PAKET ERİŞİMLERİ

Erişim konusu kütüphaneler için büyük önem taşır. Erişimde iki taraf bulunur; birisi kütüphaneyi kullanan kişiler (*client*), diğeri ise bu kütüphaneyi yazanlardır. Olaylara, hazır kütüphane kullanarak uygulama geliştiren tarafından bakılırsa, örneğin finans uygulaması yazan bir programcı, işlerin daha da hızlı yürümesi için daha önceden yazılmış ve denenmiş bir finans kütüphanesini kullanmak isteyebilir. Finans için özel geliştirilmiş kütüphanenin 1.0 uyarlamasını kullanan tasarımcı, bunun yeni uyarlamaları (2.0, 3.0...) çıktığında da hemen alıp kendi uygulamasına entegre etmek istemesi çok doğaldır. Tahmin edilebileceği üzere her yeni uyarlama bir öncekine göre daha az hata içerir ve yeni özellikler sunar. Bu nedenden dolayı, kütüphane uyarlamaları arasındaki tutarlılık çok önemlidir. Sonuç olarak, kütüphanenin her yeni uyarlamasında sisteme entegre edilmesi aşamasında, bu kütüphaneyi kullanan uygulamaları teker teker değiştirmek yaşamı çekilmez kılabilir! (yorum ekle)

Olaylara, birde kütüphane tasarımcısı açısından bakılırsa... Bir kütüphane yazdınız ve bunun kullanılması için İnternet'e koydunuz. Aradan belirli bir zaman geçti ve sizin yazdığınız kütüphane birçok kişi tarafından kullanılmaya başladı... Fakat, daha sonra, kütüphane içerisinde bazı hatalar olduğunu fark ettiniz; veya, bazı kısımları, daha verimli çalışması için geliştirilmesini istiyorsunuz. Bu arzularınız, ilgili kütüphane üzerinde nasıl bir etki oluşturur? Kütüphaneyi kullanan kişiler bundan zarar görebilir mi veya zarar görmemesi için ne yapılması gerekir?

Böylesi sorunların çözümü için devreye erişim kavramı girer. Java dili 4 adet erişim belirleyicisi sunar. Erişim belirleyiciler, en erişilebilirden erişilmeze doğru sıralanırsa, `public`, `protected`, `friendly` ve `private`'dır. Bunlar sayesinde hem kütüphane tasarımcıları özgürlüklerine kavuşur hem de kütüphaneyi kullanan programcılar kullandıkları kütüphanenin yeni bir uyarlamalarını tasarımlarına kolayca entegre edebilirler. (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

Kütüphane tasarlayan kişiler, ileride değişebilecek olan sınıfları veya sınıflara ait yordamları, kullanıcı tarafından erişilmez yaparak hem kütüphanenin rahatça gelişimini sağlarlar hem de kütüphaneyi kullanan programcıların endişelerini gidermiş olurlar. (yorum ekle)

4.1. **Paket (*Package*)**

Paketler kütüphaneyi oluşturan elemanlardır. Paket mantığının var olmasında ana nedenlerden birisi sınıf ismi karmaşasının getirmiş olduğu çözümdür. Örneğin elimizde *X* ve *Y* adlı 2 sınıf bulunsun. Bunlar içerisinde aynı isimli 2 yordam (*method*) olması, örneğin `f()` yordamı, herhangi bir karmaşıklığa neden olmayacaktır. Çünkü aynı isimdeki yordamlar ayrı sınıflarda bulunurlar. Peki sınıf isimleri? Sistemimizde bulunan aynı isimdeki sınıflar karmaşıklığa sebep vermez; eğer ki, aynı isimdeki sınıflar değişik paketlerin içerisinde bulunurlarsa... (yorum ekle)

Gösterim-4.1:

```
import java.io.BufferedReader;
```

Yukarıdaki gösterimde *BufferedReader* sınıf isminin *java.io* paketinde tek olduğunu anlıyoruz (*java.io* Java ile gelen standart bir pakettir). Fakat, başka paketlerin içerisinde *BufferedReader* sınıf ismi rahatlıkla kullanılabilir. Yukarıdaki gösterim *java.io* paketinin içerisinde bulunan *BufferedReader* sınıfını kullanacağını ifade etmektedir. Paketin içerisindeki tek bir sınıfı kullanmak yerine ilgili paketin içerisindeki tüm sınıfları tek seferde kullanmak için: (yorum ekle)

Gösterim-4.2:

```
import java.io.* ;
```

java.io paketi içerisindeki sınıfların uygulamalarda kullanılması için *import java.io.** denilmesi yeterli olacaktır. Anlatılanlar uygulama içerisinde incelenirse, (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

Örnek-4.1: *PaketKullanim.java* (yorum ekle)

```
import java.io.*;

public class PaketKullanim {

  public static void main(String args[]) throws IOException{
      BufferedReader sf = new BufferedReader(
                              new InputStreamReader(System.in));
      System.out.print("Bilgi Giriniz : ");
      String bilgi = sf.readLine();
      System.out.println("bilgi --> " + bilgi);
  }
}
```

PaketKullanim.java uygulamasında görmediğimiz yeni kavram mevcuttur, "`throws Exception`". Bu kavram istisnalar (*Exception*- 8. bölüm) konusunda detaylı bir şekilde incelenecektir. (yorum ekle)

4.2. **Varsayılan Paket (*Default Package*)**

Öncelikle belirtmek gerekirse, *.java* uzantılı fiziksel dosya derlendiği zaman buna tam karşılık *.class* fiziksel dosyası elde edilir. (*.java dosyasında hata olmadığı varsayılırsa). Fiziksel *.java* dosyasında birden fazla sınıf tanımlanmış ise, tanımlanan her sınıf için ayrı ayrı fiziksel *.class* dosyaları üretilir. (yorum ekle)

Örnek-4.2: *Test1.java* (yorum ekle)

```
public class Test1 {

  public void kos() {
```

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
  }
}

class Test2 {

  public void kos() {
  }
}
```

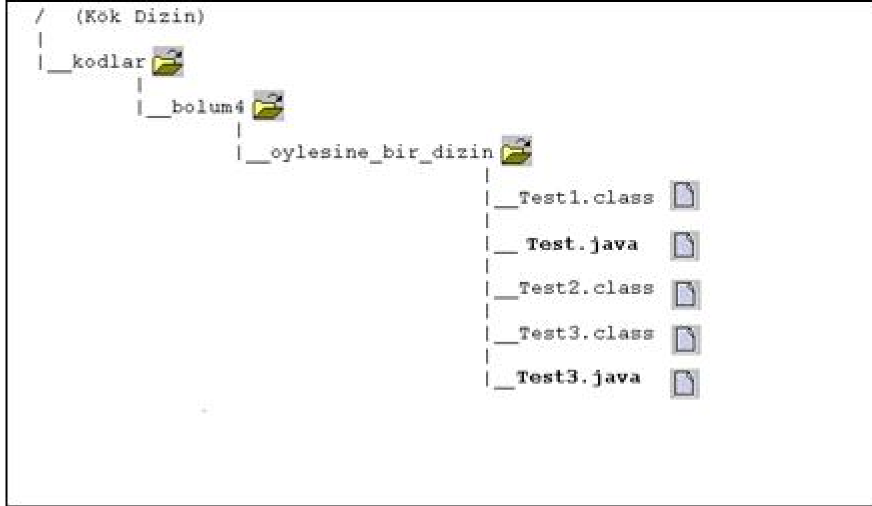
Yukarıda verilen örnek, *Test1.java* adıyla herhangi dizine kayıt edilebilir (fiziksel java uzantılı dosya ile `public` sınıfın ismi birebir aynı olmalıdır). Bu dosya `javac` komutu ile derlendiğinde adları *Test1.class* ve *Test2.class* olan 2 adet fiziksel *.class* dosyası elde edilir. *Test1.java* dosyanın en üstüne herhangi bir paket ibaresi yerleştirilmediğinden dolayı Java bu sınıfları varsayılan paket (*default package*) olarak algılayacaktır. (yorum ekle)

Örnek-4.3: *Test3.java* (yorum ekle)

