KALITIM
(Inheritance)

Java'da kalıtım nedir?

Nesne Yönelimli Programlama dillerinde kalıtım olgusu, bir sınıfta (class) tanımlanmış değişkenlerin ve/veya metotların (fonksiyon, procedure) yeniden tanımlanmasına gerek olmaksızın yeni bir sınıfa taşınabilmesidir. Bunun için yapılan iş, bir sınıftan bir alt-sınıf (subclass) türetmektir. Türetilen alt-sınıf, üst-sınıfta tanımlı olan bütün değişkenlere ve metotlara sahip olur. Bu özeliğe kalıtım özeliği (inheritance) diyoruz.

- Programcı, yeni alt-sınıfları tanımlarken, üst-sınıftan (superclass) kalıtsal olarak geleceklere ek olarak, kendisine gerekli olan başka değişken ve metotları da tanımlayabilir.
- Bu yolla, bir kez kurulmuş olan sınıfın tekrar tekrar kullanılması olanaklı olur. Böylece, programlar daha kısa olur, programın yazılma zamanı azalır ve gerektiğinde değiştirilmesi ve onarılması (debug) kolay olur.
- Alt-sınıf türetme hiyerarşik bir yapıda olur. Bir alt-sınıfın türetildiği sınıf, o alt-sınıfın üst-sınıfıdır. Java'da bir alt-sınıfın ancak bir tane üst-sınıfı olabilir (C++ 'dakinden farklı olduğuna dikkat ediniz). Ama bir sınıftan birden çok alt-sınıf türetilebilir.

Üst-sınıfa ata (parent), alt-sınıfa da oğul (child) denir.

```
class A {
  int i, j;

  void showij() {
    System.out.println("i and j: " + i + " " + j);
  }
}
class B extends A {
  int k;

  void showk() {
    System.out.println("k: " + k);
  }
  void sum() {
    System.out.println("i+j+k: " + (i+j+k));
  }
}
public class BasitInheritance {
  public static void main(String args[]) {
    A ustOb = new A();       // A 'ya ait nesne
    B altOb = new B();       // B 'ye ait nesne

        // Anlık değişkenlere atama yapılıyor.
    ustOb.i = 15;
    ustOb.j = 25;
    System.out.println(" ustOb nesnesinin öğeleri: ");
    ustOb.showij();    //ustOb 'nin öğelerini yazar
    System.out.println();
    altOb.i = 3;
    altOb.j = 5;
    altOb.k = 7;
    System.out.println("altOb nesnesinin öğeleri: ");
    altOb.showij();    //üst-sınıftaki değişkenleri yazar
    altOb.showk();     //alt-sınıftaki değişkeni yazar
    System.out.println();
    System.out.println("üst ve alt sınıftaki değişkenler toplanıyor… ");
   System.out.println("(i + j + k) = ");

    altOb.sum();
  }
}
```

`private` damgalı öğelere erişim

Üst-sınıfın private öğelerine, kendi alt sınıfı da dahil olmak üzere başka hiçbir sınıftaki kodlar erişemez. Aşağıdaki listede tanımlanan B alt-sınıfındaki kodlar, A üst-sınıfının `private` damgalı öğelerine erişemiyor.

```
        /*  Üst-sınıfın private öğelerini  alt-sınıfın kodları kullanamaz.     */
class A {
  int i;                // erişim kısıtı yok
  private int j;        // erişim kısıtlanıyor
  void setij(int x, int y) {
    i = x;
    j = y;
  }
}

        // A'nın j öğesine erişemez.
class B extends A {
  int toplam;

  void sum() {
    total = i + j;  // HATA!, alt-sınıftaki bu kod j ye erişemez
  }
}

public class DegerVer {
  public static void main(String args[]) {
    B subOb = new B();

    subOb.setij(15, 20);

    subOb.sum();
    System.out.println("Toplam : " + subOb.toplam);
  }
}
```

Süper sınıf ve alt sınıf

Kalıtım, genel bir sınıf (yani bir üst sınıf) tanımlamanızı ve daha sonra onu daha özel sınıflara (yani alt sınıflar) genişletmenizi sağlar.

