ERİŞİM BELİRLEYİCİLER (ACCESS MODİFİERS)

Bir java deyimi (belli bir iş yapan kod veya kodlardan oluşan blok) bir değişkeni çağırabiliyorsa (kullanabiliyorsa), o kod sözkonusu değişkene erişebiliyor, denir. Benzer olarak, bir java kodu bir metodu çağırabiliyorsa, o kod sözkonusu metoda erişebiliyor, diyoruz. **Bir sınıfın içindeki kodlar, başka bir sınıfın içindeki değişkenlere ve metotlara erişebiliyorsa, o sınıf sözkonusu sınıfa erişebiliyor, diyoruz.** Paketler için de benzer tanım geçerlidir.

Bir Java öğesi (değişken, metot, sınıf, paket) tanımlanırken, o öğeye kimlerin erişebileceğini belirtme olanağı vardır. Bunlara Erişim belirleyiciler (Access modifiers, access levels) denir. Java terimleriyle söylersek, erişim belirtkeleri sistemin güvenliğini sağlar. Dört tane erişim belirtkesi vardır:
friendly, private, protected, public.

Erişim Belirtkesi	*İzinler*
public	Bütün sınıflar erişebilir
private	Alt-sınıf dahil başka hiçbir sınıf erişemez
protected	Alt-sınıflar ve aynı pakettekiler erişebilir
<default> friendly	Aynı pakettekiler erişebilir

1. public

public bir değişkeni, bir metodu ya da bir sınıfı niteleyebilir. Nitelediği öğeler herkese açık olur. Başka pakette olsa bile, program içindeki, her kod onlara erişebilir. public damgalı bir sınıfın değişkenlerine ve metotlarına kendi alt-sınıfları ve dışarıdaki başka sınıflar kısıtsız erişebilir. public damgalı değişkenler ve metotlar için de kısıtsız erişim vardır. Uygulama programlarında main() metodunun public damgalı olmasının nedeni budur. Örnekler:

a. public sınıf bildirimi

```
public class class_adı {
        Class gövdesi
}
```

b. public değişken bildirimi

```
public long  r ;
```

c. public metot bildirimi

```
public  double toplam(int m, double  d){
return m + d ;
}
```

2. private

Bazı değişken, metot ya da sınıflara başka sınıftaki kodların erişmesini engellemek isteyebiliriz. Bunun için private nitelemesini kullanırız. private erişim belirleyicisi, public belirtkesinin karşıtı gibidir. private damgalı öğelere yalnız aynı sınıftaki kodlar erişebilir, başka sınıftaki kodlar erişemez. Kendi alt-sınıfları bile erişemez. Bir alt-sınıf, atasının public ve ön-tanımlı öğelerine erişebilir, ama private öğelerine erişemez. Onlara erişebilmesi için, super class interface-fonksiyonu kullanılır. Bunun nasıl olduğunu ileride göreceğiz. Sözdizimi şöyledir:

a. private sınıf bildirimi

```
private class class_adı {
        Class gövdesi
}
```

b. private değişken bildirimi

```
private long  r ;
```

c. private metot bildirimi

```
private  double toplam(int m, double  d){
return m + d ;
}
```

3. protected

Bir sınıf içindeki değişken ve metotlara alt-sınıfların erişebilmesini, ama paket içindeki ya da program içindeki başka kodların erişmesini engellemek isteyebiliriz. Bunun için sözkonusu öğeyi, protected damgası ile nitelemek yetecektir. Bu demektir ki, alt-sınıf, üst-sınıfın protected damgalı öğelerine sanki public öğelermiş gibi erişir. Görüldüğü gibi, protected belirtkesi, ön-tanımlı belirtke ile private belirtkesinin işlevleri arasında bir işleve sahiptir. Alt sınıflara erişme yetkisi verdiği için, kalıtım (inheritance) olgusunda önemli rol oynar. Sözdizimi şöyledir:

a. protected sınıf bildirimi

```
protected class class_adı {
        Class gövdesi
}
```

b. protected değişken bildirimi

```
protected long  r ;
```

c. protected metot bildirimi

```
protected  double toplam(int m, double  d){
return m + d ;
}
```

1. Friendly

Bir öğenin önüne hiçbir erişim belirtkesinin konmadığı durumdur. Erişim belirtkesi konmamışsa ön-tanımlı (default) belirtke etkin olur. Buna, bazı kaynaklarda dostça erişim (frendly access) denir.