```
 public class Test3 {
   public void kos()  {
   }
}
```

Benzer şekilde verilen örnekte, aynı dizine *Test3.java* adıyla kayıt edilebilir; derleme (*compile*) işleminden sonra genel ifade Şekil-4.1.'de gösterildiği gibi olur: (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.



Şekil-4.1. Varsayılan paket

4.3. Paket Oluşturma

Kendi paketlerimizi oluşturmanın temel amaçlarından birisi, aynı amaca yönelik iş yapan sınıfları bir çatı altında toplamaktır; böylece yazılan sınıflar daha derli toplu olurlar. Ayrıca aranılan sınıflar daha kolay bulunabilir. Peki eğer kendimiz paket oluşturmak istersek, bunu nasıl başaracağız? (yorum ekle)

Örnek-4.4: *Test1.java* (yorum ekle)

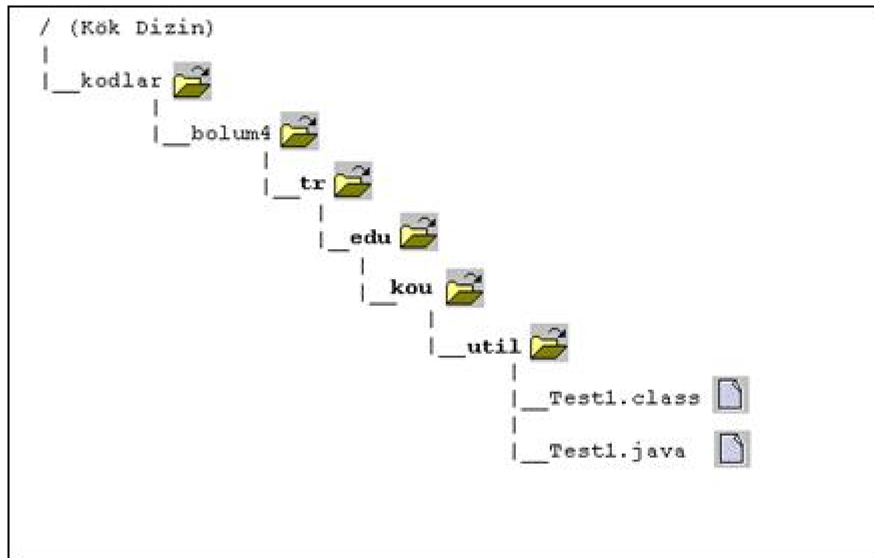
```
package tr.edu.kou.util ;


public class Test1 {
 public Test1() {
  System.out.println("tr.edu.kou.util.Test1"+
   "nesnesi olusturuluyor");
 }


  public static void main(String args[]) {
   Test1 pb = new Test1();
```

```
    }
}
```

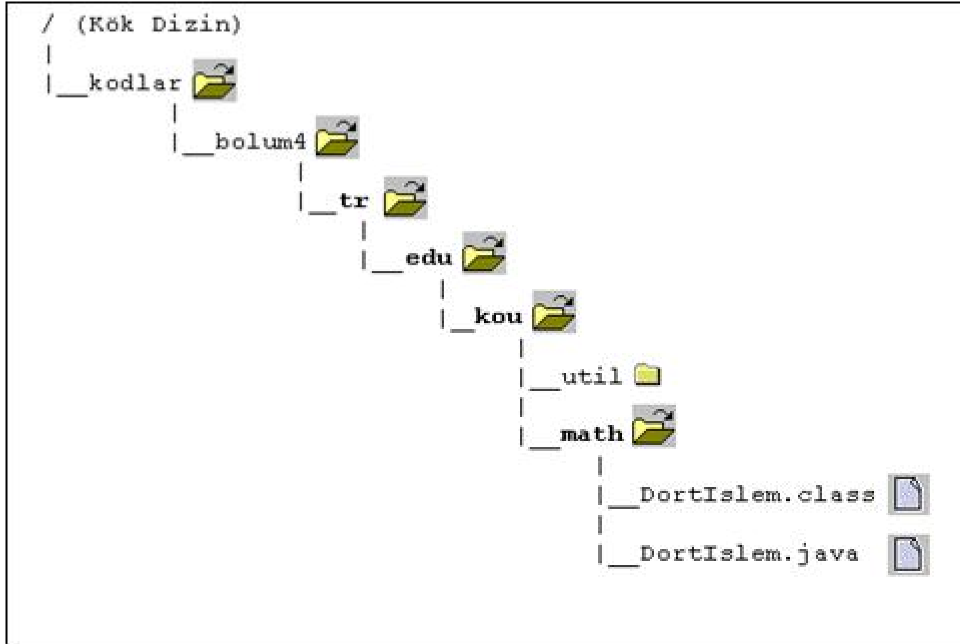
Bu örneğimizde, *Test1.java* dosyası işletim sisteminin herhangi bir dizinine yerleştirilemez; çünkü, o artık *tr.edu.kou.util* paketine ait bir sınıftır. Bundan dolayı *Test1.java* dosyası işletim sisteminin bu paket ismiyle paralel olan bir dizin yapısına kayıt edilmesi gerekir. Önemli diğer bir unsur ise sınıfın ismidir; *Test1.java* dosyası içerisinde belirtilen *Test1* sınıfının ismi artık *tr.kou.edu.util.Test1*'dir. (yorum ekle)



Şekil-4.2. *tr.edu.kou.util.Test1* sınıfı

Paket isimleri için kullanılan yapı İnternet alan isim sistemiyle (*Internet DNS*) aynıdır. Örneğin, Kocaeli Üniversitesinde matematik paketi geliştiren tasarımcı, bu paketin içerisindeki sınıfların, başka kütüphane paketleri içerisindeki sınıf isimleriyle çakışmaması için İnternet alan adı sistemini kullanmalıdır. İnternet alan adı sistemi, `www.kou.edu.tr` adresinin dünya üzerinde tek olacağını garantiler. Aynı mantık, paket isimlerine de uygulanarak, paket içerisindeki sınıf isimleri çakışma sorununa çözüm bulunmuş olunuyor. Dikkat edilirse, paket isimleri İnternet alan adlarının tersten yazılmış halleridir. (yorum ekle)

İşletim sistemleri farkı gözönüne alınarak, *math* paketine ait sınıfların içinde bulunması gereken dizin Unix veya Linux için *tr/edu/kou/math*, Windows için *tr\edu\kou\math* şekilde olmalıdır. (yorum ekle)



Şekil-4.3. ***tr.edu.kou.math.DortIslem* sınıfı**

4.4. `CLASSPATH` Ayarları

Java yorumlayıcısı (*interpreter*) işletim sistemi çevre değişkenlerinden `CLASSPATH`'e bakarak `import` ifadesindeki paketi bulmaya çalışır... (yorum ekle)

Gösterim-4.3:

```
import tr.edu.kou.math.*;
```

Diyelim ki; *math* paketi aşağıdaki dizinde bulunsun:

Gösterim-4.4:

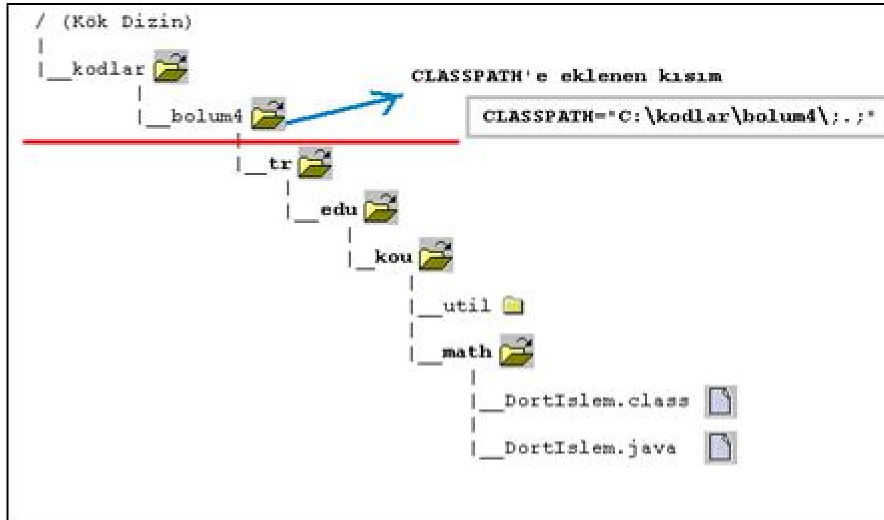
```
C:\kodlar\bolum4\tr\edu\kou\math\
```

Yorumlayıcının `import` ifadesindeki paketi bulması için aşağıdaki tanımın `CLASSATH'e` eklenmesi gerekir. (yorum ekle)

Gösterim-4.5:

```
CLASSPATH="C:\kodlar\bolum4\;."
```

`CLASSPATH`'e eklenen bu tanım sayesinde yorumlayıcı *C:\kodlar\bolum4* dizinine kadar gidip *tr\edu\kou\math* dizinini arayacaktır; bulunursa, bu dizinin altındaki tüm sınıflar uygulama tarafından kullanılabilecek hale gelecektir. (yorum ekle)



Şekil-4.4. CLASSPATH ayarları

DortIslem.java dosyasını *C:\kodlar\bolum4\tr\edu\kou\math* dizinine yerleştirelim:

Örnek-4.5: *DortIslem.java* (yorum ekle)

```
package tr.edu.kou.math ;

public class DortIslem {

  public static double topla(double a, double b) {

    return (a + b) ;

  }
```

```
  public static double cikart(double a, double b) {
     return (a - b) ;
  }

  public static double carp(double a, double b) {
     return (a * b) ;
  }

  public static double bol(double a, double b) {
     if ( (a != 0) && (b != 0) ) {
       return (a / b);
     } else {
        return 0;
     }
  }
}
```

Gerekli `CLASSPATH` tanımları yapıldıktan sonra bu paketi kullanacak olan uygulama herhangi bir dizinine yerleştirebilir. (yorum ekle)

Örnek-4.6: *Hesaplama.java* (yorum ekle)

```
import tr.edu.kou.math.*;
//dikkat!
 public class Hesaplama {
     public static void main(String args[]) {
          double sonuc = DortIslem.topla(9.6 , 8.7);
          System.out.println("9.6 + 8.7 = " + sonuc );

          sonuc = DortIslem.cikart(9.6 , 8.7);
```

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
            System.out.println("9.6 - 8.7 = " + sonuc );

            sonuc = DortIslem.carp(5.6 , 8.7);
            System.out.println("9.6 * 8.7 = " + sonuc );

            sonuc = DortIslem.bol(5.6 , 8.7);
            System.out.println("9.6 / 8.7 = " + sonuc );
        }
}
```

Uygulamanın sonucu aşağıdaki gibi olur:

```
9.6 + 8.7 = 18.299999999999997
9.6 - 8.7 =  0.9000000000000004
9.6 * 8.7 = 48.71999999999999
9.6 / 8.7 =  0.6436781609195402
```

4.4.1. Önemli Nokta

Dikkat edilirse `CLASSPATH` değişkenine değer atanırken en sona "." nokta koyuldu.

Şekil-4.5. Noktanın Önemi

Bu önemli bir ayrıntıdır. Bu noktanın konmasındaki neden varsayılan paketlerin içindeki sınıfların birbirlerini görebilmesini sağlamaktır; unutulursa, anlamsız hata mesajlarıyla karşılaşılabilir. (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

Java'yı sisteme ilk yüklendiği zaman, basit örneklerin, `CLASSPATH` değişkenine herhangi bir tanım eklemeden bile çalıştırabildiği görülür; nedeni, Java'nın, temel kütüphanelerinin bilinmesindendir. (yorum ekle)

4.5. Çakışma

Ayrı paket içerisinde aynı isimdeki sınıflar uygulamada kullanılırsa ne olur? Adları aynı olsa bile değişik paketlerde bulundukları için bir sorun yaşanmaması gerekecektir. Öncelikle *tr.edu.kou.util* paketinin içerisine kendi *ArrayList* sınıfımızı oluşturalım: (yorum ekle)

Örnek-4.7: *ArrayList.java* (yorum ekle)

```
package tr.edu.kou.util;

public class ArrayList {

   public ArrayList() {

    System.out.println("tr.edu.kou.util.ArrayList nesnesi"
+ " olusturuluyor");  }}
```

Aşağıdaki örneği işletim sisteminin herhangi bir dizinine kayıt edebiliriz.

Örnek-4.8: *Cakisma.java* (yorum ekle)

```
import java.util.*;

import tr.edu.kou.util.*;

public class Cakisma {

  public static void main(String args[]) {

    System.out.println("Baslagic..");

     ArrayList al = new ArrayList();

     System.out.println("Bitis..");

  }

}
```

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

Cakisma.java dosyası `javac` komutu ile derlendiğinde şu hata mesajıyla karşılaşılır:

```
 Cakisma.java:8: reference to ArrayList is
ambiguous, both class tr.edu.kou.util.ArrayList
in tr.edu.kou.util and class java.util.ArrayList
in java.util matchArrayList al = new
ArrayList();^Cakisma.java:8: reference to
ArrayList is ambiguous, both class
tr.edu.kou.util.  ArrayList in tr.edu.kou.util
and class java.util.ArrayList in java.util
matchArrayList al = new ArrayList();^2 errors
```

Bu hata mesajı, *ArrayList*'in hem *java.util* paketinde hem de *tr.edu.kou.util* paketinde bulunmasından kaynaklanan bir ikilemi göstermektedir. *Cakisma* sınıfının içerisinde *ArrayList* sınıfı kullanılmıştır; ancak, hangi paketin içerisindeki *ArrayList* sınıfı? Bu sorunu çözmek için aşağıdaki örneği inceleyelim:

Örnek-4.9: *Cakisma2.java* (yorum ekle)

```
import java.util.*;
import tr.edu.kou.util.*;


public class Cakisma2 {
   public static void main(String args[]) {
      System.out.println("Baslagic..");
      tr.edu.kou.util.ArrayList al = new
      tr.edu.kou.util.ArrayList();
      System.out.println("Bitis..");
```

```
    }
}
```

Eğer ortada böyle bir ikilem varsa, gerçekten hangi sınıfı kullanmak istiyorsanız, o sınıfın içinde bulunduğu paket ismini de açık bir biçimde yazarak oluşan bu ikilemi ortadan kaldırabilirsiniz. (yorum ekle)

4.6. Paket İçerisindeki Tek Başına Yürütülebilir Uygulamaları (*Standalone*) Çalıştırmak

Paket içerisindeki tek başına çalışabilen uygulamaları (*standalone*) herhangi bir dizin içerisinden çalışmak için komut satırına, ilgili "`paket ismi+sınıf ismi`" girilmesi yeterlidir. *Hesaplama.java*'nın yeni uyarlamasını*C:\kodlar\bolum4\tr\edu\kou\math* dizinine kaydedelim. (yorum ekle)

