**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

**DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH**

**Department of Computer Science & Engineering**

**Subject Name: JAVA PROGRAMMING**

**Semester: 3**

**Subject Code: CSE201**

**Academic year:2024-25**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **1.** | Demonstration of installation steps of Java,Introduction to Object Oriented Concepts, comparison of Java with other object-oriented programming languages. Introduction to JDK, JRE, JVM, Javadoc, command line argument. Introduction to Eclipse or NetBeans IDE,or BlueJ and Console Programming.  **PROGRAM CODE:**     |  | | --- | | 1.JDK (Java Development Kit) is a Kit that provides the environment to develop and execute(run) the Java program. JDK is a kit(or package) that includes two things  • Development Tools(to provide an environment to develop your java programs)  • JRE (to execute your java program).  2. JRE (Java Runtime Environment) is an installation package that provides an environment to only run(not develop) the java program(or application)onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.  3. JVM (Java Virtual Machine) is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or  JDK goes into JVM and JVM is responsible for executing the java program  line by line, hence it is also known as an interpreter.  JVM(Java Virtual Machine) acts as a run-time engine to run Java applications. JVM is the one that actually calls the main method present in a java code. JVM is a part of JRE(Java Runtime Environment).  Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustment. This is all possible because of JVM.  When we compile a *.java* file, *.class* files(contains byte-code) with the same class names present in *.java* file are generated by the Java compiler.  **JavaDoc :** JavaDoc tool is a document generator tool in Java programming language for generating standard documentation in HTML format. It generates API documentation. It parses the declarations ad documentation in a set of source file describing classes, methods, constructors, and fields.  Before using JavaDoc tool, you must include JavaDoc comments /\*\*………………..\*/ providing information about classes, methods, and constructors, etc. For creating a good and understandable document API for any java file you must write better comments for every class, method, constructor.  **Command Line Argument :** Java command-line argument is an argument  i.e. passed at the time of running the Java program. In the command line,  the arguments passed from the console can be received in the java program  and they can be used as input. The users can pass the arguments during the  execution bypassing the command-line arguments inside the main() method. | |
| **2.** | Imagine you are developing a simple banking application where you need to display the current balance of a user account. For simplicity, let's say the current balance is $20. Write a java program to store this balance in a variable and then display it to the user.  **PROGRAM CODE:**  import java.util.Scanner;  class practical2  {  public static void main(String args[])  {  System.out.print("enter bank balance: ");    Scanner s = new Scanner(System.in);  int n = s.nextInt();  System.out.println("balance:"+n);  System.out.println("Ravi Baraiya 23DCS007);  }}    **OUTPU**T**:**    **CONCLUSION:**  This program demonstrates the basic concept of storing and displaying data in Java, showcasing the use of variables and output statements. |
| **3.** | **Write a program to take the user for a distance (in meters) and the time taken (as three numbers: hours, minutes, seconds), and display the speed, in meters per second, kilometers per hour and miles per hour (hint:1 mile = 1609 meters).**  **PROGRAM CODE:**  import java.util.\*;  public class prectical3 {      public static void main(String[] args) {          float time[] = new float[3];          Scanner s = new Scanner(System.in);          int choice;          float dis, tmin, tsec, thour;          float calculate;          System.out.print("Enter Distance : ");          dis = s.nextFloat();          System.out.print("Enter Time In Hours : ");          time[0] = s.nextFloat();          System.out.print("Enter Time In Min : ");          time[1] = s.nextFloat();          System.out.print("Enter Time In Sec : ");          time[2] = s.nextFloat();          tmin = (time[0] \* 60);          tsec = ((tmin + time[1]) \* 60) + time[2];          thour = time[0] + (time[1] / 60) + (time[2] / 3600);          System.out.println("Choose Unit To Display Velocity");          System.out.println("1.m/s");          System.out.println("2.km/h");          System.out.println("3.miles/h");          choice = s.nextInt();          switch (choice) {              case 1:                  calculate = dis / tsec;                  System.out.println("Velocity = " + calculate + " m/s");                  break;              case 2:                  dis = dis / 1000;                  calculate = dis / thour;                  System.out.println("Velocity = " + calculate + " km/h");                  break;              case 3:                  dis = dis / 1609;                  calculate = dis / thour;                  System.out.println("Velocity = " + calculate + " miles/h");                  break;          }      }  }  **OUTPUT:**      **CONCLUSION:**  **This program demonstrates the ability to take user input, perform calculations, and display results in different units, showcasing fundamental programming concepts and unit conversions.** |
| **4.** | Imagine you are developing a budget tracking application. You need to calculate the total expenses for the month. Users will input their daily expenses, and the program should compute the sum of these expenses. Write a Java program to calculate the sum of elements in an array representing daily  expenses.  **PROGRAM CODE:**  import java.util.Scanner;  public class practical4  {  public static void main(String[] args)  {  System.out.println("enter daily expenses:");  Scanner s = new Scanner(System.in);  int a=5;  float[] b = new float[a];  float sum = 0;  int i;  for(i=0;i<a;i++)  {  b[i] = s.nextFloat();  sum+=b[i];  }  System.out.println(sum);  }}  **OUTPUT:**    **CONCLUSION:**  This program demonstrates the use of arrays and loops to collect and calculate the sum of daily expenses, providing a simple yet effective way to track monthly expenditures. |
| **5.** | An electric appliance shop assigns code 1 to motor,2 to fan,3 to tube and 4 for wires. All other items have code 5 or more. While selling the goods, a sales tax of 8% to motor,12% to fan,5% to tube light,7.5% to wires and 3% for all other items is charged. A list containing the product code and price in two different arrays. Write a java program using switch statement to prepare the bill.  **PROGRAM CODE:**  import java.util.\*;  class practical5  {  public static void main(String args[])  {  System.out.println("we have the list of product is ");  System.out.println("1.motor 2.fan 3.tube 4.wires 5.all other items.");  System.out.println("enter number which costumer want to buy");  Scanner s = new Scanner(System.in);  int a = s.nextInt();  float taxprice;  float[] b = new float[5];  float[] c = new float[5];  System.out.print("price of product: ");  float p = s.nextFloat();  switch(a)  {  case 1:  System.out.println(" Bill");  System.out.println("code of motor is : 1");  taxprice = p+=p\*0.08f;  System.out.print("price of motor including tax is : "+ taxprice);  break;  case 2:  System.out.println(" Bill");  System.out.println("code of fan is : 2");  taxprice = p+=p\*0.12f;  System.out.print("price of fan including tax is : "+ taxprice);  break;  case 3:  System.out.println(" Bill");  System.out.println("code of tube is : 3");  taxprice = p+=p\*0.05f;  System.out.print("price of tube including tax is : "+ taxprice);  break;  case 4:  System.out.println(" Bill");  System.out.println("code of wires is : 4");  taxprice = p+=p\*0.075f;  System.out.print("price of wires including tax is : "+ taxprice);  break;  case 5:  System.out.println(" Bill");  System.out.println("code of other items is : 5");  taxprice = p+=p\*0.03f;  System.out.print("price of other items including tax is : "+ taxprice);  break;  default:  System.out.println("entered number is wrong");  break;  }  }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates the use of a switch statement to apply different tax rates based on product codes, calculating the total bill by iterating through the arrays of codes and prices. It showcases a practical application of conditional statements in Java programming. |
