesnelerin kendi içindeki detaylara şimdi girmiyoruz. Böylece en azından örneklerimizi anlayacak kadar temel bilgileri edinmiş oluyoruz. İlerleyen konularda bu temel kavramların herbirini kendi içinde daha detaylı inceleyeceğiz.

JPanel Nesnesi

Artık genel anlamda bir JFrame nesnesi yaratmayı, bu nesneye başka nesneler eklemeyi ve eklenen bu nesneleri de takip etmeyi öğrendik. Nesnelerin JFrame üzerine eklenmesi için bir Container kullandık. Ancak bazen nesneleri direk JFrame üzerine eklemek kullanışlı olmayabilir. Bu nedenle önce bazı nesneleri kendi içinde gruplayıp tek bir nesne haline getirip bu ana nesneyi JFrame container üzerine eklemek daha anlaşılır ve daha düzenli olabilir.

JPanel kendi başına üzerinde nesne tutabilen bir yapıdır. Yani JFrame'de olduğu gibi, bir JPanel nesnesi üzerine başka bir nesne eklemek için contentPane oluşturmaya gerek yoktur. Doğrudan JPanel üzerine nesneler eklenir ve bu nesnelerin listener işlemleri tamamlanır. Daha sonra bu JPanel nesnesi artık içerisinde başka nesneleri tutan tek bir nesne gibi kullanılabilir. Aslında bu işlemleri yaparken de gerçek anlamda nesne yönelimli programcılık yaptığımızı özellikle belirtmek isterim.

Aşağıdaki örnekte bir text alanı içerisindeki yazı üzerinde renk ve boyut düzenlemeleri yapan bir uygulama geliştirilmektedir. Ancak bu düzenleme opsiyonları ayrı gruplar halinde paneller üzerinde tasarlanıp daha sonra çerçeve üzerine eklenmektedir. Uygulama çalıştıktan sonra aşağıdaki gibi bir GUI ekrana gelmektedir:

RESIM:
RESIM ADI: JpanelOrnek1Cikis.png
ALT BAŞLIK: JPanelOrnek1 Uygulamasının Çalışması

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class JPanelOrnek1 extends JFrame implements ActionListener {

   JPanel colorPanel, sizePanel, editPanel, southPanel;
   JTextArea text;
```

```
JButton yellowButton, redButton, blueButton;
JRadioButton editable, noneditable;
ButtonGroup group;
Container c;

  public JPanelOrnek1() {


      c = this.getContentPane();
      c.setLayout(new BorderLayout());

      southPanel = new JPanel();
      southPanel.setLayout(new BorderLayout());

      colorPanel = new JPanel(new FlowLayout());
      yellowButton = new JButton("Sari");
      yellowButton.addActionListener(this);
      colorPanel.add(yellowButton);
      redButton = new JButton("Kirmizi");
      redButton.addActionListener(this);
      colorPanel.add(redButton);
      blueButton = new JButton("Mavi");
      blueButton.addActionListener(this);
      colorPanel.add(blueButton);

      yellowButton.setBackground(Color.BLACK);
      yellowButton.setForeground(Color.WHITE);
      redButton.setBackground(Color.BLACK);
      redButton.setForeground(Color.WHITE);
      blueButton.setBackground(Color.BLACK);
       blueButton.setForeground(Color.WHITE);

      southPanel.add(colorPanel, BorderLayout.SOUTH);

      editPanel = new JPanel(new FlowLayout());
      group = new ButtonGroup();
      editable = new JRadioButton("Duzenlenebilir", false);
      group.add(editable);
      editable.addActionListener(this);
      editPanel.add(editable);
      noneditable = new JRadioButton("Duzenlenemez", true);
      group.add(noneditable);
      noneditable.addActionListener(this);
      editPanel.add(noneditable);

      southPanel.add(editPanel, BorderLayout.NORTH);

      text = new JTextArea();
      text.setText("Deneme Yazisi");
      text.setEditable(true);
      c.add(text, BorderLayout.CENTER);
      c.add(southPanel, BorderLayout.SOUTH);


      this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
      this.setSize(300, 300);
      this.show();
  }

  public void actionPerformed(ActionEvent e) {

      if(e.getSource() == yellowButton) {
          text.setBackground(Color.YELLOW);
```

```
            yellowButton.setBackground(Color.YELLOW);
            redButton.setBackground(Color.BLACK);
            blueButton.setBackground(Color.BLACK);
        }
        else if(e.getSource() == redButton) {
            text.setBackground(Color.RED);
            yellowButton.setBackground(Color.BLACK);
            redButton.setBackground(Color.RED);
            blueButton.setBackground(Color.BLACK);
        }
        else if(e.getSource() == blueButton) {
            text.setBackground(Color.BLUE);
            yellowButton.setBackground(Color.BLACK);
            redButton.setBackground(Color.BLACK);
            blueButton.setBackground(Color.BLUE);
        }

        if(e.getSource() == editable)
            text.setEditable(true);
        else if(e.getSource() == noneditable)
            text.setEditable(false);

        repaint();
    }


  public static void main(String[] args) {
    new JPanelOrnek1();
  }

}
```

Şimdi bu örneğimizi biraz daha derinlemesine inceleyelim. Öncekilerde olduğu gibi bu sınıf da, yine bir JFrame uygulaması ve aynı zamanda ActionListener olarak yaratılmaktadır.

Ancak bu sefer birtakım nesneler doğrudan JFrame'in Container nesnesi üzerine yerleştirilmek yerine, önce bazı paneller üzerine yerleştirilmiş ve daha sonra bu paneller JFrame üzerine eklenerek uygulama tamamlanmıştır.

Örneğimizde tüm nesneleri kendi üzerinde taşıyan JFrame nesnesinin Container'ı 18. satırda yaratılmaktadır. Daha sonra sadece renk opsiyonlarının ayarlanacağı düğmeleri barındıracak olan "colorPanel" isimli bir JPanel nesnesi yaratılmış ve 24. satır ile 33. satırlar arasında bu panelin tutacağı nesneler de tek tek yaratılarak panel üzerine eklenmiştir.

Benzer şekilde ekrana gelen text alanına yazmayı sağlayan ya da yazma opsiyonunu kaldıran seçenekleri barındıran "editPanel" isimli bir başka JPanel nesnesi ve onun tutacağı nesneler de 37. ve 46. satırlar arasında yaratılmaktadır. En son olarak üzerinde editPanel ve colorPanel panellerini tutacak olan bir başka JPanel nesnesi "southPanel" adıyla 22. satırda yaratılmış ve JFrame'in c isimli Container nesnesinin SOUTH bölgesine 54. satırda yerleştirilmiştir.

c isimli Container'ın CENTER bölgesine ise JTextArea isimli nesne 53. satırda olduğu gibi doğrudan yerleştirlmiştir.

Bunun dışında aslında uygulama genel olarak diğer nesnelerin yaratılması ve bazı özelliklerinin set edilmesi şeklinde tamamlanmaktadır. Burada karşımıza JTextArea ve JRadioButton isimli iki yeni GUI nesnesi gelmektedir.

Aslında adından da anlaşılabileceği gibi JTextArea isimli nesne bize içerisine yazı yazabileceğimiz bir text alanı sunmaktadır. Yine birçok hazır metodu sayesinde bu nesnenin bazı özelliklerini değiştirmek mümkündür.

JRadioButton nesneleri de benzer şekilde seçim düğmeleri yaratmamızı sağlar. Ancak bu nesneler yaratıldıktan sonra bir de 12. satırda yaratılan "ButtonGroup" adında bir nesne içerisine eklenmektedir. Bu ekleme, bir RadioButton seçili iken diğerinin otomatik olarak seçilmemesini sağlamaktadır. Eğer birden fazla radio button nesnesini ButtonGroup içerisine eklemezseniz, aynı anda birden fazla seçim yapabilirsiniz.

Böylece genel anlamda bir GUI uygulmasında ilk olarak hiyerarşiyi belirlemek gerektiğini ve daha sonra da bu hiyerarşiye uyarak gerekli olan GUI nesnelerini birtakım özellikleriyle yaratarak paneller üzerine eklemek gerektiğini görmekteyiz.

Seçim yapmak üzere üzerine basılan ya da seçim yapılan bu nesnelerin her biri bir ActionEvent üretmektedir. Listener'a eklenen tüm nesneler için bu ActionEvent'ler de doğrudan actionPerformed metoduna göderilmektedir. Buna göre gönderilen her ActionEvent için JTextArea nesnesinin bir özelliği değiştirilmektedir. Böylece kullanıcı asıl olarak kullandığı text alanını seçimlere göre değiştirebilme olanağına sahip olur. Bu değişimler 66. satırdaki actionPerformed metodunda ele alınmıştır.

GUI derslerimizin bundan sonra devam edecek olan kısımlarında öncelikle yukarıda bahsedilen temel kavramların kendi içinde detaylarına değinecek ve daha sonra da birçok GUI nesnesini tek tek ele alarak inceleyeceğiz. Daha sonra kapsamlı bir GUI uygulaması geliştirerek GUI derslerimizi geride bırakacağız.

Bu derste anlatılan temel bilgileri kendiniz de deneyerek test etmeli ve kavramları iyice irdelemelisiniz.

ARAŞTIRMA ÖDEVİ

Uygulama olarak; yukarıda verilen son örneğimize bir de yazı fontunun rengini değiştiren opsiyonların yeraldığı bir panelin nasıl ekleneceğini araştırabilirsiniz. Bunun için ipucu olarak JTextArea nesnesinin bir metodunu (setFont) kullanabilirsiniz. Ama ondan önce bu metodu tetikleyecek birtakım seçim düğmeleri yaratmalı ve bu düğmeleri bir panel üzerine yerleştirerek uygulamaya dahil etmelisiniz. Bu da iyi bir tekrar ve yeni araştırma gerektirmektedir. Şimdiden kolay gelsin ☺

Java ile Programlama

Java2D
Bundan önceki dersimizde arayüz tasarımına bir giriş yapmıştık. Bazı nesneleri de kullanarak genel olarak arayüz tasarımının nasıl ele alınacağını incelemiştik. Bu genel kavramları incelerken

- event handling
- container
- component

gibi kavramları da öğrenmiştik. Bu dersimizde bu kavramları kendi içinde daha detaylı incelmeye başlayacağız.

Grafik arayüz programlarında sadece nesneler ile değil bu nesneler üzerinde de yazı ve şekil gibi animasyonların da kullanıldığını görmüştük. Hatırlarsak geçen dersimizde Graphics nesnesini yardımıyla bir component içerisinde yer alan paint metodu yeniden yazılarak, ilgili componentin üzerinde istediğimiz gibi 2 boyutlu yazı ya da resimler olmasını sağlayabiliyorduk. Şimdi bu iki boyutlu animasyonlar ile daha detaylı ilgileneceğiz.

Java 2 ile birlikte gelen Graphics2D kütüphanesi ve paintComponent Metodu
Java 2 ile birlikte daha da gelişmiş olan Graphics2d isimli kütüphane bu 2 boyutlu animasyonları daha detaylı ele almamızı sağlayan metodlara sahiptir. Eğer "paintComponent" isimli metod kullanlırsa bu metoda otomatik olarak Graphics2D kütüphanesi parametre olarak tanacaktır. Bizim yapmamız gereken tek şey gelen bu Graphics2D nesnesi için bir tür dönüşümü yapmaktır.

Örnek:

```
public void paintComponent(Graphics g) {

      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D) g;

      g2.birseyler..

      ....
}
```

Graphics2D de şekiller daha nesne yönelimlidir
Graphics'D kütüphanesi içerisinde Rectangle2D, Line2D, Ellipse2D gibi şekiller daha nesne yünelimli olarak ele alınmaktadır. Örneğin yeni bir dörtgen çizmek için:

Rectangle2D rect = new Rectangle2D(...);
g2.draw(rect);

demek yeterlidir. Oysa Graphics kütüphanesinde bu çizimlerde her şekil için ayrı bir metod kullanmak gerekirdi.

Graphics2D'de birimler int pikseller değil float değerlerdir.
Graphics2D'de şekilleri ifade etmek için kullanılan boyu ve nokta bilgileri int pikseller ile ifade edilmemktedir. Bunun nedeni şekiller inch ya da milimetre gibi daha esnek birimlerle belirtme olanağı sağlamaktır. Bu nedenle birimler float değerler olarak ele alınmaktadır.

Ancak bu durumda da float değerlerin kullanılması zorunlu bir bildirim gerektirdiğinden bu değeri float olarak ele almak zorlaşmaktadır. Örneğin

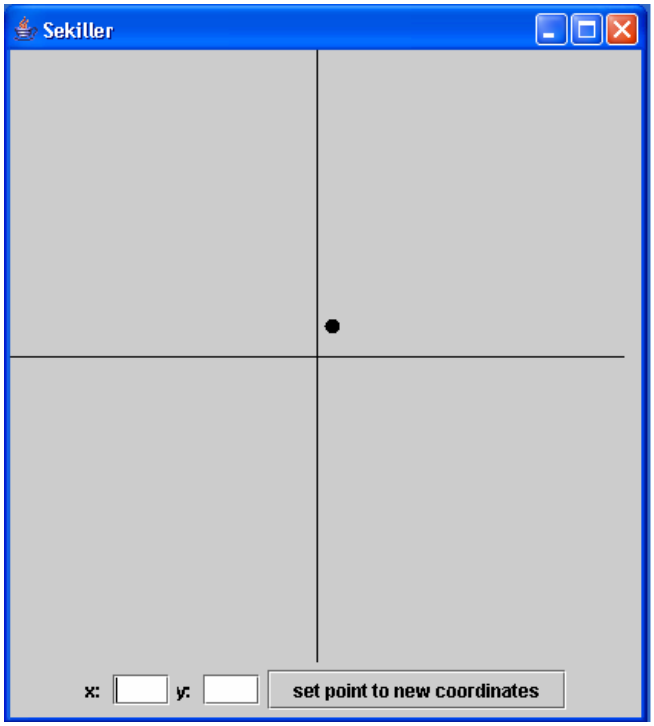
float f = 1.2 // yanlıştır. Bu değer double kabul edilir. Bunu float yapmak içi 1.2F yazılmalıdır.

Bu kargaşayı basitleştirmek amacıyla birer iç sınıf olan Rectangle2D.Double ya da Rectangle2D.Float kullanılabilir. Böylece koordinat bilgiler ister float istenirse de double türünden ele alınabilir.

NOT: İyi ama madem float değerlerde tür dönüşümü zorluk çıkarıyorsa neden sadece double değerler kullanılmıyor?
Çünkü aslında float değerler double değerlerin yarısı kadar yer kaplar ve float değerleri ile yapılan hesapmalara double değerlerden daha hzlıdır.

```
import javax.swing.*;
import java.awt.*;
import java.awt.geom.*;



public class SekilCizim extends JFrame {

	DrawPanel panel;
	Container c;
	public SekilCizim() {

		c = this.getContentPane();

		panel = new DrawPanel();
		c.add(panel);


		this.setSize(400, 400);
		this.setTitle("Sekiller");
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.show();
	}

	public static void main(String arg[]) {
		new SekilCizim();

	}

}

class DrawPanel extends JPanel {

	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		Graphics2D g2 = (Graphics2D)g;
		Rectangle2D rectangle = new Rectangle2D.Double(100, 100, 200, 150);

		Ellipse2D ellipse = new Ellipse2D.Double();
		super.setBackground(Color.BLACK);
		g2.setColor(Color.GREEN);
		ellipse.setFrame(rectangle);

		g2.draw(rectangle);
		g2.draw(ellipse);

	}
}
```

Aşağıdaki örnekte de yine Graphics2D kutuphanesi kullanılarak 2 boyutlu basit bir grafik ugulaması geliştirilmiştir:



```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
import java.awt.geom.*;

public class Eksen extends JFrame implements ActionListener {

	JTextField xpoint, ypoint;
	JButton setPointsButton;
	CoordinatePanel coordinate;
	JPanel settings;
	Container c;

	public Eksen() {

		this.prepareComponents();
		this.prepareFrame();

	}

	public void prepareFrame() {

		this.setSize(420, 470);
		this.setTitle("Sekiller");
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.show();

	}

	public void prepareComponents() {

		c = this.getContentPane();
		c.setLayout(new BorderLayout());

		coordinate = new CoordinatePanel();
```

```
            c.add(coordinate, BorderLayout.CENTER);

            xpoint = new JTextField(3);
            ypoint = new JTextField(3);
            setPointsButton = new JButton("set point to new coordinates");
            setPointsButton.addActionListener(this);
            settings = new JPanel();
            settings.setLayout(new FlowLayout());
            settings.add(new JLabel("x: "));
            settings.add(xpoint);
            settings.add(new JLabel("y: "));
            settings.add(ypoint);
            settings.add(setPointsButton);
            c.add(settings, BorderLayout.SOUTH);

      }

      public void actionPerformed(ActionEvent e) {

            if(e.getSource() == this.setPointsButton) {
                  JOptionPane.showMessageDialog(this, "dugmeye basldi");
                  coordinate.setPoint(Integer.parseInt(xpoint.getText()),
Integer.parseInt(ypoint.getText()));
                  this.repaint();
            }
      }

      public static void main(String arg[]) {

            new Eksen();
      }

}

class CoordinatePanel extends JPanel {

      int currentX, currentY;
      Ellipse2D nokta = new Ellipse2D.Double();
      int xSize = 400, ySize = 400;

      public CoordinatePanel() {

            this.setSize(this.xSize, this.ySize);
            this.currentX = 10;
            this.currentY = 20;

            this.setPoint(currentX, currentY);




      }

      public void paintComponent(Graphics g) {
            Graphics2D g2 = (Graphics2D)g;

            Line2D yline = new Line2D.Double(this.xSize/2, 0, this.xSize/2,
this.ySize);
            Line2D xline = new Line2D.Double(0, this.ySize/2, this.xSize,
this.ySize/2);

//          noktanin yeni koordinatlari relative olarak belirleniyor
```

```
            nokta.setFrame(this.xSize/2+this.currentX-5,   this.ySize/2-this.currentY-
5, 10, 10);


            g2.draw(xline); g2.draw(yline);
            g2.fill(nokta);

      }


      //Bu metod notanin yeni koordinatlarini set ediyor.
      public void setPoint(int x, int y) {
            this.currentX = x;
            this.currentY = y;
            this.repaint();
      }


}
```

SystemColor Sınıfı

Daha önce Color sınıfı yardımıyla birtakım renk nesneleri tanımlayabiliyorduk. SystemColor sınıfı içerisinde öntanımlı olarak bulunan birçok statik renk nesnesi vardır. Bu nesneler otomatik olarak kullanılan işletim sistemi nesnelerinin renklerinje göre ayarlanmıştır. Bu statik nesnelerin genel bir listesi:

- desktop
- activeCaption
- window
- windowBorder
- windowText
- menu
- menuText
- text
- textText
- textInactiveText
- textHiglight
- textHiglightText
- scrollbar
- infoText

şeklinde verilebilir. Mesela

frame.setBackground(SystemColor.desktop)

Text ve Font'lar

2 Boyutlu uygulamalar içerisinde kesinlikle ele alınması gereken nesneleri içerisinde textler ve fontlar gelmektedir. Daha önceki örneklerimizden biliyoruz ki g.drawString isimli metod yardımıyla birtakım text bilgileri component üzerine yazabiliyorduk. Ancak bu yazıda kullanılan fontu kendimiz belirleyemiyrduk.

Font Sınıfı

Java 'da tanımlı olan Font isimli sınıf sayesinde yeni bir fon belirleyebiliyoruz. Öyle ki, bu font sistemde tanımlı olan Font ailesinden birisi olacak şekilde, style ve size bilgisi ile yaratılabiliyor: Örneğin:

Font myFont = new Font("SansSerif", Font.BOLD, 12);

Font için kullnılabilecek style listesi

- Font.BOLD

- Font.ITALIC
- Font.PLAIN
- Font.BOLD + Font.ITALIC

şeklinde verilebilir.

Sistemde yer alan fon t ailelerinin isimlerini de elde etmek mümkündür. Bunun için kullanılabilecek metod GraphicsEnvironment isimli sınıfın içerisinde tanımlı olan getLocalGraphicsEnvironment isimli metodun geri dönüş değeri olan sınıfın getAvailableFontFamilyNames isimli metodu kullanmak gerekir.

```
import javax.swing.*;
import java.awt.*;


public class Fonts {

	public static void main(String[] args) {

		String str= "";
		String systemFonts[] =
GraphicsEnvironment.getLocalGraphicsEnvironment().getAvailableFontFamilyNames();



		for(int i = 0; i < systemFonts.length; i++)
			str += i+": "+systemFonts[i]+"\n";

		System.out.println(str);
	}

}
```

İmajlar

Bundan önceki kısımda 2 boyutlu basit imajların nasıl çizilebileeğini gördük. Ancak bazen sistemde yer alan fotoğraf ve resimlerden meydana gelen imajları çizmek de gerekebilir. Bu imajlar harici olarak ele alınırlar ve program içerisinde kullanılabilecek bir nesne haline getirilirler.

Öncelikle bir şekilde sistemde ya da ağ üzerinde bir imajın yeralması gerekir. Daha sonra bu nesne program içerisine Image nesnesi olarak çekilebilir. Ancak bu işlem Toolkit sınıfnın getDefaultToolkit isimli metodundan geri gelen sınıfın getImage isimli metodu ile mümkündür. Örneğin sistemimizde "sunset.jpg" isimli bir resim olsun:

```
import javax.swing.*;

import java.awt.*;

public class ImageFrame extends JFrame {
	JScrollPane scroll;
	ImagePanel imPanel;
	Container c;
	int WIDTH;
	int HEIGHT;
```

```
	public ImageFrame() {

		c = this.getContentPane();
		c.setLayout(new BorderLayout());

		imPanel = new ImagePanel();
		this.WIDTH = imPanel.getWidth();
		this.HEIGHT = imPanel.getHeight();
		c.add(imPanel);

		this.prepareFrame();
	}

	public void prepareFrame() {

		this.setSize(this.WIDTH, this.HEIGHT);
		this.setTitle("ImageFrame");
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.show();

	}

	public static void main(String[] args) {
		new ImageFrame();
	}
}

class ImagePanel extends JPanel {

	Image myImage;

	public ImagePanel() {

		myImage =
Toolkit.getDefaultToolkit().getImage("G:\\egitim_yazil_seminer\\egitimler\\"+
		"ACADEMYTECH\\javaDersleri\\derstekiOrnekler\\Sunset.jpg");

		MediaTracker tracker = new MediaTracker(this);
		tracker.addImage(myImage, 0);
		try {
			tracker.waitForID(0);
		} catch(InterruptedException e) {}

		this.setSize(new Dimension(800, 600));

	}

	public void paintComponent(Graphics g) {

		Graphics2D g2 = (Graphics2D)g;
		g2.drawImage(myImage, 0, 0, null);
	}

}
```

Araştırma Ödevi: Toolkit nesnesin ve getDefaultToolKit metodu ile geri gelen nesnenin metdolarını araştırınız.

NOT: ToolKit nesnesi yardımıyla bir frame'in ekranın ortasında görüntülenmesi:

```
Toolkit kit = Toolkit.getDefaultToolkit();
int screenX = (int)kit.getScreenSize().getWidth();
int screenY = (int)kit.getScreenSize().getHeight();
this.setLocation(screenX/2 -this.WIDTH/2, screenY/2- this.HEIGHT/2);
```

Olay Kontrolü (Event Handling)

İlk olarak olay kontrolü detaylarına bakacağız.

Daha önce temel olarak olay kontrol mekanizmasını zaten görmüştük. Herhangi bir nesnenin daha önceden belirlenen bir listener içerisine kayıt edilmesi ve bu listener içerisinde ele alınmasıyla bir nesne için exception handling yapmış oluyorduk.

Bildiğimiz gibi herhangi bir dinleyiciye kaydedilen nesne üzerinde bir olay meydana gelirse, bu olay even nesnesi içerisinde tanımlı olarak kayıt edilen dinleyiciye gönderiliyordu.

Şimdi bu layout mekanizmasını yeni bilgilerle tekrar ele alalım. aşağıdaki örnekte java'da ele alınabilen birtakım genel görüntü değişimleni sağlayan bir program yazılmaktadır. Önce örneğimizi yazalım:

```
import java.awt.*;
import javax.swing.*;
import javax.swing.plaf.basic.BasicLookAndFeel;

import java.awt.event.*;

public class PlafTest extends JFrame implements ActionListener{

        Container c;
        JButton metalButton, motifButton, windowsButton;

        public PlafTest() {

                c = this.getContentPane();
                c.setLayout(new FlowLayout());

                this.metalButton = new JButton("Metal");
                this.metalButton.addActionListener(this);
                c.add(this.metalButton);

                this.motifButton = new JButton("Motif");
                this.motifButton.addActionListener(this);
                c.add(this.motifButton);

                this.windowsButton = new JButton("Windows");
                this.windowsButton.addActionListener(this);
                c.add(this.windowsButton);

                this.setSize(400, 400);
                this.setResizable(false);
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.show();

        }


        public void actionPerformed(ActionEvent e) {

                if(e.getSource() == this.metalButton) {
```

```
                this.changeToMetal();

        }

        else if(e.getSource() == this.windowsButton) {
                this.changeToWin();

        }

        else if(e.getSource() == this.motifButton) {

                this.changeToMotif();
        }



    }

    private void changeToMetal() {

        try {

        UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
        SwingUtilities.updateComponentTreeUI(this);
        }
        catch(Exception e) {}
    }

    private void changeToWin() {

        try {

    UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
                SwingUtilities.updateComponentTreeUI(this);
                }
                catch(Exception e) {}

    }
    private void changeToMotif() {

        try {

    UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");
                SwingUtilities.updateComponentTreeUI(this);
                }
                catch(Exception e) {}

    }



    public static void main(String arg[]) {

        new PlafTest();

    }

}
```

Örneğimizde aslında temel olarak farklı bir işlem yapmıyoruz. Sadece LookAndFeel olarak anılan ve genel sistem görüntüsünü set etmeye yarayan birtakım hazır nesnleri kullanıyoruz. Bu nesneler her seferinde bu şekilde kullanıldıkları için look-and-feel konusunda daha fazla detaya girmeyeceğiz.

WindowEvent'lerin dinlenmesi

Window event, frame dediğimiz nesneler üzerinde meydana gelen olaylardır. Bir WindowEvent'i dinleyebilecek olan listener ise WindowListener nesnesidir. Bu nesne de tıpkı ActionListener gibi bir arayüzdür. O halde bir sınıfı WindowListener olarak dizayn etmek istiyorsak tıpkı ActionListener örneğinde olduğu gibi ancak bu sefer

WindwListener nesnesini implement ederek sınıfımızı WindowEvent dinleyebilecek bir dinleyici sınıf haline getirebiliriz.

bir WindowListener içerisinde aşağıdaki metodlar yeralmaktadır:

```
public interface WindowListener {

        void windowOpened(WindowEvent e);
        void windowClosing(WindowEvent e);
        void windowIconified(WindowEvent e);
        void windowActivated(WindowEvent e);
        void windowClosed(WindowEvent e);
        void windowDeiconified(WindowEvent e);
        void windowDeactivated(WindowEvent e);
}
```

Adaptör Sınıfları

İyi ama sadece tekbir olayı (mesela pencerenin kapatılması) ele alabilecek bir listener yaratmak için tüm bu metodları yazmak zorındamıyız. Aslında WindowListener bir arayüz olduğuna göre cevap evet olacaktır. İçleri boş olsa bile aslında ele almadığınız olayları da yazmak zorunda kalırsınız. Ancak javba geliştricileri bunun da bir çözümünü geliştirmişlerdir. Öyle ki sadece belirli olayları ele almak istediğiniz dinleyici sınıfları yazabilmek için önceden bu sınıfları implement eden ve gerekli tüm metodlarını boş bir şekilde yazan sınıflar tasarlanmıştır. Bu sınıflara "adaptör" sınıflar denilir.
Kullanıcı listener için arayüz kullanacağına bu arayüze ilişkin adaptör sınıf kullanır ve Polymorphism yöntemiyle sadece istediği olayları ele alan metodları yeniden yazması yeterli olacaktır.

aşağıdaki örnekte yaratılan çerçeve uygulaması bir windowListener'a kayıt edilecek ancak bu çerçevenin sadece windowClosing ve windowIconified eventleri ele alınacaktır.

```
/*WindowEvents.java*/
import javax.swing.*;

import java.awt.*;
import java.awt.event.*;

public class WindowEvents extends JFrame {

      Container c;
      PencereDinleyici dinleyici = new PencereDinleyici();

      public WindowEvents() {



            c = this.getContentPane();
            c.setLayout(new FlowLayout());

            this.addWindowListener(dinleyici);
            this.setSize(400, 400);
            this.setResizable(false);
            //this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.show();
      }

      public static void main(String arg[]) {

            new WindowEvents();
```

```
	}

}

class PencereDinleyici extends WindowAdapter {

	public void windowClosing(WindowEvent e) {
		JOptionPane.showMessageDialog(null, "Pencere kapaniyor");
		System.exit(0);
	}

	public void windowIconified(WindowEvent e) {
		JOptionPane.showMessageDialog(null, "Pencere iconfied ediliyor");
	}

}
```

Yukarıdaki örneğimizde dinleyici olarak tasarkanan sınıf WindowListener nesnesini implement edecek şekilde değil, WindowAdaptor isimli sınıfı extend edecek şekilde tasarlanmıştır. Daha önce de söylediğimiz gibi aslında WindowAdaptor isimli sınıf zaten arka planda WindowListener sınıfını implement etmiş ve bu arayüzün tüm metodlarının için boş olarak yazmıştır. Böylece biz WindowAdaptor sınıfından türetme yaparak ve polimorfizm özelliğini kullanarak sadece istediğimiz dinleyici metodlarını yeniden yazarız.

KeyEvent'lerin Dinlenmesi

Klavyeye basıldığında meydana gelen olaylar KeyEvent nesnesi şeklinde yaratılmaktadır. Bu nesnelerin dinlenmesi için kullanılabilecek dinleyici arayüzler KeyListener arayüzüdür. Ancak önceki kısımda anlattığımız gibi sadece belirli key eventler ele alındmak isteniyorsa, KeyAdaptor sınıfı da kullanılabilir. KeyListener arayüzünde yer alan metodlar

KeyListener arayüzünde yer alan metdoların listesi dökümantasyon yardımıyla elde edilebilir.

keyEvent keyReleased, keyTyped Metodları

Kullanıcı bir tuşa bastığı zaman KeyEvent isimli olay yaratılır. Benzer şekilde bir tuş bırakıldığı zaman KEY_RELEASED isimli olay tetiklenir. Bu iki olayı bele alabileceğiniz arayüz metodları keyPressed ve keyReleased isimli metodlardır. Bu ikisini birden kapsayan 3. metod ise keyTyped isimli metoddur.

VK_ terminolojisi KeyEvent sabitleri ve metodları

Herhangi bir tuşa basıldığında iki tane kavram devreye girer. 1. hangi karaktere basılmıştır? 2. si ise aslında tam olarak klavye üzerinde hangi tuşa basılmıştır. Klavye üzerindeki tuşları temsil eden saibitler VK_ ile ifade edilirler. Örneğin:

KeyEvent.VK_A -> fiziksel olarak A tuşunu temsil eder.
KeyEvent.VK_SHIFT -> fiziksel olar4ak shift tuşunu temsil eder.

Bu ifadeler keyCode adı verilir ve KeyEvent nesnesinin **getKeyCode()** metdu ile elde edilir. Bu ifaddelerin sayısal karşılıkları yerine KeyEvent.VK_A şeklinde kullanılmaları çok daha kolay olur.

Bunu yanı sıra klavyedeki control ya da shift gibi tuşların o andaki durumlarını elde etmek için yine KeyEvent nesnesinin **isShiftDown**, **isControlDown**, **isAltDown** metodlarından yararlanılabilir

Mesela

public void keyPressed(KeyEvent e) {

```
int keyCode = e.getKeyCode();
if(keyCode == KeyEvent.VK_SHIFT)
....
```

Basılan tuşun ürettiği tam karakter elde edilmek istendiğinde **getKeyChar()** metodu da kullanılabilir.

Şimdi tüm bu bilgilerimizi pekiştirmek amacıyla aşağıdaki örneği deneyelim:

```
/* SketchPanel.java*/
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.util.ArrayList;

import javax.swing.JPanel;

class SketchPanel extends JPanel {

	Point2D last;
	ArrayList lines;


	public SketchPanel() {

		last = new Point2D.Double(100,100);
		lines = new ArrayList();
		KeyHandler listener = new KeyHandler();

		this.addKeyListener(listener);
	}

	public void paintComponent(Graphics g) {

		super.paintComponent(g);
		Graphics2D g2 = (Graphics2D)g;


		//listedeki tum cizgileri ciziliyor
		for(int i = 0; i < lines.size(); i++)
			g2.draw((Line2D)lines.get(i));

	}

	public boolean isFocusTraversable() {
		return true;
	}

	public void add(int dx, int dy) {
		//last olan noktaya dx dy uzakliginda olan yeni noktayi hesapliyor
		Point2D end = new Point2D.Double(last.getX() + dx, last.getY() + dy);

		//son noktadan yeni noktaya kadar olan cizgi hazirlaniyor
		Line2D line = new Line2D.Double(last, end);

		//bu cizgi diger cizgilerin listesine kayit ediliyor
		lines.add(line);
		repaint();
```

```
            last = end;
      }

      private class KeyHandler implements KeyListener {

            public static final int LARGE_INCREMENT = 5;
            public static final int SMALL_INCREMENT = 1;

            public void keyPressed(KeyEvent e) {

                  int keyCode = e.getKeyCode();

                  //uzakligi ayarliyoruz
                  int d;
                  if(e.isShiftDown())
                        d = LARGE_INCREMENT;
                  else
                        d = SMALL_INCREMENT;

                  if(keyCode == KeyEvent.VK_RIGHT)
                        add(d, 0);
                  else if(keyCode == KeyEvent.VK_LEFT)
                        add(-d, 0);
                  else if(keyCode == KeyEvent.VK_UP)
                        add(0, -d);
                  else if(keyCode == KeyEvent.VK_DOWN)
                        add(0, d);
            }

            public void keyReleased(KeyEvent e) {


            }

            public void keyTyped(KeyEvent e) {

                  char keyChar = e.getKeyChar();

                  //uzakligi ayarliyoruz
                  int d;
                  if(Character.isUpperCase(keyChar)) {
                        d = this.LARGE_INCREMENT;
                        keyChar = Character.toLowerCase(keyChar);
                  }

                  else
                        d = this.SMALL_INCREMENT;

            }
      }
}
```

```
/*SketchFrame.java*/
import javax.swing.*;

import java.awt.event.*;
import java.awt.*;
import java.awt.geom.*;
import java.util.*;
```

```
public class SketchFrame extends JFrame {

	Container c;


	public SketchFrame() {

		c = this.getContentPane();

		SketchPanel panel = new SketchPanel();
		c.add(panel);

		this.setTitle("Sketch");
		this.setSize(400, 400);
		this.setResizable(false);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.show();
	}


	public static void main(String[] args) {
		new SketchFrame();
	}
}
```

Yukarıdaki programın tam olarak incelenmesinden önce focus kavramını irdelemekte fayda var.

Focus Hakkında (isFocusTraversable metodu)
Normalde bilgisayar üzerinde mouse yardımıyla herhangi bir pencere ya da nesne seçilerek aktif hale getirilebilir. Herhangi bir pencere aktif iken klavyeden girilen giriş bilgileri doğrudan aktif olan nesneye gönderilmektedir.

Bir java programı çalışırken java'ya ait bir pencere aktif olacaktır. İşte bu durumda klavyeden gönderilen bir giriş değeri doğrudan bir component üzerine gönderilecektir. Girişin gönderildiği bu nesneye focus denilir.

Bir pencere içerisinde en az bir nesne focus'tur.

Java penereleri içerisinde de focus lar belirli bir sırayala nesneden nesneye geçer. Ancak bazı nesneler hiçbir zaman focus olmazlar. Paneller bu tip nesnelerdendir. İşte prgramımızda override edilen isFocusTraversable metodu, panelimizin focus olmasini sağlar. Böylece klavye inputlari artik panele gönderilebilecektir.

MouseEvent'leri Ele Alma
Normalde mouse ile başka bir component tetiklenirse MouseEvent'i yakalamay gerek yoktur. Ancak çeşitli paneller ya da başka compoentler üzerinde mouse ile çizim v.b. işler yapılacaksa o zaman fare tarafından tetiklenen olaylar ele alınabilir.

Kullanıcı bir fare tuşuna bastığı zaman 3 tane listener metodu tetiklenir:
- **mousePressed(MouseEvent e)**
- **mouseReleased(MouseEvent e)**
- **mouseClicked(MouseEvent e)**

MouseEvent nesnesinin **getX** ve **getY** metodları kullanılarak mouse göstericisinin tam olarak nereye bastığı bilgisi elde edilebilir. Ayırca tek tıklama ve çift tıklama arasındaki ayrım da getClickCount metodu yardımıyla elde edilebilir.

```
public void mouseClicked(MouseEvent e) {

	int x = e.getX();
	int y = e.getY();

	if(e.isShiftDown && e.getClickCount() > 2) {
```

```
            Graphics g = getGraphics();
            g.drawString("Selam", x, y);
            g.dispose();

      }

}
```

Yukarıdaki örnekte mouse hareketlerinin dinlemenin yanı sıra graphics nesnesinin elde edilmesine ilişkin bir kullanım vardır. Burada Graphics nesnesini paintComponent metoduna gelen parametre ile elde etmek yerine kendimiz yaratıyoruz. Bunun nedeni mouse ile panele tıkladığımzıda paintCompnent nesnesinn çağrılmasını beklemeden işlemi hemen yapmak istememizdir. Bu şekilde kendimiz elde ettiğimiz graphics nesneleerin dispose metodu ile yeniden çizeriz. Ancak parametre ile gelen Graphics nesneleri repaint metodu ile çizilmelidir.

Şimdi tüm mouse eventleri ele aln bir örnek yazalım..

```
/*MouseTracker.java*/
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;


public class MouseTracker extends JFrame implements MouseListener,
MouseMotionListener {

      private JLabel statusBar;

      public MouseTracker() {
            super("Mouse olaylarini ele aliyoruz");
            statusBar = new JLabel();

            this.getContentPane().add(statusBar, BorderLayout.SOUTH);
            this.addMouseListener(this);
            this.addMouseMotionListener(this);

            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            this.setSize(400, 400);
            this.show();
      }

      public void mouseClicked(MouseEvent arg0) {
            statusBar.setText("Clicked at [" + arg0.getX() + ", " + arg0.getY() +
"]");
      }


      public void mouseEntered(MouseEvent arg0) {
            statusBar.setText("Mouse is in window");

      }


      public void mouseExited(MouseEvent arg0) {
            statusBar.setText("Mouse is not in window");
      }


      public void mousePressed(MouseEvent arg0) {
            statusBar.setText("Pressed at [" + arg0.getX() + ", " + arg0.getY() +
```

```
"]");
	}


	public void mouseReleased(MouseEvent arg0) {
		statusBar.setText("Released at [" + arg0.getX() + ", " + arg0.getY() +
"]");
	}


	public void mouseDragged(MouseEvent arg0) {
		statusBar.setText("Dragged at [" + arg0.getX() + ", " + arg0.getY() +
"]");
	}


	public void mouseMoved(MouseEvent arg0) {
		statusBar.setText("Moved at [" + arg0.getX() + ", " + arg0.getY() + "]");
	}

	public static void main(String[] args) {
		new MouseTracker();
	}
}
```

ARAŞTIRMA ÖDEVİ: Yukarıdaki uygulamaya ek olarak bir de panele çizgi çizen bir pencere-mouse uygulaması yazınız. Yani, kullanıcı mouse ile drag işlemi yaptıkça ekranda çizgi belirsin.

İpucu: üzeri çizilmek üzere bir panel yaratılsın. Bu panel MouseMotionAdapter ile dinlensin. Panelin paint metodu currentX ve currentY denilen birtakım noktalar üzerinde fillOval metodunu çağırsın...

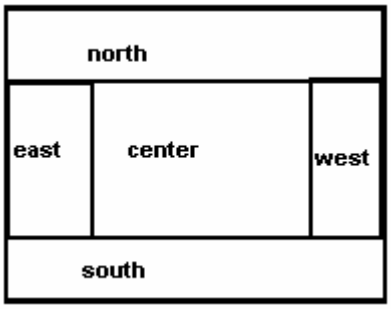
Ders 17

Geçen dersimizde GUI nin en önemli prensiplerinden birisi olan EventHandling mekanizmasını öğrenmiştik.

Layout Mekanizması
Bu dersimizde GUI nesnelerini çerçeveler üzerinde yerleştirebilmemiz için gerekli olan ve daha önce sadece özet şeklinde incelediğimiz LayoutManagement mekanizmasını biraz daha detaylı bir şekilde inceleyeceğiz.

Daha önce de söylediğimiz gibi Layout mekanizması GUI nesnelerinin çerçeve ya da panel üzerinde hangi sırayla ya da hangi hiyerarşi ile yerleştirileceğine dair bilgi vermektedir.

BorderLayout
Daha önce de bildiğimiz gibi temel bir layout mekanizması olan border layout nesneleri taşıyıcı nesneler üzerinde 5 ayrı parçada yerleştirebilmeyi sağlar.



BorderLayout JFrame için öntanımlı olan layout dur

BorderLayout'da nesnelerin pozisyonları
Aslında ilk başta Center bölgesi bütün paneli kapsar. Çerçevenin boyutu değiştikçe Center bölgesi de boyutunu değiştirir. Ancak kenar bölgeler boyutlarını değiştirmezler. Kenar bölgelerin sınırı da üzerlerine nesne eklendikçe değişir.

FlowLayout'dan farklı olarak, BorderLayout nesnelerin ilgili bölgenin tamamamını kapsamasına izin verir. Ancak FlowLayout ilgili nesneleri onların "preferred size" değerlerinde yerleşmesini sağlar.

Nesneler BorderLayout ile ayarlanmış bir panel ya da çerçeve üzerine farklı bölgelere yerleştirilebilirler:

```
Container c = getContentPane();
c.add(nesne, BorderLayout.SOUTH);
```

GridLayout

GridLayout nesnelerin yerleşme düzenini satır ve sütünlar üzerinde ele alır.

Ancak GridLayout kullanıldığı zaman her hücrenin kapladığı alan hep aynı büyüklükte olacakıtır.

GridLayout ile panel düzeni set edilirken bölünmek istenen satır ve sütün sayısı parametre olarak belirtilir:

```
panel.setLayout(new GridLayout(3, 4));
```
3 satır ve 4sütunluk bir layout mekanizması..

Box Layout

Nesnelerin tek satır ya da tek sütun içerisinde yerleştirilmesini GridLayout'dan daha esnek bir şekilde ele alan bir modeldir.

Öntanımlı layout modeli FlowLayout olan JPanel taşıyıcısı gibi, öntanımlı layout modeli BoxLayout olan "Box" adında bir taşıyıcı sınıfı daha vardır.

Box Layout modeine sahip yeni bir Box taşıyıcı örneği yaratmak için:

```
Box b = Box.createHorizontalBox();
Box b = Box.createVerticalBox();

b.add(okButton);
```

Horizontal box taşıyıcısı nesneleri soldan sağa yerleştirir. Vertical Box nesnesi nesneleri yukarıdan aşağıya yerleştirir.

Her nesne 3 adet boyut değerine sahiptir:
prefferred size
maximum size
minimum size

Aşağıda bir box layout modeline sahip horizontal Box taşıyıcısının çalışmasına ilişkin detaylar sırasıyla belirtilmektedir:

Box container;
- Önce en yüksek nesnenin maximum yüksekliğini hesaplar.
- Daha sonra tüm nesneleri yatay olarak bu yüksekliğe uzatmayı dener.
- Eğer bir nesne bu yükseklikte uzatılamıyorsa, o nesnenin y eksenideki konumu ele alınır. Bu işlem "getAlignmentY" metodu ile sağlanır. (0 align to top, 1 align to bottom, 0.5 align to center). Bu metod bu float değerlerden birtanesini geri döndürür. Böylece nesnenin y ekseni üzerindeki konumlandırılması tamamlanır.
- Her nesnenin "preferred width" değeri elde edilir ve toplanır.
- Eğer toplam uzunluklar box uzunluğundan daha az ise, o zaman nesneler "maximum size" değerleri göz önüne alınarak box uzunluğuna genişletilir.Nesneler arasında hiç boşluk bırakılmaz. Eğer nesnelerin toplam uzunluğu box uzunluğundan daha fazla ise, o zaman minimum size değerleri kullanılır. Yine de sığmazlar ise bazı nesneler görünmeyecektir.

Vertical box taşıyıcısı için yukarıdaki işlemler benzer şekilde çalışacaktır.

Öntnaımlı olarak BoxLayout modelinde nesneler arasında hiç boşluk bırakılmaz. İstenilen oranda boşluk nırakmak için bazı alan filtreleri kullanılır:
Struts
Rigid areas
Glue

Strut nesneler arasına bazı boşluklar ekler:

```
Box b = Box.cerateHorizontalBox();
b.add(label)
b.add(Box.createHorizontalStrut(10));
b.add(textField);
```

Rigid area filteri de Struts gibi boşluk bırakmak amacıyla kullanılır. Ancak bu filtre hem yatay hem de düzey alanda tek seferde belirli oranda boşluk bırakmaya olanak verir. Mesela:

```
b.add(Box.createRigidArea(new Dimension(5, 20));
```

ifadesi, horizontal bir layout kullnırken nesneler arasında 5 pixel boşluk ve box yükeskliği ile 20 pixel boşluk yaratır.

Glue filtresi, nesneler arasına mümkün olan tüm boşluğu ekler. Yani bir glue filtresi kullanılması durumunda box üzerindeki tüm boş alan tüketilir.

b.add(button1);
b.add(Box.createGlue());
b.add(button2);

```
/*** BoxLayoutTest.java *****/

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class BoxLayoutTest {
      public static void main(String arg[]) {
            new BoxLayoutFrame();
      }
}


class BoxLayoutFrame extends JFrame {

      private final int WIDTH = 300, HEIGHT = 400;



      public BoxLayoutFrame() {
            setTitle("BoxLayoutTest");
            setSize(WIDTH, HEIGHT);


            JLabel label1 = new JLabel("Name: ");

            JTextField textField1 = new JTextField(10);
            textField1.setMaximumSize(textField1.getPreferredSize());

            Box hbox1 = Box.createHorizontalBox();
            hbox1.add(label1);
            hbox1.add(Box.createHorizontalStrut(10));
            hbox1.add(textField1);


            JLabel label2 = new JLabel("Password: ");

            JTextField textField2 = new JTextField(10);
            textField2.setMaximumSize(textField2.getPreferredSize());

            Box hbox2 = Box.createHorizontalBox();
            hbox2.add(label2);
            hbox2.add(Box.createHorizontalStrut(10));
            hbox2.add(textField2);

            JButton button1 = new JButton("Ok");
            JButton button2 = new JButton("Cancel");

            Box hbox3 = Box.createHorizontalBox();
            hbox3.add(button1);
            hbox3.add(Box.createHorizontalStrut(10));
            hbox3.add(button2);

            Box vbox = Box.createVerticalBox();
            vbox.add(hbox1);
```

```
            vbox.add(hbox2);
            vbox.add(Box.createVerticalGlue());
            vbox.add(hbox3);

            this.getContentPane().add(vbox);


            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            show();


        }
}
```

GridBagLayout

GridBag layout'u limitsiz bir Grid layout gibi düşünebiliriz. Diğer layout modelleri içerisinde en karmaşık ve en güçlü olandır.

Bu modelde kolon ve satırlardaki hücrelerin boyutları değiştirilebilmektedir.

Hücreler birbirleri ile birleştirilerek daha geniş hücreler elde edilebilir.

GridBagLayout tasarımı ilk önce el yordamıyla kağıt kalem üzerinde yapılmalıdır. Tüm nesnelerin yerleri kaç satır ve kaç sütun kaplayacakları belirlenir.

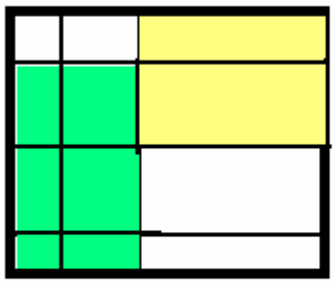
GridBagLayout kullanabilmek için öncelikle bir GridBagConstraints nesnesi tasarlanmalıdır. Bu nesne bir nesnenin GridBagLayout hücrelerine nasıl yerleşeceğini belirler.

gridx, gridy, gridwidth, gridheight

Bu özellikler (kısıtlar) bir nesnenin grid üzerinde nereye yerleşeceğini belirler.

gridx ve gridy eklenecek olan nesnenin sol üst kısmının yer alacağı hücrenin kendisini belirler.

gridwidth ve gridheight, sol üst köşesi yerleştirilen bu nesnenin kaç tane satır ve kaç tane sütun yer kaplayacağını belirler.



gridx = 2
gridy = 0
gridwidth = 1
gridheight = 2

gridx = 1
gridy = 0
gridwidth = 2
gridheight = 3

weightx, weighty

Bu değerler bir nesnenin yer aldığı hücrenin boyutlarını değişmesine ilişkin özellikleri belirler.

weightx ve weighty değerlerinin 0'a set edilmesi, taşıyıcının boyutlarının değişmesi durumunda ilgili nesnenin yer aldığı hücrenin x ya da y boyutlarının sabit kalmasına neden olur. Bu değerlerin 0 dışında set edilmesi panelin

yeniden boyutlandırılması durumunda ilgili hücredeki nesnenin de boyutlarının uzamasına nedenl olur. weight değeri daha büyük olan nesne yeniden boyutlanma sırasında daha fazla yer kaplar.

Eğer tüm nesneler için weight değerleri 0 yapılırsa, o zaman GUI boyutları değiştikçe nesneler de GUI nin ortasında yüzecektir.

fill, anchor

fill, bir nesnenin bulunduğu alan içerisinde tüm alanı kapsamaması için gerekli ayarı set eder:
GridBagConstraints.NONE: alanı hiç doldurmaz
GridBagConstraints.VERTICAL: sadece dikeyde doldurur
GridBagConstraints.HORIZONTAL: sadece yatayda doldurur
GridBagConstraints.BOTH: tum alanı doldurur

anchor, eğer nesne alanı doldurmuyorsa (hiçbiryerde doldurmuyorsa, dikeyde doldurmuyorsa ya da yatayda doldurmuyorsa) bu bölgede nerede yer alacağını belirlemek için kullanılır.
GridBagConstraints.NORTH: doldurmadığı alanda kuzeye dayalı
GridBagConstraints.CENTER: doldurmadığı alanda merkezde
GridBagConstraints.NORTHEAST: ...
GridBagConstraints.EAST: ...
...

GridBagLayout modelinin sağladığı bu esnekliği kullanabilmek için aslında aşağıdaki 4 temel adım uygulanmaktadır:

1) GridBagLayout türünde bir örnek yaratılır. Ancak hiç bir değer verimez. Sadece öntanımlı olarak bu nesne yaratılır.
2) Taşıyıcı nesnenin layout modeli bu örnek türüne set edilir.
3) GridBagConstraints türünde bir örnek yaratıolır. Bu nesne aslında diğer nesnelerin GridBag içerisinde nasıl yerleşeceğini belirleyecek özellikleri tutar ve set eder.
4) Eklencek olan her nesne için, istenilen GridBagConstraints özellikleri set edilir ve aşağıdaki add metodu ile nesneler taşıyıcı üzerine eklenir:

add(component, constraints);

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class GridBagDemo extends JFrame {
   private Container container;
   private GridBagLayout gbLayout;
   private GridBagConstraints gbConstraints;

   public GridBagDemo()
   {
      super( "GridBagLayout" );

      container = getContentPane();
      gbLayout = new GridBagLayout();
      container.setLayout( gbLayout );

      // instantiate gridbag constraints
      gbConstraints = new GridBagConstraints();
```

```
    JTextArea ta = new JTextArea( "TextArea1", 5, 10 );
    JTextArea tx = new JTextArea( "TextArea2", 2, 2 );
    String names[] = { "Iron", "Steel", "Brass" };
    JComboBox cb = new JComboBox( names );
    JTextField tf = new JTextField( "TextField" );
    JButton b1 = new JButton( "Button 1" );
    JButton b2 = new JButton( "Button 2" );
    JButton b3 = new JButton( "Button 3" );

    // text area
    // weightx and weighty are both 0: the default
    // anchor for all components is CENTER: the default

    gbConstraints.fill = GridBagConstraints.BOTH;
    //gbConstraints.fill = GridBagConstraints.HORIZONTAL;
    addComponent( ta, 0, 0, 1, 3 );

    // button b1
    // weightx and weighty are both 0: the default
    gbConstraints.fill = GridBagConstraints.HORIZONTAL;
    addComponent( b1, 0, 1, 2, 1 );

    // combo box
    // weightx and weighty are both 0: the default
    // fill is HORIZONTAL
    addComponent( cb, 2, 1, 2, 1 );

    // button b2
    gbConstraints.weightx = 1000;  // can grow wider
    gbConstraints.weighty = 1;     // can grow taller
    gbConstraints.fill = GridBagConstraints.BOTH;
    addComponent( b2, 1, 1, 1, 1 );

    // button b3
    // fill is BOTH
    gbConstraints.weightx = 0;
    gbConstraints.weighty = 0;
    addComponent( b3, 1, 2, 1, 1 );

    // textfield
    // weightx and weighty are both 0: fill is BOTH
    addComponent( tf, 3, 0, 2, 1 );

    // textarea
    // weightx and weighty are both 0: fill is BOTH
    addComponent( tx, 3, 2, 1, 1 );

    setSize( 300, 150 );
    show();
}

// addComponent is programmer defined
private void addComponent( Component c,
    int row, int column, int width, int height )
{
    // set gridx and gridy
    gbConstraints.gridx = column;
    gbConstraints.gridy = row;

    // set gridwidth and gridheight
    gbConstraints.gridwidth = width;
    gbConstraints.gridheight = height;
```

```
      // set constraints
      gbLayout.setConstraints( c, gbConstraints );
      container.add( c );      // add component
   }

   public static void main( String args[] )
   {
      GridBagDemo app = new GridBagDemo();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
        }
      );
   }
}
```

```
/*** FontDialog.java ***/
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FontDialog {

      public static void main(String[] args) {
            new FontDialogFrame();
      }
}


class FontDialogFrame extends JFrame {

      JComboBox face, size;
      JCheckBox bold, italic;
      JTextArea sample;

      public FontDialogFrame() {

            setTitle("Font Dalog");
            setSize(400, 400);


            /*Contaner ve Layout yaratiliyor*/
            Container c = this.getContentPane();
            GridBagLayout gbl = new GridBagLayout();
            c.setLayout(gbl);

            ActionListener listener = new FontAction();

            JLabel faceLabel = new JLabel("Face: ");
```

```
            face = new JComboBox(new String[] {"Serif", "SansSerif", "Monospaced",
"Dialog", "DialogInput"});
            face.addActionListener(listener);

            JLabel sizeLabel = new JLabel("Size: ");
            size = new JComboBox(new String[] {"8", "10", "12", "15", "18", "24"} );
            size.addActionListener(listener);

            bold = new JCheckBox("Bold");
            bold.addActionListener(listener);

            italic = new JCheckBox("Italic");
            italic.addActionListener(listener);

            sample = new JTextArea();
            sample.setText("Bugun, bundan sonraki gunlerinin ilk gunudur");
            sample.setEditable(false);
            sample.setLineWrap(true); //satir bitince alt satira gecis
            sample.setBorder(BorderFactory.createEtchedBorder());


            /*Grid tanimlamalarimini hazirliyoruz*/
            GridBagConstraints constraints = new GridBagConstraints();

            constraints.fill = GridBagConstraints.NONE;
            constraints.anchor = GridBagConstraints.EAST;
            constraints.weightx = 0;
            constraints.weighty = 0;

            add(faceLabel, constraints, 0, 0, 1, 1);
            add(sizeLabel, constraints, 0, 1, 1, 1);

            constraints.fill = GridBagConstraints.HORIZONTAL;
            constraints.weightx = 100;

            add(face, constraints, 1, 0, 1, 1);
            add(size, constraints, 1, 1, 1, 1);

            constraints.weighty = 100;
            constraints.fill = GridBagConstraints.NONE;
            constraints.anchor = GridBagConstraints.CENTER;

            add(bold, constraints, 0, 2, 2, 1);
            add(italic, constraints, 0, 3, 2, 1);

            constraints.fill = GridBagConstraints.BOTH;
            add(sample, constraints, 2, 0, 1, 4);

            show();

      }

      public void add(Component c, GridBagConstraints constraints, int x, int y, int
w, int h) {

            constraints.gridx = x;
            constraints.gridy = y;
            constraints.gridwidth = w;
            constraints.gridheight = h;
            getContentPane().add(c, constraints);

      }
```

```
	private class FontAction implements ActionListener {

		public void actionPerformed(ActionEvent e) {

			String fontFace = (String)face.getSelectedItem();
			int fontStyle = (bold.isSelected() ? Font.BOLD : 0) +
(italic.isSelected() ? Font.ITALIC : 0);
			int fontSize = Integer.parseInt( (String)size.getSelectedItem() );

			sample.setFont(new Font(fontFace, fontStyle, fontSize));
			sample.repaint();
		}

	}

}
```

Swing Nesneleri Üzerinde Bir Tur

Model-view-controller (MVC) yapısı swing nesnelerinin tasarlanmasında ve kullanılmasında dikkat edilen 3 temel yaklaşımı temsil eder.

Model: sadece nesnenin içeriğidir. Yani gorsel açıdan hiçbirsey ifade etmez. Model içerik hakkında bilgiyi alan ve içerigi degistirne metodların tasarımıdır.

View: Bir nesnenin ekrandaki goruntusunu temsil eder.

Controller: Kullanıcı girislerini ele alır.

JButton
Bir dugmeye basıldıgı zaman ActionEvent yaratılır. Bu ActionEven uzerinden kullanılabilecek metodlar:

getActionCommand()
getMnemonic() : dugme uzerine fare imleci ile geldiginiziden ekranda beliren ipucu kutucugu
isArmed(): Eger dugmeye basildiysa ve mouse hala uzerindeyse true
isEnabled(): eger dugme secilebilir nitelikte ise true
isPressed(): Eger dugmeye basildiysa ve mouse hala release edilmediyse true
isRollover(): Eger mouse hala uzerindeyse true
isSelected() : Eger dugme tek tıklama gbi bir hareketle secildiyse

JTextFields
Text girislerini elde alamak amacıyla kullanılır.

void setText(String t)
String getText()
void setEditable(boolean b)
setFont(new Font(…)): alan icerisindeki font degerini degistirmek icin kulanılabilir.
JTextField(String str, int colsize)

Text field icerisindeki degisiklikler
Text field icerisindeki her degisikligi control etmek icin biraz efor harcamak gerekir. Bu islem icin key listener dinlemek pek mantıklı olmayabilir. Cunku bazı tuslar zaten bir tex girisi yapmazlar (ok tuslari gibi) Ayrıca text girisleri fare ile de yapılabilir.

Butun text nesneleri icin "model", Document denilen bir arayüz tarafından ifade edilir. Kontrol edilmek istenen text nesnesi icin bir document listener yuklenerek- nesne uzerindeki her degisiklik elde edilebilir:

textField.getDocument().addDocumentListener(listener);
//text nesnesni temsil eden document nesnesi elde edilerek bir document listener nesnesine kaydediliyor.

Bu durumda Text nesnesinde meydana gelen her degisiklikjte asagidaki metodlardan birisi tetiklenir,

void insertUpdate(DocumentEvent e)
void removeUpdate(DocumentEvent e)
void changeUpdate(DocumentEvent e)

3 metod her text nesnesi icin cagrilmaz (style- size v.b. degisiklikler.) Ancak ilk iki metod text alanına yapılan bir giris ya da slime isleminde tetiklenir:

Kontrollu klavye girisine iliskin bir ornek

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;

public class TextTest extends JFrame implements ActionListener, KeyListener {

	Container c;
	JButton dugme;
	JTextField text;

	public TextTest() {
		c = getContentPane();
		c.setLayout(new FlowLayout());


		dugme = new JButton("Gonder");
		dugme.setToolTipText("int bilgiyi gonderir");
		dugme.addActionListener(this);
		dugme.setBackground(Color.BLUE);
		dugme.setForeground(Color.WHITE);
		c.add(dugme);

		text = new JTextField("0", 10);
		text.addKeyListener(this);
		c.add(text);

		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(240, 120);
		setResizable(false);
		show();


	}
```

```
	public void actionPerformed(ActionEvent e) {
		int strInt = strInt = Integer.parseInt(text.getText());

		if(strInt == 42)
			System.exit(0);
		else
			JOptionPane.showMessageDialog(this, ""+strInt);
	}

	public void keyPressed(KeyEvent e) {

		int k = e.getKeyCode();

		if(k != KeyEvent.VK_0 && k != KeyEvent.VK_1 && k != KeyEvent.VK_2 &&
		   k != KeyEvent.VK_3 && k != KeyEvent.VK_4 && k != KeyEvent.VK_5 &&
		   k != KeyEvent.VK_6 && k != KeyEvent.VK_7 && k != KeyEvent.VK_8 &&
		   k != KeyEvent.VK_9 && k != KeyEvent.VK_BACK_SPACE) {


				JOptionPane.showMessageDialog(this, "rakam giriniz..");

				String str = text.getText();
				str = str.substring(0, str.length()-1);
				text.setText(str);

				text.validate();
		}

		c.repaint();
	}

	public void keyReleased(KeyEvent e) {}

	public void keyTyped(KeyEvent e) {}

	public static void main(String arg[]) {

		new TextTest();
	}

}
```

JPasswordField

Password alanları text girileri gibidir. Ancak kullanıcı tarafından girilen bilgi ekranda * tuslari ile gorunur.

```
JPasswordField(String str, int colsize)
void setEchoChar(char echo): gizleyici karakter sembolu
char [] getPassword(): password alanı icerisinde saklanan password bilgisini geri gonderir.
```

Ancak bu bilgi String olarak gonderilmez. Bunun nedeni Stringler garbage collector devreye girene kadar hazfızadan silinmezler.

JTextArea

Bazen kullanıcıdan alınacak olan bilgi textfield gibi kucuk alanlarda degil daha buyuk alanlarda saklanma istenebilir.

JTextArea(line, column)

void setLineWrap(boolean b): Text alanının kolon uzunlugundan daha uzun satirlar bolunur. Ancak bu bolme islemi sanaldir. Yani orjinal text icerisine "\n" eklenmez. Ancak kullanıcı kendisi Enter tusuna basarsa bu bilgi orjinal tex icerisine eklenir.

Java da bir TextArea nın kendisine ait scroll bar'ı yoktur. Bu nedenle gerektiginde kullanıcı kendisi TextArea nesnesini bir ScrollBar icerisine eklemelidir.

JScrollPane scroll = new JScrollPane(new JTextArea(8, 40));
contentPane.add(scroll, BorderLayout.CENTER);

Gereginden fazla text girisi yapildiginda scroll bar otomatik olarak belirecektir.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TextAreaTest
{
   public static void main(String[] args)
   {
      TextAreaFrame frame = new TextAreaFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.show();
   }
}

/**
   A frame with a text area and buttons for text editing
*/
class TextAreaFrame extends JFrame
{
   public TextAreaFrame()
   {
      setTitle("TextAreaTest");
      setSize(WIDTH, HEIGHT);

      Container contentPane = getContentPane();

      buttonPanel = new JPanel();

      // add button to append text into the text area

      JButton insertButton = new JButton("Insert");
      buttonPanel.add(insertButton);
      insertButton.addActionListener(new
         ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               textArea.append("The quick brown fox "
                  + "jumps over the lazy dog. ");
            }
         });

      // add button to turn line wrapping on and off

      wrapButton = new JButton("Wrap");
      buttonPanel.add(wrapButton);
      wrapButton.addActionListener(new
         ActionListener()
```

```
         {
            public void actionPerformed(ActionEvent evt)
            {
               boolean wrap = !textArea.getLineWrap();
               textArea.setLineWrap(wrap);
               scrollPane.validate();
               wrapButton.setText(wrap ? "No Wrap" : "Wrap");
            }
         });

      contentPane.add(buttonPanel, BorderLayout.SOUTH);

      // add a text area with scroll bars

      textArea = new JTextArea(8, 40);
      scrollPane = new JScrollPane(textArea);

      contentPane.add(scrollPane, BorderLayout.CENTER);
   }

   public static final int WIDTH = 300;
   public static final int HEIGHT = 300;

   private JTextArea textArea;
   private JScrollPane scrollPane;
   private JPanel buttonPanel;
   private JButton wrapButton;
}


import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class TextEditTest
{
   public static void main(String[] args)
   {
      TextEditFrame frame = new TextEditFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.show();
   }
}

/**
   A frame with a text area and components for search/replace.
*/
class TextEditFrame extends JFrame
{
   public TextEditFrame()
   {
      setTitle("TextEditTest");
      setSize(WIDTH, HEIGHT);

      Container contentPane = getContentPane();

      JPanel panel = new JPanel();

      // add button, text fields and labels

      JButton replaceButton = new JButton("Replace");
      panel.add(replaceButton);
      replaceButton.addActionListener(new ReplaceAction());
```

```
      from = new JTextField("brown", 8);
      panel.add(from);

      panel.add(new JLabel("with"));

      to = new JTextField("purple", 8);
      panel.add(to);
      contentPane.add(panel, BorderLayout.SOUTH);

      // add text area with scroll bars

      textArea = new JTextArea(8, 40);
      textArea.setText
         ("The quick brown fox jumps over the lazy dog.");
      JScrollPane scrollPane = new JScrollPane(textArea);
      contentPane.add(scrollPane, BorderLayout.CENTER);
   }

   public static final int WIDTH = 400;
   public static final int HEIGHT = 200;

   private JTextArea textArea;
   private JTextField from;
   private JTextField to;

   /**
      The action listener for the replace button.
   */
   private class ReplaceAction implements ActionListener
   {
      public void actionPerformed(ActionEvent event)
      {
         String f = from.getText();
         int n = textArea.getText().indexOf(f);
         if (n >= 0 && f.length() > 0)
            textArea.replaceRange(to.getText(), n,
               n + f.length());
      }
   }
}
```

Etiketler ve Nesnelerin Etiketlendirilmesi

```
JLabel(String str, int alignment)
SwingConstants.LEFT, RIGHT, CENTER, NORTH, EAST
setText
setIcon(Icon a)
```

JCheckBox

Kullanıcıdan sadece "evet" ya da "hayir" girişlerini almak icin check box kullanılabilir. Bu nesneler yaratılırken label ile birlikte yaratılırlar.

JCheckBox(String str)
void setSelected(Boolean b)
boolean isSelected()

JRadioButton

Bu nesneler de check box lar gibi olumlu ya da olumsuz bir seçimi ifade ederler.

JRadioButton(String label, boolean selection)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


public class ButtonsTest extends JFrame implements ActionListener {

	Container c;
	JLabel l;
	JRadioButton r1, r2;
	JPanel bp, lp;
	JCheckBox c1, c2;
	ButtonGroup group;
	Font f;
	int boldvalue = 0, italicvalue = 0, size = 24;

	public ButtonsTest() {

		c = getContentPane();

		l = new JLabel("Hakan Gozutok");
		f = new Font("Serif", boldvalue + italicvalue, size);
		l.setFont(f);

		r1 = new JRadioButton("Blue");
		r1.addActionListener(this);
		r2 = new JRadioButton("Red");
		r2.addActionListener(this);
		group = new ButtonGroup();
		group.add(r1);
		group.add(r2);

		c1 = new JCheckBox("Bold");
		c1.addActionListener(this);
		c2 = new JCheckBox("Italic");
		c2.addActionListener(this);

		bp = new JPanel();
		bp.add(r1);
		bp.add(r2);
		bp.add(c1);
		bp.add(c2);

		/*label icin hazirlanan panel*/
		lp = new JPanel();
		lp.add(l);

		c.add(lp, BorderLayout.CENTER);
		c.add(bp, BorderLayout.SOUTH);
```

```
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setSize(300, 400);
            show();
      }

      public void actionPerformed(ActionEvent e) {

            if(e.getSource() == r1)
                  l.setForeground(Color.BLUE);
            else if(e.getSource() == r2)
                  l.setForeground(Color.RED);

            if(e.getSource() == c1 || e.getSource() == c2) {

                  boldvalue = c1.isSelected() ? Font.BOLD : 0;
                  italicvalue = c2.isSelected() ? Font.ITALIC : 0;

                  f = new Font("Serif", boldvalue + italicvalue, size);
                  l.setFont(f);

            }

            l.validate();
            c.repaint();


      }

      public static void main(String[] args) {

            new ButtonsTest();
      }
}
```

JComboBox

Birden fazla seçim yapmak gereken durumlarda Combox nesnelerinin kullanmak iyi bir çözüm olabilir.

```
faceCombo = new JComboBox();
faceCombo.setEditable(true);
faceCombo.addItem("Serif");
faceCombo.addItem("SansSerif");


Object getSelectedItem()
JComboBox(String []str)
void removeItemAt(int index)
void removeItem(String item)

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


public class ButtonsTest extends JFrame implements ActionListener {

      Container c;
      JLabel l;
      JRadioButton r1, r2;
      JPanel bp, lp;
      JCheckBox c1, c2;
      ButtonGroup group;
      Font f;
```

```
int boldvalue = 0, italicvalue = 0, size = 24;
JComboBox sizecombo;

public ButtonsTest() {

      c = getContentPane();

      l = new JLabel("Hakan Gozutok");
      f = new Font("Serif", 0, 24);
      l.setFont(f);

      r1 = new JRadioButton("Blue");
      r1.addActionListener(this);
      r2 = new JRadioButton("Red");
      r2.addActionListener(this);
      group = new ButtonGroup();
      group.add(r1);
      group.add(r2);

      c1 = new JCheckBox("Bold");
      c1.addActionListener(this);
      c2 = new JCheckBox("Italic");
      c2.addActionListener(this);

      sizecombo = new JComboBox(new String [] {"8", "12", "24", "48"});
      sizecombo.setSelectedItem("24");
      sizecombo.setEditable(true);
      sizecombo.addActionListener(this);

      bp = new JPanel();
      bp.add(r1);
      bp.add(r2);
      bp.add(c1);
      bp.add(c2);
      bp.add(sizecombo);


      /*label icin hazirlanan panel*/
      lp = new JPanel();
      lp.add(l);

      c.add(lp, BorderLayout.CENTER);
      c.add(bp, BorderLayout.SOUTH);

      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setSize(600, 400);
      show();
}

public void actionPerformed(ActionEvent e) {

      if(e.getSource() == r1)
            l.setForeground(Color.BLUE);
      else if(e.getSource() == r2)
            l.setForeground(Color.RED);

      if(e.getSource() == c1 || e.getSource() == c2) {

            boldvalue = c1.isSelected() ? Font.BOLD : 0;
            italicvalue = c2.isSelected() ? Font.ITALIC : 0;


            l.setFont(new Font("Serif", boldvalue + italicvalue, size));
```

```
            }

            if(e.getSource() == sizecombo) {
                  m("combo");
                  size = Integer.parseInt(sizecombo.getSelectedItem().toString());
                  l.setFont(new Font("Serif", boldvalue + italicvalue, size));
            }

            l.validate();
            c.repaint();


      }
      public void m(String str) {
            JOptionPane.showMessageDialog(null, str);

      }

      public static void main(String[] args) {

            new ButtonsTest();
      }
}
```

JSlider
Bu nesneler birtakım devam eden sürekli secimler uzerinde kullanıcının secim yapmasını sağlayan nesnelerdir. Kullanıcı bu secim rastgele degil sırayla yapabilir. (Biliyoruz ki combo box lar kullanılırken secim rastgle yapılıyordu)

En genell sekliyle bir JSlider nesnesi asagidaki gibi yaratılır:
new JSlider(min, max, initialValue)

Eger yatay degil de dikey bir JSlider yaratılmak istenirse:

JSlider sl = new JSlider(SwingConstants.VERTICAL, min, max, initialValue)

Ancak bu slider yaratma yöntemleri sadece düz bir slider yaratır. Slider'ların bir dekorasyon ile de yaratılabileceğini belirtmek gerekir:

ChangeListener
Slider hareket ettigi zaman ChangeEvent yaratılır. Bu nedenle Slider'ların dinlenmesi için onları ChangeListener nesnesine eklemek gerekir.

Bir ChangeListener arayüzü içerisinde:

stateChanged(ChangeEvent e)

metodu yer alır.

Bir Slider dekoratif olarak da yaratılabilir. Bunun için:

slider.setMajorTickSpacing(20)
slider.setMinorTickSpacing(5);
slider.setPaintTicks(true): Bu dekorların gorunur olmasını sağlar.
slider.setPaintLabels(true)

metoldarından yararlanılır.

slider.setPaintLabels(true): Major tick space' leri gorunur kılar

Slider boyutlandirmasi için:
sl.setPreferredSize(new Dimension(550, 50));
sl.setMinimumSize(new Dimension(550, 50));
sl.setMaximumSize(new Dimension(550, 50));

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;


public class ButtonsTest extends JFrame implements ActionListener, ChangeListener {

	Container c;
	JLabel l;
	JRadioButton r1, r2;
	JPanel bp, lp;
	JCheckBox c1, c2;
	ButtonGroup group;
	Font f;
	int boldvalue = 0, italicvalue = 0, size = 24;
	JComboBox sizecombo;
	JSlider sl;
	JPanel np;

	public ButtonsTest() {

		c = getContentPane();

		l = new JLabel("Hakan Gozutok");
		f = new Font("Serif", 0, 24);
		l.setFont(f);

		r1 = new JRadioButton("Blue");
		r1.addActionListener(this);
		r2 = new JRadioButton("Red");
		r2.addActionListener(this);
		group = new ButtonGroup();
		group.add(r1);
		group.add(r2);

		c1 = new JCheckBox("Bold");
		c1.addActionListener(this);
		c2 = new JCheckBox("Italic");
		c2.addActionListener(this);

		sizecombo = new JComboBox(new String [] {"8", "12", "24", "48"});
		sizecombo.setSelectedItem("24");
		sizecombo.setEditable(true);
		sizecombo.addActionListener(this);

		sl = new JSlider(0, 100, 24);
		sl.setMajorTickSpacing(5);
		sl.setMinorTickSpacing(10);
		sl.setPaintTicks(true);
```

```
		sl.setPaintLabels(true);
		sl.addChangeListener(this);



		bp = new JPanel();
		bp.add(r1);
		bp.add(r2);
		bp.add(c1);
		bp.add(c2);
		bp.add(sizecombo);


		/*label icin hazirlanan panel*/
		lp = new JPanel();
		lp.add(l);

		np = new JPanel();
		np.add(sl);

		c.add(lp, BorderLayout.CENTER);
		c.add(bp, BorderLayout.SOUTH);
		c.add(np, BorderLayout.NORTH);

		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		setSize(600, 400);
		show();
	}

	public void actionPerformed(ActionEvent e) {

		if(e.getSource() == r1)
			l.setForeground(Color.BLUE);
		else if(e.getSource() == r2)
			l.setForeground(Color.RED);

		if(e.getSource() == c1 || e.getSource() == c2) {

			boldvalue = c1.isSelected() ? Font.BOLD : 0;
			italicvalue = c2.isSelected() ? Font.ITALIC : 0;


			l.setFont(new Font("Serif", boldvalue + italicvalue, size));

		}

		if(e.getSource() == sizecombo) {
			m("combo");
			size = Integer.parseInt(sizecombo.getSelectedItem().toString());
			l.setFont(new Font("Serif", boldvalue + italicvalue, size));
		}

		l.validate();
		c.repaint();


	}
	public void m(String str) {
		JOptionPane.showMessageDialog(null, str);

	}

	public static void main(String[] args) {
```

```
            new ButtonsTest();
      }


      public void stateChanged(ChangeEvent e) {

            l.setFont(new Font("Serif", boldvalue + italicvalue, sl.getValue()));
      }
}
```

Ayrıca slider uzerindeki bu bilgileri String ya da icon etiketler seklinde de ifade etmek mumkundur. Bunun icin:

setLabelTable(HashTable ltable)

metodunu kullanırız. Asagida buna iliskin orek kodlar yer alır.

```
      HashTable labelTable = new HashTable();
      labelTable.put(new Integer(0), new JLabel("A"));
      labelTable.put(new Integer(20), new JLabel("B"));
      ….
      labelTable.put(new Integer(100), new JLabel("F"));
      slider.setLabelTable(labelTable);
```

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;

public class SliderTest
{
   public static void main(String[] args)
   {
      SliderTestFrame frame = new SliderTestFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.show();
   }
}

/**
   A frame with many sliders and a text field to show slider
   values.
*/
class SliderTestFrame extends JFrame
{
   public SliderTestFrame()
   {
      setTitle("SliderTest");
      setSize(WIDTH, HEIGHT);

      sliderPanel = new JPanel();
      sliderPanel.setLayout(new FlowLayout(FlowLayout.LEFT));

      // common listener for all sliders
      listener = new
         ChangeListener()
         {
            public void stateChanged(ChangeEvent event)
            {
```

```
            // update text field when the slider value changes
            JSlider source = (JSlider)event.getSource();
            textField.setText("" + source.getValue());
         }
      };

   // add a plain slider

   JSlider slider = new JSlider();
   addSlider(slider, "Plain");

   // add a slider with major and minor ticks

   slider = new JSlider();
   slider.setPaintTicks(true);
   slider.setMajorTickSpacing(20);
   slider.setMinorTickSpacing(5);
   addSlider(slider, "Ticks");

   // add a slider that snaps to ticks

   slider = new JSlider();
   slider.setPaintTicks(true);
   slider.setSnapToTicks(true);
   slider.setMajorTickSpacing(20);
   slider.setMinorTickSpacing(5);
   addSlider(slider, "Snap to ticks");

   // add a filled slider

   slider = new JSlider();
   slider.setPaintTicks(true);
   slider.setMajorTickSpacing(20);
   slider.setMinorTickSpacing(5);
   slider.putClientProperty("JSlider.isFilled",
      Boolean.TRUE);
   addSlider(slider, "Filled");

   // add a filled and inverted slider

   slider = new JSlider();
   slider.setPaintTicks(true);
   slider.setMajorTickSpacing(20);
   slider.setMinorTickSpacing(5);
   slider.putClientProperty("JSlider.isFilled",
      Boolean.TRUE);
   slider.setInverted(true);
   addSlider(slider, "Inverted");

   // add a slider with numeric labels

   slider = new JSlider();
   slider.setPaintTicks(true);
   slider.setPaintLabels(true);
   slider.setMajorTickSpacing(20);
   slider.setMinorTickSpacing(5);
   addSlider(slider, "Labels");

   // add a slider with alphabetic labels

   slider = new JSlider();
   slider.setPaintLabels(true);
   slider.setPaintTicks(true);
```

```
      slider.setMajorTickSpacing(20);
      slider.setMinorTickSpacing(5);

      Hashtable labelTable = new Hashtable();
      labelTable.put(new Integer(0), new JLabel("A"));
      labelTable.put(new Integer(20), new JLabel("B"));
      labelTable.put(new Integer(40), new JLabel("C"));
      labelTable.put(new Integer(60), new JLabel("D"));
      labelTable.put(new Integer(80), new JLabel("E"));
      labelTable.put(new Integer(100), new JLabel("F"));

      slider.setLabelTable(labelTable);
      addSlider(slider, "Custom labels");

      // add a slider with icon labels

      slider = new JSlider();
      slider.setPaintTicks(true);
      slider.setPaintLabels(true);
      slider.setSnapToTicks(true);
      slider.setMajorTickSpacing(20);
      slider.setMinorTickSpacing(20);

      labelTable = new Hashtable();

      // add card images

      labelTable.put(new Integer(0),
         new JLabel(new ImageIcon("nine.gif")));
      labelTable.put(new Integer(20),
         new JLabel(new ImageIcon("ten.gif")));
      labelTable.put(new Integer(40),
         new JLabel(new ImageIcon("jack.gif")));
      labelTable.put(new Integer(60),
         new JLabel(new ImageIcon("queen.gif")));
      labelTable.put(new Integer(80),
         new JLabel(new ImageIcon("king.gif")));
      labelTable.put(new Integer(100),
         new JLabel(new ImageIcon("ace.gif")));

      slider.setLabelTable(labelTable);
      addSlider(slider, "Icon labels");

      // add the text field that displays the slider value

      textField = new JTextField();
      Container contentPane = getContentPane();
      contentPane.add(sliderPanel, BorderLayout.CENTER);
      contentPane.add(textField, BorderLayout.SOUTH);
   }

   /**
      Adds a slider to the slider panel and hooks up the listener
      @param s the slider
      @param description the slider description
   */
   public void addSlider(JSlider s, String description)
   {
      s.addChangeListener(listener);
      JPanel panel = new JPanel();
      panel.add(s);
      panel.add(new JLabel(description));
      sliderPanel.add(panel);
```

```
    }

    public static final int WIDTH = 350;
  public static final int HEIGHT = 450;

  private JPanel sliderPanel;
  private JTextField textField;
  private ChangeListener listener;
}
```

Ders 19

Menuler

Menuler bir GUI uygulamasının en vazgecilmez parcalarından birisidir. Bir menu cubugu uzerinde birtakım menu nesneleri ve alt menuler yer almaktadır. Bu kısımdan menuleri nasıl ele alacağımızı göreceğiz.

```
JMenuBar menuBar = new JMenuBar();
```

Yukarıda yaratılan bu menu bar istedigimiz yere ekleyebileceğimiz bir nesnedir.

Genelde bir menu, bir JFrame nesnesinin ust kısmına yerlestirilir. Bu islem asagfidaki gibi yapilabilir:

```
frame.setJMenuBar(menuBar);
```

menubar icerisine bir takım menuler eklenebilir

```
JMenu editMenu = new JMenu("Edit");
menuBar.addMenu(editMenu);
```

Her menu icerisinde menu nesneleri vardır.

```
JMenuItem pasteItem = new JMenuItem("Paste");
JMenuItem cutItem = new JMenuItem("Cut");
(JMenuItem iconedItem = new JMenuItem("Iconed", icon);
editMenu.add(pasteItem);
editMenu.add(cutItem);
```

Menu nesneleri arasına ayırac da eklenebilir:

```
JMenu optionMenu = new JMenu("Select");
editMenu.addSperator();
editMenu.add(optionsMenu)
```

Bir menu ya da menu nesnesi secildiginde ActionEvent nesnesi yaratılır. Bu nedenle daha once de ogrendigimiz sekilde menu nesnelerini ActionListener nesnelerine kaydetmek mumkundur.

MenuItem'lar JCheckBoxMenuItem ya da JRadioButtonMenuItem nesnelerinden de olabilirler.

Popup Menuleri

Popup menu de tıpkı JMenu nesnesi gibi bir menudur. Ancak tek farkı herhangi bir yerde sabit bir sekilde yerlesmezler.

```
JPopupMenu popup = new JPopupMenu();
JMenuItem item = new JMenuItem("Cut");
item.addActionListener(listener);
popup.add(item);
```

Bir popup menusunu her seferinde açıkça gostermek gerekir. Bunun için

```
popup.show(panel, x, y)
```

metodu kullanılır. panel nesnesi popup menusunu "parent" olan nesnesini temsil eder. X ve y de menunun gpsterilecegi lokasyon bilgisini belirtir.

Bir popup menusu asagidaki sekilde ele alınabilir:

```
1. bir mouse listener yuklenir.
2. public void mouseReleased(MouseEvent e) {
        if(e.isPopupTrigger())
            popup.show(e.getComponent(), e.getX(), e.getY());
   }
```

Menulerde Tus Kombinasyonları

Menuler kullanılırken fareye ihtiyaç duymadan bazı klavye tuş kombinasyonları yardımıyla menu nesnelerine ulaşılabilir. Java'da bu işelm bir menu nesnesi yaratılırken ayarlanabilir:

```
JMenuItem cutItem = new JMenuItem("Cut", 'T');
```

Bunun yerine bir menu nesnesine sonradan tuş kombinasyonu atamak da mümkündür:

```
cutItem.setMnemonic('T');
```

NOT: Bu tuş kombinasyonları sadece MenuItem constructor içerisinde tanımlanır. Eğer menu başlangıç metodu içerisinde de bir tanımlama yapmak istiyorsak o zaman setMnemonic metodu ile sonradan bunu belirtmek gerekir.

Hızlandırıcılar: Doğrudan Alt Menulere Erişim

Menulere tanımlanan bu tuş kombinadyonlarınn yanı sıra hızlandırıcılar denilen ve doğrudan alt menu elemanlarına erişimi sağlayan ayarlar da vardır. Çünkü sadece bu mnemonic tanımlamaları ile belirlenen menu elemanlarına erişmek için öncelikle ara menunun açılması gerekir. Ancak hızlandırıcılar ara menu açmaya gereksinim duymaksızın alt menu nesnelerine de erişimi sağlar.

Örneğin bir çok programda tanımlı olan Ctrl+O ve Ctrl+S hızlandırıcılar, doğrudan alt menu olan Save ve Open seçeneklerine erişmeyi sağlar.

Bunun için örneğin:

```
openItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, InputEvent.CTRL_MASK));
```

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.InputEvent;
import java.awt.event.KeyEvent;

import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextArea;
import javax.swing.KeyStroke;

public class JPanelOrnek1 extends JFrame implements ActionListener {

    JPanel colorPanel, sizePanel, editPanel, southPanel;
    JTextArea text;
```

```
  JButton yellowButton, redButton, blueButton;
  JRadioButton editable, noneditable;
  ButtonGroup group;
  Container c;
  JMenu fontMenu;
  JMenu fontStyleMenu;
  JCheckBoxMenuItem boldItem, italicItem;
  JMenuBar menuBar, menuBar2;

JComboBox comboMenu;

  int boldValue = 0, italicValue = 0;
  String face = "Serif";
  int size = 24;

  public JPanelOrnek1() {


      c = this.getContentPane();
      c.setLayout(new BorderLayout());

      southPanel = new JPanel();
      southPanel.setLayout(new BorderLayout());

      colorPanel = new JPanel(new FlowLayout());
      yellowButton = new JButton("Sari");
      yellowButton.addActionListener(this);
      colorPanel.add(yellowButton);
      redButton = new JButton("Kirmizi");
      redButton.addActionListener(this);
      colorPanel.add(redButton);
      blueButton = new JButton("Mavi");
      blueButton.addActionListener(this);
      colorPanel.add(blueButton);

      yellowButton.setBackground(Color.BLACK);
      yellowButton.setForeground(Color.WHITE);
      redButton.setBackground(Color.BLACK);
      redButton.setForeground(Color.WHITE);
      blueButton.setBackground(Color.BLACK);
      blueButton.setForeground(Color.WHITE);

      southPanel.add(colorPanel, BorderLayout.SOUTH);

      editPanel = new JPanel(new FlowLayout());
      group = new ButtonGroup();
      editable = new JRadioButton("Duzenlenebilir", false);
      group.add(editable);
      editable.addActionListener(this);
      editPanel.add(editable);
      noneditable = new JRadioButton("Duzenlenemez", true);
      group.add(noneditable);
      noneditable.addActionListener(this);
      editPanel.add(noneditable);

      southPanel.add(editPanel, BorderLayout.NORTH);

      text = new JTextArea();
      text.setText("Deneme Yazisi");
      text.setEditable(true);
      c.add(text, BorderLayout.CENTER);
      c.add(southPanel, BorderLayout.SOUTH);
```

```
        menuBar = new JMenuBar();

        fontMenu = new JMenu("Font");
        fontMenu.setToolTipText("Fontlarla ilgili ayarlari yapabilirsiniz");
        fontMenu.setMnemonic('F');

        fontStyleMenu = new JMenu("StyLe");


        boldItem = new JCheckBoxMenuItem("bold");
        boldItem.addActionListener(this);
        boldItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_B,
InputEvent.CTRL_MASK));
        fontStyleMenu.add(boldItem);

        italicItem = new JCheckBoxMenuItem("italic");
        italicItem.addActionListener(this);
        fontStyleMenu.add(italicItem);

        fontMenu.add(fontStyleMenu);

        this.comboMenu = new JComboBox(new String[] {"12", "24", "48"});
        this.comboMenu.setMaximumSize(new Dimension(50, 30));
        this.comboMenu.addActionListener(this);

        menuBar.add(fontMenu);
        menuBar.add(comboMenu);

        this.setJMenuBar(menuBar);







        this.setDefaultCloseOperation(this.EXIT_ON_CLOSE);
        this.setSize(300, 300);
        //this.show();
    }




    public void actionPerformed(ActionEvent e) {

        if(e.getSource() == yellowButton) {
            text.setBackground(Color.YELLOW);
            yellowButton.setBackground(Color.YELLOW);
            redButton.setBackground(Color.BLACK);
            blueButton.setBackground(Color.BLACK);
        }
        else if(e.getSource() == redButton) {
            text.setBackground(Color.RED);
            yellowButton.setBackground(Color.BLACK);
            redButton.setBackground(Color.RED);
            blueButton.setBackground(Color.BLACK);
        }
        else if(e.getSource() == blueButton) {
            text.setBackground(Color.BLUE);
```

```
            yellowButton.setBackground(Color.BLACK);
            redButton.setBackground(Color.BLACK);
            blueButton.setBackground(Color.BLUE);
        }


        if(e.getSource() == editable)
            text.setEditable(true);
        else if(e.getSource() == noneditable)
            text.setEditable(false);

        if(e.getSource() == this.boldItem || e.getSource() == this.italicItem ||
e.getSource() == this.comboMenu) {

            this.boldValue = this.boldItem.isSelected() ? Font.BOLD : 0;
            this.italicValue = this.italicItem.isSelected() ? Font.ITALIC : 0;

            this.size = Integer.parseInt((String)this.comboMenu.getSelectedItem());

            this.text.setFont(new Font(this.face, this.boldValue + this.italicValue,
this.size));
        }


        this.repaint();
    }


    public static void main(String[] args) {
        JPanelOrnek1 jp = new JPanelOrnek1();
        jp.show();
    }


}
```

NOT: Hızlandırıcılar sadece menu nesnelerine eklenebilir. Menulere hızlandırıcı ataması yapılmaz.

Menu Nesnelerinin Geçerli ya da Geçersiz Hale Getirilmesi

```
JMenuItem openItem = new JMenuItem("Open", 'O');
openItem.setEnabled(false);
```

ya da

```
openItem.setEnabled(true);
```

javax.swing.awt paketi içerisinde MenuListener isminde bir arayüz tanımlıdır ve bu arayüzün metodları

```
class MenuListener {
      void menuSelected(MenuEvent e) ;
      void menuCanceled(MenuEvent e);
      void menuDeselected(MenuEvent e) ;
}
```

Araç Çubukları (Toolbars)

Araç çubukları tahmin edebileceğimiz gibi üzerinde 1 ya da 1 den fazla düğme barındıran görünür menuler olarak tanımlanabilir.

NOT: Eğer içerisinde tanımlandığı Container 'ın layout modeli BorderLayout ise toolbar'ın drag özelliği çalışacaktır.

Ornek:
```
JToolBar bar = new JToolBar();
bar.add(blueButton);
bar.addSeperator();
```

Ornek:
```
contentPane.add(bar, BorderLayout.NORTH);
```

Ornek::
```
bar = new JToolBar("Title", SwingConstants.VERTICAL);
```

Aşağıdaki örnek Sun Microsytems tarafından yayınlanan Core Java 1 kitabından aynen alınmıştır.

```
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import javax.swing.*;

public class ToolBarTest
{
   public static void main(String[] args)
   {
      ToolBarFrame frame = new ToolBarFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.show();
   }
}

/**
   A frame with a toolbar and menu for color changes.
*/
class ToolBarFrame extends JFrame
{
   public ToolBarFrame()
   {
      setTitle("ToolBarTest");
      setSize(WIDTH, HEIGHT);

      // add a panel for color change

      Container contentPane = getContentPane();
      panel = new JPanel();
      contentPane.add(panel, BorderLayout.CENTER);

      // set up actions

      Action blueAction = new ColorAction("Blue",
         new ImageIcon("blue-ball.gif"), Color.blue);
      Action yellowAction = new ColorAction("Yellow",
         new ImageIcon("yellow-ball.gif"), Color.yellow);
      Action redAction = new ColorAction("Red",
         new ImageIcon("red-ball.gif"), Color.red);

      Action exitAction = new
         AbstractAction("Exit", new ImageIcon("exit.gif"))
```

```
         {
            public void actionPerformed(ActionEvent event)
            {
               System.exit(0);
            }
         };
      exitAction.putValue(Action.SHORT_DESCRIPTION, "Exit");

      // populate tool bar

      JToolBar bar = new JToolBar();
      bar.add(blueAction);
      bar.add(yellowAction);
      bar.add(redAction);
      bar.addSeparator();
      bar.add(exitAction);
      contentPane.add(bar, BorderLayout.NORTH);

      // populate menu

      JMenu menu = new JMenu("Color");
      menu.add(yellowAction);
      menu.add(blueAction);
      menu.add(redAction);
      menu.add(exitAction);
      JMenuBar menuBar = new JMenuBar();
      menuBar.add(menu);
      setJMenuBar(menuBar);
   }

   public static final int WIDTH = 300;
   public static final int HEIGHT = 200;

   private JPanel panel;

   /**
      The color action sets the background of the frame to a
      given color.
   */
   class ColorAction extends AbstractAction
   {
      public ColorAction(String name, Icon icon, Color c)
      {
         putValue(Action.NAME, name);
         putValue(Action.SMALL_ICON, icon);
         putValue(Action.SHORT_DESCRIPTION,
            name + " background");
         putValue("Color", c);
      }

      public void actionPerformed(ActionEvent evt)
      {
         Color c = (Color)getValue("Color");
         panel.setBackground(c);
         panel.repaint();
      }
   }
}
```

DialogPencereleri

Oldukça fazla sayıda hazır dialog pencereleri kullanılabilir:

Aşaıdaki örnekte hazır olarak kullanılabilecek tüm dialog pencereleri oldukça detaylı bir şekilde ele alınmıştır:

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;

public class OptionDialogTest
{
   public static void main(String[] args)
   {
      OptionDialogFrame frame = new OptionDialogFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.show();
   }
}

/**
    A panel with radio buttons inside a titled border.
*/
class ButtonPanel extends JPanel
{
   /**
      Constructs a button panel.
      @param title the title shown in the border
      @param options an array of radio button labels
   */
   public ButtonPanel(String title, String[] options)
   {
      setBorder(BorderFactory.createTitledBorder
            (BorderFactory.createEtchedBorder(), title));
      setLayout(new BoxLayout(this,
         BoxLayout.Y_AXIS));
      group = new ButtonGroup();

      // make one radio button for each option
      for (int i = 0; i < options.length; i++)
      {
         JRadioButton b = new JRadioButton(options[i]);
         b.setActionCommand(options[i]);
         add(b);
         group.add(b);
         b.setSelected(i == 0);
      }
   }

   /**
      Gets the currently selected option.
      @return the label of the currently selected radio button.
   */
   public String getSelection()
   {
      return group.getSelection().getActionCommand();
   }

   private ButtonGroup group;
}

/**
   A frame that contains settings for selecting various option
```

```
   dialogs.
*/
class OptionDialogFrame extends JFrame
{
   public OptionDialogFrame()
   {
      setTitle("OptionDialogTest");
      setSize(WIDTH, HEIGHT);

      JPanel gridPanel = new JPanel();
      gridPanel.setLayout(new GridLayout(2, 3));

      typePanel = new ButtonPanel("Type",
         new String[]
         {
            "Message",
            "Confirm",
            "Option",
            "Input"
         });

      messageTypePanel = new ButtonPanel("Message Type",
         new String[]
         {
            "ERROR_MESSAGE",
            "INFORMATION_MESSAGE",
            "WARNING_MESSAGE",
            "QUESTION_MESSAGE",
            "PLAIN_MESSAGE"
         });

      messagePanel = new ButtonPanel("Message",
         new String[]
         {
            "String",
            "Icon",
            "Component",
            "Other",
            "Object[]"
         });

      optionTypePanel = new ButtonPanel("Confirm",
         new String[]
         {
            "DEFAULT_OPTION",
            "YES_NO_OPTION",
            "YES_NO_CANCEL_OPTION",
            "OK_CANCEL_OPTION"
         });

      optionsPanel = new ButtonPanel("Option",
         new String[]
         {
            "String[]",
            "Icon[]",
            "Object[]"
         });

      inputPanel = new ButtonPanel("Input",
         new String[]
         {
            "Text field",
            "Combo box"
```

```
      });

   gridPanel.add(typePanel);
   gridPanel.add(messageTypePanel);
   gridPanel.add(messagePanel);
   gridPanel.add(optionTypePanel);
   gridPanel.add(optionsPanel);
   gridPanel.add(inputPanel);

   // add a panel with a Show button

   JPanel showPanel = new JPanel();
   JButton showButton = new JButton("Show");
   showButton.addActionListener(new ShowAction());
   showPanel.add(showButton);

   Container contentPane = getContentPane();
   contentPane.add(gridPanel, BorderLayout.CENTER);
   contentPane.add(showPanel, BorderLayout.SOUTH);
}

/**
Gets the currently selected message.
   @return a string, icon, component or object array,
   depending on the Message panel selection
*/
public Object getMessage()
{
   String s = messagePanel.getSelection();
   if (s.equals("String"))
      return messageString;
   else if (s.equals("Icon"))
      return messageIcon;
   else if (s.equals("Component"))
      return messageComponent;
   else if (s.equals("Object[]"))
      return new Object[]
      {
         messageString,
         messageIcon,
         messageComponent,
         messageObject
      };
   else if (s.equals("Other"))
      return messageObject;
   else return null;
}

/**
   Gets the currently selected options.
   @return an array of strings, icons or objects, depending
   on the Option panel selection
*/
public Object[] getOptions()
{
   String s = optionsPanel.getSelection();
   if (s.equals("String[]"))
      return new String[] { "Yellow", "Blue", "Red" };
   else if (s.equals("Icon[]"))
      return new Icon[]
      {
         new ImageIcon("yellow-ball.gif"),
         new ImageIcon("blue-ball.gif"),
```

```
            new ImageIcon("red-ball.gif")
         };
      else if (s.equals("Object[]"))
         return new Object[]
         {
            messageString,
            messageIcon,
            messageComponent,
            messageObject
         };
      else
         return null;
   }

   /**
      Gets the selected message or option type
      @param panel the Message Type or Confirm panel
      @return the selected XXX_MESSAGE or XXX_OPTION constant
      from the JOptionPane class
   */
   public int getType(ButtonPanel panel)
   {
      String s = panel.getSelection();
      try
      {
         return JOptionPane.class.getField(s).getInt(null);
      }
      catch(Exception e)
      {
         return -1;
      }
   }

   /**
      The action listener for the Show button shows a
      Confirm, Input, Message or Option dialog depending
      on the Type panel selection.
   */
   private class ShowAction implements ActionListener
   {
      public void actionPerformed(ActionEvent evt)
      {
         if (typePanel.getSelection().equals("Confirm"))
            JOptionPane.showConfirmDialog(
               OptionDialogFrame.this,
               getMessage(),
               "Title",
               getType(optionTypePanel),
               getType(messageTypePanel));
         else if (typePanel.getSelection().equals("Input"))
         {
            if (inputPanel.getSelection().equals("Text field"))
               JOptionPane.showInputDialog(
                  OptionDialogFrame.this,
                  getMessage(),
                  "Title",
                  getType(messageTypePanel));
            else
               JOptionPane.showInputDialog(
                  OptionDialogFrame.this,
                  getMessage(),
                  "Title",
                  getType(messageTypePanel),
```

```
                  null,
                  new String[] { "Yellow", "Blue", "Red" },
                  "Blue");
         }
         else if (typePanel.getSelection().equals("Message"))
            JOptionPane.showMessageDialog(
               OptionDialogFrame.this,
               getMessage(),
               "Title",
               getType(messageTypePanel));
         else if (typePanel.getSelection().equals("Option"))
            JOptionPane.showOptionDialog(
               OptionDialogFrame.this,
               getMessage(),
               "Title",
               getType(optionTypePanel),
               getType(messageTypePanel),
               null,
               getOptions(),
               getOptions()[0]);
      }
   }

   public static final int WIDTH = 600;
   public static final int HEIGHT = 400;

   private ButtonPanel typePanel;
   private ButtonPanel messagePanel;
   private ButtonPanel messageTypePanel;
   private ButtonPanel optionTypePanel;
   private ButtonPanel optionsPanel;
   private ButtonPanel inputPanel;

   private String messageString = "Message";
   private Icon messageIcon = new ImageIcon("blue-ball.gif");
   private Object messageObject = new Date();
   private Component messageComponent = new SamplePanel();
}

/**
   A panel with a painted surface
*/

class SamplePanel extends JPanel
{
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      Graphics2D g2 = (Graphics2D)g;
      Rectangle2D rect = new Rectangle2D.Double(0, 0,
         getWidth() - 1, getHeight() - 1);
      g2.setPaint(Color.yellow);
      g2.fill(rect);
      g2.setPaint(Color.blue);
      g2.draw(rect);
   }

   public Dimension getMinimumSize()
   {
      return new Dimension(10, 10);
   }
}
```

Yeni Dialog Pencerelerinin Yaratılması

Yeni bir dialog penceresi yaratmak için JDialog sınfından türetme yapmak gerekir. Aslında yeni bir dialog penceresi yaratmak, yeni bir JFrame nesnesi yaratmaya benzer. Tıpkı JFrame yaratır gibi çeşitli nesneleri JDialog nesnesi üzerinde yerleştirerek istediğimiz gibi bir dilog penceresi yaratılabilir.

Aşağıda oldukça basit bir dialog yaratma işlemi gösterilmektedir:

```
class AboutDialog extends JDialog
{
   public AboutDialog(JFrame owner)
   {
      /*owner parametresi bu dialog nesnesinin parent olan nesnesini
      belirler*/
      super(owner, "About DialogTest", true);
      Container contentPane = getContentPane();

      // add HTML label to center
      contentPane.add(new JLabel(
         "<HTML><H1><I>Core Java</I></H1><HR>"
         + "By Cay Horstmann and Gary Cornell</HTML>"),
         BorderLayout.CENTER);

      // Ok button closes the dialog

      JButton ok = new JButton("Ok");
      ok.addActionListener(new
         ActionListener()
         {
            public void actionPerformed(ActionEvent evt)
            {
               setVisible(false);
            }
         });

      // add Ok button to southern border

      JPanel panel = new JPanel();
      panel.add(ok);
      contentPane.add(panel, BorderLayout.SOUTH);

      setSize(250, 150);
   }
}
```

```
import java.awt.Container;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;

/**
 * @author DATAPRO
 *
```

```
 * TODO To change the template for this generated type comment go to
 * Window - Preferences - Java - Code Style - Code Templates
 */
public class AboutTestFrame extends JFrame {


	Container c;
	JButton button = new JButton("About");
	AboutDialog d;

	public AboutTestFrame() {
		c = getContentPane();
		c.setLayout(new FlowLayout());

		d = new AboutDialog(this);

		button.addActionListener(new ActionListener() {

				public void actionPerformed(ActionEvent e) {

					d.show();

				}

		});


		c.add(button);

		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setSize(300, 300);
		this.show();
		this.setResizable(false);

	}

	public static void main(String[] args) {
		new AboutTestFrame();
	}
}
```

Ancak yukarıdaki örnekte eksik olan şey bu dialog pencersinin bir değer üretmemesidir. Yani önemli olan şey dialog pencereleri yardımıyla kullanıcıya birtakım bilgiler vermenin yanısıra aynı zamanda kullanıcıdan birtakım verileri almaktır.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;

public class CustomDialog extends JDialog implements ActionListener {

	private String entered;
	JTextField tf;


	public CustomDialog(JFrame owner) {

		super(owner, "Ozel Dialog Penceresi", true);
		Container c = getContentPane();
		c.setLayout(new FlowLayout());
```

```
            tf = new JTextField("Enter val", 10);
            JButton b = new JButton("Tamam");
            b.addActionListener(this);

            c.add(tf);
            c.add(b);

            setSize(300, 80);
            setResizable(false);
      }

      String getValue() {

            show();

            return entered;

      }

      public void actionPerformed(ActionEvent e) {

            entered = tf.getText();
            setVisible(false);

      }
}



import java.awt.event.*;
import java.awt.*;
import javax.swing.*;

public class DialogTesFrame extends JFrame implements ActionListener {


      Container c;

      DialogTesFrame() {
            c = getContentPane();
            c.setLayout(new FlowLayout());

            JButton dugme = new JButton("Dialog Goster");
            dugme.addActionListener(this);
            c.add(dugme);

            this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setSize(600, 600);
            show();

      }

      public void actionPerformed(ActionEvent e) {

            CustomDialog cd = new CustomDialog(this);
            System.out.print(cd.getValue());
      }

      public static void main(String arg[]) {

            new DialogTesFrame();
```

```
        }

}
```

FileDialog Nesneleri
Kullandığımız programların birçoğunda bilinen şekliyle open ve save menu seçenekleriyle birlikte gelen ve sistemde herhangi birdosyayı "gözat" yöntemiyle bulup seçebileceğimiz dialog pencerelerine FileDialog pencereleri adı verilir.

Java'da bu işlemi JFileChooser nesnesiyle kolaylıkla yapabiliyoruz. Bu nesnenin iki tane metodu ile dosya açmak ya da dosya kaydetmek için pencereyi görüntülemek mümkündür:

- showOpenDialog
- showSaveDialog

```
JFileChooser  fc = new JFileChooser();
fc.setCurrentDirectory(new File(".”));
fc.setMultiSelectionEnabled(true);   //birden fazla dosya seçimine izin verir.

fc.setFileSelectionMode(FileChooser.FILES_ONLY);
fc.setFileSelectionMode(FileChooser.DIRECTORIES_ONLY);
fc.setFileSelectionMode(FileChooser.FILES_AND_DIRECTORIES);


int result = fc.showOpenDialog(parent);
int result = fc.showSaveDialog(parent);
//JFileChooser.APPROVE_OPTION ya da JFileChooser.CANCEL_OPTION geri dönecektir.

String filename = fc.getSelectedFile().getPath();
```

Paketler
Java da bir grup sınıf ya da paketi bazı genel paketler alttında topladığımızı daha önceden biliyoruz. Bunun nedeni hem aynı konuya ilişkin çalışmaları tek bir hiyerarşi altında toplamak hem de birtakım isim çakışmalarını önlemektir.

Öyle ki eğer A isimli bir sınıfımızı kendi paketimiz içerisinde yazarsak, bir başka A isimli sınıf ancak başka bir paket içerisinde yer alabilir ve böylece isim çakışmaları engellenmiş olur.

Bir sınıf kendi paketi içerisindeki tüm sınıflara ve başka paketler içerisindeki public sınıflara erişebilir. Bu erişim için ya ilgili paketin tam adını kullnarak sınıf adıyla erişim ya da sadece paketi import ederk sınıf adıyla erişim yapmak gerekir. Örneğin com,academytech isimli paket içerisinde A isimli bir sınıfa erişmek için:

a) import com.academytech.A ya da com.academytech.*;
   ...
   A a = new A();
   ...

b) ...
   com.academytech.A a = new com.academytech.A();

şeklinde erişim yapılır.

Java da bir paketin içerisindeki tüm sınıflar import edildiği zaman gereksiz bir kod büyümesi olmaz. Bu konuda bir performans düşüklüğü olmaz.

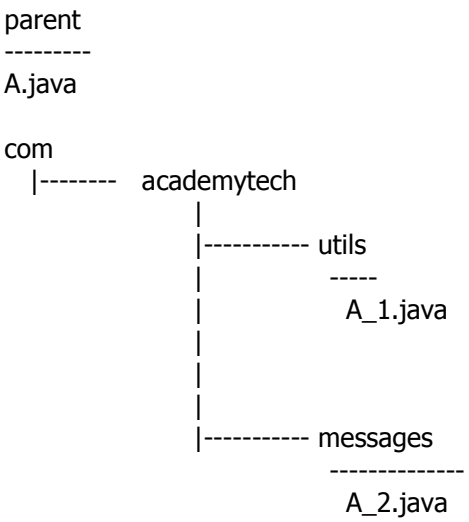
Paketlere Sınıfların Eklenmesi

Java da bir sınıfı belirli paketin içerisine dahil etmek istiyorsak, dahil edilmek istenen sınıf bildiriminin başına "package" anahtar sözcüğü ile ait olduğu hiyerarşi yapısını yazarız.

Mesela;

A herhangi bir yerde bir sınıf olarak yaratılsın.
A_1 sınıfı **com.academytech.utils** paketine ait olsun.
A_2 sınıfı **com.academytech**. **messages** paketinde olsun.

Diyelim ki A sınıfı kendi içinde A_1 ve A_2 sınıflarını kullansın. Bu durumda bu sınıflar disk üzerinde aşağıda dizin yapısına göre yerleştirilir:



A_1 ve A_2 sınıflarının bildirimleri şu şekilde yapılır:

```
/**** A_1.java ***/
package com.academytech.utils;
class A_1 {

}


/*** A_2.java ***/
package com.academytech.messages;
class A_2 {

}
```

Bu paketler içerisindeki A_1.java ve A_2.java sınıflarının derlenme ve yorumlanma aşamaları da şu şekildedir:

```
# javac com/academytech/utils/A_1.java
# java com.academytech.utils.A_1
```

Bu durumda A sınıf içerisinde A_1 ve A_2 nesneleri kullanabilmek için A sınıfının içinde:

```
/*** A.java ***/
import com.academytech.utlis.*;
import com.academytech.messages.*;

class A {
```

```
        ...
        A_1 a = new A_1(...);
        ...
        A_2 b = new A_2(...);

}
```

JVM'nin Paketleri Bulması
Yukarıdaki örnekte görmekteyiz ki A sınıfı içerisinde kullanılan harici paketler sınıf ile aynı dizin altında yer alıyor.

Ancak bu durum çok esnek değildir. Yani her zaman kullanılan paketler, bu paketleri kullnadığımız sınıflarla aynı dizinde yer alamyabilir. Bu durum zaten projenin esnekliğini ve taşınabilirliğini de ortadan kaldıracağı için pek tercih edilmez.

Birden fazla farklı paketin çalışılan dizinden bağımsız olarak kullanılabilmesi için:

1) paketlerin yer alacağı bir dizin yaratılır. Örneğin: **/home/academytech/javapaketleri**

2) Daha önce sizin ya da başkalarının yazdığı bu paketler parent dizinleri ile birlikte (com gibi) bu ana dizin altına kopyalanır. Bizim örneğimiz için **com/academytech** dizini **/home/academytech/javapaketleri** dizini altına kopyalanır.

3) class path ayarı yapılır. CLASSPATH, JVM'ye aranacak paketlerin nerelerde aranacağını belirten bir ayardır. (linux'de export ile windows'da environment variables ayarı ile yapılabilir. Bkz. ders 1)

Herhangi bir yerdeki sınıfı derlerken derleme ve yorumlama aşamasında da classpath bilgisi verilebilir:

```
# javac –classpath (ya da -cp) /home/academytech/javapaketleri:.  A.java
# java –classpath /home/academytech/javapaketleri:.    A
```

Bu örnekte A.java com.academytech paketi altındaki A_1 ve A_2 sınıflarını kullanıyor.

Normalde compiler ilk olarak jre/lib ve jre/lib/ext dizinleri altındaki sistem kütüphanelerinde gerekli sınıfları arar. Burada bulamadığı sınıfları da classpath içerisinde arar.

Paket Faliyet Alanı

Bir sınıf, metod ya da değişken public ise başka herhangi bir sınıf tarafından kullanılabilir.

Bir sınıf, metod ya da değişken private ise sadece kendi dosyası içerisindeki başka sınıflar tarafından kullanılabilir.

Bir sınıf, metod ya da değişken bildiriminde public ya da private anahtar sözcüğü yer almıyorsa o zaman o sınıf, metod ya da değişkene sadece kendi paketindeki diğer sınıflar erişebilir.

Dökümantasyon Yaratma (javadoc)
Java'da kod içerisine yazılacak birtakım etiketler sayesinde profosyonel bir dökümantasyon hazırlamak mümkündür.

Aslında java'nın kendi dökümantasyonu da az sonra bahsedilecek olan yöntemlerle hazırlanmıştır.

javadoc denilen komut kod içerisindeki bu özel etiketleri algılayarak istenilen dizinler altında dökümantasyonu .html sayfaları olarak üretir.

Genel olarak dökümantasyon paketler, public sınıflar ve arayüzler, public ve protected metodlar ile public ve protected veri elemanları için hazırlanır.

Her dökümantasyon komutu /** ve */ sembolleri arasında ve dökümante edilecek paket, sınıf, metod ya da değişkenin hemen üstünde yer alır.

```
@author   yazar adi
@return <açıklama>: metodun geri dönüş değerine ilişkin bir açıklama
@param <deger> <aciklama>: parametrenin açıklanması
@see "<bir nesne>" : See Also kısmının hazırlanması
Bu etiketlerin yanı sıra bilinen tüm html etiketleri de kullanılabilir. Bunlardan
bazıları:

<code>  : Kod stili şeklinde yazma
<img >   : imaj ekleme
<strong> : strong vurgu yapmak
```

gibi..

Eclipse'de File/Export menu seçeneği yardımıyla otomatik olarak oluşturulabilir. Ancak bu ouşturma aşamasında Eclipse sizden "javadoc" komutunun lokasyonunu girmenizi isteyecektir.

JAR Dosyalarının Oluşturulması

Jar dosyalarını oluşturmaktaki temel amaç uygulamanın tek bir arşivde toplanması hatta tek bir dosya şeklinde çalıştırılmasını sağlamaktır.

Aslında genel olarak bakıldığında jar dosyası oluşturmak birkaç küçük adımdan ibarettir.

Bir jar arşivi yaratmak için:

#jar cvf <arsiv_adi>.jar <arsivlenecek_dosya1> [ve/veya] <arşivlenecek_dosya2> [ve/veya] *.java

Örneğin bulundugum dizin altındaki A.java, B.java ve C.java dosyaları ile a.gif dosyasını içeren bir .jar arşivi oluşturmak için:

#jar cvf program.jar A.class B.class C.class a.gif

yazmak yeterlidir.

Bir arşivin içerisine ayrıca "manifest" adı verilen özel bir dosya daha yerleştirilir. Bu dosya aslında arşiv hakkında birtakım konfigurasyon bilgileri verir.

Aslında genel java sistem kütüphaneleri bu şekilde arşivler içerisinde tutularak kullanılmaktadır. SDK'nın runtime kütüphanesi olan rt.jar böyle bir arşivdir.

Sonuç olarak .jar arşivleri bir programın ihtiyaç duyacağı tüm dosyaları barındıran sıkıştırılmış bir dosyadır. Ayırca arşivin tüm teknik bilgilerini barındıran MANIFEST.MF dosyası da arşivin içerisinde yer alır.

Manifest Dosyasının İçeriği

/*** MANIFEST.MF ***/
Manifest-Version: 1.0
<arsivi tanimlayan satirlar>

Name: Person.class
<Sınıfı tanımlayan satırlar>

Name: com/academytech/

<bu paketi tanımlayan satırlar>

Bu şekilde bir manifest dosyasını, com/academytech paketini ve Genel sınıfını içeren bir arşiv yaratmak için:

#jar cfm program.jar manifest.mf com/academytech/*.class A.class

Kendi başına çalışabilen bir arşiv yaratmak için manifest dosyası içerisinde bir de bu programı çalıştıracak main metodunu içeren sınıfn bilgisi de eklenmelidir:

Main-Class: com/academytech/Main

Bundan sonra arşiv şu şekilde çalıştırılabilir:

java –jar program.jar

Konsolda basit bir uygulama:

1) Aşağıdaki iki sinifi yaziniz.

```
import com.academytech.*;

class A {


        public static void main(String arg[]) {
                B b = new B();
                b.show();
        }
}
```

```
package com.academytech;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class B extends JFrame implements ActionListener {

        Container c;

        public B() {

                c = getContentPane();
                c.setLayout(new FlowLayout());
                JButton b = new JButton("submit");
                b.addActionListener(this);
                c.add(b);

                setSize(300, 400);

        }

        public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "Merhaba");
        }

        public static void main(String arg[]) {
                B b = new B();
                b.show();
        }
}
```

2) B.java sınıfını com/academytech dizini altına yerleştiriniz. Daha sonra bu com dizinini yerleştirdiğniz lokasyonu classpath olarak ayarlayınız.

3) Aşağıdaki gibi manifest.mf dosyasını oluşturunuz:

   /*** manifest.mf ***/
   Manifest-Version: 1.0
   Main-Class: A

   /*************/**
   NOT: Main-Class: A yazdıktan sonra bir satır boşluk bırakıldığına dikkat ediniz.
4) komut: java cvfm program.jar manifest.mf A.class com\academytech\B.class
5) sonuc: java –jar program.jar

Java Derleri
Bölüm 20

Java Collections
Bu kısımda programcılık açısından oldukça önemli olan veri yapıları kavramlarının Java'da nasıl ele alındığını inceleyeceğiz.

Aslında veri yapıları kavramları genel olarak oldukça fazal ve detaylı bilgileri içerir. Biz ise burada sadece en temel olanları ele alacağız. Bu kısım tamalandıktan sonra elinizdeki herhangi bir veri yapısını Java Programalama dilinde kolaylıkla çevirebileceğinizi umuyoruz.

Collection ve Iterator Arayüzü Kavramı
Java'da veri yspıları mekanizmalarını ele alan ve herbirini daha sonra ele alacağımız bu taşıyıcıların hepsi Collection denilen bir arayüzden türemiştir.

Bir Collection arayüzünün temel olan iki metodu

```
boolean add(Object obj)
Iterator iterator()
```

itarator isimli metod kullanılan taşıyıcı üzerinde Iterator sınıfını implement eden bir sınıfı geri gönderir. Bu Iterator arayzünün temel metodları:

```
Object next()
boolean hasNext()
void remove()
```

bu Iterator nesnesi yardımıyla taşıyıcı üerindeki elemanlar tek tek ziyaret edilebilir. Ancak taşıyıcının sonuna gelindiğinde NoSuchElementException üretilir. Bu nedenle taşıyıcı üzerinde gezerken hasNext metoduyla kontrol yapılarak gezme işlemi sürdürülür:

```
//c bir collection olarak yaratılmış olsun
Iterator iter = c.iterator();
while(iter.hasNext()) {
      Object obj = iter.next();
      …
}
```

remove metodu ile iteraturun next ile elde edilen son eleman listeden silinir. Yani bu kullanım bir eleman elde edilmeden silinmesine engel olmaktadır. Önce next ile eleman kontrol edilmeli ve sonra remove edilmelidir. Bir eleman next ile geçilmeden önce remove ile silinemez.

```
Iterator iter = c.iterator();
Object obj = iter.next();  sıradki eleman alındı.
```

```
iter.remove(); az once alınan eleman silindi.
```

Normalde kullanıcılar kendileri de isterlerse Collection ve Iterator sınıflarını implement edebilirler. Hatta bunu arayüzlerin tüm metdlarını implement etmek yğrucu gelebileceği için AbstrackCollection sınıfı da implement edilebilir. Bu arayüzlerin içerdikleri metdoların listesine dökümantasyon üzerinden erişilebilir.

LinkedList
LinkedList nesnesi bilinen bağlı liste implementasyonunu gerçekler.

```
LinkedList  ogrenciler = new LinkedList();
ogrenciler.add("Academytech");
ogrenciler.add("Caglar");
ogrenciler.add("Ahmet");
```

LinkedList nesnesi için kullanılan iterator ListIterator olarak isimlendirilir. Bu iterator'de Iterator arayüzünden olmayan "add" metodu yer alır. Böylece iterator bağlı liste üzerinde istenilen pozisyona getirilerek yeni bir eleman araya eklenebilir.

```
ListIterator  iter = ogrenciler.iterator();
```

Ayrıca ListIterator nesnesinde yer alan "previous" ve "hasPrevious" metodları ile de geri yönlü bir tarama yapılabilir.

ListIterator nesnesinin "set" isimli metodu next ile elde edilen son elemanı yeni bir elemanla değiştirmeye yarar.

```
Object eskiDeger = iter.next();
iter.set(yeniDeger);
```

ÖDEV:
Aşağıdaki örnekte henüz operasyonları tanımlanmamış olan Next, Previous, Add New Object ve Remove Object metodlarını hazırlayınız ve programı tamamlayınız.

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.LinkedList;
import java.util.ListIterator;

import javax.swing.JButton;
```

```
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;


public class LinkedListHandler extends JFrame implements ActionListener {

	JButton add_1, list, next, remove, previous, add_2;
	JTextArea screen;
	JPanel tourPanel, mainPanel;
	Container c;
	LinkedList objectlist;

	public LinkedListHandler() {

		prepareComponents();
		prepareFrame(600, 300, false, true);

	}

	private void prepareFrame(int x_size, int y_size, boolean resizable, boolean
show) {

		this.setSize(x_size, y_size);
		this.setResizable(resizable);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		if(show)
			this.show();



	}

	private void prepareComponents() {

		c = getContentPane();


		add_1 = new JButton("Add");
		add_1.addActionListener(this);
```

```
        list = new JButton("List All");
        list.addActionListener(this);

        next = new JButton("Next Object");
        next.addActionListener(this);

        previous = new JButton("Previous Object");
        previous.addActionListener(this);

        remove = new JButton("Remove Current Object");
        remove.addActionListener(this);

        add_2 = new JButton("Add New Object");
        add_2.addActionListener(this);

        tourPanel = new JPanel();
        tourPanel.add(next);
        tourPanel.add(previous);
        tourPanel.add(remove);
        tourPanel.add(add_2);
        c.add(tourPanel, BorderLayout.SOUTH);

        mainPanel = new JPanel();
        mainPanel.add(list);
        mainPanel.add(add_1);
        c.add(mainPanel, BorderLayout.NORTH);

        screen = new JTextArea();
        screen.setEditable(false);
        screen.setBackground(Color.BLACK);
        screen.setForeground(Color.GREEN);
        screen.setFont(new Font("Serif", Font.BOLD, 24));
        c.add(screen, BorderLayout.CENTER);

    }

    private void prepareButton(JButton button, String button_name, ActionListener
listener) {

        button = new JButton(button_name);
        button.addActionListener(listener);
    }
```

```
	public void actionPerformed(ActionEvent e) {

		if(e.getSource() == add_1) {

			if(this.objectlist == null)
				this.objectlist= new LinkedList();

			if(this.objectlist.add(JOptionPane.showInputDialog(this, "New
Record")));
				screen.setText("Yeni kayit eklendi...");

			screen.validate();

		}

		else if(e.getSource() == this.list) {

			StringBuffer buf = new StringBuffer("");

			if(this.objectlist != null) {
				ListIterator iter = (ListIterator)objectlist.iterator();
				while(iter.hasNext())
					buf.append("\n"+(String)iter.next());
			}

			else
				buf.append("nothing is recorded yet!!!");

			screen.setText(buf.toString());
			screen.validate();
		}
	}

	public static void main(String[] args) {
		new LinkedListHandler();
	}
}
```

Vector Nesnesi
Bu nesne aslında diziler gibi bir veri yapısını meydana getirir .Ancak dizilerde olmayan dinamik olarak büyüme özelliği de Vector nesneleri ile birlikte gelir.

Herhangi bir zaman dilimi içerisinde bir Vector nesnesi kendi kapasitesinden daha az ya da eşit sayıda nesne barındırır. Eğer kapasitesi dolduysa ve büyümesi gerekiyorsa o zaman istenilen miktarda dinamik olarak genişletilebilir.

Eğer kullanıcı herhangi bir kapasite artış miktarı belirtmezse o zaman vektor kendi boyutunun iki katı genişliğe büyütülür.

Vector nesnesi de diğer yapılar gibi Object nesnelerini taşır.

NOT: Vector nesnelerinin ArrayList nesnelerinden farkı senkron edilmelidir. Eğer birden fazla thread Vector üzerinde kullnılıyorsa Vector nesnesi sürekli senkron olur. Ancak ArrayList böyle bir işlem yapmaz. Eğer birden fazla thread kullanılımıyorsa senkron için zaman harcamadığından dolayı ArrayList tercih edilebilir.

Enumaration Nesnesi ile Vector içerisinde dolaşmak
Aşağıdaki örnekte kullanılan Enumaration nesnesi Vector sınıfının elements isimli metodu tarafından gönderilen bir nesnedir. Bu nesne vector içerisinde dolaşabilmeyi sağlayan iki tane metoda sahiptir:

```
nextElement()
hasMoreElements()
```

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class VectorTest extends JFrame {

   public VectorTest()
   {
      super( "Vector Example" );

      final JLabel status = new JLabel();
      Container c = getContentPane();
      final Vector v = new Vector( 1 );

      c.setLayout( new FlowLayout() );

      c.add( new JLabel( "Enter a string" ) );
      final JTextField input = new JTextField( 10 );
      c.add( input );

      JButton addBtn = new JButton( "Add" );
      addBtn.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
```

```
            v.addElement( input.getText() );
            status.setText( "Added to end: " +
                            input.getText() );
            input.setText( "" );
         }
      }
   );
   c.add( addBtn );      // add the input value

   JButton removeBtn = new JButton( "Remove" );
   removeBtn.addActionListener(
      new ActionListener() {
         public void actionPerformed( ActionEvent e )
         {
            if ( v.removeElement( input.getText() ) )
               status.setText( "Removed: " +
                               input.getText() );
            else
               status.setText( input.getText() +
                               " not in vector" );
         }
      }
   );
   c.add( removeBtn );

   JButton firstBtn = new JButton( "First" );
   firstBtn.addActionListener(
      new ActionListener() {
         public void actionPerformed( ActionEvent e )
         {
            try {
               status.setText( "First element: " +
                               v.firstElement() );
            }
            catch ( NoSuchElementException exception ) {
               status.setText( exception.toString() );
            }
         }
      }
   );
   c.add( firstBtn );

   JButton lastBtn = new JButton( "Last" );
   lastBtn.addActionListener(
      new ActionListener() {
         public void actionPerformed( ActionEvent e )
         {
            try {
               status.setText( "Last element: " +
                               v.lastElement() );
            }
            catch ( NoSuchElementException exception ) {
               status.setText( exception.toString() );
            }
         }
      }
   );
   c.add( lastBtn );
```

```
JButton emptyBtn = new JButton( "Is Empty?" );
emptyBtn.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         status.setText( v.isEmpty() ?
            "Vector is empty" : "Vector is not empty" );
      }
   }
);
c.add( emptyBtn );

JButton containsBtn = new JButton( "Contains" );
containsBtn.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         String searchKey = input.getText();

         if ( v.contains( searchKey ) )
            status.setText( "Vector contains " +
                            searchKey );
         else
            status.setText( "Vector does not contain " +
                            searchKey );
      }
   }
);
c.add( containsBtn );

JButton locationBtn = new JButton( "Location" );
locationBtn.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         status.setText( "Element is at location " +
                         v.indexOf( input.getText() ) );
      }
   }
);
c.add( locationBtn );

JButton trimBtn = new JButton( "Trim" );
trimBtn.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         v.trimToSize();
         status.setText( "Vector trimmed to size" );
      }
   }
);
c.add( trimBtn );

JButton statsBtn = new JButton( "Statistics" );
statsBtn.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         status.setText( "Size = " + v.size() +
            "; capacity = " + v.capacity() );
      }
   }
```

```
        );
        c.add( statsBtn );

        JButton displayBtn = new JButton( "Display" );
        displayBtn.addActionListener(
           new ActionListener() {
              public void actionPerformed( ActionEvent e )
              {
                 Enumeration enum = v.elements();
                 StringBuffer buf = new StringBuffer();

                 while ( enum.hasMoreElements() )
                    buf.append(
                           enum.nextElement() ).append( "   " );

                 JOptionPane.showMessageDialog( null,
                    buf.toString(), "Display",
                    JOptionPane.PLAIN_MESSAGE );
              }
           }
        );
        c.add( displayBtn );
        c.add( status );

        setSize( 300, 200 );
        show();
     }

     public static void main( String args[] )
     {
        VectorTest app = new VectorTest();

        app.addWindowListener(
           new WindowAdapter() {
              public void windowClosing( WindowEvent e )
              {
                 System.exit( 0 );
              }
           }
        );
     }
}
```

Stack Nesnesi

Bu nesne Vector nesnesinde türetilerek elde edilmiş ve Stack adı verilen veri yapılarını ele almak için tasarlanmıştır.

Vector sınıfı gibi, Stack sınıfı içerisinde Object türünden nesneler saklanmaktadır. Eğer primitive type verileri bu yapılar içinde saklamak isterseniz bunları daha önce incelediğimiz Integer, Double gibi sargı sınıfları yardımıyla nesne haline getirerek kullanabilirsiniz.

Bir stack nesnesi içerisine yeni bir eklemek "push" denilen bir operasyon ile gerçekleşir. Stack içerisine bir veri push ile eklendikten sonra "pop" ile elde dilebilir. Bu durumda Stack veri yapısı üzerinde LIFO (Last In First Out) mantığı geçerlidir.

Aşağıdaki örnek Deitel&Deitel tarafından yazılan Java HowTo Program, Third Edition kitabından değiştirilmeden alınmıştır.

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class StackTest extends JFrame {

   public StackTest()
   {
      super( "Stacks" );

      Container c = getContentPane();

      final JLabel status = new JLabel();
      final Stack s = new Stack();

      c.setLayout( new FlowLayout() );
      c.add( new JLabel( "Enter a string" ) );
      final JTextField input = new JTextField( 10 );
      c.add( input );

      JButton pushBtn = new JButton( "Push" );
      pushBtn.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
               status.setText( "Pushed: " +
                  s.push( input.getText() ) );
            }
         }
      );
      c.add( pushBtn );

      JButton popBtn = new JButton( "Pop" );
      popBtn.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
               try {
                  status.setText( "Popped: " + s.pop() );
               }
               catch ( EmptyStackException exception ) {
                  status.setText( exception.toString() );
               }
            }
         }
      );
      c.add( popBtn );

      JButton peekBtn = new JButton( "Peek" );
      peekBtn.addActionListener(
         new ActionListener() {
```

```
      public void actionPerformed( ActionEvent e )
      {
         try {
            status.setText( "Top: " + s.peek() );
         }
         catch ( EmptyStackException exception ) {
            status.setText( exception.toString() );
         }
      }
   }
);
c.add( peekBtn );

JButton emptyBtn = new JButton( "Is Empty?" );
emptyBtn.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         status.setText( s.empty() ?
            "Stack is empty" : "Stack is not empty" );
      }
   }
);
c.add( emptyBtn );

JButton searchBtn = new JButton( "Search" );
searchBtn.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         String searchKey = input.getText();
         int result = s.search( searchKey );

         if ( result == -1 )
            status.setText( searchKey + " not found" );
         else
            status.setText( searchKey +
               " found at element " + result );
      }
   }
);
c.add( searchBtn );

JButton displayBtn = new JButton( "Display" );
displayBtn.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         Enumeration enum = s.elements();
         StringBuffer buf = new StringBuffer();

         while ( enum.hasMoreElements() )
            buf.append(
               enum.nextElement() ).append( " " );

         JOptionPane.showMessageDialog( null,
             buf.toString(), "Display",
             JOptionPane.PLAIN_MESSAGE );
      }
   }
);
c.add( displayBtn );
c.add( status );
```

```
        setSize( 675, 100 );
        show();
    }

    public static void main( String args[] )
    {
        StackTest app = new StackTest();

        app.addWindowListener(
            new WindowAdapter() {
                public void windowClosing( WindowEvent e )
                {
                    System.exit( 0 );
                }
            }
        );
    }
}
```

Örnekte kullanılan "peek" isimli metod stack nesnesi üzerinde herhangi "pop" işlemi yapmadan en yukarıdaki elemanı görüntülemeye yarar.

Bu örnek içerisinde de stack içerisinde dolaşmak için yine Enumaration isimli nesne elde dilmiştir. Stack nesnesinin aslında Vector nesnesinden bir türetme olduğu buradan da anlaşılabilir.

HashTable Nesnesi

Hashing işlemi ve bu işlemi gerçekleştirmek için kullanılan algoritma oldukça detaylı bir konudur. Burada önemli olan HashTable nesnesinin verileri nasıl ele aldığıdır.

HashTable verilerin fazla sayıda olduğu ve sırasından ziyade çabuk ulaşılabilir olmasıın tercih edildiği durumlarda kullanılan bir yapıdır. İç mekanizması hashing işlemine dayanan bu yapı verilere çok hızlı bir şekilde erişebilmeyi sağlar. Ancak bu erişimde verilerin yerleşimi Vector ve LinkedList yapılarında olduğu gibi belirli bir sırada değildir. Onun yerine her veri için özel bir hesapla atanan "hash number" anahtarı yardımıyla her elemanı LinkedList olan bir diziye bu veriler yerleştirilir. Bu hash number aslında verinin lokasyonu konusunda bilgi içerdiğinden verilere erişim oldukça hızlı bir şekilde gerçekleşir.

put metodu belirli bir key ile birlikte yeni bir nesne eklemeye yarar. Ancak bu metodu geri dönüş değeri Object'tir. Eğer eklenecek olan anahtar tabloda zaten varsa, bu abahtarın gösterdiği nesne kullanıcıya geri döner. Eğer eklenecek olan anahtar tabloda yoksa o zaman null değeri gei dönecektir.

get metodu kendisine verilen anahtarla eşleşmiş olan Object nesnesini geri gönderir. Eğer böyle bir nahtarla ilişkilendirilmiş bir nesne tabloda yoksa, o zaman null değer geri dönecktir.

elements ve keys metodları, tablodaki anahtar ve nesnelerin iteratörlerini yani Enumaration nesnelerini geri gönderir.

Aşağıdaki örnek Deitel&Deitel tarafından yazılan Java HowTo Program, Third Edition kitabından değiştirilmeden alınmıştır.

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class HashTableTest extends JFrame {

   public HashTableTest()
   {
      super( "Hashtable Example" );

      final JLabel status = new JLabel();
      final Hashtable table = new Hashtable();
      final JTextArea display = new JTextArea( 4, 20 );
      display.setEditable( false );

      JPanel northPanel = new JPanel();
      northPanel.setLayout( new BorderLayout() );
      JPanel northSubPanel = new JPanel();
      northSubPanel.add( new JLabel( "First name" ) );
      final JTextField fName = new JTextField( 8 );
      northSubPanel.add( fName );

      northSubPanel.add( new JLabel( "Last name (key)" ) );
      final JTextField lName = new JTextField( 8 );
      northSubPanel.add( lName );
      northPanel.add( northSubPanel, BorderLayout.NORTH );
      northPanel.add( status, BorderLayout.SOUTH );

      JPanel southPanel = new JPanel();
      southPanel.setLayout( new GridLayout( 2, 5 ) );
      JButton put = new JButton( "Put" );
      put.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
               Employee emp = new Employee(
                  fName.getText(), lName.getText() );
               Object val = table.put( lName.getText(), emp );

               if ( val == null )
                  status.setText( "Put: " + emp.toString() );
               else
                  status.setText( "Put: " + emp.toString() +
                     "; Replaced: " + val.toString() );
            }
         }
      );
      southPanel.add( put );

      JButton get = new JButton( "Get" );
      get.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
               Object val = table.get( lName.getText() );

               if ( val != null )
```

```
            status.setText( "Get: " + val.toString() );
         else
            status.setText( "Get: " + lName.getText() +
                            " not in table" );
      }
   }
);
southPanel.add( get );

JButton remove = new JButton( "Remove" );
remove.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         Object val = table.remove( lName.getText() );

         if ( val != null )
            status.setText( "Remove: " +
                            val.toString() );
         else
            status.setText( "Remove: " +
               lName.getText() + " not in table" );
      }
   }
);
southPanel.add( remove );

JButton empty = new JButton( "Empty" );
empty.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         status.setText( "Empty: " + table.isEmpty() );
      }
   }
);
southPanel.add( empty );

JButton containsKey = new JButton( "Contains key" );
containsKey.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         status.setText( "Contains key: " +
            table.containsKey( lName.getText() ) );
      }
   }
);
southPanel.add( containsKey );

JButton clear = new JButton( "Clear table" );
clear.addActionListener(
   new ActionListener() {
      public void actionPerformed( ActionEvent e )
      {
         table.clear();
         status.setText( "Clear: Table is now empty" );
      }
   }
);
southPanel.add( clear );

JButton listElems = new JButton( "List objects" );
```

```
      listElems.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
               StringBuffer buf = new StringBuffer();

               for ( Enumeration enum = table.elements();
                     enum.hasMoreElements(); )
                  buf.append(
                     enum.nextElement() ).append( '\n' );

               display.setText( buf.toString() );
            }
         }
      );
      southPanel.add( listElems );

      JButton listKeys = new JButton( "List keys" );
      listKeys.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
               StringBuffer buf = new StringBuffer();

               for ( Enumeration enum = table.keys();
                     enum.hasMoreElements(); )
                  buf.append(
                     enum.nextElement() ).append( '\n' );

                  JOptionPane.showMessageDialog( null,
                     buf.toString(), "Display",
                     JOptionPane.PLAIN_MESSAGE );
            }
         }
      );
      southPanel.add( listKeys );
      Container c = getContentPane();
      c.add( northPanel, BorderLayout.NORTH );
      c.add( new JScrollPane( display ), BorderLayout.CENTER );
      c.add( southPanel, BorderLayout.SOUTH );

      setSize( 540, 300 );
      show();
   }

   public static void main( String args[] )
   {
      HashTableTest app = new HashTableTest();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               System.exit( 0 );
            }
         }
      );
   }
}

class Employee {
   private String first, last;
```

```
    public Employee( String fName, String lName )
    {
       first = fName;
       last = lName;
    }

    public String toString() { return first + " " + last; }
}
```

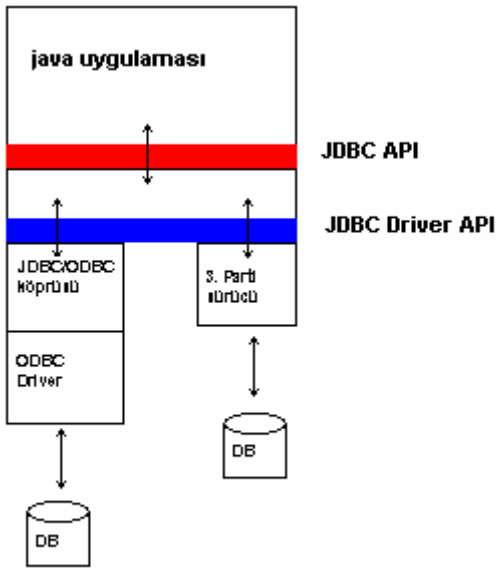
JDBC Teknolojisi

1996 yazında Sun JDBC aracının ilk versiyonunu duyurdu. Böyle bir aracın geliştirilmesindeki asıl maksat tek bir arayüz üzerinde birbirinden bağımsız veri tabanlarına erişimi sağlamaktı.

Bu erişim sağlandıktan sonra standart SQL dili ile veri tabanı üzerinde sorgu yapılabilecekti.

Daha özel olarak bahsetmek gerekirse JDBC iki katmandan meydana gelir. En yukarıda JDBC arayüzü vardır. Bu API JDBC sürücü yöneticisi ile haberleşerek ona birtakı SQL sorguları gönderir.

Sürücü yöneticisi de kendisi de API den gelen bu sorguları ele alır ve 3. parti veri tabanı sistemleri ile iletişim kurarak sorguları bu veri tabanları üzerinde çalıştırır ve sonuçları tekrar API'ye gönderir.



Class.forName("COM.ibm.db2.jdbc.app.DB2Driver"):

Bu bildirim yukarıdaki şemada belirtilen JDBC Driver API katmanının ihtiyaç duyacağı 3. parti sürücüyü kayıt etmek için (yüklemek için) kullanılır. O halde aslında bu bildirim için gerekli olan bu sürücü (burada **"COM.ibm.db2.jdbc.app.DB'Driver")** 3. parti veri tabanı üreticileri tarafından sunulmalıdır.

Böyle bir sürücü sisteme indirildikten sonra CLASSPATH tarafından görülen bir lokasyona ya da java'nın default olarak görebildiği herhangi bir lokasyona yerleştirilir. Böylece artık yukarıdaki bildirimi kullanmak için bir engel kalmaz. İndirdiğiniz sürücü .zip ya ya da .jar uzantılı olabilir. CLASSPATH içerisine koyulacak olan path bilgisinde bu .zip ya da .jar uzantılı dosya da yer almalıdır. Örneğin sürücümüz mysql.jar olsun:

CLASSPATH = /bir/yer; ....; /home/drivers/ java/mysql.jar

Database URL
Database sistemde yüklendikten ve yukarıdaki gibi sürücü yüklendikten sonra veri tabanına bağlanma aşamasına gelinmiştir. Bu aşamada bağlanmak için kullanılan veri tabanı sistemine uygun bir URL ile bağlantı kurulur. Bu URL'nin en genel hali:

jdbc:<alt protokol ismi>:<diğer bilgiler>

şeklindedir. Bu URL 3. parti veri tabanı sistemi tarafından sağlanır. Bu url kullanılarak

Connection conn=DriverManager.getConnection(url, username, password);

Bilgisi ile veri tabanına JDBC API üzerinden bağlantı sağlanır.

SQL Komutlarının Çalıştırılması
SQL komutlarının çalıştırılması için öncelikle Statement türünden bir nesne yaratılır. Bunun için öceden yaratılmış olan Connection nesnesi üzerinden createStatement() metodu çağrılır.

Statement st = conn.createStatement();

Bu Statement nesnesi üzerinde

int executeUpdate(String query):

```
st.executeUpdate("UPDATE Books SET Price = Price – 500 WHERE Title NOT LIKE '%HTML 3%'");
```

gibi bir tablo güncelleme komutu çalıştırılabilir. Bu metodun geri dönüş değeri bu güncellemeden etkilenen satır sayısıdır.

executeUpdate komutu ile INSERT, UPDATE, DELETE, CREATE TABLE ve DROP TABLE operasyonları çalıştırılabilir.

ResultSet executeQuery(String query):

Ancak bir sorgu çalıştırıldığında gelecek olan sonuçlarla ilgileniriz. Statement nesnesinin executeQuery metodu bize bu sonuçları içeren bir ResultSet nesnesi gönderir.

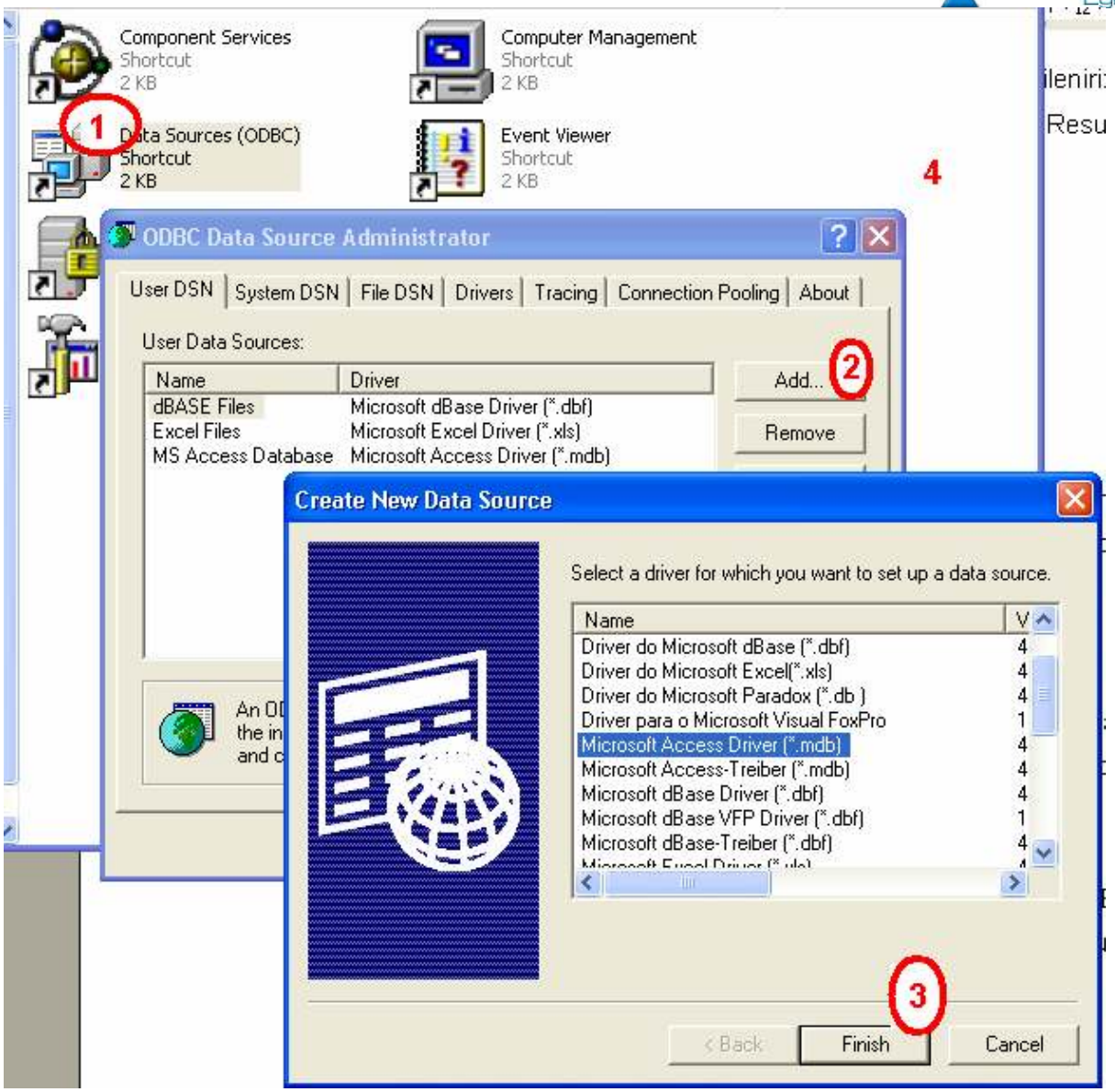
```
ResultSet rs = st.executeQuery("SELECT * FROM Books");
while(rs.next()) {
      String name = rs.getString("Name");
      ...
}
```

Böylece ResultSet içerisine gelen her satır rs.next() ile elde edilirken bu satır içerisindeki her sütun kendi adıyla birlikte rs.getString("<sütun adi>") nesnesi içerisinde kullanılır ve değerler elde edilir.

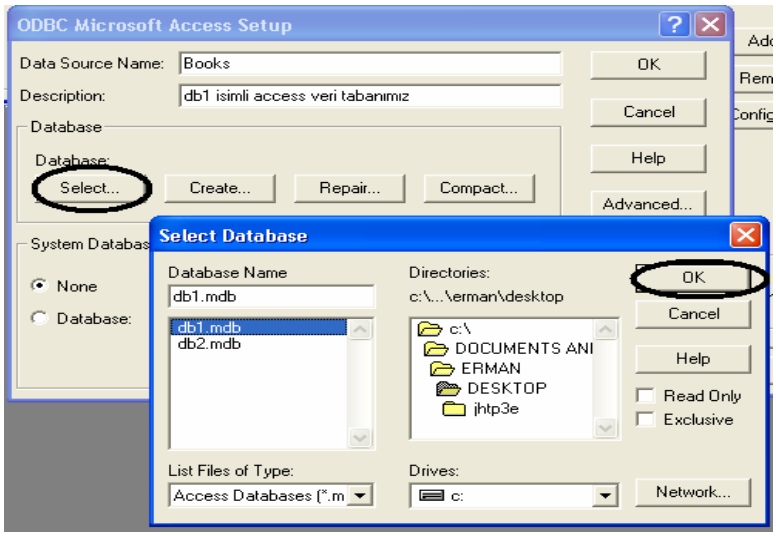
JDBC-ODBC Köprüsü ile Access Veri Tabanına Bağlantı

1. Sistemde Access veri tabanı yüklü olmalıdır. Ayrıca bu veri tabanı sistemi üzerinde oluşturulmuş bir veri tabanı hazır olmalıdır. (Bu örnekte Books.mdb kullanılacak)

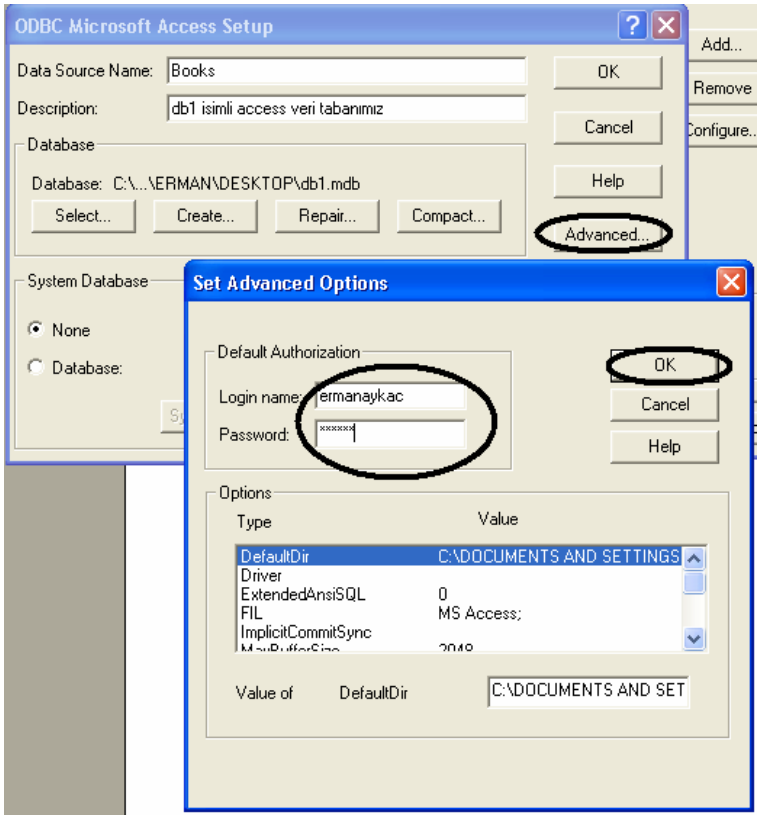
2. Elde edilen bu veri tabanı ODBC üzerinde kaydedilmelidir. Bu işlem için Microsoft Control Panel üzerinde Administrative Task -> Data Sources (ODBC) seçilir. Daha sonra aşağıdaki 3 adım gerçekleştirilir.



Daha sonra önceden hazırlanmış olan Access Veri tabanı sistemden seçilir:



Son olarak bu veri tabanına erişim için gerekli olan şifre ve parola bilgisi girilir:



Artık JDBC-ODBC köprüsü hazırlanmıştır. Geriye sadece uygun bir program yazmak ve program içerisinde bu veri tabanına erişimi sağlamaktır. Aşağıdaki örnek kod parçası bu erişimin nasıl yapılacağını göstermektedir:

```
String url = "jdbc:odbc:Books";  //Books veri tabanı ismi
String username = "ermanaykac";
String password = "197919";

try {

  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                //Bu url JDBC-ODBC köprüsü için gerekli

  connection = DriverManager.getConnection(url, username, password );

} catch ( ClassNotFoundException cnfex ) {
    System.err.println( "Failed to load JDBC/ODBC driver." );
    cnfex.printStackTrace();
    System.exit( 1 );  // terminate program
}
catch ( SQLException sqlex ) {
    System.err.println( "Unable to connect" );
    sqlex.printStackTrace();
    System.exit( 1 );  // terminate program
}
```

Aşağıdaki örnekte bütünüyle bu bağlantı ve bağlantı üzerinde kullanılan sorgular incelenmektedir.

```
import java.sql.*;
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class DisplayQueryResults extends JFrame {
   // java.sql types needed for database processing
   private Connection connection;
   private Statement statement;
   private ResultSet resultSet;
   private ResultSetMetaData rsMetaData;

   // javax.swing types needed for GUI
   private JTable table;
   private JTextArea inputQuery;
   private JButton submitQuery;

   public DisplayQueryResults()
   {
      super( "Enter Query. Click Submit to See Results." );

      // The URL specifying the Books database to which
      // this program connects using JDBC to connect to a
      // Microsoft ODBC database.
      String url = "jdbc:odbc:Books";
      String username = "ermanaykac";
      String password = "197919";

      // Load the driver to allow connection to the database
      try {
         Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver" );
         connection = DriverManager.getConnection(url, username, password );
      }
      catch ( ClassNotFoundException cnfex ) {
         System.err.println(
            "Failed to load JDBC/ODBC driver." );
         cnfex.printStackTrace();
         System.exit( 1 );  // terminate program
      }
      catch ( SQLException sqlex ) {
         System.err.println( "Unable to connect" );
         sqlex.printStackTrace();
         System.exit( 1 );  // terminate program
      }

      // If connected to database, set up GUI
      inputQuery =new JTextArea( "SELECT * FROM Authors", 4, 30 );
      submitQuery = new JButton( "Submit query" );
      submitQuery.addActionListener(
         new ActionListener() {
            public void actionPerformed( ActionEvent e )
            {
               getTable();
            }
         }
      );

      JPanel topPanel = new JPanel();
      topPanel.setLayout( new BorderLayout() );
      topPanel.add( new JScrollPane( inputQuery),
                    BorderLayout.CENTER );
      topPanel.add( submitQuery, BorderLayout.SOUTH );
```

```
      table = new JTable( 4, 4 );

      Container c = getContentPane();
      c.setLayout( new BorderLayout() );
      c.add( topPanel, BorderLayout.NORTH );
      c.add( table, BorderLayout.CENTER );

      getTable();

      setSize( 500, 500 );
      show();
   }

   private void getTable()
   {
      try {
         String query = inputQuery.getText();

         statement = connection.createStatement();
         resultSet = statement.executeQuery( query );
         displayResultSet( resultSet );
      }
      catch ( SQLException sqlex ) {
         sqlex.printStackTrace();
      }
   }

   private void displayResultSet( ResultSet rs )
      throws SQLException
   {
      // position to first record
      boolean moreRecords = rs.next();

      // If there are no records, display a message
      if ( ! moreRecords ) {
         JOptionPane.showMessageDialog( this, "ResultSet contained no records" );
         setTitle( "No records to display" );
         return;
      }

      Vector columnHeads = new Vector();
      Vector rows = new Vector();

      try {
         // get column heads
         ResultSetMetaData rsmd = rs.getMetaData();

         for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
            columnHeads.addElement( rsmd.getColumnName( i ) );

         // get row data
         do {
            rows.addElement( getNextRow( rs, rsmd ) );
         } while ( rs.next() );

         // display table with ResultSet contents
         table = new JTable( rows, columnHeads );
         JScrollPane scroller = new JScrollPane( table );
         Container c = getContentPane();
         c.remove( 1 );
         c.add( scroller, BorderLayout.CENTER );
         c.validate();
      }
```

```
      catch ( SQLException sqlex ) {
         sqlex.printStackTrace();
      }
   }

   private Vector getNextRow( ResultSet rs,
                              ResultSetMetaData rsmd )
       throws SQLException
   {
      Vector currentRow = new Vector();

      for ( int i = 1; i <= rsmd.getColumnCount(); ++i )
         switch( rsmd.getColumnType( i ) ) {
            case Types.VARCHAR:
            case Types.LONGVARCHAR:
                  currentRow.addElement( rs.getString( i ) );
               break;
            case Types.INTEGER:
                  currentRow.addElement(
                     new Long( rs.getLong( i ) ) );
               break;
            default:
               System.out.println( "Type was: " +
                  rsmd.getColumnTypeName( i ) );
         }

      return currentRow;
   }

   public void shutDown()
   {
      try {
         connection.close();
      }
      catch ( SQLException sqlex ) {
         System.err.println( "Unable to disconnect" );
         sqlex.printStackTrace();
      }
   }

   public static void main( String args[] )
   {
      final DisplayQueryResults app =
         new DisplayQueryResults();

      app.addWindowListener(
         new WindowAdapter() {
            public void windowClosing( WindowEvent e )
            {
               app.shutDown();
               System.exit( 0 );
            }
         }
      );
   }
}
```