Method Overloading

- It is legal for a class to have two or more methods with the same name.
- However, Java has to be able to uniquely associate the invocation of a method with its definition relying on the number and types of arguments.
- Therefore the same-named methods must be distinguished:

  1) by the number of arguments, or

  2) by the types of arguments

- Overloading and inheritance are two ways to implement polymorphism.

Example: Overloading 1

```
class OverloadDemo {
        void test() {
                System.out.println("No parameters");
        }
        void test(int a) {
                System.out.println("a: " + a);
        }
        void test(int a, int b) {
                System.out.println("a and b: " + a + " " + b);
        }
        double test(double a) {
                System.out.println("double a: " + a); return a*a;
        }
}
```

Example: Overloading 2

```
class Overload {
        public static void main(String args[]) {
        OverloadDemo ob = new OverloadDemo();
                double result;
                ob.test();
                ob.test(10);
                ob.test(10, 20);
                result = ob.test(123.2);
                System.out.println("ob.test(123.2): " + result);
        }
}
```

Different Result Types

- Different result types are insufficient.
- The following will not compile:
- double test(double a) {

```
System.out.println("double a: " + a);
return a*a;
}
int test(double a) {
        System.out.println("double a: " + a);
        return (int) a*a;
}
```

Overloading and Conversion 1

- When an overloaded method is called, Java looks for a match between the arguments used to call the method and the method's parameters.
- When no exact match can be found, Java's automatic type conversion can aid overload resolution:
- class OverloadDemo {

```
void test() {
        System.out.println("No parameters");
        }
void test(int a, int b) {
        System.out.println("a and b: " + a + " " + b);
}
```

Overloading and Conversion

- void test(double a) {
  System.out.println("Inside test(double) a: " + a);
  }
  }
  class Overload {
  public static void main(String args[]) {
  OverloadDemo ob = new OverloadDemo();
  int i = 88;
  ob.test();
  ob.test(10, 20);
  ob.test(i);
  ob.test(123.2);
  }
  }

Constructor Overloading

- Why overload constructors? Consider this:
- ```
  class Box {
        double width, height, depth;
        Box(double w, double h, double d) {
              width = w; height = h; depth = d;
        }
        double volume() {
              return width * height * depth;
        }
  }
  ```
- All Box objects can be created in one way: passing all three dimensions.
  ```

Constructor Overloading

- Three constructors: 3-parameter, 1-parameter, parameter-less.

```
class Box {
        double width, height, depth;
        Box(double w, double h, double d) {
                width = w; height = h; depth = d;
                }
        Box() {
                width = -1; height = -1; depth = -1;
                }
        Box(double len) {
                width = height = depth = len;
                }
                double volume() { return width * height * depth; }
        }
```

Example: Overloading 2

```
class OverloadCons {
        public static void main(String args[]) {
        Box mybox1 = new Box(10, 20, 15);
        Box mybox2 = new Box();
        Box mycube = new Box(7);
        double vol;
        vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);
        vol = mybox2.volume();
        System.out.println("Volume of mybox2 is " + vol);
        vol = mycube.volume();
        System.out.println("Volume of mycube is " + vol);
        }
}
```

Using Objects as Parameters

- class Box {

```
        double width;
        double height;
        double depth;
// construct clone of an object
 Box(Box ob) { // pass object to constructor
            width = ob.width;
            height = ob.height;
            depth = ob.depth;
   }
```

Using Objects as Parameters

```
Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }
    double volume() {
        return width * height * depth;
    }
}
```

```
class ObjectCons {
    public static void main(String args[]) {
// create boxes using the various constructors
        double vol;
        Box mybox1 = new Box(10, 20, 15);
    // get volume of first box
    vol = mybox1.volume();
        System.out.println("Volume of mybox1 is " + vol);
        Box myclone = new Box(mybox1);
//get volume of clone
        vol = myclone.volume();
        System.out.println("Volume of clone is " + vol);
         }
}
```

Argument Passing

- There are two ways that a computer language can pass an argument to a subroutine.
- The first way is ***call-by-value***. *This method copies the value of an argument* into the formal parameter of the subroutine.
- Therefore, changes made to the parameter of the subroutine have no effect on the argument.
- The second way an argument can be passed is ***call-by-reference.***
- *In this method, a reference to an argument (not the value of* the argument) is passed to the parameter. Inside the subroutine, this reference is used to access the actual argument specified in the call. This means that changes made to the parameter will affect the argument used to call the subroutine.

call-by-value

```
// Simple Types are passed by value.
class Test {
        void meth(int i, int j) {
                i *= 2;
                j /= 2;
        }
}
```

call-by-value

```
class CallByValue {
  public static void main(String args[]) {
        Test ob = new Test();
          int a = 15, b = 20;
System.out.println("a and b before call: " + a + " " + b);
        ob.meth(a, b);
  System.out.println("a and b after call: " + a + " " + b);
        }
}
```

call-by-reference.

```
class Test {
        int a, b;
        Test (int i, int j) {
                a = i;  b = j;
        }
// pass an object
        void meth(Test o) {
                o.a *=  2;   o.b /= 2;
        }
}
```

call-by-reference.

- class CallByRef {

  public static void main(String args[]) {

  Test ob = new Test(15, 20);

  System.out.println("ob.a and ob.b before call: " + ob.a + " " + ob.b);

  ob.meth(ob);

  System.out.println("ob.a and ob.b after call: " + ob.a + " " + ob.b);

- }
- }