| **6.** | Create a Java program that prompts the user to enter the number of days (n) for which they want to generate their exercise routine. The program should then calculate and display the first n terms of the Fibonacci series, representing the exercise duration for each day.  **PROGRAM CODE:**  import java.util.Scanner;  public class practical6  {  public static void main(String[] args)  {  System.out.print("enter number of days that you want to exercise: ");  Scanner s = new Scanner(System.in);  int n = s.nextInt();  int first = 0,second = 1,next;  for(int i = 0;i<n;i++)  {  System.out.println(i+" day hours of exercise is "+first);  next = first+second;  first = second;  second = next;  }  }  }  OUTPUT:    **CONCLUSION:**  This program demonstrates the use of a loop to generate the Fibonacci series, calculating the exercise duration for each day based on the user-input number of days. It showcases a practical application of mathematical concepts in Java programming, providing a fun and interactive way to generate an exercise routine. |

|  |  |  |
| --- | --- | --- |
| **7.** | | Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;  front\_times('Chocolate', 2) → 'ChoCho'  front\_times('Chocolate', 3) → 'ChoChoCho'  front\_times('Abc', 3) → 'AbcAbcAbc'  **PROGRAM CODE:**  import java.util.Scanner;  public class practical7  {  static void front\_times(String str,int a)  {      for(int i=0;i<a;i++)      {      System.out.print(str.substring(0,3));      }      System.out.println();  }  static void array\_count9(Scanner sc)  {  int count = 0;      System.out.print("enter size of array: ");      int size = sc.nextInt();      int arr[] = new int[size];      System.out.println("enter array values");      for(int r =0;r<size;r++)      {      arr[r] = sc.nextInt();      }      System.out.print("enter the number that you want to know how many time that is revised: ");      int target = sc.nextInt();      for(int j=0;j<size;j++)      {      if(arr[j] == target)      count++;      }  System.out.println("count of 9: "+count);  }  public static void main(String[] args)  {      System.out.print("enter string : ");      Scanner sc = new Scanner(System.in);      String str = sc.nextLine();      System.out.print("enter number that you want repeat: ");      int a = sc.nextInt();      front\_times(str,a);      array\_count9(sc);  }  }  **OUTPUT:**    **CONCLUSION:**  To solve the problem of generating n copies of the front part of a string in Java, use the substring function to extract the first three characters (or the entire string if it's shorter). Then, repeat this segment n times.This method ensures correct handling even if the string is shorter than three characters. The solution is efficient, using simple string slicing and repetition. |
| **8.** | | Given an array of ints, return the number of 9's in the  array. array\_count9([1, 2, 9]) → 1  array\_count9([1, 9, 9]) → 2  array\_count9([1, 9, 9, 3, 9]) → 3  **PROGRAM CODE:**  import java.util.Scanner;  public class practical8 {      public static void main(String[] args) {          System.out.print("enter the string line: ");          Scanner s = new Scanner(System.in);          String str = s.nextLine();          System.out.print("enter the string that you want changed : ");          String a = s.nextLine();          System.out.print("enter the string from which that you want to changed : ");          String b = s.nextLine();          String replaced = str.replace(a, b);          String repl = replaced;          System.out.println("replaced string is: " + repl);      }  }  **OUTPUT:**    **CONCLUSION:**  In this practical we learn how to use function argument,use the count varible  And print number of int in the array is same. |
| **9** | | Given a string, return a string where for every char in the  original, there are two chars.  double\_char('The') → 'TThhee'  double\_char('AAbb') → 'AAAAbbbb'  double\_char('Hi-There') → 'HHii--TThheerree'  **PROGRAM CODE:**  import java.util.Scanner;  public class practical9  {  static void double\_char(String str)  {  int length = str.length();  for(int j = 0;j<length;j++){      char ch = str.charAt(j);      for(int i=0;i<2;i++)      {      System.out.print(ch);      }      }  }  public static void main(String[] args)  {  System.out.print("enter a string: ");  Scanner s=new Scanner(System.in);  String str = s.nextLine();  double\_char(str);  }  }  **OUTPUT:**    **CONCLUSION:**  To double every character in a string in Java, iterate through each character, appending it twice to a new string. This method ensures each character is duplicated, creating a new string with repeated characters. It provides an efficient and straightforward solution for string manipulation tasks. |
| **10.** | Perform following functionalities of the string:  ● Find Length of the String  ● Lowercase of the String  ● Uppercase of the String  ● Reverse String, Sort the string  **PROGRAM CODE:**  import java.util.Arrays;      class practical10{      public static int findLength(String str) {      return str.length();      }    public static String toLowercase(String str) {      return str.toLowerCase();      }    public static String toUppercase(String str) {      return str.toUpperCase();      }  public static String reverseString(String str) { StringBuilder sb = new StringBuilder(str); return sb.reverse().toString();  }  public static String sortString(String str) { char chars[] = str.toCharArray(); Arrays.sort(chars);  return new String(chars);  }  public static void main(String args[]){ String str = "ravi";  int length = findLength(str); System.out.println("Length: " + length);  String lowercase = toLowercase(str); System.out.println("Lowercase: " + lowercase);  String uppercase = toUppercase(str); System.out.println("Uppercase: " + uppercase);  String reversed = reverseString(str); System.out.println("Reversed: " + reversed);  String sorted = sortString(str); System.out.println("Sorted: " + sorted);  }  }  **OUTPUT:**    **CONCLUSION:**  In this practical we learn how to use different type of function use like,how to find length using length(), toLowercase(),toUpperCase(),reversed(),and short the String and display the output. | |
| **11.** | Perform following Functionalities of the string:  “CHARUSAT UNIVERSITY”  ● Find length  ● Replace ‘H’ by ‘FIRST LATTER OF YOUR NAME’  ● Convert all character in lowercase  **PROGRAM CODE:**  public class practical11 {      public static void main(String[] args) {          String str = "CHARUSAT UNIVERSITY";            int length = str.length();          System.out.println("Length of the string: " + length);            char firstLetterOfYourName = 'R';          String replacedStr = str.replace('H', firstLetterOfYourName);          System.out.println("String after replacing 'H' with " + firstLetterOfYourName + " : " + replacedStr);            String lowerCaseStr = str.toLowerCase();          System.out.println("Lowercase of the string: " + lowerCaseStr);  }  }  **OUTPUT:**    **CONCLUSION:**  This code finds the length of the string, replaces 'H' with 'your name first latter', and converts all characters to lowercase, printing the results. | |

**PART-III Object Oriented Programming: Classes, Methods, Constructors**

|  |  |
| --- | --- |