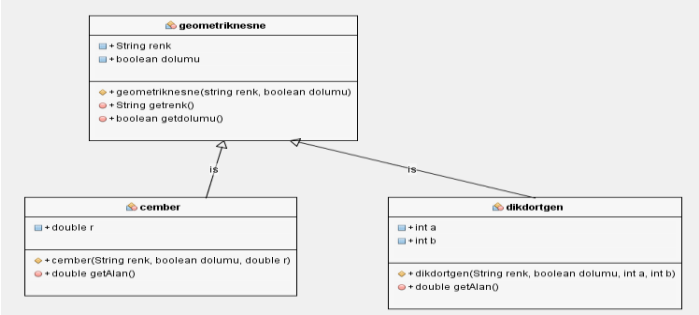
Aynı türdeki nesneleri modellemek için bir sınıf kullanırsınız. Farklı sınıflar, diğer sınıflar tarafından paylaşılabilen bir sınıfta genelleştirilebilen bazı ortak özelliklere ve davranışlara sahip olabilir. Genelleştirilmiş sınıfı genişleten özel bir sınıf tanımlayabilirsiniz. Özelleştirilmiş sınıfları, özellikleri ve yöntemleri genel sınıftan alır.

Aşağıdaki şekilde geometrik nesne bir süper sınıf çember ve dikdörtgen sınıfları ise alt sınıf olarak tanımlanmıştır.

Dolayısıyla geometriknesen sınıfındaki public metot ve değişkenler iki alt sınıf tarafından kullanılır.

public class cember extends geometriknesne{ }

public class dikdortgen extends geometriknesne{ }




```java
package kalitim;
public class geometriknesne{
        String renk;
        boolean dolumu;
        public geometriknesne(String renk, boolean dolumu) {
            this.renk = renk;
            this.dolumu = dolumu;
        }
        public String getrenk(){
            return renk;
        }
        public boolean getdoulumu(){
            return dolumu;
        }
    }
```

```java
package kalitim;
public class cember extends geometriknesne{
    double r;
    public cember(double r, String renk, boolean dolumu) {
        super(renk, dolumu);
        this.r = r;
    }
    double alanHesapla(){        return Math.PI*r*r;     }
}
```

```java
package kalitim;
public class dikdortgen extends geometriknesne{
    double a,b;

    public dikdortgen(double a, double b, String renk, boolean dolumu) {
        super(renk, dolumu);
        this.a = a;
        this.b = b;
    }
    public double getAlan(){return a*b;}
}
```

Süper Anahtar Kelimesi

Super anahtar kelimesi, üst sınıfı ifade eder ve üst sınıfın yöntemlerini ve kurucu metotlarını çağırmak için kullanılabilir.

Kurucu metot, sınıfın bir örneğini oluşturmak için kullanılır. Özelliklerden ve yöntemlerden farklı olarak, bir üst sınıfın kurucuları bir alt sınıf tarafından miras alınmaz. Yalnızca süper anahtar kelimesini kullanarak alt sınıfların kurucularından çağrılabilirler.

Üst sınıfın yapıcısını çağırmak için sözdizimi şudur: super () veya super (argümanlar);

Super () ifadesi, üst sınıfının parametre almayan kurucusunu çağırır ve super (arguments) ifadesi, bağımsız değişkenlerle eşleşen üst sınıf kurucusunu çağırır. Super () veya super (argümanlar) ifadesi, alt sınıfın yapıcısının ilk ifadesi olmalıdır; bir üst sınıf yapıcısını açıkça çağırmanın tek yolu budur. Yani süper kelimesi ilk komu olmalıdır.

Daha önceki örnekte super(renk, dolumu) isimli metot süper sınıf olan geometriknesne sınıfının kurucu metodunu çağırmaktadır. Dikkat edilirse bu komut kurucu metotta ilk komut olarak verilmiştir.

NOT: Üst sınıf kurucusunu çağırmak için super anahtar sözcüğünü kullanmalısınız ve çağrı kurucudaki ilk ifade olmalıdır. Üst sınıf kurucusunun adını bir alt sınıfta çağırmak sözdizimi hatasına neden olur.