Örnek-4.10: *Hesaplama.java* (yorum ekle)

```
package tr.edu.kou.math ;
// dikkat!

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

      double sonuc = DortIslem.topla(9.6 , 8.7);
      System.out.println("9.6 + 8.7 = " + sonuc
);
      sonuc = DortIslem.cikart(9.6 , 8.7);
      System.out.println("9.6 - 8.7 = " + sonuc
);
      sonuc = DortIslem.carp(5.6 , 8.7);
      System.out.println("9.6 * 8.7 = " + sonuc
```

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
);
      sonuc = DortIslem.bol(5.6 , 8.7);
      System.out.println("9.6 / 8.7 = " + sonuc );
   }
}
```

Artık *Hesaplama* sınıfımız *tr.edu.kou.math* paketinin yeni bir üyesidir. *Hesaplama* sınıfımızı **`java`** komutu kullanarak çalıştırmayı deneyelim. *C:\kodlar\bolum4* dizininin `CLASSPATH` değişkeninde tanımlı olduğunu varsayıyorum. (yorum ekle)

Gösterim-4.6:

```
java Hesaplama
```

Bir aksilik var! Ekrana yazılan hata mesajı aşağıdaki gibidir:

```
 Exception in thread "main" java.lang.NoClassDefFoundError:
Hesaplama (wrong name: tr/edu/kou/math/Hesaplama)  at
java.lang.ClassLoader.defineClass0(Native Method)  at
java.lang.ClassLoader.defineClass(ClassLoader.java:509)  at
java.security.SecureClassLoader.defineClass
                       (SecureClassLoader.java:123)  at
java.net.URLClassLoader.defineClass
(URLClassLoader.java:246)  at
java.net.URLClassLoader.access$100(URLClassLoader.java:54)
at java.net.URLClassLoader$1.run(URLClassLoader.java:193)
at java.security.AccessController.doPrivileged(Native
```

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
Method)  at
java.net.URLClassLoader.findClass(URLClassLoader.java:186)
at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
at sun.misc.Launcher$AppClassLoader.loadClass
(Launcher.java:265)  at
java.lang.ClassLoader.loadClass(ClassLoader.java:262)  at
java.lang.ClassLoader.loadClassInternal
(ClassLoader.java:322)
```

Hesaplama sınıfı bulanamıyor diye bir hata mesajı? Nasıl olur ama orada… Evet orada; ama, o artık *Hesaplama* sınıfı değildir; *tr.edu.kou.math.Hesaplama* sınıfıdır. Yani, artık bir paketin üyesi olmuştur. Aşağıdaki ifade sayesinde herhangi bir dizinden(tabii `CLASSPATH` değişkeninin değeri doğru tanımlanmış ise) *tr.edu.kou.math. Hesaplama* sınıfına ulaşıp onu `java` komutu ile çalıştırabiliriz. (yorum ekle)

Gösterim-4.7:

```
java tr.edu.kou.math.Hesaplama
```

Uygulamanın çıktısı aşağıdaki gibidir.

```
9.6 + 8.7 = 18.299999999999997
9.6 - 8.7 = 0.9000000000000004
9.6 * 8.7 = 48.71999999999999
9.6 / 8.7 = 0.6436781609195402
```

4.7. JAR Dosyaları (The Java™ Archive File)

JAR dosya formatı dosyaların arşivlenmesine ve sıkıştırılmasına olanak tanır. Olağan durumunda JAR dosyaları içerisinde sınıf dosyaları (**.class*) bulunur; bazen, özellikle A*ppletler* de, yardımcı dosyalar da (*gif, jpeg...*) JAR dosyası içerisine konulabilir. (yorum ekle)

JAR dosyasının sağladığı yararlar şöyledir:

- **Güvenlik:** Dijital olarak JAR dosyasının içeriğini imzalayabilirsiniz. Böylece sizin imzanızı tanıyan kişiler JAR dosyasının içeriğini rahatlıkla kullanabilirler. (yorum ekle)
- **Sıkıştırma:** Bir çok dosyayı güvenli bir şekilde arşivleyip sıkıştırılabilir. (yorum ekle)
- **İndirme (*download*) zamanını azaltması:** Arşivlenmiş ve sıkıştırılmış dosyalar internet üzerinde daha çabuk indirilebilir. (yorum ekle)
- **Paket mühürleme (*versiyon* 1.2):** Versiyon uyumluluğunu sağlamak amacı ile JAR dosyasının içerisindeki paketler mühürlenebilir. JAR dosyasının içerisinde paket mühürlemekten kasıt edilen paket içerisinde bulunan sınıfların aynı JAR dosyasında bulunmasıdır. (yorum ekle)
- **Paket uyarlama (*versiyon* 1.2):** JAR dosyaları, içindeki dosyalar hakkında bilgiler saklayabilirler, örneğin üretici firmaya ait bilgiler, versiyon bilgileri gibi. (yorum ekle)
- **Taşınabilirlik:** Java Platformunun standart bir üyesi olan JAR dosyaları kolaylıkla taşınabilir. (yorum ekle)

Oluşturulan paketler JAR dosyası içerisine yerleştirilerek daha derli toplu bir görüntü elde etmiş olunur; *tr.edu.kou.math* ve *tr.edu.kou.util* paketlerini tek bir JAR dosyasında birleştirmek için Gösterim-4.8'de verilen komutun kullanılması yeterli olur; ancak, JAR dosyası oluşturmak için komutun hangi dizinde yürütüldüğü önemlidir. Tablo-4.1'de JAR dosyası işlemleri için gerekli olan bazı komutlar verilmiştir: (yorum ekle)

Tablo-4.1. JAR dosyaları (yorum ekle)

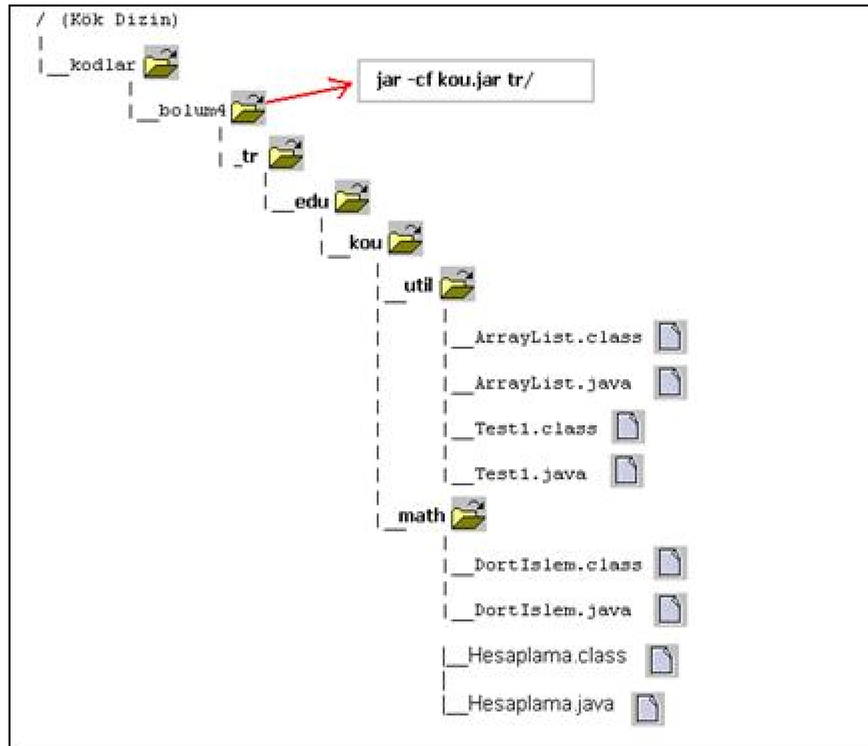
Açıklama	**Komut**
JAR dosyası oluşturmak için	jar -cf jar-dosya-ismi içeriye-atılacak-dosya(lar)
JAR dosyasının içeriği bakmak için	jar -tf jar-dosya-ismi