| **12.** | Imagine you are developing a currency conversion tool for a travel agency. This tool should be able to convert an amount in Pounds to Rupees. For simplicity, we assume the conversion rate is fixed: 1 Pound = 100 Rupees. The tool should be able to take input both from command-line arguments and interactively from the user.  **PROGRAM CODE:**  public class prec12 {      public static void main(String []args)      {          int a = Integer.parseInt(args[0]);            int c = a\*100;          System.out.println(c + " Pound");      }  }  **OUTPUT:**    **CONCLUSION:** Learnt about the working of command line argument.  s |
| **13.** | Create a class called Employee that includes three pieces of information as instance variables—a first name (type String), a last name (type String) and a monthly salary (double). Your class should have a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named EmployeeTest that demonstrates class Employee’s capabilities. Create two Employee objects and display each object’s yearly salary. Then give each Employee a 10% raise and display each Employee’s yearly salary again.  **PROGRAM CODE:**  class Employee {  private String firstName;  private String lastName;  private double monthlySalary;  Employee(String firstName, String lastName, double monthlySalary) {  this.firstName = firstName;  this.lastName = lastName;  if (monthlySalary > 0) {  this.monthlySalary = monthlySalary;  } else {  this.monthlySalary = 0.0;  }  }  public void setFirstName(String firstName) {  this.firstName = firstName;  }  public String getFirstName() {  return firstName;  }  public void setLastName(String lastName) {  this.lastName = lastName;  }  public String getLastName() {  return lastName;  }  public void setMonthlySalary(double monthlySalary) {  if (monthlySalary > 0) {  this.monthlySalary = monthlySalary;  } else {  this.monthlySalary = 0.0;  }  }  public double getMonthlySalary() {  return monthlySalary;  }  public double getYearlySalary() {  return monthlySalary \* 12;  }  public void giveRaise(double percentage) {  if (percentage > 0) {  monthlySalary += monthlySalary \* (percentage / 100);  }  }  }  class practical13{  public static void main(String[] args) {  Employee emp1 = new Employee("John", "Doe", 3000.00);  Employee emp2 = new Employee("Jane", "Smith", 4000.00);  System.out.printf("%s %s's yearly salary: %.2f%n", emp1.getFirstName(), emp1.getLastName(), emp1.getYearlySalary());  System.out.printf("%s %s's yearly salary: %.2f%n", emp2.getFirstName(), emp2.getLastName(), emp2.getYearlySalary());  emp1.giveRaise(10);  emp2.giveRaise(10);  System.out.printf("After 10%% raise, %s %s's yearly salary: %.2f%n", emp1.getFirstName(), emp1.getLastName(), emp1.getYearlySalary());  System.out.printf("After 10%% raise, %s %s's yearly salary: %.2f%n", emp2.getFirstName(), emp2.getLastName(), emp2.getYearlySalary());  }  }  **OUTPUT:**    **CONCLUSION:** In this practical we learnt about constructors and methods. |
| **14** | Create a class called Date that includes three pieces of information as instance variables—a month (type int), a day (type int) and a year (type int). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test application named DateTest that demonstrates class Date’s capabilities  **PROGRAM CODE:**  class Date  {  private int month;  private int day;  private int year;  public Date(int a,int b,int c)  {  month = a;  day = b;  year = c;  }  public void setmonth(int a)  {  month = a;  }  public int getmonth()  {  return month;  }  public void setday(int b)  {  day = b;  }  public int getday()  {  return day;  }  public void setyear(int c)  {  year = c;  }  public int getyear()  {  return year;  }  public void displayDate()  {  System.out.println("Date: "+month+"/"+day+"/"+year);  }  }  class practical14{  public static void main(String[] args)  {  Date d1 = new Date(7,30,2024);  Date d2 = new Date(1,26,1949);  System.out.print("Date 1: ");  d1.displayDate();  System.out.print("Date 2: ");  d2.displayDate();  d1.setmonth(1);  d1.setday(1);  d1.setyear(2021);  System.out.print("Modified Date 1: ");  d1.displayDate();    d2.setmonth(10);  d2.setday(31);  d2.setyear(2023);  System.out.print("Modified Date 2: ");  d2.displayDate();  }  }  **OUTPUT:**    **CONCLUSION:** In this practical we created class by which we can see date in DD/MM/YY format. |
| **15** | Write a program to print the area of a rectangle by creating a class named 'Area' taking the values of its length and breadth as parameters of its constructor and having a method named 'returnArea' which returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.  **PROGRAM CODE:**  class area {  int length, breadth;  area() {  }  area(int len, int brth) {  length=len;  breadth=brth;  }  int getarea()  {  return length\*breadth;  }  }  public class practical15 {  public static void main(String[] args) {  area a1=new area(50, 20);  System.out.println("Area : "+ a1.getarea());  }  }  **OUTPUT:**    **CONCLUSION:** In this practical we created a class by which we can calculate area of rectangle. |
| **16** | Print the sum, difference and product of two complex numbers by creating a class named ‘Complex’ with separate methods for each operation whose real and imaginary parts are entered by user.  **PROGRAM CODE:**  import java.util.Scanner;  class complex  {  int r,i;  Scanner s=new Scanner(System.in);  void getvalue()  {  System.out.println("Enter Real Part ");  r=s.nextInt();  System.out.println("Enter Imaginary Part ");  i=s.nextInt();  }  void add(complex cx)  {  int sumr=r+cx.r;  int sumi=i+cx.i;  System.out.println("Sum :" + sumr + "+" + sumi + "i");  }  void sub(complex cx)  {  int sumr=r-cx.r;  int sumi=i-cx.i;  System.out.println("Subtraction :" + sumr + "+" + sumi + "i");  }  void mul(complex cx)  {  int sumr=(r\*cx.r)-(i\*cx.i);  int sumi=(r\*cx.r)+(i\*cx.i);  System.out.println("multiplication :" + sumr + "+" + sumi + "i");  }  }  public class practical16 {  public static void main(String[] args) {  complex c1=new complex();  complex c2=new complex();  c1.getvalue();  c2.getvalue();  c1.add(c2);  c1.sub(c2);  c1.mul(c2);      }  }  **OUTPUT:**    **CONCLUSION:** In this practical we created a class by which we can do addition, subtraction and multiplication of complex numbers. |
| **17** | Create a class with a method that prints "This is parent  class" and its subclass with another method that prints "This  is child class". Now, create an object for each of the  class and call 1 - method of parent class by object of parent.  **PROGRAM CODE:**  class parent {      public void method() {          System.out.println("this is parent class");      }  }  class child {      public void method() {          System.out.println("this is child class");      }  }  class practical17 {      public static void main(String[] args) {          parent p = new parent();          child c = new child();          p.method();      }  }  **OUTPUT:**    **CONCLUSION:**  The example demonstrates inheritance in Java, where the Parent class has a method that prints a message, and the Child class extends it with its own method. An object of the Parent class is used to call the inherited method, showcasing method invocation through class instances. |
