**Overloading methods and constructors**

**Overloading Methods**

In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different.

When this is the case, the methods are said to be *overloaded,* and the process is referred to as

*method overloading.*

Method overloading is one of the ways that Java implements polymorphism.

// Demonstrate method overloading.

class OverloadDemo

{

void test() {

System.out.println("No parameters");

}

// Overload test for one integer parameter.

void test(int a)

{

System.out.println("a: " + a);

}

// Overload test for two integer parameters.

void test(int a, int b) { System.out.println("a and b: " + a + " " + b);

}

// overload test for a double parameter double test(double a) { System.out.println("double a: " + a); return a\*a;

}

}

class Overload {

public static void main(String args[]) { OverloadDemo ob = new OverloadDemo(); double result;

// call all versions of test() ob.test();

ob.test(10);

ob.test(10, 20);

result = ob.test(123.25);

System.out.println("Result of ob.test(123.25): " + result);

}

}

This program generates the following output: No parameters

a: 10

a and b: 10 20

double a: 123.25

Result of ob.test(123.25): 15190.5625

As you can see, **test( )** is overloaded four times.

**Overloading Constructor**

In addition to overloading normal methods, you can also overload constructor methods. In fact, for most real-world classes that you create.

class Box { double width; double height; double depth;

// This is the constructor for Box. Box(double w, double h, double d)

{ width = w; height = h; depth = d;

}

// compute and return volume double volume() {

return width \* height \* depth;

}

}

**Argument/Parameter passing**

In general, 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. As you will see, Java uses both approaches, depending upon what is passed.

For example, consider the following program: // Simple types are passed by value. class Test {

void meth(int i, int j)

{ i \*= 2; j /= 2;

}

}

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);

}

}

The output from this program is shown here: a and b before call: 15 20 a and b after call: 15 20