a. paket bildirimi

Paketler yalnızca ön-tanımlı erişime sahiptir, başka erişim belirtkesi almazlar. Paket içindeki her sınıf pakette olan her değişken ve metoda erişebilir. Ama başka paketlerdeki sınıflar erişemez.

```
  package paket_adı {
        Paket gövdesi  }
```

b. erişim belirtecsiz (default) sınıf bildirimi

Sınıf bidiriminde, sınıfın önüne hiçbir erişim belirtkesi konmazsa, o sınıf içindeki değişken ve metotlara o sınıfı içeren paketteki bütün kodlar erişebilir. Kuruluş yapısı şöyledir:

```
class class_adı {
   Class gövdesi
}
```

c. erişim belirteçsiz (default) metot bildirimi

Metot bildiriminde hiçbir erişim belirtkesi konulmazsa, default belirke geçerlidir. Bu halde, class'ın kendisi, alt-sınıflar ve aynı paket içindeki diğer sınıflar erişebilir. Örnek:

```
int topla (int m, int n) {
  return m+n ;
}
```

d. erişim belirteçsiz (default) değişken bildirimi

Değişken bildiriminde hiçbir erişim belirtkesi konulmazsa, ön-tanımlı (default) belirke geçerlidir. Bu halde, sınıfın kendisi, alt-sınıfları ve aynı paket içindeki diğer sınıflar erişebilir. Örnek:

```
float kesir;
```

Örnek-1:

```
class Test {
  int a;              // friendly erişim
  public int b;       // public erişim
  private int c;      // private erişim
      //  c  ye erişen metot
  void setc(int i) { // c  ye değer atar
    c = i;
  }
  int getc() {        // c  nin değeri
    return c;
  }
}
public class erisimbelirleyiciler {
    public static void main(String[] args) {

    Test ob = new Test();
    // a ile b ye direkt erişilebilir
    ob.a = 10;
    ob.b = 20;
    //  ob.c = 100;          // Hata!
    ob.setc(100); // geçerli
    System.out.println("a, b, ve c: " + ob.a + " " +
                       ob.b + " " + ob.getc());
  }
}
```

Örnek-2:

```
class Stack {
  private int stk[] = new int[10];
  private int top;
  Stack() {    top = -1;  }
  void push(int item) {
    if(top==9)
      System.out.println("Stack doldu.");
    else
      stk[++top] = item; }
  int pop() {
    if(top < 0) {
      System.out.println("Stack dolmadı.");
      return 0;    }
    else
      return stk[top--];
  }
}
public class erisim2 {
  public static void main(String args[]) {
    Stack st1 = new Stack();
    Stack st2 = new Stack();
    for(int i=0; i<10; i++) st1.push(i);
    for(int i=10; i<20; i++) st2.push(i);
    System.out.println("Stack in st1:");
    for(int i=0; i<10; i++)
      System.out.println(st1.pop());
    System.out.println("Stack in st2:");
    for(int i=0; i<10; i++)
      System.out.println(st2.pop());
  }}
```

PAKETLER (PACKAGES)

Paketler sınıfları düzenlemek için kullanılabilir. Bunu yapmak için, programdaki ilk yorumsuz ve boş olmayan ifade olarak aşağıdaki satırı eklemeniz gerekir:

Package paketadi;

Bir sınıf package deyimi olmadan tanımlanırsa, varsayılan pakete yerleştirildiği söylenir.

```
package  p1;

public class C1 {
  public int x;
  int y;
  private int z;

  public void m1() {
  }
  void m2() {
  }
  private void m3() {
  }
}
```

```
package  p1;

public class C2 {
  void aMethod() {
    C1 c1 = new C1();
    can access c1.x;
    can access c1.y;
    cannot access o.z;

    can invoke c1.m1();
    can invoke c1.m2();
    cannot invoke c1.m3();
  }
}
```

```
package p2;

public class C3 {
  void aMethod() {
    C1 c1 = new C1();
    can access c1.x;
    cannot access c1.y;
    cannot access c1.z;

    can invoke c1.m1();
    cannot invoke c1.m2();
    cannot invoke c1.m3();
  }
}
```

Paketler kurulurken şu kurallara uyulur:

1. Her paketin bir adı vardır. Paket adından hemen önce package anahtar sözcüğü yer alır.
2. Paketler erişim belirleyicisi almaz, ancak içerdiği sınıflar erişim belirtecine sahip olabilir.
3. Sınıftan ayırmak için, paket adları küçük harfle başlatılır.
4. Bir paket içinde aynı adı taşıyan iki sınıf ya da arayüz olamaz. Ama, ayrı paketlerde aynı adı taşıyan sınıflar ve arayüzler olabilir.

Veri Alanı Kapsülleme(Data Encapsulation)

Veri alanlarını private yapmak verileri korur ve sınıfın bakımını kolaylaştırır.

Cember sınıfındaki r ve nesnesay veri alanları doğrudan değiştirilebilir (örn. C1.r = 5 veya Cember.nesnesay = 10). Bu iki durumdan dolayı iyi bir uygulama değildir.