| **18** | Create a class named 'Member' having the following  members: Data members  1 - Name  2 - Age  3 - Phone number  4 - Address  5 – Salary  It also has a method named 'printSalary' which prints the  salary of the members. Two classes 'Employee' and  'Manager' inherits the 'Member' class. The 'Employee' and  'Manager' classes have data members 'specialization' and  'department' respectively. Now, assign name, age, phone  number, address and salary to an employee and a manager  by making an object of both of these classes and print the  same.  **PROGRAM CODE:**  class Member {      public String name, address;      public int age, salary;      public long phonenum;      public void printsalary() {          System.out.println("salary: " + salary);      }  }  class Employee extends Member {      String specialization, department;      public Employee() {          this.name = "Ravi";          this.age = 18;          this.phonenum = 9876558679L;          this.address = "charusat university";          this.salary = 85000;          specialization = "python";          department = "depstar";      }      public void detail() {          System.out.println("Employee detail");          System.out.println("name: " + this.name);          System.out.println("age: " + this.age);          System.out.println("phone number: " + this.phonenum);          System.out.println("address: " + this.address);          this.printsalary();          System.out.println("specialization: " + specialization);          System.out.println("department: " + department);      }  }  class Manager extends Member {      String specialization, department;      public Manager() {          this.name = "ajay";          this.age = 21;          this.phonenum = 7894587592L;          this.address = "charusat university";          this.salary = 87000;          specialization = "designing";          department = "depstar";      }      public void detail() {          System.out.println("Employee detail");          System.out.println("name: " + this.name);          System.out.println("age: " + this.age);          System.out.println("phone number: " + this.phonenum);          System.out.println("address: " + this.address);          this.printsalary();          System.out.println("specialization: " + specialization);          System.out.println("department: " + department);      }  }  class practical18 {      public static void main(String[] args) {          Member m = new Member();          Employee e1 = new Employee();          e1.detail();          Manager m1 = new Manager();          m1.detail();      }  }  **OUTPUT:**    **CONCLUSION:**  The example demonstrates inheritance in Java, where Employee and Manager classes inherit common attributes from the Member class while adding their own specific properties. This illustrates code reusability and the hierarchical structure of classes in object-oriented programming. |
| **19** | Create a class named 'Rectangle' with two data members  'length' and 'breadth' and two methods to print the area and  perimeter of the rectangle respectively. Its constructor  having parameters for length and breadth is used to  initialize length and breadth of the rectangle. Let class  'Square' inherit the 'Rectangle' class with its constructor  having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area  and perimeter of a rectangle and a square. Also use array  of objects.  **PROGRAM CODE:**  import java.util.Scanner;  class Rectangle {      public int length, breadth;      public Rectangle(int l, int b) {          length = l;          breadth = b;      }      public int area() {          return length \* breadth;      }      public int perimeter() {          return 2 \* (length + breadth);      }  }  class Square extends Rectangle {      public Square(int side) {          super(side, side);      }  }  public class practical19 {      public static void main(String[] args) {          Scanner s = new Scanner(System.in);          int i;          Rectangle[] rectangles = new Rectangle[2];          Square[] squares = new Square[2];          for (i = 0; i < 2; i++) {              System.out.println("Enter the value of rectangle " + (i + 1));              System.out.print("Enter the length of Rectangle: ");              int len = s.nextInt();              System.out.print("Enter the breadth of Rectangle: ");              int bre = s.nextInt();              rectangles[i] = new Rectangle(len, bre);              int resulta = rectangles[i].area();              System.out.println("Area of Rectangle: " + resulta);              int resultp = rectangles[i].perimeter();              System.out.println("Perimeter of Rectangle: " + resultp);              System.out.println();              System.out.print("Enter the side of Square: ");              int side = s.nextInt();              squares[i] = new Square(side);              resulta = squares[i].area();              System.out.println("Area of Square: " + resulta);              resultp = squares[i].perimeter();              System.out.println("Perimeter of Square: " + resultp);          }          s.close();      }  }  **OUTPUT:**    **CONCLUSION:**  The example demonstrates inheritance and method overriding in Java, where the Square class inherits properties from the Rectangle class. It showcases how to initialize objects using constructors and calculate the area and perimeter of both shapes using an array of objects. |
| **20** | Create a class named 'Shape' with a method to print "This  is This is shape". Then create two other classes named  'Rectangle', 'Circle' inheriting the Shape class, both  having a method to print "This is rectangular shape" and  "This is circular shape" respectively. Create a subclass  'Square' of 'Rectangle' having a method to print "Square  is a rectangle". Now call the method of 'Shape' and  'Rectangle' class by the object of 'Square' class.  **PROGRAM CODE:**  class Shape {      public void method() {          System.out.println("This is shape");      }  }  class Rectangle extends Shape {      @Override      public void method() {          System.out.println("This is rectangular shape");      }  }  class Circle extends Shape {      @Override      public void method() {          System.out.println("This is circular shape");      }  }  class Square extends Rectangle {      @Override      public void method() {          System.out.println("Square is a rectangle");      }  }  public class practical20 {      public static void main(String[] args) {          // Create an object of Square          Square square = new Square();            // Call the method of Square class          square.method(); // This will print "Square is a rectangle"            // To call the method from Rectangle or Shape, you can use casting          ((Shape) square).method(); // This will print "This is rectangular shape"          ((Rectangle) square).method(); // This will print "This is rectangular shape"      }  }  **OUTPUT:**    **CONCLUSION:**  This example demonstrates inheritance and method overriding, allowing subclasses to extend functionality. The Square class showcases polymorphism by invoking methods from both the Shape and Rectangle classes. |
| **21** | Create a class 'Degree' having a method 'getDegree' that  prints "I got a degree". It has two subclasses namely  'Undergraduate' and 'Postgraduate' each having a method  with the same name that prints "I am an Undergraduate"  and "I am a Postgraduate" respectively. Call the method  by creating an object of each of the three classes.  **PROGRAM CODE:**  class degree  {      void getdegree()      {          System.out.println("i get a degree");      }  }  class undergraduate extends degree  {      void getdegree()      {          System.out.println("i am an undergraduate");      }  }  class postgraduate extends degree  {      void getdegree()      {          System.out.println("i am an postgraduate");      }  }  public class practical21 {      public static void main(String[] args)      {          degree d = new degree();          d.getdegree();          undergraduate u = new undergraduate();          u.getdegree();          postgraduate p = new postgraduate();          p.getdegree();      }    }  **OUTPUT:**    **CONCLUSION:**  The program illustrates the concept of inheritance through a parent class and its subclasses, each implementing their version of a method. This showcases how subclasses can provide specific behavior while maintaining a shared interface. |
