OBJECT ORIENTED PROGRAMMING THROUGH JAVA

(R24A0583)

LABORATORY MANUAL

B.TECH

(II YEAR-I SEM)

(2025-26)

PREPARED BY
B.Aruna Kumari

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MALLAREDDY COLLEGE OF ENGINEERING &TECHNOLOGY

(Autonomous Institution–UGC, Govt. of India)

Recognizedunder2(f)and12(B) of UGCACT1956
(AffiliatedtoJNTUH,Hyderabad,ApprovedbyAICTE-AccreditedbyNBA&NAAC–'A'Grade-ISO9001:2015Certified)
Maisammaguda,Dhulapally(PostVia.Hakimpet),Secunderabad–500100,TelanganaState,India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision

To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students in to competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

- To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address the challenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

- To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

- To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

- To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science and Information Technology.The graduates will have the following Program Specific Outcomes:

1. Fundamentals and critical knowledge of the Computer System:- Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2. The comprehensive and Applicative knowledge of Software Development: Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneousplatforms individually or working in teams.

3. Applications of Computing Domain & Research: Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates should possess the following:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design / development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis ofthe information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and needfor sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.

12 .Life- long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
    a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
    b. Laboratory Record updated up to the last session experiments and other utensils (ifany) needed in the lab.
    c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during workinghourswill be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

HEAD OF THE DEPARTMENT **PRINCIPAL**

INDEX

S.No	Date	Name of the Activity/Experiment	Grade/ Marks	Faculty Signature

INDEX

PROGRAM-1 WRITE A JAVA PROGRAM TO FIND THE FIBONACCI SERIES USING RECURSIVE AND NON RECURSIVE FUNCTIONS:

```
//Class to write the recursive and non recursive functions.
class fib
{
int a,b,c;
//Non recursive function to find the Fibonacci series.
void nonrecursive(int n)
{ a=0; b=1;
c=a+b;
System.out.print(b);
while(c<=n)
{
System.out.print(c);
a=b;
b=c;
c=a+b;
}
}
// Recursive function to find the Fibonacci series. int recursive(int n)
{
if(n==0) return (0);
if(n==1) return (1);
else
return(recursive(n-1)+recursive(n-2));
}
}
```

```
// Class that calls recursive and non recursive functions
class fib1
{
public static void main(String args[])
{
int n;
// Accepting the value of n at run time.
n=Integer.parseInt(args[0]);
System.out.println("the recursion using non recursive is"); // Creating object for the fib class.
fib f=newfib();
// Calling non recursive function of fib class.
f.nonrecursive(n);
System.out.println("the recursion using recursive is");
for(int i=0;i<=n;i++)
{
// Calling recursive function of fib class. int F1=f.recursive(i); System.out.print(F1);
}}}
```

Three Test Outputs:

OOPS THROUGH JAVA 2024 - 2025

PROGRAM-2 WRITE A JAVA PROGRAM TO MULTIPLY TWO GIVEN MATRICES.

```
// Class to find multiplication of matrices.
class matri
{
public static void main(String args[])
{
// Accept the number of rows and columns at run time. int
m=Integer.parseInt(args[0]);
int n=Integer.parseInt(args[1]);
// Initialize the arrays.
int a[][]=new int[m][n];
int b[][]=new int[m][n];
int c[][]=new int[m][n]; int i=2;
// Loop to accept the values into a matrix.
for(int j=0;j<m;j++)
{for(int k=0;k<n;k++)
{
a[j][k]=Integer.parseInt(args[i]); i++;
}
}
// Loop to accept the values into b matrix. for(int j=0;j<m;j++)
{
for(int k=0;k<n;k++)
{
b[j][k]=Integer.parseInt(args[i]);
i++;
}
}
```

```
// Loop to multiply two matrices .
for(int j=0;j<m;j++)
{
for(int k=0;k<n;k++)
{ c[j][k]=0;
for(int l=0;l<m;l++)
{
c[j][k]=c[j][k]+(a[j][l]*b[l][k]);
}
}
}
// Loop to display the result . for(int j=0;j<m;j++)
{
for(int k=0;k<n;k++)
{
System.out.print(c[j][k]);
}
System.out.println();
}
}
}
```