Kurucu Zinciri

Bir kurucu metot aşırı yüklenmiş bir kurucu veya onun üst sınıf kurucusunu çağırabilir. Her ikisi de açıkça çağrılmazsa, derleyici otomatik olarak super () öğesini kurucudaki ilk ifade olarak koyar.

```java
public ClassName() {
  // some statements
}
```

Equivalent

```java
public ClassName() {
  super();
  // some statements
}
```

```java
public ClassName(parameters) {
  // some statements
}
```

Equivalent

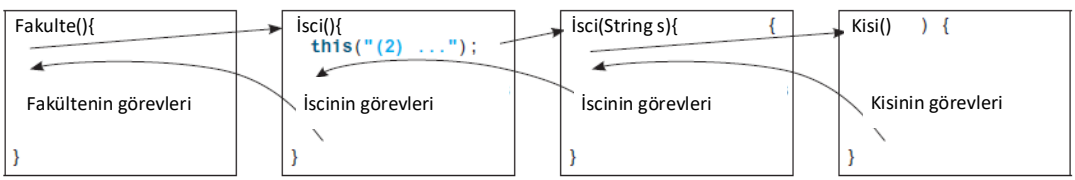
```java
public ClassName(parameters) {
  super();
  // some statements
}
```

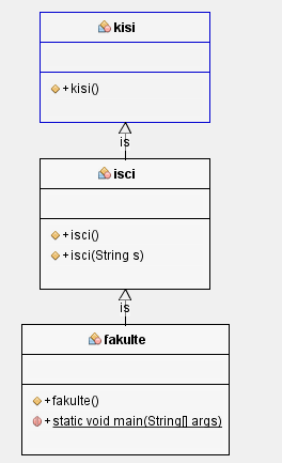
Herhangi bir durumda, bir sınıf nesnesi oluşturmak, miras zinciri boyunca tüm üst sınıfların kurucularını çağırır. Bir alt sınıfın bir nesnesini oluştururken, alt sınıf kurucusu kendi görevlerini gerçekleştirmeden önce üst sınıf kurucusunu çağırır. Üst sınıf başka bir sınıftan türetilmişse, üst sınıf kurucusu kendi görevlerini gerçekleştirmeden önce üst sınıf kurucusunu çağırır. Bu süreç, miras hiyerarşisindeki son kurucu çağrılıncaya kadar devam eder. Buna kurucu zinciri denir.
Örnek:

```
package kalitim;

public class fakulte extends isci {
 public static void main(String[] args) {
 new fakulte();
 }
 public fakulte() {
 System.out.println("(4) Fakultenin görevleri");
 }
 }
 class isci extends kisi {
 public isci() {
 this("(2) isci asırı yükleme parametreli metot");
 System.out.println("(3) iscinini gorevleri ");
 }
 public isci(String s) {
 System.out.println(s);
 }
 }
class kisi {
 public kisi() {
 System.out.println("(1) kisinin gorevleri");
 }
 }
```

```
Output - kalitim (run)
   run:
   (1) kisinin gorevleri
   (2) isci asırı yükleme parametreli metot
   (3) iscinini gorevleri
   (4) Fakultenin görevleri
```





Süper Sınıfın Metodunu Çağırma

Süper anahtar sözcüğü, üst sınıftaki kurucudan başka bir yönteme başvurmak için de kullanılabilir. Sözdizimi

super.metotadi (parametreler);

Örnek: Süper sınıfın veri alanına erişim

```
class arac
{
    int makshiz = 120;
}
/* araba alt sınıfı araçtan extends edildi */
class araba extends arac
{
    int maxSpeed = 180;

    void display()
    {
        /* super sınıfın makismum hızı:*/
        System.out.println("Maksimum hız: " + super.makshiz);
    }
}
public class superanahtar
{
    public static void main(String[] args)
    {
        araba small = new araba();
        small.display();
    }
}
```

