LAB # 7

INTRODUCING CLASSES

OBJECTIVE

To Study Java class, constructors, method overloading, constructor overloading.

THEORY

The class is at the core of Java. It is the logical construct upon which the entire Java language is built because it defines the shape and nature of an object. As such, the class forms the basis for object-oriented programming in Java. Any concept you wish to implement in a Java program must be encapsulated within a class.

A class is declared by use of the **class** keyword. Notice that the general form of a class does not specify a **main( )** method. Java classes do not need to have a **main( )** method. You only specify one if that class is the starting point for your program.

The general form of a **class** definition is shown here:

```
class classname {
        type instance-variable1;
        type instance-variable2;
        // ...
        type instance-variableN;
        type methodname1(parameter-list) {
              // body of method              }
        type methodname2(parameter-list) {
             // body of method              }
         // ...
                    }
```

Declaring Objects

As just explained, when you create a class, you are creating a new data type. You can use this type to declare objects of that type. However, obtaining objects of a class is a two-step process. First, you must declare a variable of the class type. This variable does not define an object. Instead, it is simply a variable that can *refer* to an object. Second, you must acquire an actual, physical copy of the object and assign it to that variable. You can do this using the **new** operator. The **new** operator dynamically allocates (that is, allocates at run time) memory for an object and returns a reference to it. This reference is, more or less, the address in memory of the object allocated by **new**. This reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated. Let's look at the details of this procedure.

A line similar to the following is used to declare an object of type **Box**:

```
Box mybox = new Box();
```

This statement combines the two steps just described. It can be rewritten like this to show each step more clearly:

```
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```

Constructors

A *constructor* initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the **new** operator completes. Constructors look a little strange because they have no return type, not even **void**.

```
/* Here, Box uses a parameterized constructor to
initialize the dimensions of a box.
*/
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;
                                }
              }

class BoxDemo{
             public static void main(String args[]) {
       // declare, allocate, and initialize Box objects
             Box mybox1 = new Box(10, 20, 15);
             Box mybox2 = new Box(3, 6, 9);
             double vol;
      // get volume of first box
             vol = mybox1.volume();
             System.out.println("Volume is " + vol);
    // get volume of second box
            vol = mybox2.volume();
             System.out.println("Volume is " + vol);
```

```
                                                    }
                    }
```

The output from this program is shown here:

```
Volume is 3000.0
Volume is 162.0
```

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
```

Overloading Constructors

In addition to overloading normal methods, you can also overload constructor methods.

```
/* Here, Box defines three constructors to initialize
the dimensions of a box various ways.
*/
class Box {
         double width;
         double height;
        double depth;
// constructor used when all dimensions specified
          Box(double w, double h, double d) {
                   width = w;
                   height = h;
                   depth = d;
                                                    }
// constructor used when no dimensions specified
          Box() {
                  width = -1; // use -1 to indicate
                   height = -1; // an uninitialized
                   depth = -1; // box
                  }
// constructor used when cube is created
           Box(double len) {
                  width = height = depth = len;
                                }
// compute and return volume
          double volume() {
                   return width * height * depth;
                                }
               }
```

```
class OverloadCons {
            public static void main(String args[]) {
// create boxes using the various constructors
                Box mybox1 = new Box(10, 20, 15);
                Box mybox2 = new Box();
                Box mycube = new Box(7);
               double vol;
// get volume of first box
                vol = mybox1.volume();
                System.out.println("Volume of mybox1 is " + vol);
// get volume of second box
                vol = mybox2.volume();
                System.out.println("Volume of mybox2 is " + vol);
// get volume of cube
                vol = mycube.volume();
                System.out.println("Volume of mycube is " + vol);
                                                    }
                        }
```

The output produced by this program is shown here:

```
Volume of mybox1 is 3000.0
Volume of mybox2 is -1.0
Volume of mycube is 343.0
```

LAB TASK

1. Create a class Point, with two properties x and y. Write all the methods to manipulate the values of Point. Write a method that can check two Objects of Point class for Equality.

1. Create a class with a name Calculate. Add two basic arithmetic functions to it, such as add() and subtract() to perform mathematical calculations. Now overload these methods so that they can take three types of values, an int, a double or a char. Note that if characters are passed to a method, it should return char, if double is sent to a method it should give its answer in double, and so on.