JAR dosyasının içeriği toptan dışarı çıkartmak için	jar -xf jar-dosya-ismi
Belli bir dosyayı JAR dosyasından dışarı çıkartmak için	jar -xf jar-dosya-ismi arşivlenmiş dosya(lar)
JAR olarak paketlenmiş uygulamayı çalıştırmak için	java -classpath jar-dosya-ismi MainClass

Gösterim-4.8:

```
jar -cf kou.jar tr/
```

Gösterim-4.8'deki ifadeyi *C:\kodlar\bolum4* dizinin içerisinde iken çalıştırmalıyız ki, JAR dosyasının içerisine doğru yapıdaki dizinleri yerleştirelim. (yorum ekle)



Şekil-4.6. Dizinlerin JAR dosyasına atılmasına

Oluşmuş olan bu JAR dosyasını CLASSPATH ekleyerek, Java'nın bu paketleri bulmasını sağlayabilirsiniz. Aşağıdaki ifade yerine: (yorum ekle)

Gösterim-4.9:

```
CLASSPATH="C:\kodlar\bolum4\;."
```

Artık Gösterim-4.10'daki ifade kullanılabilir; *kou.jar* dosyası, *C:\kodlar\bo-lum4*'nin altındaki dizin yapılarının aynısını kendi içerisinde barındırır. Bu nedenle *kou.jar* dosyası en alakasız dizine kopyalanabilir; ancak, tek bir koşulu unutmamak kaydıyla... Bu koşul da *kou.jar* dosyası sistemin `CLASSSPATH` değişkenin de tanımlı olmasıdır. (yorum ekle)

Gösterim-4.10:

```
CLASSPATH="C:\muzik\kou.jar;."
```

Java, `CLASSPATH` değerlerinden yola çıkarak JAR dosyasını bulup açar; *tr\edu\kou\util* ve *tr\edu\kou\math* dizinlerine erişebileceğinden bir sorun yaşanmayacaktır. Yani, JAR dosyasının hangi dizinde olduğu önemli değildir, önemli olan ilgili *jar*dosyasının sistemin `CLASSPATH` değişkenin tanımlı olmasıdır. Tabii, paketlerin içerisindeki sınıflar geliştikçe güncelliği korumak adına JAR dosyasını tekrardan oluşturmak (jar -cvf.....) gerekebilir. (yorum ekle)

4.7.1. JAR Dosyası İçerisindeki Bir Uygulamayı Çalıştırmak

JAR dosyası içeriğini dışarı çıkartılmadan tek başına çalışabilir (*standalone*) java uygulamalarını yürütmek olasıdır. Örneğin, JAR dosyası içerisinde *tr.edu.kou.math* paketi altındaki *Hesaplama* sınıfı çalıştırılsın, (yorum ekle)

Gösterim-4.11:

```
> java -classpath C:\muzik\kou.jar
tr.edu.kou.math.Hesaplama
```

Eğer *kou.jar* dosyası, sistemin `CLASSPATH` değişkeninde tanımlı değilse ve `CLASSPATH` ayarlarıyla uğraşılmak istenmiyorsa, *java* komutuna kullanılacak JAR dosyası adresi tam olarak `-classpath` parametresiyle birlikte verilebilir. Daha sonra, ilgili. JAR

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

dosyasındaki hangi sınıf çalıştırılmak isteniyorsa, bu dosya düzgün bir biçimde yazılmalıdır. Gösterim-4.11'deki komutun oluşturacağı ekran sonucu aşağıdaki gibi olur:. (yorum ekle)

```
9.6 + 8.7 = 18.299999999999997
9.6 - 8.7 =  0.9000000000000004
9.6 * 8.7 = 48.71999999999999
9.6 / 8.7 =  0.6436781609195402
```

4.8. **Erişim Belirleyiciler**

Java dilinde 4 tür erişim belirleyicisi vardır; bunlar `friendly`, `public`, `protected` ve `private`'dir. Bu erişim belirleyiciler global alanlar (statik veya değil) ve yordamlar (statik veya değil) için kullanılabilir. Ayrıca sınıflar içinde (dahili sınıflar hariç –inner class) sadece `public` ve `friendly` erişim belirleyicilerini kullanılabilir. (yorum ekle)

4.8.1. **`friendly`**

`friendly` erişim belirleyicisi global alanlara (statik veya değil), yordamlara (statik veya değil) ve sınıflara atanabilir. `friendly` türünde erişim belirleyicisine sahip olan global alanlar (statik veya değil) içerisinde bulundukları paketin diğer sınıfları tarafından erişilebilirler. Fakat, diğer paketlerin içerisindeki sınıflar tarafından erişilemezler. Yani, diğer paketlerin içerisindeki sınıflara karşı `private` erişim belirleyici etkisi oluşturmuş olurlar. (yorum ekle)
`friendly` yordamlarda, yalnız, paketin kendi içerisindeki diğer sınıflar tarafından erişilebilirler. Diğer paketlerin içerisindeki sınıflar tarafından erişilemezler. Aynı şekilde, sınıflara da `friendly` erişim belirleyicisi atayabiliriz, böylece `friendly` erişim belirleyicisine sahip bu sınıfa, aynı paket içerisindeki diğer sınıflar tarafından erişilebilir; ancak, diğer paketlerin içerisindeki sınıflar tarafından erişilemezler. (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

Şimdi, *tr\edu\kou* dizini altına yeni bir dizin oluşturalım; ve, ismini *gerekli* verelim. Yani *tr\edu\kou\gerekli* paketini oluşturmuş olduk; bunun içerisine adları *Robot* ve *Profesor* olan 2 adet `friendly` sınıf yazalım: (yorum ekle)

Örnek-4.11: *Robot.java* (yorum ekle)

```
package tr.edu.kou.gerekli;
 class Robot {
    int calisma_sure = 0;
    String renk = "beyaz";
    int motor_gucu = 120;
    Robot() {
      System.out.println("Robot olusturuluyor");
  }
}
```

Örnek-4.12: *Profesor.java* (yorum ekle)

```
package tr.edu.kou.gerekli;
class Profesor {
      public void kullan() {
         Robot upuaut = new Robot(); // sorunsuz
      }
}
```

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

Verilen örneklerden anlaşılacağı gibi, bir global alan veya sınıf `friendly` yapılmak isteniyorsa önüne hiç bir erişim belirleyicisi konulmaz. Şimdi, bu iki sınıf aynı paketin içerisinde olduklarına göre *Profesor* sınıfı rahatlıkla `Robot` sınıfına erişebilecektir. Peki başka bir paket içerisindeki sınıf *Robot* sınıfına erişebilir mi? (yorum ekle)

Örnek-4.13: *Asistan.java* (yorum ekle)