Yukarıdaki örnekte hem üst sınıfta hem alt sınıfta makshiz değişkeni vardır. Üst sınıftaki değişkene ulaşmak için süper anahtar kelimesi kullanılmıştır.

```java
class kisi{
    void mesaj()
    {
        System.out.println("Kişi sınıfı");
    }
}
class ogrenci extends kisi{
    void mesaj()
    {
        System.out.println("ogrenci sınıfı");
    }
    void goster()
    {
        // mevcut sınıftaki mesaj metodu çağrılır.
        mesaj();

        // super sınıftaki mesaj metodu çağrılır.
        super.mesaj();
    }
}
public class supermetot {
     public static void main(String args[])
    {
        ogrenci s = new ogrenci();
        s.goster();
    }
}
```

Üzerine Yazma/Geçersiz Kılma (Overriding)

Bir yöntemi geçersiz kılmak için, yöntem alt sınıfında üst sınıfındakiyle aynı imza kullanılarak tanımlanmalıdır. Bir alt sınıf, yöntemleri bir üst sınıftan devralır. Bazen, alt sınıfın üst sınıfta tanımlanan bir yöntemin uygulanmasını değiştirmesi gerekir. Buna yöntem geçersiz kılma denir.

Aşırı Yükleme(Overloading)- Üzerine Yazma Karşılaştırması

Aşırı yükleme, aynı ada ancak farklı imzalara sahip birden çok yöntem tanımlamak anlamına gelir. **Geçersiz kılma**, alt sınıftaki bir yöntem için yeni bir uygulama sağlamak anlamına gelir.

Bir yöntemi geçersiz kılmak için, yöntemin aynı sınıf ve aynı veya uyumlu dönüş türü kullanılarak alt sınıfta tanımlanması gerekir.

Geçersiz kılma ve aşırı yükleme arasındaki farkları göstermek için bir örnek ele alalım. Aşağıda (a) 'da, A sınıfındaki p (double i) yöntemi, B sınıfında tanımlanan aynı yöntemi geçersiz kılar. (b)' de, A sınıfı iki aşırı yüklenmiş yönteme sahiptir: p (double i) ve p (int i).

yöntem p (çift i) B'den miras alınır.

```java
public class TestOverriding {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

(a)

```java
public class TestOverloading {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

(b)

Aşağıdakilere dikkat etmek gerekir:

- Geçersiz kılınan yöntemler miras ile ilgili farklı sınıflardadır; aşırı yüklenmiş yöntemler aynı sınıfta veya kalıtımla ilişkili farklı sınıflarda olabilir.
- Geçersiz kılınan yöntemlerin aynı imzası vardır; aşırı yüklenmiş yöntemler aynı ada ancak farklı parametre listelerine sahiptir.

Object sınıfı ve toString() Metodu

Java'daki her sınıf, java.lang.Object sınıfından alınmıştır.
Bir sınıf tanımlandığında miras belirtilmezse, sınıfın üst sınıfı varsayılan olarak Object'dir. Örneğin, aşağıdaki iki sınıf tanımı aynıdır:

```
public class ClassName {
  ...
}
```

Equivalent

```
public class ClassName extends Object {
  ...
}
```

Sınıflarınızda kullanabilmeniz için Object sınıfı tarafından sağlanan yöntemlere aşina olmanız önemlidir. Bu bölüm, Object sınıfındaki toString yöntemini tanıtır. ToString () yönteminin imzası:
public String toString ()
Bir nesne üzerinde toString () çağrıldığında, nesneyi tanımlayan bir String döndürülür. Varsayılan olarak, nesnenin örnek olduğu bir sınıf adı, bir at işareti (@) ve nesnenin bellek adresini onaltılık olarak içeren bir String döndürür.Örneğin aşağıdaki üçgen sınıfı siniflar.ucgen@4554617c gibi bir değer döndürür.

```
class ucgen{
    double a,b,c;
    public ucgen(double a, double b, double c) {
        this.a = a;
        this.b = b;
        this.c = c;
    }

    double alanhesapl(){
        double s=cevrehesap()/2;
        double alan=Math.sqrt(s*(s-a)*(s-b)*(s-c));
        return alan;
    }
    double cevrehesap(){
        return a+b+c;
    }
    public String toString(){
        return "Kenarları "+a+" "+b+" "+c+" olan üçgenin alanı:"+alanhesapl()+" çevresi:"+cevrehesap();
    }
}
public class overridemetot{
    public static void main(String[] args) {
        ucgen u=new ucgen(3.2, 4.5, 6.8);
        System.out.println(u);
    }
}
```