1. Veriler üzerinde oynanabilir. Örneğin, nesnesay oluşturulan nesne sayısını saymakla birlikte, yanlışlıkla rastgele bir değere ayarlanmış olabilir (örneğin, Cember. nesnesay = 10).

2. Sınıfın bakımı zorlaşır ve hatalara karşı savunmasız hale gelir. Diğer programlar sınıfı kullandıktan sonra yarıçapın negatif olmamasını sağlamak için Cember sınıfını değiştirmek istediğinizi varsayalım. Kullanıcılar yarıçapı doğrudan değiştirmiş olabileceğinden yalnızca Cember sınıfını değil, onu kullanan programları da değiştirmeniz gerekir (örn., C1.r = –5). Veri alanlarının doğrudan değiştirilmesini önlemek için, private erişim belirleyicisi kullanarak veri alanlarını private olarak bildirmeniz gerekir. Bu, veri alanı kapsülleme olarak bilinir.

private veri alanına, sınıfın dışındaki özel alanı tanımlayan bir nesne erişemez. Ancak, bir kullanıcının genellikle bir veri alanını alması ve değiştirmesi gerekir. private bir veri alanını erişilebilir hale getirmek için değerini döndürmek için bir get metodu tanımlamak gerekir. private veri alanının değerini değiştirmek için, yeni bir değer ayarlamak üzere bir set metodu tanımlanır.

Get metodu:

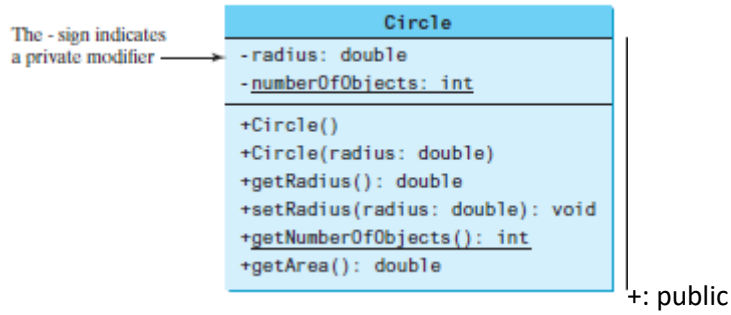
public returnType getmetodadi()

Set Metodu:

public void setmetodadi(veritipi ozellikadi)

```
class cember{
    private double r;
     private static int nesnesay=0;
    cember(){
        r=1.0;
        nesnesay++;   }
    cember(double nr){
        r=nr;
        nesnesay++; }
    double getAlan(){         return Math.PI*r*r;    }
    public double getR(){        return r;     }
    public void setR(double nr){   r=nr;    }
    double getCevre(){
        return 2*Math.PI*r;
    }
   public  int getnesnesay(){         return nesnesay;     }
}
public class Siniflar {
    public static void main(String[] args) {
     cember c1=new cember();
        System.out.println("Nesne sayısı:"+c1.getnesnesay());
       // cember.nesnesay=4;
     cember c2=new cember(2.7);
        System.out.println("Nesne sayısı:"+c1.getnesnesay());
        System.out.println(c1.getR());
        System.out.println(c2.getR());
        System.out.println(Math.PI);
    }

}
```



+: public

- :private

Altı çizili: static

Nesne Alan Metotlar

Nesneleri metotlara parametre olarak verebilirsiniz. Aynı şekilde bir metot bir nesne döndürebilir. Bir diziyi geçirmek gibi, bir nesneyi geçirmek de aslında nesnenin referansını geçmektedir.

```
public class Test {
public static void main(String[] args) {
cember c = new cember(5.0);
printCember(c);
}

public static void printCember(cember c) {
System.out.println("The area of the circle of radius "
+ c.getRadius() + " is " + c.getArea());
```

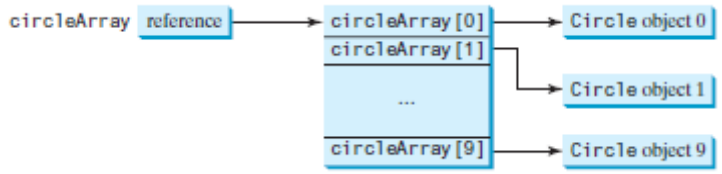
}
}

Nesne Dizileri

Bir dizi, ilkel tip değerlerin yanı sıra nesneleri de tutabilir.
Nesnelerin dizilerini de oluşturabilirsiniz. Örneğin, aşağıdaki ifade 10 Circle nesnesinden oluşan bir dizi bildirir ve oluşturur:
int [] x=new int[10];