Three test outputs:

PROGRAM-3 WRITE A JAVA PROGRAM FOR METHOD OVERLOADING AND CONSTRUCTOR OVERLOADING

```
class DisplayOverloading
{
 //adding two integer numbers
 int add(int a, int b)
 {
  int sum = a+b;
  return sum;
 }
 //adding three integer numbers
 int add(int a, int b, int c)
 {
  int sum = a+b+c;
  return sum;
 }
}
class JavaExample
{
 public static void main(String args[])
 {
  DisplayOverloading obj = new DisplayOverloading();
  System.out.println(obj.add(10, 20));
  System.out.println(obj.add(10, 20, 30));
 }
}
```

```
// Constructor Overloading
        public class Student {
        //instance variables of the class
        int id;
        String name;

        Student(){
        System.out.println("this a default constructor");
        }

        Student(int i, String n){
        id = i;
        name = n;
        }

        public static void main(String[] args) {
        //object creation
        Student s = new Student();
        System.out.println("\nDefault Constructor values: \n");
        System.out.println("Student Id : "+s.id + "\nStudent Name : "+s.name);

        System.out.println("\nParameterized Constructor values: \n");
        Student student = new Student(10, "David");
        System.out.println("Student Id : "+student.id + "\nStudent Name : "+student.name);
        }
        }
```

PROGRAM -4 WRITE A PROGRAM TO DEMONSTRATE EXECUTION OF STATIC BLOCKS, STATIC VARIABLES & STATIC METHODS.

```
public class Demo {
  static int x = 10;
  static int y;
  static void func(int z) {
    System.out.println("x = " + x);
    System.out.println("y = " + y);
    System.out.println("z = " + z);
  }
  static {
    System.out.println("Running static initialization block.");
    y = x + 5;
  }
  public static void main(String args[]) {
    func(8);
  }
}
```

Three test outputs:

PROGRAM-5 WRITE A JAVA PROGRAM TO DISPLAY THE EMPLOYEE DETAILS USING SCANNER CLASS

```
importjava.util.*;
class EmployeeDetails
{
public static void main(String args[])
{
System.out.println("entername,id,age,salary");
Scanner sc=newScanner(System.in);
String n=sc.next();
int i=sc.nextInt();
int a=sc.nextInt();
float s=sc.nextFloat();
System.out.println("name is"+n+"idis"+i+"ageis"+a+"salaryis"+s);
}
}
```

Three test Outputs:

PROGRAM-6 WRITE A PROGRAM FOR SORTING A GIVEN LIST OF NAMES IN ASCENDING ORDER

```
// Java Program to Sort Names in an Alphabetical Order
import java.io.*;
import java.util.*;
 class GFG {
   public static void main(String[] args)
   {
      // storing input in variable
      int n = 4;
      // create string array called names
      String names[]
         = { "Rahul", "Ajay", "Gourav", "Riya" };
      // inbuilt sort function
      Arrays.sort(names);
      // print output array
      System.out.println(
         "The names in alphabetical order are: ");
      for (int i = 0; i < n; i++) {
         System.out.println(names[i]);
      }
   }
}
```

Three Test Outputs:

PROGRAM-7 WRITE A PROGRAM TO IMPLEMENT SINGLE AND MULTI-LEVEL INHERITANCE

```
// Single Inheritance
class Animal
 {
 void sleep(){System.out.println("sleeping.  ");}
 }
 class Dog extends Animal
 {
 void bark(){System.out.println("barking.  ");}
 }
 class TestInheritance
 {
 public static void main(String args[])
 {
 Dog d=new Dog();
 d.bark();
 d.sleep();
 }
 }
```

```
// Multi Level Inheritance
class Animal
 {
 void eat(){System.out.println("eating.  ");}
 }
 class Dog extends Animal{
 void sleep(){System.out.println("sleeping.  ");}
 }
 class BabyDog extends Dog
```

```
void weep(){System.out.println("weeping.  ");}
}
class TestInheritance2
{
public static void main(String args[])
{
BabyDog d=new BabyDog();
d.weep();
d.sleep();
d.eat();
}
}
```

Three Test Outputs:

PROGRAM-8 WRITE A PROGRAM TO IMPLEMENT HIERARCHICAL INHERITANCE.

```
class Animal
 {
 void eat(){System.out.println("eating...");}
 }
 class Dog extends Animal
 {
 void sleep(){System.out.println("sleeping...");}
 }
 class Cat extends Animal
 {
 void meow(){System.out.println("meowing...");}
 }
 class TestInheritance3
 {
 public static void main(String args[])
 {
 Cat c=new Cat(); c.meow();
 c.eat();
 //c.bark();//C.T.Error
 }
 }
```

Three Test Outputs

PROGRAM-9 WRITE A PROGRAM TO IMPLEMENT METHOD OVERRIDING.

```
//Creating a parent class. class
Vehicle{
  //defining a method
  void run(){System.out.println("Vehicle is running");}
}
//Creating a child  class class
Bike2 extends Vehicle{
  //defining the same method as in the parent class
  void run(){System.out.println("Bike is running safely");}

  public static void main(String args[]){ Bike2
  obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Three Test Outputs:

PROGRAM-10 WRITE A PROGRAM TO CREATE AN ABSTRACT CLASS NAMED SHAPE THAT CONTAINS TWO INTEGERS AND AN EMPTY METHOD NAMED PRINTAREA (). PROVIDE THREE CLASSES NAMED RECTANGLE, TRIANGLE AND CIRCLE SUCH THAT EACH ONE OF THE CLASSES EXTENDS THE CLASS SHAPE. EACH ONE OF THE CLASSES CONTAINS ONLY THE METHOD PRINTAREA () THAT PRINTS THE AREA OF THE GIVEN SHAPE.

```
import java.util.*; abstract
class shape
{
int x,y;
abstract void area(double x,double y);
}
class Rectangle extends shape
{
void area(double x,double y)
{
System.out.println("Area of rectangle is :"+(x*y));
}
}
class Circle extends shape
{
void area(double x,double y)
{
System.out.println("Area of circle is :"+(3.14*x*x));
}
```

```
class Triangle extends shape
{
void area(double x,double y)
{
System.out.println("Area of triangle is :"+(0.5*x*y));
}
}
public class AbstactDDemo
{
public static void main(String[] args)
{
Rectangle r=new Rectangle();
r.area(2,5);
Circle c=new Circle();
c.area(5,5);
Triangle t=new Triangle();
t.area(2,5);
}
}
```

Three Test Outputs:

PROGRAM-11 WRITE A PROGRAM TO IMPLEMENT INTERFACE

```
interface printable{
void print();
}
class A6 implements printable{
public void print(){System.out.println("Hello");}

public static void main(String args[]){
A6 obj = new A6();
obj.print();
 }
}
```

Three Test Outputs:

PROGRAM-12 WRITE A PROGRAM TO IMPLEMENT MULTIPLE AND HYBRID INHERITANCE

```
// concept of Multiple inheritance
import java.io.*;
import java.lang.*;
import java.util.*;
interface One {
    public void print_geek();
}
 interface Two {
    public void print_for();
}
 interface Three extends One, Two {
    public void print_geek();
}
class Child implements Three {
    @Override public void print_geek()
    {
        System.out.println("Geeks");
    }
     public void print_for() { System.out.println("for"); }
}
 // Drived class
public class Main {
    public static void main(String[] args)
    {
        Child c = new Child();
        c.print_geek();
        c.print_for();
        c.print_geek();
    }
}
```

```
// Hybrid Inheritance
class HumanBody
{
public void displayHuman()
{
System.out.println("Method defined inside HumanBody class");
}
}
interface Male
{
public void show();
}
interface Female
{
public void show();
}
public class Child extends HumanBody implements Male, Female
{
public void show()
{
System.out.println("Implementation of show() method defined in interfaces Male and Female");
}
public void displayChild()
{
System.out.println("Method defined inside Child class");
```

```
    public static void main(String args[])
{
Child obj = new Child();
System.out.println("Implementation of Hybrid Inheritance in Java");
obj.show();
obj.displayChild();
}
}
```

Three Test Outputs:

PROGRAM-14 WRITE A PROGRAM TO CREATE USER DEFINED PACKAGE AND DEMONSTRATE VARIOUS ACCESS MODIFIERS.

com/example/Calculator.java