Polymorphism

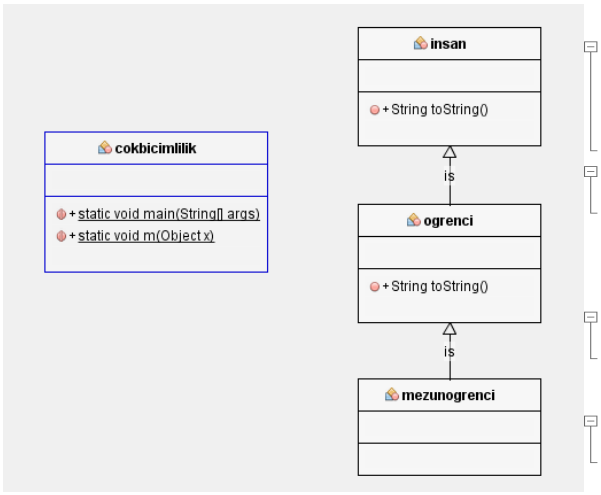
Polimorfizm, bir süper tip değişkeninin bir alt tip nesnesine işaret edebileceği anlamına gelir. Nesneye yönelik programlamanın üç temeli; kapsülleme, kalıtım ve polimorfizmdir. İlk ikisini zaten öğrendiniz. Bu bölüm polimorfizmi tanıtır. Kalıtım ilişkisi, bir alt sınıfın, ek yeni özelliklerle üst sınıfından özellikler devralmasını sağlar. Bir alt sınıf, üst sınıfının özelleştirilmesidir; her alt sınıf örneği aynı zamanda üst sınıfının bir örneğidir, ancak tam tersi doğru değildir. Örneğin, her daire geometrik bir nesnedir, ancak her geometrik nesne bir daire değildir. Bu nedenle, alt sınıfın bir örneğini her zaman üst sınıf türündeki bir parametreye iletebilirsiniz.

```java
package kalitim;
public class Kalitim {
    public static void main(String[] args) {
      cember c=new cember(2, "mavi", false);
      dikdortgen d=new dikdortgen(2, 3, "sari", true);
      System.out.println(c.getrenk());
      System.out.println(c.getdoulumu());
      goster(c);
        goster(d);

    }
    public static void goster(geometriknesne g){
        System.out.println("Geometrik nesnenin rengi:"+g.getrenk()+" dolu olma durumu:"+g.dolumu);
    }

}
```

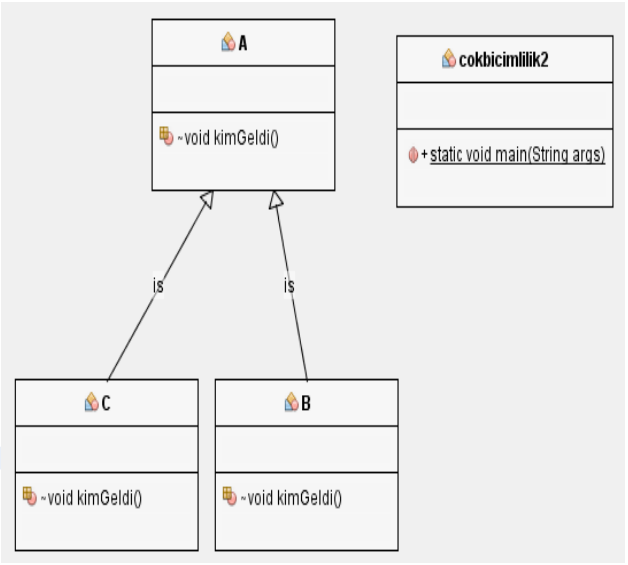
Yukarıdaki örnekte her çember ve dikdörtgen aynı zamanda bir geometrik nesnedir. Dolayısıyla geometrik nesneyi parametre alan bir metoda hem çember sınıfından bir nesne hem de dikdörtgen sınıfından bir nesne parametre olarak gönderilebilir. Bu çok biçimliliğin kullanımıdır.




```java
package kalitim;
public class cokbicimlilik {
    public static void main(String[] args) {
        m(new mezunogrenci());
        m(new ogrenci());
        m(new insan());
        m(new Object());
    }
    public static void m(Object x) {
        System.out.println(x.toString());
    }
}
class mezunogrenci extends ogrenci {
}
class ogrenci extends insan {
    public String toString() {
        return "Ogrenci";
    }
}
class insan extends Object {
    public String toString() {
        return "İnsan";
    }
}
```

