Overriding

Bir alt-sınıfta tanımlanan bir metodun veri-tipi (değer bölgesi), adı ve parametrelerinin sayısı ve tipleri, üst-sınıfında tanımlı bir başka metodun veri-tipi (değer bölgesi), adı ve parametrelerinin sayısı ve tipleri ile, karşılıklı olarak, aynı ise, alt-sınıftaki metot üst-sınıftakini örtüyor (overriding) denilir. Ön-tanımlı (default) olarak, alt-sınıf öncelikle kendi öğelerini görür, üst-sınıftakini görmez. Üst sınıftaki öğeleri görebilmesi için özel yöntemler gerekir. Değişkenler için bunun nasıl yapıldığını önceki bölümlerde öğrendik. Metotlar için de benzer yöntem uygulanır.

Liste 10:

Aşağıdaki program overriding metoduna örnektir.

```
        // overriding metot.
class A {
  int i, j;

  A(int a, int b) {
    i = a;
    j = b;
  }

        // i ile j yi yazar
  void show() {
    System.out.println("i and j: " + i + " " + j);
  }
}

class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }

                // k 'yı göster – A içindeki show() metodunu örter.
  void show() {
    System.out.println("k: " + k);
  }
}

class Override {
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);

    subOb.show();       // B 'deki show() metodunu çağırır
  }
}
```

Liste 11:

Üst-sınıfta örtülen metodu çağırmak istersek, super anahtar sözcüğünü kullanabiliriz. Aşağıdaki program bunun nasıl yapılacağını göstermektedir.

```
class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }

  void show() {
    super.show();       // A 'daki show() metodunu çağırır
    System.out.println("k: " + k);
  }
}
```

Override ve overload arasındaki fark

Bir alt-sınıfta tanımlanan bir metodun veri-tipi (değer bölgesi), adı ve parametrelerinin sayısı ve tipleri, üst-sınıfında tanımlı bir başka metodun veri-tipi (değer bölgesi), adı ve parametrelerinin sayısı ve tipleri ile, karşılıklı olarak, aynı ise, alt-sınıftaki metot üst-sınıftakini örtüyor (overriding) denilir.

Bir sınıf içindeki iki metodun adları aynı olduğu halde formal parametreleri farklı ise bu iki metot aşkındır (overloaded). Aşkın metotlardan biri alt-sınıfta öteki üst sınıfta da olabilir.

Liste 12:

Aşağıdaki program overridden ile overloaded arasındaki farkı göstermektedir.

```
        // overloaded metot -- not overridden'den farklıdır.
class A {
  int i, j;

  A(int a, int b) {
    i = a;
    j = b;
  }

        // i ile j  yi göster
  void show() {
    System.out.println("i and j: " + i + " " + j);
  }
}

        // A 'nın alt-sınıfı
class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }

        // overload show()
  void show(String msg) {
    System.out.println(msg + k);
  }
}

class Override {
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);

    subOb.show("This is k: "); // this calls show() in B
    subOb.show();       // A 'daki show() metodunu çağırır
  }
}
```

`show()` metodu A üst-sınıfında ve B alt_sınıfında aynı ad ve aynı tiptendir. A üst-sınıfındaki `show()` metodunun parametresi yoktur. B 'deki `show()` metodunun ise `String` tipinden bir parametresi vardır. Dolayısıyla, B' deki `show()` metodu A' dakini overload eder; override değil.

```
//Liste 16
// Dynamic Method Dispatch
class A {
   void callme() {
     System.out.println("Inside A's callme method");
  }
}
```

```
class B extends A {
  // override callme()
  void callme() {
    System.out.println("Inside B's callme method");
  }
}

class C extends A {
  // override callme()
  void callme() {
    System.out.println("Inside C's callme method");
  }
}

class Dispatch {
  public static void main(String args[]) {
    A a = new A(); // object of type A
    B b = new B(); // object of type B
    C c = new C(); // object of type C
    A r; // obtain a reference of type A

    r = a; // r refers to an A object
    r.callme(); // calls A's version of callme

    r = b; // r refers to a B object
    r.callme(); // calls B's version of callme

    r = c; // r refers to a C object
    r.callme(); // calls C's version of callme
  }
}



//Liste 17
// Using run-time polymorphism.
class Figure {
  double dim1;
  double dim2;

  Figure(double a, double b) {
    dim1 = a;
    dim2 = b;
  }

  double area() {
    System.out.println("Area for Figure is undefined.");
    return 0;
  }
}

class Rectangle extends Figure {
  Rectangle(double a, double b) {
    super(a, b);
  }

  // override area for rectangle
  double area() {
    System.out.println("Inside Area for Rectangle.");
    return dim1 * dim2;
  }
}

class Triangle extends Figure {
  Triangle(double a, double b) {
    super(a, b);
  }

  // override area for right triangle
  double area() {
    System.out.println("Inside Area for Triangle.");
```

```
    return dim1 * dim2 / 2;
  }
}

class FindAreas {
  public static void main(String args[]) {
    Figure f = new Figure(10, 10);
    Rectangle r = new Rectangle(9, 5);
    Triangle t = new Triangle(10, 8);

    Figure figref;

    figref = r;
    System.out.println("Area is " + figref.area());

    figref = t;
    System.out.println("Area is " + figref.area());

    figref = f;
    System.out.println("Area is " + figref.area());
  }
}
```

```
//Liste 18
// A Simple demonstration of abstract.
abstract class A {
  abstract void callme();

  // concrete methods are still allowed in abstract classes
  void callmetoo() {
    System.out.println("This is a concrete method.");
  }
}

class B extends A {
  void callme() {
    System.out.println("B's implementation of callme.");
  }
}

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

    b.callme();
    b.callmetoo();
  }
}
```

```
//Liste 19
// Using abstract methods and classes.
abstract class Figure {
  double dim1;
  double dim2;

  Figure(double a, double b) {
    dim1 = a;
    dim2 = b;
  }

  // area is now an an abstract method
  abstract double area();
}

class Rectangle extends Figure {
  Rectangle(double a, double b) {
    super(a, b);
```

```
  }

  // override area for rectangle
  double area() {
    System.out.println("Inside Area for Rectangle.");
    return dim1 * dim2;
  }
}

class Triangle extends Figure {
  Triangle(double a, double b) {
    super(a, b);
  }

  // override area for right triangle
  double area() {
    System.out.println("Inside Area for Triangle.");
    return dim1 * dim2 / 2;
  }
}

class AbstractAreas {
  public static void main(String args[]) {
  // Figure f = new Figure(10, 10); // illegal now
    Rectangle r = new Rectangle(9, 5);
    Triangle t = new Triangle(10, 8);

    Figure figref; // this is OK, no object is created

    figref = r;
    System.out.println("Area is " + figref.area());

    figref = t;
    System.out.println("Area is " + figref.area());
  }
}
```

```
//Liste 20
class A {
  final void meth() {
    System.out.println("This is a final method.");
  }
}

class B extends A {
  void meth() { // ERROR! Can't override.
    System.out.println("Illegal!");
  }
}
```

```
//Liste 21
final class A {
  // ...
}

// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
  // ...
}
```

Liste 4:
Bu liste Kutu sınıfına renk değerini ekleyen bir alt sınıf yaratıyor.

```
// Kutu'ın alt-sınıfı tanımlanıyor;
   alt-sınıf color adlı bir değişken içeriyor.
class ColorKutu extends Kutu {
  int color;                // Kutu'ın rengi

  ColorKutu(double e, double b, double y, int c) {
    en = e;
    boy = b;
    yukseklik = y;
    color = c;
  }
}
```