```
package  com.example;
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
    public int subtract(int a, int b) {
        return a - b;
    }
    public int multiply(int a, int b) {
        return a * b;
    }
    public int divide(int a, int b) {
        if (b != 0) {
            return a / b;
        } else {
            throw new ArithmeticException("Cannot divide by zero!");
        }
    }
}
```

Now, create another file outside the "com" folder to access the Calculator class from the user-defined package.

PackageExample.java

```
import com.example.Calculator;
public class PackageExample {
```

```
 public static void main(String[] args) {
        Calculator calculator = new Calculator();
        int result = calculator.add(5, 3);
        System.out.println("Addition: " + result);
        result = calculator.subtract(5, 3);
        System.out.println("Subtraction: " + result);
        result = calculator.multiply(5, 3);
        System.out.println("Multiplication: " + result);
        result = calculator.divide(10, 2);
        System.out.println("Division: " + result);
    }
}
```

Three Test Outputs:

PROGRAM-15 WRITE A PROGRAM TO DEMONSTRATE THE USE OF SUPER AND FINAL KEYWORDS.

```
// Super Keyword
class employee {
int wt = 8;
}
class clerk extends employee {
 int wt = 10;  //work time
 void display() {
  System.out.println(super.wt); //will print work time of clerk
 }
 public static void main(String args[]) {
  clerk c = new clerk();
  c.display();
 }
}
```

```
// Final Keyword
class stud {
 final void show() {
  System.out.println("Class - stud : method defined");
 }
}
class books extends stud {
 void show() {
  System.out.println("Class - books : method defined");
 }
 public static void main(String args[]) {
  books B2 = new books();
  B2.show();
  } }
```

PROGRAM-16 WRITE A PROGRAM IF NUMBER IS LESS THAN 10 AND GREATER THAN 50 IT GENERATE THE EXCEPTION OUT OF RANGE. ELSE IT DISPLAYS THE SQUARE OF NUMBER.

```
classCustomTest {
 public static void main(String arr[]) {
 try {
  int a=Integer.parseInt(arr[0]);
 if(a<0||a>50)
 throw(new outofRangeException("valid range is 10 to 50"));
 {
 int s=a*a;
 System.out.println("Square is:"+s);
  }
 }catch(Exception ex)
 {
 System.out.println(ex);
 }
  } }
```

Three test outputs

PROGRAM-17 WRITE A PROGRAM WITH MULTIPLE CATCH STATEMENTS.

```
public class MultipleCatchBlock1 {

  public static void main(String[] args) {

      try{
          int a[]=new int[5];
          a[5]=30/0;
         }
         catch(ArithmeticException e)
           {
            System.out.println("Arithmetic Exception occurs");
           }
         catch(ArrayIndexOutOfBoundsException e)
           {
            System.out.println("ArrayIndexOutOfBounds Exception occurs");
           }
         catch(Exception e)
           {
            System.out.println("Parent Exception occurs");
           }
         System.out.println("rest of the code");
   }
}
```

Three test outputs:

PROGRAM-18 WRITE A PROGRAM TO IMPLEMENT NESTED TRY

```
public class NestedTryBlock{
 public static void main(String args[]){
 //outer try block
  try{
  //inner try block 1
    try{
     System.out.println("going to divide by 0");
     int b =39/0;
    }
    //catch block of inner try block 1
    catch(ArithmeticException e)
    {
      System.out.println(e);
    }

     //inner try block 2
    try{
    int a[]=new int[5];

    //assigning the value out of array bounds
     a[5]=4;
     }
      //catch block of inner try block 2
    catch(ArrayIndexOutOfBoundsException e)
    {
       System.out.println(e);
                                                                    }
```

```
  System.out.println("other statement");
 }
 //catch block of outer try block
 catch(Exception e)
 {
  System.out.println("handled the exception (outer catch)");
 }

 System.out.println("normal flow..");
}
}
```

Three test outputs:

PROGRAM -19 WRITE A PROGRAM TO IMPLEMENT SIMPLE THREAD BY EXTENDING THREAD CLASS AND IMPLEMENTING RUNNABLE INTERFACE.