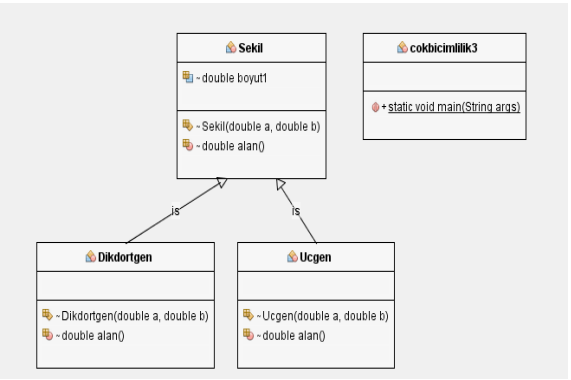
Örnek-2:

```java
package kalitim;
class A {
   void kimGeldi() {
     System.out.println("Şimdi A 'daki kimGeldi methodu çağrıldı.");
  }
}
class B extends A {
       // override kimGeldi()
  void kimGeldi() {
    System.out.println("Şimdi B 'deki kimGeldi methodu çağrıldı. ");
  }
}
class C extends A {
       // override kimGeldi()
  void kimGeldi() {
    System.out.println("Şimdi C 'deki kimGeldi methodu çağrıldı. ");
  }
}
public class cokbicimlilik2 {
  public static void main(String args[]) {
    A a = new A();     // A tipinden nesne
    B b = new B();     // B tipinden nesne
    C c = new C();     // C tipinden nesne
    A r;               // A tipinden bir referans (işaretçi, pointer)
    r = a;             // r  referansı A 'nın bir nesnesini işaret ediyor
    r.kimGeldi();      // A ' daki kimGeldi metodunu çağırıyor
    r = b;             // r  referansı B 'nin bir nesnesini işaret ediyor
    r.kimGeldi();      // B ' deki kimGeldi metodunu çağırıyor
    r = c;             // r  referansı C 'nin bir nesnesini işaret ediyor
    r.kimGeldi();      // C ' deki kimGeldi metodunu çağırıyor
  }
}
```



Örnek-3:

```java
class Sekil {
  double boyut1,boyut2;
  Sekil(double a, double b) {
    boyut1 = a;  boyut2 = b;  }
double alan() {
    System.out.println("Şeklin alanı henüz tanımlı değil! ");
    return 0;  }}
class Dikdortgen extends Sekil {
  Dikdortgen(double a, double b) {    super(a, b);  }
   // Dikdörtgen için override alan
  double alan() {
    System.out.println("Dikdörtgenin alanı.");
    return boyut1 * boyut2;  }
}
class Ucgen extends Sekil {
  Ucgen(double a, double b) {
    super(a, b);  }
  double alan() {
    System.out.println("Üçgenin alanı.");
    return boyut1 * boyut2 / 2;  }}
public class cokbicimlilik3 {
  public static void main(String args[]) {
    Sekil f = new Sekil(15, 15);
    Dikdortgen r = new Dikdortgen(7, 4);
    Ucgen t = new Ucgen(9, 6);
    Sekil ref=r;
    System.out.println("Alan = " + ref.alan());
    ref = t;
    System.out.println("Alan = " + ref.alan());
    ref = f;
    System.out.println("Alan = " + ref.alan());
  }
}
```