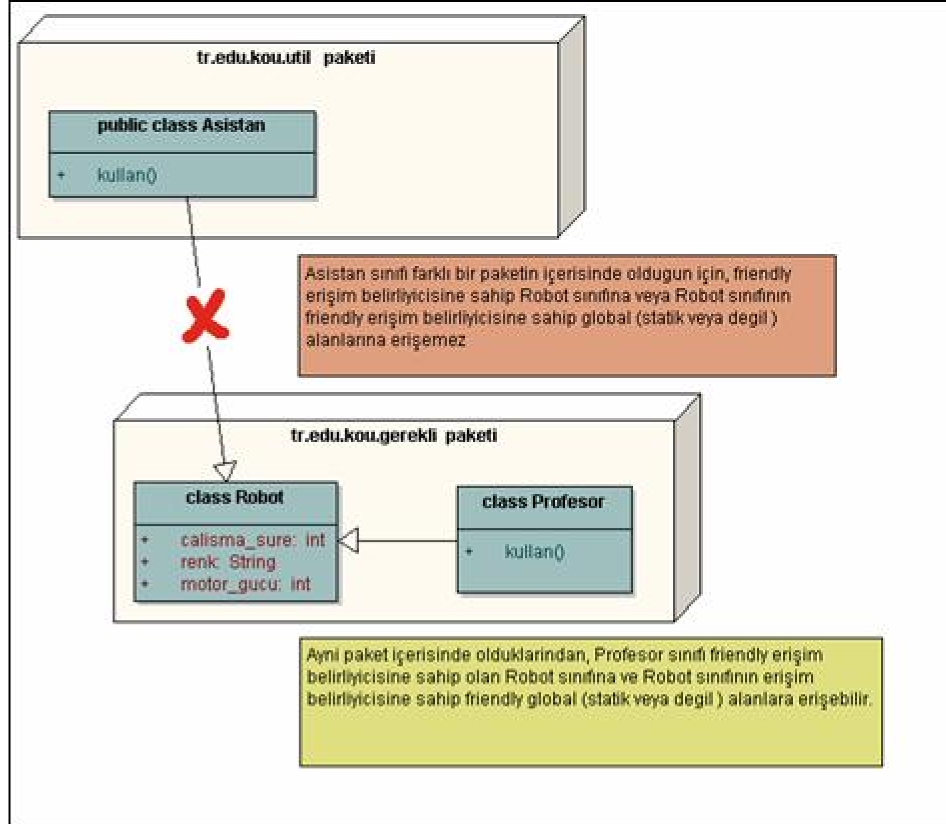
```
package tr.edu.kou.util;


import tr.edu.kou.gerekli.*;


public class Asistan {
     public void arastir() {
       System.out.println("Asistan arastiriyor");
     }
     public void kullan() {
      //Robot upuaut = new Robot(); Hata! erişemez
     }
}
```

Daha önce belirtildiği gibi, `friendly` erişim belirleyicisine sahip olan global alanlara veya sınıflara yalnızca içerisinde bulunduğu paketin diğer sınıfları tarafından erişilebilirdi. Diğer paketlerin içerisindeki sınıflar tarafından erişilemezler. Yukarıdaki örneğimizde, *Asistan* sınıfı *tr.edu.kou.util* paketi altında tanımlandığı için *tr.edu.kou.gerekli* paketi altında tanımlı olan *Robot* sınıfına hiç bir şekilde erişemez. Anlatılanlar Şekil-4.7'de çizimsel olarak gösterilmeye çalışılmıştır. (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.



Şekil-4.7. ***friendly*** **erişim belirleyicisinin etkisi**

4.8.1.1. Varsayılan Paketlerde (*Default Package*) Erişim

Aşağıdaki 2 sınıfı herhangi bir dizini kaydedelim: (yorum ekle)

Örnek-4.14: *AltKomsu.java* (yorum ekle)

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

     UstKomsu uk = new UstKomsu();
     uk.merhaba();
   }
}
```

Örnek-4.15: *UstKomsu.java* (yorum ekle)

```
class UstKomsu {
  void merhaba() {
   System.out.println("Merhaba");
  }
}
```

Bu iki sınıf `friendly` erişim belirleyicisine sahiptir yani aynı paketin içerisindeki sınıflar tarafından erişilebilirler ama ortada herhangi bir paket ibaresi bulunmamaktadır. Aynı dizinde olan fakat bir paket olarak tanımlanmamış sınıflar, Java tarafındanvarsayılan paket çatısı altında toplanmaktadır. Bu iki sınıfın birbirini görebilmesi için CLASSPATH değişkenin değerinde **"."** (nokta) ibaresinin olması şarttır (bkz: Şekil-4.5.1 önemli nokta). (yorum ekle)

4.8.2. `public` (Herkese Açık)

`public` erişim belirleyicisi sahip olabilen sınıflar, global alanlar ve yordamlar herkes tarafından erişilebilir. Bu erişim belirleyicisi yerleştirilmeden önce iki kez düşünmelidir! Bu erişim belirleyicisine sahip olan global alanlar veya yordamlar herhangi bir yerden doğrudan çağrılabildiklerinden dolayı dış dünya ile arasındaki arabirim rolünü üstlenirler. (yorum ekle)

Örnek-4.16: *Makine.java* (yorum ekle)

```
package tr.edu.kou.util;
public class Makine {
  int devir_sayisi;
  public String model = "2002 model" ;
  public int degerAl() {
    return devir_sayisi;
```

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
  }


  public void degerAta(int deger) {
   this.devir_sayisi = deger;
   calis();
  }
  void calis() {
     for (int i = 0 ; i < devir_sayisi ; i++)
{
       System.out.println("devir_sayisi=" +
i);
     }
  }
}
```

tr.edu.kou.util paketinin içerisindeki *Makine* sınıfının 2 adet global alanı bulunmaktadır; bunlardan `int` türündeki `devir_sayisi` alanı `friendly` erişim belirleyicisine sahiptir. Yani, sadece *tr.edu.kou.util* paketinin içerisindeki diğer sınıflar tarafından doğrudan erişilebilir. Diğer *String* tipindeki `model` alanı ise her yerden erişilebilir. Çünkü `public` erişim belirleyicisine sahiptir. `degerAl()` yordamı `public` erişim belirleyicisine sahiptir yani her yerden erişilebilir. Aynı şekilde `degerAta(int deger)` yordamı da her yerden erişilebilir; ancak, `calis()` yordamı `friendly` belirleyicisine sahip olduğundan sadece *tr.edu.kou.util* paketinin içerisindeki sınıflar tarafından erişilebilir. (yorum ekle)

Örnek-4.17: *UstaBasi.java* (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
import tr.edu.kou.util.*;
public class UstaBasi {
  public UstaBasi() {
     Makine m = new Makine();
     // int devir_sayisi = m.devir_sayisi ; ! Hata ! erişemez
     m.degerAta(6);      int devir_sayisi =
     m.degerAl();
     String model = m.model;
     // m.calis() ; ! Hata ! erişemez
   }
}
```

Yukarıdaki uygulamada *tr.edu.kou.util* paketinin altındaki tüm sınıflar kullanılmak istendiği belirtmiştir. *Ustabasi* sınıfının yapılandırıcısında `public` erişim belirleyicisine sahip olan *Makine* sınıfına ait bir nesne oluşturulabilmesine karşın, bu nesnenin**friendly** erişim belirleyicisine sahip olan `devir_sayisi` alanına ve `calis()` yordamına erişilemez. Çünkü, *Ustabasi* sınıfı *tr.edu.kou.util* paketinin içerisinde değillerdir. (yorum ekle)

4.8.3. `private` (Özel)

`private` olan global alanlara veya yordamlara (sınıflar `private` olamazlar; dahili sınıfar-*inner class* hariç) aynı paket içerisinden veya farklı paketlerden erişilemez. Ancak, ait olduğu sınıfın içinden erişilebilir. `private` belirleyicisine sahip olan yordamların içerisinde devamlı değişebilecek/geliştirilebilecek olan kodlar yazılmalıdır. (yorum ekle)

Örnek-4.18: *Kahve.java* (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
package tr.edu.kou.gerekli;
class Kahve {
     private int siparis_sayisi;

     private Kahve() {   }

     private void kahveHazirla() {
       System.out.println(siparis_sayisi + " adet kahve"
        +" hazirlandi");
     }

     public static Kahve siparisGarson(int sayi) {
         Kahve kahve = new Kahve(); //dikkat
         kahve.siparis_sayisi = sayi ;
         kahve.kahveHazirla();
         return kahve;
     }
}
```