| **22** | Write a java that implements an interface  AdvancedArithmetic which contains amethod signature  int divisor\_sum(int n). You need to write a class  calledMyCalculator which implements the interface.  divisorSum function just takes an integer as input and  return the sum of all its divisors.  For example, divisors of 6 are 1, 2, 3 and 6, so  divisor\_sum should return 12. The value of n will be at  most 1000.  **PROGRAM CODE:**  interface AdvancedArithmetic {      int divisor\_sum(int n);  }  class MyCalculator implements AdvancedArithmetic {      public int divisor\_sum(int n) {          int sum = 0;          for (int i = 1; i <= n; i++) {              if (n % i == 0) {                  sum += i;              }          }          return sum;      }  }  public class practical22{      public static void main(String[] args) {          MyCalculator calculator = new MyCalculator();          System.out.println(calculator.divisor\_sum(6));      }  }  **OUTPUT:**    **CONCLUSION:**  This example shows how to define an interface in Java and implement its methods in a class, highlighting the importance of interfaces for creating flexible, reusable code. It emphasizes the contract between classes and methods that must be fulfilled. |
| **23** | Assume you want to capture shapes, which can be either  circles (with a radiusand a color) or rectangles (with a  length, width, and color). You also want to be able to  create signs (to post in the campus center, for example),  each of which has a shape (for the background of the sign)  and the text (a String) to put on the sign. Create classes  and interfaces for circles, rectangles, shapes, and signs.  Write a program that illustrates the significance of  interface default method.  **PROGRAM CODE:**  interface Shape {      String getColor();      default void display() {          System.out.println("Displaying shape with color: " + getColor());      }  }  interface Drawable {      void draw();  }  class Circle implements Shape, Drawable {      private double radius;      private String color;      public Circle(double radius, String color) {          this.radius = radius;          this.color = color;      }      public double getRadius() {          return radius;      }      @Override      public String getColor() {          return color;      }      @Override      public void draw() {          System.out.println("Drawing a circle with radius " + radius + " and color " + color);      }  }  class Rectangle implements Shape, Drawable {      private double length;      private double width;      private String color;      public Rectangle(double length, double width, String color) {          this.length = length;          this.width = width;          this.color = color;      }      public double getLength() {          return length;      }      public double getWidth() {          return width;      }      @Override      public String getColor() {          return color;      }      @Override      public void draw() {          System.out.println("Drawing a rectangle with length " + length + ", width " + width + " and color " + color);      }  }  class Sign {      private Shape shape;      private String text;      public Sign(Shape shape, String text) {          this.shape = shape;          this.text = text;      }      public void display() {          shape.display();          System.out.println("Sign text: " + text);      }  }  public class practical23 {      public static void main(String[] args) {          Shape circle = new Circle(5.0, "Red");          Shape rectangle = new Rectangle(10.0, 6.0, "Blue");          Sign circleSign = new Sign(circle, "Welcome to the Park!");          Sign rectangleSign = new Sign(rectangle, "Campus Center");          System.out.println("Circle Sign:");          circleSign.display();          System.out.println("\nRectangle Sign:");          rectangleSign.display();      }  }  **OUTPUT:**    **CONCLUSION:**  The example illustrates the significance of interfaces in Java, highlighting the use of default methods and the ability to define shared behavior among classes while allowing individual implementations. |
| **24** | Write a java program which takes two integers x & y as  input, you have to compute x/y. If x and y are not integers  or if y is zero, exception will occur and you have to  report it.  **PROGRAM CODE:**  import java.util.Scanner;  public class practical24{      public static void main(String[] args) {          Scanner s = new Scanner(System.in);          try {              System.out.print("Enter the first integer (x): ");              int x = Integer.parseInt(s.nextLine());              System.out.print("Enter the second integer (y): ");              int y = Integer.parseInt(s.nextLine());              int result = x / y;              System.out.println("Result of " + x + " / " + y + " = " + result);          } catch (ArithmeticException e) {              System.out.println("Error: " + e.getMessage());          } catch (Exception e) {              System.out.println("An unexpected error occurred: " + e.getMessage());          }      }  }    **OUTPUT:**    **CONCLUSION:**  The program emphasizes robust exception handling when performing division, capturing invalid inputs and preventing division by zero. This ensures the program can handle errors gracefully without crashing. |
| **25** | Write a Java program that throws an exception and catch  it using a try-catch block.  **PROGRAM CODE:**  class MyCustomException extends Exception {      public MyCustomException(String message) {          super(message);      }  }  public class practical25 {      public static void riskyMethod() throws MyCustomException {          throw new MyCustomException("Something went wrong!");      }      public static void main(String[] args) {          try {              riskyMethod();          } catch (MyCustomException e) {              System.out.println("Caught a custom exception: " + e.getMessage());          }      }  }  **OUTPUT:**    **CONCLUSION:**  This example demonstrates how to throw and catch exceptions in Java, emphasizing the importance of proper error handling. It illustrates how exceptions can be managed to maintain program stability during runtime. |
| **26** | Write a java program to generate user defined exception  using “throw” and “throws” keyword.  Also Write a java that differentiates checked and  unchecked exceptions. (Mention at least two checked and  two unchecked exceptions in program).  **PROGRAM CODE:**  import java.io.IOException;  import java.sql.SQLException;  class MyCustomException extends Exception {      public MyCustomException(String message) {          super(message);      }  }  public class practical26 {      public static void riskyMethod() throws MyCustomException {          throw new MyCustomException("User-defined exception occurred!");      }      public static void methodWithCheckedExceptions() throws IOException, SQLException {          throw new IOException("This is a checked IOException.");      }      public static void methodWithUncheckedExceptions() {          throw new NullPointerException("This is an unchecked NullPointerException.");      }      public static void main(String[] args) {          try {              riskyMethod();          } catch (MyCustomException e) {              System.out.println("Caught a custom exception: " + e.getMessage());          }          try {              methodWithCheckedExceptions();          } catch (IOException e) {              System.out.println("Caught an IOException: " + e.getMessage());          } catch (SQLException e) {              System.out.println("Caught an SQLException: " + e.getMessage());          }            try {              methodWithUncheckedExceptions();          } catch (NullPointerException e) {              System.out.println("Caught a NullPointerException: " + e.getMessage());          } catch (ArithmeticException e) {              System.out.println("Caught an ArithmeticException: " + e.getMessage());          }      }  }  **OUTPUT:**    **CONCLUSION:**  The program shows how to create user-defined exceptions and differentiate between checked and unchecked exceptions. This highlights effective error management in Java, allowing developers to handle specific conditions gracefully. |