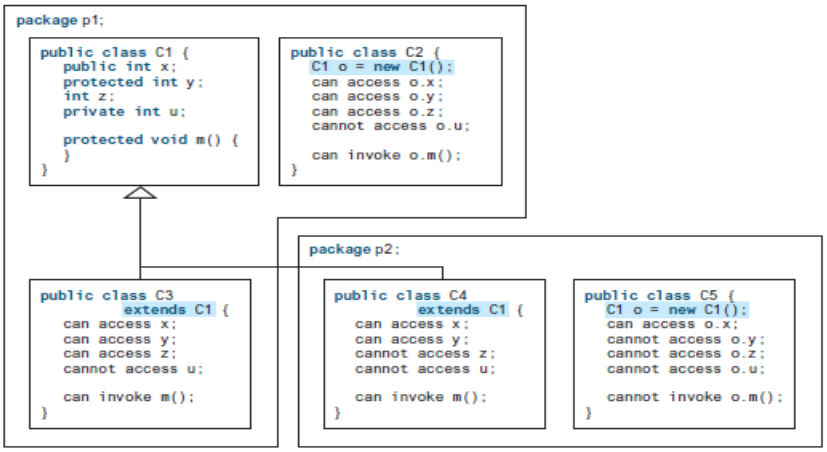
Protected Erişim Belirleyicisi

Bir sınıfın protected bir üyesine bir alt sınıftan erişilebilir. Veri alanlarına ve yöntemlerine sınıfın dışından erişilip erişilemeyeceğini belirtmek için şimdiye kadar private ve public anahtar kelimeleri kullandık. private üyelere yalnızca sınıfın içinden erişilebilir ve public üyelere diğer sınıflardan erişilebilir.

Genellikle, alt sınıfların üst sınıfta tanımlanan veri alanlarına veya yöntemlerine erişmesine izin vermek, ancak farklı paketlerdeki alt sınıfların bu veri alanlarına ve yöntemlerine erişmesine izin vermemek arzu edilir. Bunu yapmak için korumalı anahtar kelimeyi kullanabilirsiniz. Bu şekilde, bir üst sınıftaki korumalı veri alanlarına veya yöntemlerine alt sınıflarından erişebilirsiniz.

Sınıfın üyelerini doğrudan sınıfın dışından erişemeyecekleri şekilde gizlemek için private belirleyicisini kullanın. Sınıf üyelerine aynı paket içindeki herhangi bir sınıftan doğrudan erişilmesine izin vermek, ancak diğer paketlerden izin vermemek için herhangi bir erişim belirleyicisi kullanmayın(friendly).Sınıf üyelerine aynı paketteki veya herhangi bir paketteki alt sınıflardan erişimi sağlamak için protect erişim belirleyicisini kullanın. Sınıf üyelerine herhangi bir sınıf tarafından erişilmesini sağlamak için public erişim belirleyicisini kullanın.

Modifier on Members in a Class	*Accessed from the Same Class*	*Accessed from the Same Package*	*Accessed from a Subclass in a Different Package*	*Accessed from a Different Package*
Public	✓	✓	✓	✓
Protected	✓	✓	✓	–
Default (no modifier)	✓	✓	–	–
Private	✓	–	–	–



Miras Alma ve Override'ı Geçersiz Kılma

Bazı durumlarda bir sınıfın hiçbir değişikliğe uğratılmadan korunması gerekir. Bu durumlarda, o sınıfın hiçbir alt-sınıfının oluşturulmaması gerekir. Bunu yapmak için, sınıf tanımında final nitelemini kullanırız. Final nitelemi ile damgalı bir sınıfın metotları da kendiliğinden final nitelemini alırlar. Aynı şekilde bir metodun alt sınıf tarafından override edilmesini engellemek için metot final olarak tanımlanır. Örneğin Java'daki Math sınıfı final olup extend edilemez. Aşağıdaki A sınıfı final olduğu için extend edilemez.

```
public final class A {

}
```

Son olarak bir yöntem de tanımlayabilirsiniz; final bir yöntem alt sınıfları tarafından override edilemez.

```
public class Test {

public final void m() {

}
}
```

Örnek:

```
class A {
  final void deneme() {
    System.out.println("Bu bir final metot'tur.");
  }
}

class B extends A {
  void deneme() { // HATA! Override edilemez.
    System.out.println("Geçersidir!");
  }
```

```
}
```

Örnek:

```
final class A {
        // ...
}

        // bu alt sınıf yaratılamaz.
class B extends A {          // HATA!
        // ...
}
```