```
// Java program to illustrate defining Thread
// by implements Runnable interface
class Geeks {
   public static void m1()
   {
      System.out.println("Hello Visitors");
   } }
  // Here we can extends any other class
class Test extends Geeks implements Runnable {
   public void run()
   {
      System.out.println("Run method executed by child Thread");
   }
   public static void main(String[] args)
   {
      Test t = new Test();
      t.m1();
      Thread t1 = new Thread(t);
      t1.start();
      System.out.println("Main method executed by main thread");
   } }
```

Three test outputs:

PROGRAM-20 WRITE A PROGRAM THAT IMPLEMENTS A MULTI-THREAD APPLICATION THAT HAS THREE THREADS

```
import java.util.*;
// class for Even Number
class EvenNum implements Runnable {
   public int a;
   public EvenNum(int a) {
      this.a = a;
   }
   public void run() {
      System.out.println("The Thread "+ a +" is EVEN and Square of " + a + " is : " + a * a);
   }
} // class for Odd Number
class OddNum implements Runnable {
   public int a;
   public OddNum(int a)  {
      this.a = a;
   }
   public void run()  {
      System.out.println("The Thread "+ a +" is ODD and Cube of " + a + " is: " + a * a * a);
   }
       }
```

```
// class to generate random number
class RandomNumGenerator extends Thread {
   public void run() {
      int n = 0;
      Random rand = new Random();
      try {
         for (int i = 0; i < 10; i++) {
            n = rand.nextInt(20);
            System.out.println("Generated Number is " + n);
            // check if random number is even or odd
            if (n % 2 == 0) {
               Thread thread1 = new Thread(new EvenNum(n));
               thread1.start();
            }
            else {
               Thread thread2 = new Thread(new OddNum(n));
               thread2.start();
            }
      // thread wait for 1 second
      Thread.sleep(1000);
      System.out.println("-----------------------------------------");
         }
      }
      catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }
                                                                    }
```

```
    // Driver class
public class MultiThreadRandOddEven {
    public static void main(String[] args) {
        RandomNumGenerator rand_num = new RandomNumGenerator();
        rand_num.start();
}
}
```

Three test outputs:

PROGRAM 21. WRITE A PROGRAM TO SET AND PRINT THREAD PRIORITIES

// Set Prioroties

```
public class A implements Runnable
{
public void run()
{
  System.out.println(Thread.currentThread()); // This method is static.
}
public static void main(String[] args)
{
 A a = new A();
 Thread t = new Thread(a, "NewThread");
 t.setPriority(2); // Setting the priority of thread.

 System.out.println("Priority of Thread: " +t.getPriority());
 System.out.println("Name of Thread: " +t.getName());
 t.start();
  }
}
```

Example 2:

```
public class A implements Runnable
{
public void run()
{
  System.out.println(Thread.currentThread()); // This method is static.
}
public static void main(String[] args)
{
A a = new A();
```

```
Thread t1 = new Thread(a, "First Thread");
Thread t2 = new Thread(a, "Second Thread");
Thread t3 = new Thread(a, "Third Thread");

t1.setPriority(4); // Setting the priority of first thread.
t2.setPriority(2); // Setting the priority of second thread.
t3.setPriority(8); // Setting the priority of third thread.

t1.start();
t2.start();
t3.start();
 }
}
```

Three test outputs

PROGRAM-22 WRITE A PROGRAM TO IMPLEMENT FOLLOWING COLLECTIONS

A)ARRAY LIST B) VECTOR C)HASH TABLE D)STACK

```
// Array List
import java.util.*;
class TestJavaCollection1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();//Creating arraylist
list.add("Ravi");//Adding object in arraylist
list.add("Vijay");
list.add("Ravi");
list.add("Ajay");
//Traversing list through Iterator
Iterator itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

// Vector

```
import java.util.*;
public class TestJavaCollection3{
public static void main(String args[]){
Vector<String> v=new Vector<String>();
v.add("Ayush");
v.add("Amit");
v.add("Ashish");
v.add("Garima");
Iterator<String> itr=v.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

// stack

