NEW EDITION

PROGRAMMING IN JAVA LABORATORY

(SEMESTER-VI of B.Tech)

As per the curricullam and syllabus
of

Bharath Institute of Higher Education & Research

Programming in Java laboratory

Bharath
INSTITUTE OF HIGHER EDUCATION AND RESEARCH
(Declared as Deemed - to - be - University under section 3 of UGC Act 1956)
ACCREDITED WITH 'A' GRADE BY NAAC

PREPARED BY
Mr.A V Allin Geo

Bharath

INSTITUTE OF HIGHER EDUCATION AND RESEARCH

(Declared as Deemed-to-be University under section 3 of UGC Act, 1956)

(Vide Notification No. F.9-5/2000 - U.3, Ministry of Human Resource Development, Govt. of India, dated 4th July 2002)

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

LAB MANUAL

SUBJECT NAME: PROGRAMMING IN JAVA

SUBJECT CODE: BCS6L3

Regualtion 2015

(2015-2016)

BCS6L3	**PROGRAMMING IN JAVA LABORATORY**	**L**	**T**	**P**	**C**
	Total Contact Hours - 30	0	0	3	2
	Prerequisite –Fundamentals of Computing and Programming, Object Oriented Programming Using C++				
	Lab Manual Designed by – Dept. of Computer Science and Engineering.				

OBJECTIVES
The Main objective of this Lab manual is Develop CUI and GUI application using Java Programming.

COURSE OUTCOMES (COs)

CO1	Identify classes, objects, members of a class and the relationships among them for a Specific problem.
CO2	Develop programs using appropriate packages for Inter –thread Communication and Synchronization.
CO3	Develop GUI applications to handle events.
CO4	Develop client server based applications.
CO5	Design, develop, test and debug Java programs using object-oriented principles in Conjunction with development tools including integrated development environments.
CO6	Develop Applets Programs.

MAPPING BETWEEN COURSE OUTCOMES & PROGRAM OUTCOMES
(3/2/1 INDICATES STRENGTH OF CORRELATION) 3- High, 2- Medium, 1-Low

COs	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**
CO1		3	3	3	3				3		3	2		2	
CO2														2	
CO3	2	2	3	3	3			2		2	3			2	
CO4									3			3		2	
CO5		3	3	2										2	
CO6	1	3	2		2			3	2		2	2		2	

Category	Professional Core (PC)
Approval	37th Meeting of Academic Council, May 2015

JAVA PROGRAMS USING FOLLOWING CONCEPTS

1. Classes& Objects.
2. Constructors & Destructors.
3. Methods Overloading.
4. Inheritance.
5. Interface.
6. Multithreading.
7. Package.
8. Creating Java Applets.

PROGRAMMING IN JAVA LABORATORY- BCS6L3

LIST OF EXPERIMENTS

S.NO	NAME OF THE EXPERIMENT
1	Classes & Objects
2	Constructors & Destructors
3	Methods Overloading
4	Inheritance
5	Interface
6	Multithreading
7	Package
8	Creating Java Applets

CONTENT

EX.No.1 CLASSES AND OBJECTS

AIM:

Write a Program in Java to implement the Classes and Objects.

ALGORITHM:

STEP 1:	Start the Program
STEP 2:	Create Class
STEP 3:	Declare the Input and Output Variables
STEP 3:	Create object and access the method
STEP 4:	Implement it with return type and without parameter list
STEP 5:	Implement it with return type and with parameter list
STEP 6:	Implement the constructor by creating classes and objects

SOURCE CODE:

```
class Student{
 int id;
 String name;
}
class TestStudent3{
 public static void main(String args[]){
  //Creating objects
  Student s1=new Student();
  Student s2=new Student();
  //Initializing objects
  s1.id=101;
  s1.name="Sonoo";
  s2.id=102;
  s2.name="Amit";
  //Printing data
  System.out.println(s1.id+" "+s1.name);
  System.out.println(s2.id+" "+s2.name);
 }
}

class Student{
 int rollno;
 String name;
 void insertRecord(int r, String n){
  rollno=r;
  name=n;
 }
 void displayInformation(){System.out.println(rollno+" "+name);}
}
class TestStudent4{
 public static void main(String args[]){
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

#Creating multiple objects by one type only

```
//Java Program to illustrate the use of Rectangle class which
//has length and width data members
class Rectangle{
 int length;
 int width;
 void insert(int l,int w){
  length=l;
  width=w;
 }
 void calculateArea(){System.out.println(length*width);}
}
class TestRectangle2{
 public static void main(String args[]){
```

```
 Rectangle r1=new Rectangle(),r2=new Rectangle();//creating two objects
 r1.insert(11,5);
 r2.insert(3,15);
 r1.calculateArea();
 r2.calculateArea();
}
}
```

OUTPUT:

```
101 Sonoo
102 Amit
111 Karan
222 Aryan
```

```
55
45
```

```
//Java Program to demonstrate the working of a banking-system
//where we deposit and withdraw amount from our account.
//Creating an Account class which has deposit() and withdraw() methods
class Account{
int acc_no;
String name;
float amount;
//Method to initialize object
void insert(int a,String n,float amt){
acc_no=a;
name=n;
amount=amt;
}
//deposit method
void deposit(float amt){
amount=amount+amt;
System.out.println(amt+" deposited");
}
//withdraw method
void withdraw(float amt){
if(amount<amt){
System.out.println("Insufficient Balance");
}else{
amount=amount-amt;
System.out.println(amt+" withdrawn");
}
}
//method to check the balance of the account
void checkBalance(){System.out.println("Balance is: "+amount);}
//method to display the values of an object
void display(){System.out.println(acc_no+" "+name+" "+amount);}
}
//Creating a test class to deposit and withdraw amount
class TestAccount{
public static void main(String[] args){
Account a1=new Account();
a1.insert(832345,"Ankit",1000);
a1.display();
a1.checkBalance();
a1.deposit(40000);
a1.checkBalance();
a1.withdraw(15000);
a1.checkBalance();
}}
```

```
832345 Ankit 1000.0
Balance is: 1000.0
40000.0 deposited
Balance is: 41000.0
15000.0 withdrawn
Balance is: 26000.0
```

RESULT:

Thus the Java program to implement classes and objects was written, executed and the output was verified successfully.

EX.No.2 CONSTRUCTORS & DESTRUCTORS

AIM:

To write a program in java with Constructors and destructors

ALGORITHM:

STEP 1: Start the Program

STEP 2: Declare and Initialize the input variables

STEP 3: Create the Constructors

STEP 4: Create various methods for Subclass

STEP 5: In derived class extend the previous class

STEP 6: In main class specify the values and create the object

STEP 7: Stop

SOURCE CODE:

```
// import java.io.*;


class Geek

{

    int num;

    String name;


    // this would be invoked while an
object

    // of that class is created.

    Geek()

    {

        System.out.println("Constructor
called");
```

```
    }
}
class GFG
{

    public static void main (String[]
args)
    {
        // this would invoke default
constructor.
        Geek geek1 = new Geek();
          // Default constructor provides
the default
        // values to the object like 0, null
        System.out.println(geek1.name);
        System.out.println(geek1.num);
    }
}
```

OUTPUT:

```
Constructor called
null
0
Param
```

RESULT:

Thus the program in java with Constructors and destructors is executed successfully and the output is verified

EX.No.3 METHOD OVERLOADING

AIM:

To write a program in java to implement method overloading

ALGORITHM:

STEP 1: Start the Program

STEP 2: Initialize the File Pointer

STEP 3: Create the class Sum

STEP 4: Overload Sum with two parameters

STEP 5: Create int sum

STEP 6: Create another overloaded sum with two double parameters

STEP 7: In main function create the object and call the methods

STEP 8: Print the data in the file

SOURCE CODE:

```
// Java program to demonstrate working of method
// overloading in Java.

public class Sum {

  // Overloaded sum(). This sum takes two int parameters
  public int sum(int x, int y)
  {
          return (x + y);
  }

  // Overloaded sum(). This sum takes three int parameters
  public int sum(int x, int y, int z)
  {
          return (x + y + z);
  }

  // Overloaded sum(). This sum takes two double parameters
  public double sum(double x, double y)
  {
          return (x + y);
  }
  // Driver code
  public static void main(String args[])
  {
          Sum s = new Sum();
          System.out.println(s.sum(10, 20));
          System.out.println(s.sum(10, 20, 30));
          System.out.println(s.sum(10.5, 20.5));
  }
}
```

OUTPUT:

```
30
60
31.0
```

```
  #Adding Numbers

  static int plusMethodInt(int x, int y) {

    return x + y;

  }

  static double plusMethodDouble(double x, double y) {

    return x + y;

  }

  public static void main(String[] args) {

    int myNum1 = plusMethodInt(8, 5);

    double myNum2 = plusMethodDouble(4.3, 6.26);

    System.out.println("int: " + myNum1);

    System.out.println("double: " + myNum2);

  }
```

OUTPUT:

```
int: 13

double: 10.559999999999999
```

RESULT:

Thus the Java program to implement method overloading and the output was verified successfully.

EX. No.4 INHERITANCE

AIM:

To write a program in Java to implement Inheritance.

ALGORITHM:

```
STEP 1:    Start the Program
STEP 2:    Declare and Initialize the input variables
STEP 3:    Create the class Bicycle
STEP 4:    Create the constructor for Bicycle class.
STEP 5:    Create various methods for Subclass
STEP 6:    In derived class extend the previous class
STEP 7:    In main class specify the values and create the object
STEP 8:    Stop
```

SOURCE CODE:

```
class Bicycle {
    // the Bicycle class has two fields
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }

    // the Bicycle class has three methods
    public void applyBrake(int decrement)
    {
        speed -= decrement;
    }

    public void speedUp(int increment)
    {
        speed += increment;
    }

    // toString() method to print info of Bicycle
    public String toString()
    {

  return ("No of gears are " + gear + "\n"
                + "speed of bicycle is " + speed);
    }
}

