LAB NO: 5 **Date:**

CLASSES AND METHODS

Objectives:

1. To understand the fundamentals of class
2. To know the usage and importance of constructors
3. To be familiar with method overloading and constructor overloading
4. To write simple Java programs to demonstrate the usage of classes, constructors and overloading concepts of Java

5.1 Fundamentals of class

A class defines the structure and behavior (data and code) that will be shared by a set of objects. A class is a template for an object, and an object is an instance of a class. When a class is created it will specify the code and data constituting that class. Collectively, these elements are called members of the class. Specifically, the data defined by the class are referred to as member variables or instance variables. The code that operates on that data is referred to as member methods or just methods. The methods define how the member variables can be used. This means that the behavior and interface of a class are defined by the methods that operate on its instance data.

General form of a class is as shown below:

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

```
/* A simple Java program to illustrate the class concept.
Call this file BoxDemo.java
*/
class Box {
double width;
double height;
```

```
double depth;
}
// This class declares an object of type Box.
class BoxDemo {
public static void main(String args[]) {
        Box mybox = new Box();
        double vol;
        // assign values to mybox's instance variables
        mybox.width = 10;
        mybox.height = 20;
        mybox.depth = 15;
        // compute volume of box
        vol = mybox.width * mybox.height * mybox.depth;

        System.out.println("Volume is " + vol);
}
}
```

```
C://>javac BoxDemo.java
C://>java BoxDemo
Voulme is 3000
```

5.2 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**. This is because the implicit return type of a class constructor is the class type itself. It is the constructor's job to initialize the internal state of an object so that the code creating an instance will have a fully initialized, usable object immediately.

```
/*  Box uses a constructor to initialize the dimensions of a box.*/
class Box {
        double width;
        double height;
        double depth;
        // This is the constructor for Box.
        Box() {
        System.out.println("Constructing Box");
        width = 10;
        height = 10;
        depth = 10;
        }
// compute and return volume
double volume() {
return width * height * depth;
} }
```

```
class BoxDemoConstructor {
        public static void main(String args[]) {
        // declare, allocate, and initialize Box objects
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        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);
        }}
```

Output:
Constructing Box
Constructing Box
Volume is 1000.0
Volume is 1000.0

5.3 Method overloading and Constructor overloading

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.

When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call. Thus, overloaded methods must differ in the type and/or number of their parameters. While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method. When Java encounters a call to an overloaded method, it simply executes the version of the method whose parameters match the arguments used in the call.

```
// 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(123.2);
        System.out.println("Result of ob.test(123.2): " + result);
        }
}
```

This program generates the following output:

```
No parameters
a: 10
a and b: 10 20
double a: 123.2
Result of ob.test(123.2): 15178.24
```

In this example, **test( )** is overloaded four times. The first version takes no parameters, the second takes one integer parameter, the third takes two integer parameters, and the fourth takes one **double** parameter. The fact that the fourth version of **test( )** also returns a value is of no consequence relative to overloading, since return types do not play a role in overload resolution.

```
/* sample program for constructor overloading
 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 exercises

1. Create a class Box that uses a parameterized method to initialize the dimensions of a box. (dimensions are width, height, depth of double type). The class should have a method that can return volume. Obtain an object and print the corresponding volume in main() function.

2. Define a class Employee with data members: employee name, city, basic salary, dearness allowance (DA%) and house rent (HRA%). Define getdata(), calculate(), and display() functions. Calculate method should find the total salary and display method should display it.

   Total = basic+basic*da/100+basic*hra/100;

3. Create a Time class that has separate integer member data for hours, minutes and seconds. One constructor should initialize these data to zero and another should initialize to fixed value. A method should display time in hh:mm:ss format. Finally a method should add 2 objects of time passed as argument.

4. Create a complex class. Use method overloading to find the sum.
   add(integer, complex number)
   add(complex number, complex number)

5. Create class Number with only one private instance variable as a double primitive type. Include the following methods (include respective constructors) isZero( ), isPositive(), isNegative( ), isOdd( ), isEven( ), isPrime(), isArmstrong(). The above methods return boolean primitive type.

6. Write a program to define a class called Book with title,author and edition fields. Define suitable constructors for the Book class. Create a list of 6 Book objects in ascending order. Display only those books' details written by an author taken as an user input.

Additional exercises:

1. The annual examination results of 3 students are tabulated as follows:-

Roll No.	**Subject 1**	**Subject 2**	**Subject 3**

Create a class Result with 2D array and 1D array as its data members. And write methods to perform the following tasks:-
   a. Store marks of 3 subjects obtained by 3 students in 2D array
   b. To store total marks obtained by each student in 1D array.
   c. To find the highest marks in each subject and the roll number of the student who secured it.
   d. To find the student who obtained the highest total marks.

2. Create a class with integer array of size 10 and write methods to perform following:-
   a. Input values into an array
   b. Display the values
   c. Display the largest value
   d. Display the average
   e. Sort the array in ascending order

3. Swap two values using call by value and call by reference.

4. Write a Java program to implement stack class.

5. Write a JAVA program which contains a method square() such that square(3) returns 9, square(0.2) returns 0.04.