| **27** | Write a program that will count the number of lines in  each file that is specified on the command line. Assume  that the files are text files. Note that multiple files can be  specified, as in "java Line Counts file1.txt file2.txt  file3.txt". Write each file name, along with the number of  lines in that file, to standard output. If an error occurs  while trying to read from one of the files, you should print  an error message for that file, but you should still  process all the remaining files.  **PROGRAM CODE:**  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  public class practical27 {      public static void main(String[] args) {          if (args.length == 0) {              System.out.println("Usage: java LineCounter <file1> <file2> ...");              return;          }          for (String filename : args) {              try {                  int lineCount = countLines(filename);                  System.out.println(filename + ": " + lineCount + " lines");              } catch (IOException e) {                  System.err.println("Error reading file: " + filename);              }          }      }      private static int countLines(String filename) throws IOException {          try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {              int lines = 0;              while (reader.readLine() != null) {                  lines++;              }              return lines;          }      }  }  **OUTPUT:**    **CONCLUSION:**  This example illustrates how to count lines in multiple text files while handling errors gracefully. Even if some files fail to read, the program continues processing, showcasing resilience in file operations. |
| **28** | Write an example that counts the number of times a particular character, such as e, appears in a file. The  character can be specified at the command line. You can  use xanadu.txt as the input file.  **PROGRAM CODE:**  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  public class practical28 {      public static void main(String[] args) {          if (args.length != 2) {              System.out.println("Usage: java CharacterCounter <filename> <character>");              return;          }          String filename = args[0];          char targetChar = args[1].charAt(0);          int count = 0;          try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {              String line;              while ((line = reader.readLine()) != null) {                  for (int i = 0; i < line.length(); i++) {                      if (line.charAt(i) == targetChar) {                          count++;                      }                  }              }              System.out.println("The character '" + targetChar + "' appears " + count + " times in the file.");          } catch (IOException e) {              System.out.println("An error occurred while reading the file.");              e.printStackTrace();          }      }  }  **OUTPUT:**    **CONCLUSION:**  The program demonstrates how to count the occurrences of a specified character in a file using command-line input. This showcases practical file handling and character manipulation in Java. |
| **29** | Write a Java Program to Search for a given word in a  File. Also show use of Wrapper Class with an example.  **PROGRAM CODE:**  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  import java.util.Scanner;  class practical29 {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);            System.out.print("Enter the filename: ");          String filename = scanner.nextLine();          System.out.print("Enter the word to search: ");          String wordToSearch = scanner.nextLine();          int lineCount = 0;          boolean wordFound = false;          try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {              String line;              while ((line = reader.readLine()) != null) {                  lineCount++;                    if (line.contains(wordToSearch)) {                      System.out.println("Word found on line " + lineCount + ": " + line);                      wordFound = true;                  }              }              if (!wordFound) {                  System.out.println("The word '" + wordToSearch + "' was not found in the file.");              }          } catch (IOException e) {              System.out.println("An error occurred while reading the file.");              e.printStackTrace();          }          int primitiveInt = 42;          Integer wrapperInt = Integer.valueOf(primitiveInt);          System.out.println("Primitive int: " + primitiveInt);          System.out.println("Wrapper Integer: " + wrapperInt);      }  }  **OUTPUT:**    **CONCLUSION:**  This example illustrates searching for a specific word in a file while utilizing wrapper classes. It reinforces the importance of file I/O operations and string manipulation in Java applications. |
| **30** | Write a program to copy data from one file to another file.  If the destination file does not exist, it is created  automatically.  **PROGRAM CODE:**  import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  public class practical30 {      public static void main(String[] args) {          String sourceFile = "source.txt";          String destinationFile = "destination.txt";          try (BufferedReader reader = new BufferedReader(new FileReader(sourceFile));              BufferedWriter writer = new BufferedWriter(new FileWriter(destinationFile))) {                String line;                while ((line = reader.readLine()) != null) {                  writer.write(line);                  writer.newLine();              }                System.out.println("Data copied from " + sourceFile + " to " + destinationFile + " successfully.");            } catch (IOException e) {              System.out.println("An error occurred while copying the file.");              e.printStackTrace();          }      }  }  **OUTPUT:**    **CONCLUSION:**  The program shows how to copy data from one file to another, automatically creating the destination file if it doesn't exist. This highlights practical file handling and ensures data integrity during file operations. |
| **31** | Write a program to show use of character and byte stream.  Also show use of  BufferedReader/BufferedWriter to read console input  and write them into a file.  **PROGRAM CODE:**  import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.FileWriter;  import java.io.IOException;  import java.io.InputStreamReader;  public class practical31 {      public static void main(String[] args) {          String fileName = "output.txt";          try (BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));              BufferedWriter fileWriter = new BufferedWriter(new FileWriter(fileName))) {              System.out.println("Enter text to write to the file (type 'exit' to finish):");              String inputLine;              while (!(inputLine = consoleReader.readLine()).equalsIgnoreCase("exit")) {                  fileWriter.write(inputLine);                  fileWriter.newLine();              }              System.out.println("Data has been written to " + fileName);          } catch (IOException e) {              System.out.println("An error occurred while reading from console or writing to the file.");              e.printStackTrace();          }          try (FileInputStream fileInputStream = new FileInputStream(fileName);              FileOutputStream fileOutputStream = new FileOutputStream("copy\_output.txt")) {              System.out.println("Copying data from " + fileName + " to copy\_output.txt...");              int byteData;              while ((byteData = fileInputStream.read()) != -1) {                  fileOutputStream.write(byteData);              }              System.out.println("Data has been copied to copy\_output.txt");          } catch (IOException e) {              System.out.println("An error occurred while copying the file.");              e.printStackTrace();          }      }  }  **OUTPUT:**    **CONCLUSION:**  This example illustrates the use of character and byte streams, employing BufferedReader and BufferedWriter for efficient I/O operations. It highlights the importance of handling console input and file output in a streamlined manner. |
| **32** | Write a program to create thread which display “Hello  World” message. A. by extending Thread class B. by using  Runnable interface.  **PROGRAM CODE:**  class practical32 extends Thread {      public void run() {          System.out.println("hello world");      }      public static void main(String[] args) {          practical32 thread = new practical32();          thread.start();      }  }  **OUTPUT:**    **CONCLUSION:**  The program demonstrates creating threads using both the Thread class and the Runnable interface. This showcases different approaches to multithreading in Java and how they can be effectively utilized. |
| **33** | Write a program which takes N and number of threads as  an argument. Program should distribute the task of  summation of N numbers amongst number of threads and  final result to be displayed on the console.  **PROGRAM CODE:**  class First extends Thread {      private int start, end;      private int sum;      public First(int start, int end) {          this.start = start;          this.end = end;      }      public void run() {          sum = 0;          for (int i = start; i <= end; i++) {              sum += i;          }      }      public int getSum() {          return sum;      }  }  class Second extends Thread {      private int start, end;      private int sum;      public Second(int start, int end) {          this.start = start;          this.end = end;      }      public void run() {          sum = 0;          for (int i = start; i <= end; i++) {              sum += i;          }      }      public int getSum() {          return sum;      }  }  public class practical33 {      public static void main(String[] args) {          int N = 10;          int n = 2;          int a = N / n;          First f = new First(1, a);          Second s = new Second(a + 1, N);          f.start();          s.start();          try {              f.join();              s.join();          } catch (InterruptedException e) {              e.printStackTrace();          }          int sum = f.getSum() + s.getSum();          System.out.println("Sum: " + sum);      }  }  **OUTPUT:**    **CONCLUSION:**  This example illustrates distributing tasks among multiple threads for efficient computation of a summation. It emphasizes the performance benefits of using multithreading in applications that require parallel processing. |
| **34** | Write a java program that implements a multi-thread  application that has three threads. First thread generates  random integer every 1 second and if the value is even,  second thread computes the square of the number and  prints. If the value is odd, the third thread will print the  value of cube of the number.  **PROGRAM CODE:**  import java.util.Random;  class Randomclass extends Thread {      public static int number;      public void run() {          Random rnum = new Random();          while (true) {              number = rnum.nextInt(100);              System.out.println("Generated Number: " + number);              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  System.out.println("RandomNumberGenerator interrupted.");              }          }      }  }  class Square extends Thread {      public void run() {          while (true) {              if (Randomclass.number % 2 == 0) {                  int square = Randomclass.number \* Randomclass.number;                  System.out.println("Square of " + Randomclass.number + " is: " + square);              }              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  System.out.println("Square interrupted.");              }          }      }  }  class Cube extends Thread {      public void run() {          while (true) {              if (Randomclass.number % 2 != 0) {                  int cube = Randomclass.number \* Randomclass.number \* Randomclass.number;                  System.out.println("Cube of " + Randomclass.number + " is: " + cube);              }              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  System.out.println("Cube interrupted.");              }          }      }  }  public class practical34 {      public static void main(String[] args) {          Randomclass random = new Randomclass();          Square square = new Square();          Cube cube = new Cube();          random.start();          square.start();          cube.start();      }  }  **OUTPUT:**    **CONCLUSION:**  The program shows coordination between threads that generate and process random numbers based on their even or odd nature. This illustrates conditional execution in multithreaded applications, enhancing functionality. |
| **35** | Write a program to increment the value of one variable by  one and display it after one second using thread using  sleep() method.  **PROGRAM CODE:**  class practical35 extends Thread {      private int value;      public practical35(int value) {          this.value = value;      }      public void run() {          try {              Thread.sleep(1000);              value++;              System.out.println("value after increment: " + value);          } catch (InterruptedException e) {              System.out.println("thread interrupted");          }      }      public static void main(String[] args) {          practical35 thread = new practical35(5);          thread.start();      }  }  **OUTPUT:**    **CONCLUSION:**  This example demonstrates using threads to increment a variable with a delay, showcasing thread control. It highlights how to utilize the sleep() method for timing in concurrent execution. |
| **36** | Write a program to create three threads ‘FIRST’,  ‘SECOND’, ‘THIRD’. Set the priority of the ‘FIRST’  thread to 3, the ‘SECOND’ thread to 5(default) and the  ‘THIRD’ thread to 7.  **PROGRAM CODE:**  class practical36 extends Thread {      public practical36(String name) {          super(name);      }      public void run() {          System.out.println(Thread.currentThread().getName() + "with priority " + Thread.currentThread().getPriority());      }      public static void main(String[] args) {          practical36 firstt = new practical36("FIRST");          practical36 secondt = new practical36("SECOND");          practical36 thirdt = new practical36("THIRD");          firstt.setPriority(3);          secondt.setPriority(Thread.NORM\_PRIORITY);          thirdt.setPriority(7);          firstt.start();          secondt.start();          thirdt.start();      }  }  **OUTPUT:**    **CONCLUSION:**  The program illustrates setting thread priorities in Java, emphasizing how priority affects thread execution order. This showcases the ability to control thread behavior and performance in a multithreaded environment. |
| **37** | Write a program to solve producer-consumer problem  using thread synchronization.  **PROGRAM CODE:**  class Produce extends Thread {      int n;      boolean produced = false;      Produce(int n) {          this.n = n;      }      public synchronized void run() {          for (int i = 1; i <= n; i++) {              while (produced) {  // Wait if something is already produced                  try {                      wait();                  } catch (InterruptedException e) {                      Thread.currentThread().interrupt();                  }              }              System.out.println("Produced: " + i);              produced = true;              notify();  // Notify the consumer that production is done          }      }  }  class Consume extends Thread {      int n;      Produce producer;      Consume(int n, Produce producer) {          this.n = n;          this.producer = producer;      }   public synchronized void run() {          for (int i = 1; i <= n; i++) {              synchronized (producer) {                  while (!producer.produced) {  // Wait until something is produced                      try {                          producer.wait();                      } catch (InterruptedException e) {                          Thread.currentThread().interrupt();                      }                  }                  System.out.println("Consumed: " + i);                  producer.produced = false;                  producer.notify();  // Notify the producer to produce more              }          }      }  }  public class practical37 {      public static void main(String[] args) {          int n = 10;  Produce p = new Produce(n);          Consume c = new Consume(n, p);   p.start();          c.start();      }}  **OUTPUT:**    **CONCLUSION:**  This example shows how to solve the producer-consumer problem using thread synchronization, demonstrating inter-thread communication. It highlights the importance of managing shared resources to avoid race conditions and ensure data consistency. |