// derived class
class MountainBike extends Bicycle {

    // the MountainBike subclass adds one more field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear, int speed,
                        int startHeight)
    {
        // invoking base-class(Bicycle) constructor
        super(gear, speed);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one more method
    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }

    // overriding toString() method
    // of Bicycle to print more info
    @Override public String toString()
    {
        return (super.toString() + "\nseat height is "
                + seatHeight);
    }
}

// driver class
public class Test {
    public static void main(String args[])
    {

        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());
    }
}
```

Output

```
No of gears are 3
Speed of bicycle is 100
Seat height is 25
```

Result:

Thus the program in java to implement Inheritance is executed successfully and the output is verified.

EX.No.5 **INTERFACE**

AIM:

To write a program in Java to implement Interface

ALGORITHM:

```
STEP 1:     Start the Program
STEP 2:     Import the GUI packages
STEP 3:     Create new frame and set sizes
STEP 4:     In showeventdemo add button and listeners
STEP 5:     In buttonclicklistener check whether the button is clicked
STEP 6:     STOP
```

SOURCE CODE

```
import java.io.*;
  // A simple interface
interface In1
{
    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}
  // A class that implements the interface.
class TestClass implements In1
{
    // Implementing the capabilities of
    // interface.
    public void display()
    {
        System.out.println("Geek");
    }

    // Driver Code
    public static void main (String[] args)
    {
        TestClass t = new TestClass();
        t.display();
        System.out.println(a);
    }
}
```

RESULT:

Thus the program in java to implement Interface is executed successfully and the output is verified

EX. No. 6 MULTI THREADING

AIM:

To write a Program in Java to implement Multi Thread.

ALGORITHM:

STEP 1:	Start the Program
STEP 2:	Declare and Initialize the Variables
STEP 3:	Create the class Multi-Threading Demo
STEP 4:	Declare the method run
STEP 5:	Create the class Multithreads
STEP 6:	Specify number of Threads
STEP 7:	In main method create the object and start
STEP 8:	Stop

SOURCE CODE:

```
// Java code for thread creation by extending
// the Thread class
class MultithreadingDemo extends Thread {
        public void run()
        {
                try {
                        // Displaying the thread that is running
                        System.out.println(
                                "Thread " + Thread.currentThread().getId()
                                + " is running");
                }
                catch (Exception e) {
                        // Throwing an exception
                System.out.println("Exception is caught");
            }
        }
}
// Main Class
public class Multithread {
        public static void main(String[] args)
        {
                int n = 8; // Number of threads
                for (int i = 0; i < n; i++) {
                        MultithreadingDemo object
                                = new MultithreadingDemo();
                        object.start();
                }
        }
}
```

OUTPUT:

```
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 12 is running
Thread 11 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
```

```
// Java code for thread creation by implementing
// the Runnable Interface
class MultithreadingDemo implements Runnable {
  public void run()
  {
        try {
              // Displaying the thread that is running
              System.out.println(
                    "Thread " + Thread.currentThread().getId()
                    + " is running");
        }
        catch (Exception e) {
              // Throwing an exception
              System.out.println("Exception is caught");
        }
  }
}

// Main Class
class Multithread {
  public static void main(String[] args)
  {
        int n = 8; // Number of threads
        for (int i = 0; i < n; i++) {
              Thread object
                    = new Thread (new Multithreading Demo());
              object.start ();
        }
  }}
```

OUTPUT:

```
Thread 13 is running
Thread 11 is running
Thread 12 is running
Thread 15 is running
Thread 14 is running
Thread 18 is running
Thread 17 is running
Thread 16 is running
```

RESULT:

Thus the Java program using Thread class and runnable interface for implementing Thread was written, executed and the output was verified successfully.

EX. No. 7 PACKAGES

AIM:

To write a program in Java to implement Packages

ALGORITHM:

```
STEP 1.   START
STEP 2.   Import package
STEP 3.   Create Class A
STEP 4.   Create Class B
STEP 5.   Get output.
```

SOURCE CODE:

```java
package pack;
public class A {
    public void msg() {
        System.out.println("Hello");
    }
}

//save by B.java
package mypack;
class B {
    public static void main(String args[]) {
        pack.A obj = new pack.A();  //using fully qualified name
        obj.msg();
    }
}
```

Output

```
Hello
```

Result:

Thus the program in java to implement Packages is executed successfully and the output is verified

EX. No. 8 CREATING JAVA APPLETS

AIM:

To write a program in java to create Java Applets

ALGORITHM:

STEP 1:	start the program
STEP2:	Import applet
STEP3:	Import graphics
STEP 4:	Add labels
STEP 5:	STOP

SOURCE CODE:

```
import java.applet.Applet;
import java.awt.Graphics;
 /*
<applet code="HelloWorld" width=200 height=60>
</applet>
*/
 // HelloWorld class extends Applet
public class HelloWorld extends Applet
{
   // Overriding paint() method
   @Override
   public void paint(Graphics g)
   {
      g.drawString("Hello World", 20, 20);
   }
     }
```

Output

```
appletviewer
HelloWorld
```

RESULT:

Thus the program in Java to create Java Applets is executed successfully and the output is verified