```
import java.util.*;
public class TestJavaCollection4{
public static void main(String args[]){
Stack<String> stack = new Stack<String>();
stack.push("Ayush"); stack.push("Garvit");
stack.push("Amit");
stack.push("Ashish");
stack.push("Garima");
stack.pop();
Iterator<String> itr=stack.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
} } }
```

// Hash Table

```
import java.util.*;
public class TestJavaCollection7{
public static void main(String args[]){
//Creating HashSet and adding elements
HashSet<String> set=new HashSet<String>();
set.add("Ravi");
set.add("Vijay");
set.add("Ravi");
set.add("Ajay");
//Traversing elements
Iterator<String> itr=set.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Three test outputs:

PROGRAM-23 WRITE A PROGRAM TO DEMONSTRATE LAMBDA EXPRESSIONS.

```
// A Java program to demonstrate simple lambda expressions
import java.util.ArrayList;
class Test {
    public static void main(String args[])
    {
        // Creating an ArrayList with elements {1, 2, 3, 4}
        ArrayList<Integer> arrL = new ArrayList<Integer>();
        arrL.add(1);
        arrL.add(2);
        arrL.add(3);
        arrL.add(4);
         // Using lambda expression to print all elements of arrL
        arrL.forEach(n -> System.out.println(n));

        // Using lambda expression to print even elements
        // of arrL
        arrL.forEach(n -> {
            if (n % 2 == 0)
                System.out.println(n);
        });
    }
}
```

Three test outputs:

PROGRAM-24 WRITE A PROGRAM FOR PRODUCER AND CONSUMER PROBLEM USING THREADS

```
// Java program to implement solution of producer
//  consumer problem.
 import java.util.LinkedList;
 public class Threadexample {
   public static void main(String[] args)
      throws InterruptedException
   {
      // Object of a class that has both produce()
      // and consume() methods
      final PC pc = new PC();

      // Create producer thread
      Thread t1 = new Thread(new Runnable() {
         @Override
         public void run()
         {
            try {
               pc.produce();
            }
            catch (InterruptedException e) {
               e.printStackTrace();
            }
          }
});
```

```
    // Create consumer thread
      Thread t2 = new Thread(new Runnable() {
          @Override
          public void run()
          {
              try {
                  pc.consume();
              }
              catch (InterruptedException e) {
                  e.printStackTrace();
              }
          }
      });

      // Start both threads
      t1.start();
      t2.start();

      // t1 finishes before t2
      t1.join();
      t2.join();
  }

  // This class has a list, producer (adds items to list and consumer (removes items).
  public static class PC {

      // Create a list shared by producer and consumer Size of list is 2.
      LinkedList<Integer> list = new LinkedList<>();
    int capacity = 2;
```

```
    // Function called by producer thread
    public void produce() throws InterruptedException
    {
        int value = 0;
        while (true) {
            synchronized (this)
            {
                // producer thread waits while list
                // is full
                while (list.size() == capacity)
                    wait();

                System.out.println("Producer produced-" + value);

                // to insert the jobs in the list
                list.add(value++);

                // notifies the consumer thread that
                // now it can start consuming
                notify();

                // makes the working of program easier
                // to understand
                Thread.sleep(1000);
            }
        }
}
```

```
        // Function called by consumer thread
        public void consume() throws InterruptedException
        {
            while (true) {
                synchronized (this)
                {
                    // consumer thread waits while list
                    // is empty
                    while (list.size() == 0)
                        wait();

                    // to retrieve the first job in the list
                    int val = list.removeFirst();

                    System.out.println("Consumer consumed-" + val);

                    // Wake up producer thread
                    notify();

                    // and sleep
                    Thread.sleep(1000);
                }
            }
        }
    }
}
```