0	0	0	0	0	0	0	0	0	0

cember [] c1 = yeni cember [10];
C1 i başlatmak için, aşağıdaki gibi bir for döngüsü kullanabilirsiniz.
for (**int** i = **0**; i < c1.length; i++) {
c1[i] = **new** cember();
}



Değişkenlerin Kapsamı

Örnek ve statik değişkenlerin kapsamı, değişkenlerin nerede bildirildiğine bakılmaksızın tüm sınıftır.
Yerel değişkenler bir yöntem içinde yerel olarak bildirilir ve kullanılır. Bu bölümde, bir sınıf bağlamındaki tüm değişkenlerin kapsam kuralları açıklanmaktadır.
Sınıftaki örnek ve statik değişkenler, sınıfın değişkenleri veya veri alanları olarak adlandırılır. Bir method içinde tanımlanan bir değişkene yerel değişken denir. Bir sınıfın değişkenlerinin kapsamı, değişkenlerin nerede bildirildiğine bakılmaksızın tüm sınıftır.

```
public class Circle {
  public double getArea() {
    return radius * radius * Math.PI;
  }

  private double radius = 1;
}
```

```
public class F {
  private int i;
  private int j = i + 1;
}
```

Yerel bir değişken sınıf değişkeniyle aynı ada sahipse, yerel değişken öncelik kazanır ve sınıfın aynı ada sahip değişkeni gizlenir. Örneğin, aşağıdaki programda, x hem örnek değişkeni hem de yöntemde yerel değişken olarak tanımlanır:

```
public class F {
  private int x = 0; // Instance variable
  private int y = 0;

  public F() {
  }

  public void p() {
    int x = 1; // Local variable
    System.out.println("x = " + x);
    System.out.println("y = " + y);
  }
}
```

this Anahtar Kelimesi

Bu anahtar kelimenin nesnenin kendisini ifade eder.
Aynı sınıftaki başka bir kurucuyu çağırmak için bir kurucu içinde de kullanılabilir.
Bu anahtar kelime, bir nesnenin kendisine başvurmak için kullanabileceği bir referansın adıdır.
Bu anahtar kelimeyi, nesnenin örnek üyelerine başvurmak için kullanabilirsiniz.

Örneğin, (a) 'daki aşağıdaki kod bunu nesnenin yarıçapına başvurmak için kullanır ve getAlan () yöntemini açıkça çağırır.
Bu referans normal olarak (b) 'de gösterildiği gibi kısalık için atlanmıştır.
Ancak, bu başvuru, bir yöntem veya yapıcı parametresi tarafından gizlenen bir veri alanına başvurmak veya aşırı yüklenmiş bir yapıcı çağırmak için gereklidir.

```
public class Circle {
  private double radius;

  ...

  public double getArea() {
    return this.radius * this.radius * Math.PI;
  }

  public String toString() {
    return "radius: " + this.radius
       + "area: " + this.getArea();
  }
}
```

(a)

Equivalent

```
public class Circle {
  private double radius;

  ...

  public double getArea() {
    return radius * radius * Math.PI;
  }

  public String toString() {
    return "radius: " + radius
       + "area: " + getArea();
  }
}
```

(b)

this Anahtar Kelimesini Veri Alanlarına Değer Atamak için Kullanım

this anahtar kelimesi veri alanı için sınıfın veri alanını temsil etmek için kullanılır. Bir metodun lokal değişkeni ile sınıfın değişkeni aynı isimli olabilir. Dolayısıyla aşağıdaki kodda da gösterildiği gibi this.r sınıfın r'sini, diğer r ise parametre olarak gelen r'yi ifade eder.
Örnek:

```
class cember {
double r;
void setR(double r){
this.r=r;
}
```

Aşağıdaki kullanım ise yanlıştır.

```
class cember {
double r;
void setR(double r){
r=r;}
```

Gizli bir statik değişkene sadece sinifadi.staticdegisken başvurusu kullanılarak erişilebilir.

```
public class F {
  private int i = 5;
  private static double k = 0;

  public void setI(int i) {
    this.i = i;
  }

  public static void setK(double k) {
    F.k = k;
  }

  // other methods omitted
}
```

(a)

```
Suppose that f1 and f2 are two objects of F.

Invoking f1.setI(10) is to execute
   this.i = 10, where this refers f1

Invoking f2.setI(45) is to execute
   this.i = 45, where this refers f2

Invoking F.setK(33) is to execute
   F.k = 33. setK is a static method
```

(b)

Kurucuyu Çağırmak için this Kullanımı

this anahtar kelimesi, aynı sınıftan başka bir kurucu çağırmak için kullanılabilir. Örneğin, Cember sınıfını aşağıdaki gibi yeniden yazabilirsiniz:

```
public class cember {
private double r;
public cember(double r) {
```

```
this.r = r;
}
public cember() { this(1.0);}
...
}
```