| **38** | Design a Custom Stack using ArrayList class, which implements following functionalities of stack. My Stack  -list ArrayList<Object>: A list to store elements.  +isEmpty: boolean: Returns true if this stack is empty.  +getSize(): int: Returns number of elements in this stack.  +peek(): Object: Returns top element in this stack without  removing it.  +pop(): Object: Returns and Removes the top elements in  this stack.  +push(o: object): Adds new element to the top of this  stack.  **PROGRAM CODE:**  import java.util.\*;  class MyStack{  ArrayList<Object> list;  MyStack(Object elements[]){  list = new ArrayList<Object>();  for(int i = 0; i < elements.length; i++){  list.add( elements[i] );  }  }  MyStack(){  list = new ArrayList<Object>();  }  boolean isEmpty(){  return (list.size() == 0);  }  Object peek(){  return list.get( list.size()-1 );  }  Object pop(){  Object ob = list.get( list.size()-1 );  list.remove( list.size()- 1 );  return ob;  }  void push(Object o){  list.add(o);  }  }  public class PRACT38{  public static void main(String[] args){  Integer arr[] = new Integer[]{1,2,3,4};  MyStack s = new MyStack( arr );  System.out.println("Current top = " + s.peek());  System.out.println("Pushing 7,8,9 in the stack");  s.push(7);  s.push(8);  s.push(9);  s.pop();  System.out.println("Elements in the stack are: ");  while(!s.isEmpty()){  System.out.println(s.pop());  }  }  }  **OUTPU**T**:**    **CONCLUSION:**  Designing a custom stack using the ArrayList class demonstrates the implementation of fundamental stack operations like push, pop, and peek. This exercise enhances understanding of data structures, specifically how dynamic arrays can be used to manage collections of objects efficiently. |
| **39** | Imagine you are developing an e-commerce application.  The platform needs to sort lists of products based on  different criteria, such as price, rating, or name. Each  product object implements the Comparable interface to  define the natural ordering. To ensure flexibility and  reusability, you need a generic method that can sort any  array of Comparable objects. Create a generic method in  Java that sorts an array of Comparable objects. This method  should be versatile enough to sort arrays of different types  of objects (such as products, customers, or orders) as long  as they implement the Comparable interface.  **PROGRAM CODE:**  public class practical39 {  public static <T extends Comparable<T>> void sortArray(T[] array) {  int n = array.length;  boolean swapped;  for (int i = 0; i < n - 1; i++) {  swapped = false;  for (int j = 0; j < n - 1 - i; j++) {  if (array[j].compareTo(array[j + 1]) > 0) {  T temp = array[j];  array[j] = array[j + 1];  array[j + 1] = temp;  swapped = true;  }  }  if (!swapped) {  break;  }  }  }  public static void main(String[] args) {  Product[] products = {  new Product("Laptop", 1200, 4.5),  new Product("Phone", 800, 4.3),  new Product("Headphones", 150, 4.7),  new Product("Monitor", 300, 4.4)  };  sortArray(products);  for (Product p : products) {  System.out.println(p);  }  }  }  class Product implements Comparable<Product> {  String name;  double price;  double rating;  public Product(String name, double price, double rating) {  this.name = name;  this.price = price;  this.rating = rating;  }  @Override  public int compareTo(Product other) {  return Double.compare(this.price, other.price);  }  @Override  public String toString() {  return name + " - $" + price + " - Rating: " + rating;  }  }  **OUTPU**T**:**    **CONCLUSION:**  This program demonstrates the flexibility of generic methods in Java, allowing you to sort arrays of various types that implement the Comparable interface. It ensures reusability by creating a method that works with any class, promoting efficient code organization. |
| **40** | Write a program that counts the occurrences of words in  a text and displays the words and their occurrences in  alphabetical order of the words. Using Map and Set  Classes.  **PROGRAM CODE:**  import java.util.\*;  public class practical40 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.print("Enter the text: ");  String inputText = sc.nextLine();    inputText = inputText.toLowerCase();  HashMap<String, Integer> wordCountMap = new HashMap<>();    StringBuilder currentWord = new StringBuilder();    for (int i = 0; i < inputText.length(); i++) {  char c = inputText.charAt(i);    if (Character.isLetter(c) || Character.isDigit(c)) {  currentWord.append(c);  } else {  if (currentWord.length() > 0) {  String word = currentWord.toString();  wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);  currentWord.setLength(0);  }  }  }  if (currentWord.length() > 0) {  String word = currentWord.toString();  wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);  }  TreeSet<String> sortedWords = new TreeSet<>(wordCountMap.keySet());  System.out.println("Word occurrences:");  for (String word : sortedWords) {  System.out.println(word + ": " + wordCountMap.get(word));  }  sc.close();  }  }  **OUTPU**T**:**    **CONCLUSION:**  This example showcases the use of Map and Set to count occurrences of words in a text and display them in alphabetical order. It highlights the efficiency of collections in managing and organizing data, making it easier to analyze text. |
| **41** | Write a code which counts the number of the keywords in  a Java source file. Store all the keywords in a HashSet  and use the contains () method to test if a word is in the  keyword set.  **PROGRAM CODE:**  import java.io.BufferedReader;  import java.io.File;  import java.io.FileReader;  import java.io.IOException;  import java.util.HashSet;  import java.util.Scanner;  import java.util.StringTokenizer;  public class practical41{      public static void main(String[] args) throws IOException{          Scanner sc = new Scanner(System.in);          System.out.print("Enter the file name you want to scan : ");          String f = sc.nextLine();          File file = new File(f);          FileReader br = new FileReader(file);          BufferedReader fr = new BufferedReader(br);          String keywords[] = new String[]{"abstract","assert ","boolean","break","byte","case","catch","char","class",                                          "continue","default","do","double","else","enum ","extends","final","finally",                                          "float","for","if","implements","import","instanceof","int","interface","long",                                          "native","new","package","private","protected","public","return","short","static",                                          "strictfp","super","switch","synchronized","this","throw","throws","transient","try",                                          "void","volatile","while"};          HashSet<String> set = new HashSet<String>();          for(int i =0;i < keywords.length; ++i){              set.add(  keywords[i] );          }          String st;          int count =0 ;          while ((st = fr.readLine()) != null){              StringTokenizer str = new StringTokenizer( st, " +-/\*%<>;:=&|!~()");              while(str.hasMoreTokens()){                  String swre = str.nextToken();                  if(set.contains(swre )){                      count++;                  }              }          }          System.out.println("Total keywords are : " + count);          fr.close();          sc.close();      }  }    **OUTPU**T**:**    **CONCLUSION:**  This program utilizes a HashSet to efficiently store and check Java keywords, highlighting the importance of collections for keyword analysis in source files. |