PROGRAM-25 WRITE A PROGRAM TO LIST ALL THE FILES IN A DIRECTORY INCLUDING THE FILES PRESENT IN ALL ITS SUBDIRECTORIES.

```
// Java program to print all files in a folder(and sub-folders)

import java.io.File;

public class GFG {
   static void RecursivePrint(File[] arr, int index, int level)
   {
      // terminate condition
      if (index == arr.length)
         return;

         // tabs for internal levels
        for (int i = 0; i < level; i++)
            System.out.print("\t");

      // for files
      if (arr[index].isFile())
         System.out.println(arr[index].getName());

      // for sub-directories
      else if (arr[index].isDirectory()) {
         System.out.println("[" + arr[index].getName()
                        + "]");

         // recursion for sub-directories
         RecursivePrint(arr[index].listFiles(), 0,
                     level + 1);
                                                                    }
```

```
 // recursion for main directory
      RecursivePrint(arr, ++index, level);
   }

   // Driver Method
   public static void main(String[] args)
   {
      // Provide full path for directory(change accordingly)
      String maindirpath  = "C:\\Users\\Gaurav Miglani\\Desktop\\Test";

      // File object
      File maindir = new File(maindirpath);

      if (maindir.exists() && maindir.isDirectory()) {
            // array for files and sub-directories of directory pointed by maindir
         File arr[] = maindir.listFiles();
          System.out.println( "**********************************************");
         System.out.println( "Files from main directory : " + maindir);
         System.out.println("**********************************************");
          // Calling recursive method
         RecursivePrint(arr, 0, 0);
      }
   }
}
```

Three test outputs:

PROGRAM-26 WRITE A PROGRAM TO READ THE CONTENT OF A FILE LINE BY LINE

```
import java.io.FileReader;
import java.io.BufferedReader;
class Main {
  public static void main(String[] args) {

    // Creates an array of character
    char[] array = new char[100];
    try {

   // Creates a FileReader
      FileReader file = new FileReader("input.txt");
      // Creates a BufferedReader
      BufferedReader input = new BufferedReader(file);
      // Reads characters
      input.read(array);
      System.out.println("Data in the file: ");
      System.out.println(array);

    // Closes the reader
      input.close();
    }
    catch(Exception e) {
      e.getStackTrace();
    }
  }
                                                                    }
```

PROGRAM-27 WRITE A PROGRAM THAT CONNECTS TO A DATABASE USING JDBC DISPLAY ALL RECORDS IN A TABLE.

```
import java.sql.*;
public class jdbcResultSet {
  public static void main(String[] args) {
    try {
      Class.forName("org.apache.derby.jdbc.ClientDriver");
    } catch(ClassNotFoundException e) {
      System.out.println("Class not found "+ e);
    }
    try {
      Connection con = DriverManager.getConnection(
        "jdbc:derby://localhost:1527/testDb","username", "password");

      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT * FROM employee");
      System.out.println("id  name    job");

      while (rs.next()) {
        int id = rs.getInt("id");
        String name = rs.getString("name");
        String job = rs.getString("job");
        System.out.println(id+"  "+name+"    "+job);
      }
    } catch(SQLException e) {
      System.out.println("SQL exception occured" + e);
    }
  }
}
```

PROGRAM-28 WRITE A PROGRAM TO CONNECT TO A DATABASE USING JDBC AND INSERT VALUES INTO IT.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JDBCExample {
  static final String DB_URL = "jdbc:mysql://localhost/TUTORIALSPOINT";
  static final String USER = "guest";
  static final String PASS = "guest123";
  public static void main(String[] args) {
    // Open a connection
    try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
      Statement stmt = conn.createStatement();
    ) {
      // Execute a query
      System.out.println("Inserting records into the table...");
      String sql = "INSERT INTO Registration VALUES (100, 'Zara', 'Ali', 18)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration VALUES (101, 'Mahnaz', 'Fatma', 25)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration VALUES (102, 'Zaid', 'Khan', 30)";
      stmt.executeUpdate(sql);
      sql = "INSERT INTO Registration VALUES(103, 'Sumit', 'Mittal', 28)";
      stmt.executeUpdate(sql);
      System.out.println("Inserted records into the table...");
    } catch (SQLException e) {
      e.printStackTrace();
    }
  }
}
```

PROGRAM-29 WRITE A JAVA PROGRAM TO CONNECT TO A DATABASE USING JDBC AND DELETE VALUES FROM IT

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBCExample {
  static final String DB_URL = "jdbc:mysql://localhost/TUTORIALSPOINT";
  static final String USER = "guest";
  static final String PASS = "guest123";
  static final String QUERY = "SELECT id, first, last, age FROM Registration";

  public static void main(String[] args) {
    // Open a connection
    try(Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
      Statement stmt = conn.createStatement();
    ) {
      String sql = "DELETE FROM Registration " +
        "WHERE id = 101";
      stmt.executeUpdate(sql);
      ResultSet rs = stmt.executeQuery(QUERY);
      while(rs.next()){
        //Display values
        System.out.print("ID: " + rs.getInt("id"));
        System.out.print(", Age: " + rs.getInt("age"));
        System.out.print(", First: " + rs.getString("first"));
        System.out.println(", Last: " + rs.getString("last"));
}
```

```
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Three test outputs:

PROGRAM 30 WRITE A PROGRAM THAT WORKS AS A SIMPLE CALCULATOR. USE A GRID LAYOUT TO ARRANGE BUTTONS FOR DIGITS AND FOR THE + - * % OPERATIONS. ADD A TEXT FIELD TO DISPLAY THE RESULT.