Örnek-4.19: *Musteri.java* (yorum ekle)

```
package tr.edu.kou.gerekli;
public class Musteri {
     public static void main(String args[]) {

        // Kahve kh = new Kahve() ;    // Hata
!
        // kh.kahveHazirla() ;         // Hata
```

```
!
        // kh.siparis_sayisi = 5 ;      // Hata !

        Kahve kh = Kahve.siparisGarson(5);
      }
 }
```

Kahve sınıfının yapılandırıcısı (*constructor*) `private` olarak tanımlanmıştır. Bundan dolayı herhangi bir başka sınıf, *Kahve* sınıfının yapılandırıcısını doğrudan çağıramaz; aynı paketin içerisinde olsa bile... Ancak, bu `private` yapılandırıcı aynı sınıfın içerisindeki yordamlar tarafından rahatlıkla çağırılabilir (Kahve sınıfının statik `siparisGarson()` yordamına dikkat edeniz). Aynı şekilde `private` olarak tanımlanmış global alanlara veya yordamlara aynı paket içerisinde olsun veya olmasın, kesinlikle erişilemez. Anlatılanlar Şekil-4.8'de çizimsel olarak gösterilmeye çalışılmıştır. (yorum ekle)

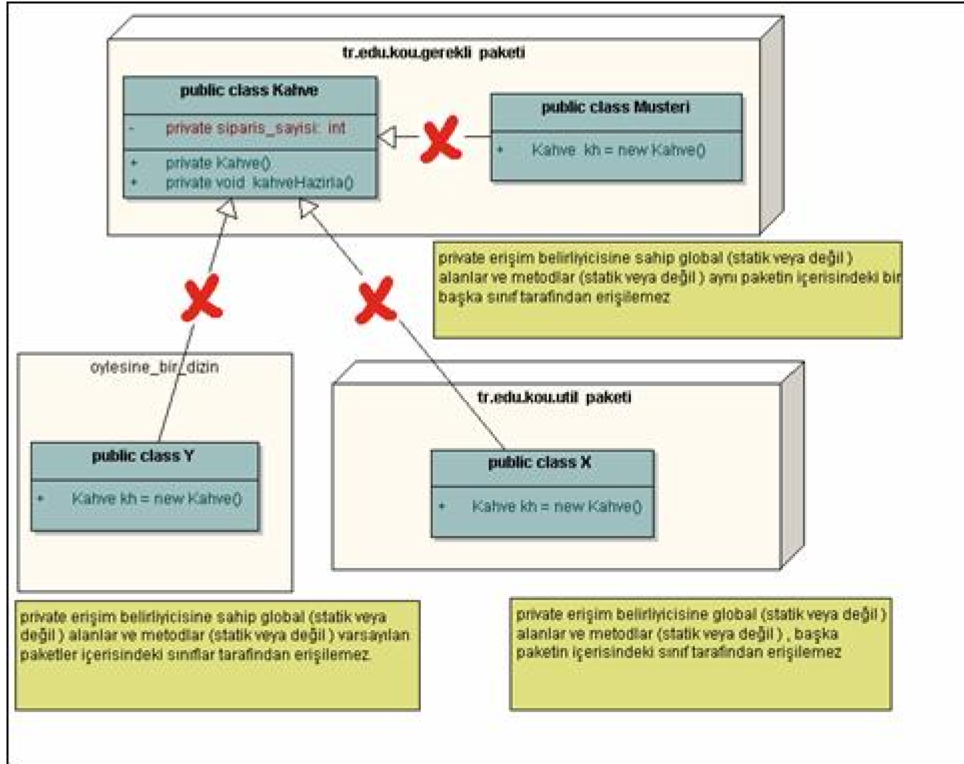
4.8.4. `protected` (Korumalı Erişim)

Sadece global alanlar ve yordamlar `protected` erişim belirleyicisine sahip olabilirler. Sınıflar `protected` erişim belirleyicisine sahip olmazlar (dahili sınıflar-inner class hariç); ancak, sınıflar `friendly` veya `public` erişim belirleyicisine sahip olabilirler.`protected` erişim belirleyicisi kalıtım (*inheritance*) konusu ile sıkı sıkıya bağlıdır. Kalıtım konusunu bir sonraki ayrıtta ele alınmıştır. Kalıtım konusu hakkında kısaca, ana sınıftan diğer sınıfların türemesi denilebilir. (yorum ekle)

<u>Gösterim-4.12:</u>

```
class Kedi extends Hayvan
```

Yukarıda şu ifade edilmiştir: Her Kedi bir Hayvandır. Yani *Hayvan* sınıfından *Kedi* üretildi; bizim oluşturacağımız her *Kedi* nesnesi bir *Hayvan* olacaktır. Ancak, kendisine has kedisel özellikler de taşıyacaktır. (yorum ekle)



Şekil-4.8. private erişim belirleyicisi

`protected` erişim belirleyicisini evdeki buzdolabımızın kapısına vurulmuş bir kilit olarakda düşünebiliriz. Örneğin evimizde bir buzdolabı var ve içinde her türlü yiyecek ve içecek mevcut. Biz aile büyüğü olarak bu buzdolabına herkesin erişmesini istemiyoruz. (`public` yaparsak) aksi takdirde yiyecek ve içecek kısa sürede bitip ailenin aç kalma tehlikesi oluşacaktır, aynı şekilde bu buzdolabına erişimi tamamen kesersek de aile bireyleri aç kalacaktır (`private` yaparsak). Tek çare özel bir erişim belirleyicisi kullanmaktır yani sadece aileden olanlara (aynı paketin içerisindeki sınıflara) bu buzdolabının erişmesine izin veren bir erişim belirleyicisi yani `protected` erişim belirleyicisi. (yorum ekle)

Örnek-4.20: *Hayvan.java* (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
package tr.edu.kou.util;
  public class Hayvan {
    protected String a = "Hayvan.a";
    String b = "Hayvan.b"; //friendly
    private String c = "Hayvan.c";
    public String d = "Hayvan.d";
  }
```

Hayvan sınıfından türetilen *Kedi* sınıfı *tr.edu.kou.gerekli* paketi içerisine yerleştirildi. (yorum ekle)

Örnek-4.21: *Kedi.java* (yorum ekle)

```
Package tr.edu.kou.gerekli;İmport
tr.edu.kou.util.*;

public class Kedi extends Hayvan {
   public Kedi() {
     System.out.println("Kedi
olusturuluyor");
     System.out.println(a);
     System.out.println(b); //  ! Hata ! erisemez
     System.out.println(c); //  ! Hata ! erisemez
     System.out.println(d);
    }

    public static void main(String args[]) {
     Kedi k = new Kedi();
```

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

```
    }
}
```

Kedi.java dosyasını önce derleyip (*compile*)

Gösterim-4.13:

```
javac Kedi.java
```

Sonrada çalıştıralım.

Gösterim-4.14:

```
java tr.edu.kou.gerekli.Kedi
```

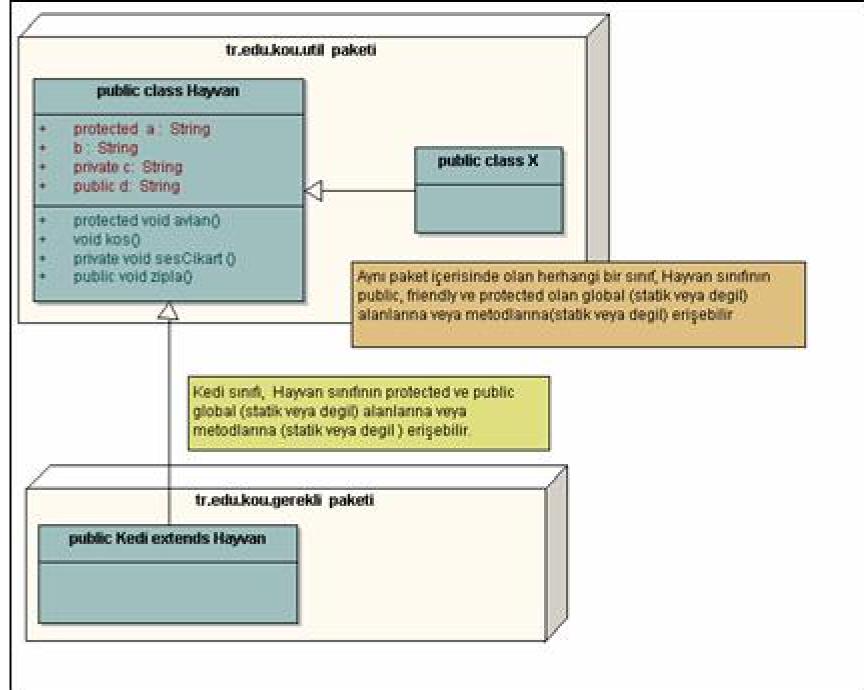
Uygulamanın sonucu aşağıdaki gibi olacaktır:

```
Kedi olusturuluyorHayvan.aHayvan.d
```

Anlatılanları şekil üzerinde incelenirse,

Şekil-4.9'dan görülebileceği gibi, *tr.edu.kou.gerekli.Kedi* sınıfı, *tr.edu.kou.util.Hayvan* sınıfının `public` ve `protected` erişim belirleyicilerine sahip olan global alanlarına ve yordamlarına erişebilme fırsatı bulmuştur. (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.



Şekil-4.9. protected erişim belirleyicisi

4.9. **Kapsüllenme (*Encapsulation*)**

Nesneye yönelik programlama özelliklerinden birisi kapsüllenmedir; bu, dışarıdaki başka bir uygulamanın bizim nesnemiz ile sadece arabirimler (`public`) sayesinde iletişim kurması gerektiğini, ancak, arka planda işi yapan esas kısmın gizlenmesi gerektiğini söyler. Olaylara bu açıdan bakılırsa, nesneleri 2 kısma bölmeliyiz; arabirimler -ki nesnenin dünya ile iletişim kurabilmesi için gerekli kısımlar ve gemiyi yürüten kısım... (yorum ekle)

Örnek-4.22: *Makine2.java* (yorum ekle)

```
package tr.edu.kou.util;

public class Makine2 {
      private int alinan = 0;
      private int geridondurulen = 0 ;
```

```
      public int get() {
           return geridondurulen;
      }

      public void set(int i ) {
        alinan = i;
        calis();
      }

      private void calis() {
         for (int j = 0 ; j < alinan ; j++ )
{
            System.out.println("Sonuc =
"+j);
         }
         geridondurulen = alinan * 2 ;
      }
}
```

Bir önce verilen örnekte bu *Makine2* türündeki nesneye yalnızca `get()` ve `set()` yordamlarıyla ulaşılabiliriz; geriye kalan global nesne alanlarına veya `calis()` yordamına ulaşım söz konusu değildir. Kapsüllenme kavramının dediği gibi nesneyi 2 kısımdan oluşturduk: ara birimler (- `get()`, `set()` -) ve gemiyi yürüten kısım (- `calis()` -). (yorum ekle)

Başka paket içerisinde olan başka bir uygulama, *tr.edu.kou.util.Makine2* sınıfının sadece iki yordamına erişebilir, `get()` ve `set()`.(yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

Örnek-4.23: *GetSet.java* (yorum ekle)

```
package tr.edu.kou.gerekli;

import tr.edu.kou.util.*;

public class GetSet {
   public static void main(String args[])
{
      Makine2 m2 = new Makine2() ;
      m2.set(5);
      int deger = m2.get();
      // m2.calis() ;     // Hata !
      // m2.alinan ;      // Hata !
      //  m2.geridondurulen; // Hata !
      System.out.println("Deger =" +
deger);
    }
}
```

4.10. Genel bir bakış

Sınıflar için erişim tablosu aşağıdaki gibidir:

	Aynı Paket	Ayrı Paket	Ayrı paket-türetilmiş
`public`	erişebilir	erişebilir	erişebilir
`protected`	-	-	-

`friendly`	erişebilir	erişemez	erişemez
`private`	-	-	-

Sınıflar `protected` veya `private` olamazlar, bu bağlamda bu tablomuzu şöyle okuyabiliriz; örneğin elimizde *A* sınıfı bulunsun (yorum ekle)

- `public` *A* sınıfına aynı paketin içerisindeki başka bir sınıf tarafından erişebilir. (yorum ekle)
- `public` *A sınıfına* ayrı paketin içerisindeki başka bir sınıf tarafından erişebilir. (yorum ekle)
- `public` *A sınıfına* ayrı paketten erişebildiğinden buradan yeni sınıflar türetilebilir. (yorum ekle)
- `friendly` *A sınıfına* aynı paketin içerisindeki başka bir sınıf tarafından erişebilir. (yorum ekle)
- `friendly` *A sınıfına* ayrı paketin içerisindeki başka bir sınıf tarafından erişemez. (yorum ekle)
- `friendly` *A'ya* ayrı paketten erişilemediğinden, buradan yeni sınıflar türetilemez. (yorum ekle)

Statik veya statik olmayan yordamlar için, erişim tablosu aşağıdaki gibidir. (yorum ekle)

	Aynı Paket	**Ayrı Paket**	**Ayrı paket-türetilmiş**
public	erişebilir	erişebilir	public
protected	erişebilir	erişemez	erişebilir
friendly	erişebilir	erişemez	erişemez
private	erişemez	erişemez	erişemez

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.

Yordamlar `public`, `protected`, `friendly` ve `private` olabilirler.

Örneğin, `public` *X* sınıfının içerisinde `f()` yordamı olsun: (yorum ekle)

- `public f()` yordamı, aynı paket içerisinden erişilebilir. (yorum ekle)
- `protected f()` yordamı, hem aynı paket içerisinden, hem de *X* sınıfından türetilmiş ayrı paketteki bir sınıf tarafından erişilebilir. (yorum ekle)
- `friendly f()` yordamı, yalnızca aynı paket içerisinden erişilebilir. (yorum ekle)
- `private f()` yordamına, yalnızca kendi sınıfı içerisinden erişilebilir. Başka bir sınıfın bu yordama erişmesi mümkün değildir. (yorum ekle)

Statik veya statik olmayan global alanlar için erişim tablosu aşağıdaki gibidir: (yorum ekle)

	Aynı Paket	**Ayrı Paket**	**Ayrı paket-türetilmiş**
public	erişebilir	erişebilir	erişebilir
protected	erişebilir	erişemez	erişebilir
friendly	erişebilir	erişemez	erişemez
private	erişemez	erişemez	erişemez

Global alanlar `public`, `protected`, `friendly`, `private` olabilirler. Örneğin `public` *X* sınıfının içerisindeki *String* sınıfı tipindeki `uzunluk` adında bir alanımız olsun: (yorum ekle)

- `public uzunluk` alanı, aynı paket içerisinden erişilebilir. (yorum ekle)
- `protected uzunluk` alanı, hem aynı paket içerisinden, hem de **X** sınıfından türetilmiş ayrı paketteki bir sınıf tarafından erişilebilir. (yorum ekle)

- `friendly uzunluk` alanı, yalnızca aynı paket içerisinden erişilebilir. (yorum ekle)
- `private uzunluk` alanı, yalnızca kendi sınıfı içerisinden erişilebilir. Başka bir sınıfın bu alana erişmesi mümkün değildir. (yorum ekle)

Altuğ Bilgin ALTINTAŞ - 2010 - Bu dökümanın her hakkı saklıdır.