```
/* Program to create a Simple Calculator */ import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
/*
<applet code="MyCalculator" width=300 height=300>
</applet>
*/
public class MyCalculator extends Applet implements ActionListener {
   int num1,num2,result;
   TextField T1;
   Button NumButtons[]=new Button[10];
   Button Add,Sub,Mul,Div,clear,EQ;
   char Operation;
   Panel nPanel,CPanel,SPanel;
   public void init() {
      nPanel=new Panel();
      T1=new TextField(30);
      nPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
      nPanel.add(T1);
          CPanel=new Panel();
      CPanel.setBackground(Color.white);
      CPanel.setLayout(new GridLayout(5,5,3,3));
      for(int i=0;i<10;i++) {
            NumButtons[i]=new Button(""+i);
}
```

```
Add=new Button("+");
Sub=new Button("-");
Mul=new Button("*");
Div=new Button("/");
clear=new Button("clear");
EQ=new Button("=");
T1.addActionListener(this);
for(int i=0;i<10;i++) {
   CPanel.add(NumButtons[i]);
}
CPanel.add(Add);
CPanel.add(Sub);
CPanel.add(Mul);
CPanel.add(Div);
CPanel.add(EQ);
SPanel=new Panel();
SPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
SPanel.setBackground(Color.yellow);
SPanel.add(clear);
for(int i=0;i<10;i++) {
   NumButtons[i].addActionListener(this);
}
 Add.addActionListener(this);
Sub.addActionListener(this);
Mul.addActionListener(this);
Div.addActionListener(this);
 clear.addActionListener(this);
 EQ.addActionListener(this);
this.setLayout(new BorderLayout());
```

```
  add(nPanel,BorderLayout.NORTH);
   add(CPanel,BorderLayout.CENTER);
   add(SPanel,BorderLayout.SOUTH);
}
public void actionPerformed(ActionEvent ae) {
   String str=ae.getActionCommand ();
   char ch=str.charAt(0);
   if(Character.isDigit(ch))
      T1.setText(T1.getText()+str);
   else
      if(str.equals("+")){
         num1=Integer.parseInt (T1.getText());
         Operation='+';
         T1.setText ("");
      }
      if(str.equals("-")){
         num1=Integer.parseInt(T1.getText());
         Operation='-';
         T1.setText("");
      }
      if(str.equals("*")){
         num1=Integer.parseInt(T1.getText());
         Operation='*';
         T1.setText("");
      }
      if(str.equals("/")){
         num1=Integer.parseInt(T1.getText());
         Operation='/';
         T1.setText("");
      }
```

```
    if(str.equals("%")){
       num1=Integer.parseInt(T1.getText());
       Operation='%';
       T1.setText("");
    }
    if(str.equals("=")) {
       num2=Integer.parseInt(T1.getText());
       switch(Operation)
       {
          case '+':result=num1+num2;
                break;
          case '-':result=num1-num2;
                break;
          case '*':result=num1*num2;
                break;
          case '/':try {
                   result=num1/num2;
                }
                catch(ArithmeticException e) {
                   result=num2;
                   JOptionPane.showMessageDialog(this,"Divided by zero");
                }
                break;
       }
       T1.setText(""+result);
    }
    if(str.equals("clear")) {
       T1.setText("");
    }
}}
```

OUTPUT: