**CHAROTAR UNIVERSITY OF SCIENCE TECHNOLOGY**

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

Department of Computer Science & Engineering

**Subject Name: java programming**

**Semester: 3rd**

**Subject Code: CSE201**

**Academic year: 2024**

**Part - I**

|  |  |
| --- | --- |
| **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.  **Java Installation Steps**  1. Download JDK: Visit Oracle's JDK download page and select the version for your OS.  2. Install JDK: Run the installer and follow instructions.  3. Set Environment Variables: Add JDK's `bin` directory to your system PATH.  4. Verify Installation: Use `java -version` and `javac -version` in Command Prompt.  **Object-Oriented Concepts**  - Inheritance, Polymorphism, Encapsulation, Abstraction: Key principles for structuring code.  **Comparison with Other Languages**  - C++: Manual memory management.  - Python: Slower, dynamically typed.  - C#: Tied to Microsoft ecosystem.  **Java Components**  - JDK: Development kit.  - JRE: Runtime environment.  - JVM: Executes Java bytecode.  - Javadoc: Generates documentation.  - Command Line Arguments: Pass configuration info to `main(String[] args)`.  **IDEs and Console Programming**  - Eclipse, NetBeans, BlueJ: Popular IDEs for Java development.  - Console Programming: Compile with `javac` and run with `java`. |
| **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 :**  public class prac2 {      public static void main (String [] args){          String balance;          balance="$20";          System.out.print("current balance is : ");          System.out.println(balance);          System.out.println("23DCS088 RUDRA A. PATEL");      }  }  **OUTPUT:**    **CONCLUSION:**  This practical exercise helps in understanding Variable declaration and initialization in Java. Printing output to the console using System.out.print() and System.out.println().Basic structure and syntax of a Java program. |
| **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 prac3 {      public static void main(String[] args) {          Scanner sc=new Scanner (System.in);          System.out.print("Enter the distance in meters: ");          float distance = sc.nextFloat();          System.out.print("Enter the time (in hours): ");          int hours = sc.nextInt();          System.out.print("Enter the time (in minutes): ");          int minutes = sc.nextInt();          System.out.print("Enter the time (in seconds): ");          int seconds = sc.nextInt();          int totalSeconds;          totalSeconds = seconds+(minutes\*60)+(hours\*3600);          float speed1= distance / totalSeconds;          float speed2= (distance / 1000) / (totalSeconds / 3600.0f);          float speed3= (distance / 1609) / (totalSeconds / 3600.0f);          System.out.println("Speed in meters/second: " + speed1);          System.out.println("Speed in kilometers/hour: " + speed2);          System.out.println("Speed in miles/hour: " + speed3);          System.out.println("23DCS088 RUDRA A. PATEL");      }    }  **OUTPUT:**    **CONCLUSION:**  This Java program calculates and displays the speed based on user-inputted distance and time values in hours, minutes, and seconds. It converts and prints the speed in meters/second, kilometers/hour, and miles/hour. Finally, it prints identifying information. |
| **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.\*;  public class prac4{      public static void main(String[] args) {          Scanner sc=new Scanner (System.in);          int i;          int sum=0;          int[] arr=new int[30];          for(i=0;i<30;i++){              System.out.print("day ");              System.out.print(i+1);              System.out.print(" expense is: ");              arr[i]=sc.nextInt();              sum = sum + arr[i];          }          System.out.println("total expenses : " + sum);          System.out.println("23DCS088 RUDRA A. PATEL");      }  }    **OUTPUT:**    **CONCLUSION:**  This Java program calculates the total expenses over 30 days based on user-inputted expenses for each day. It stores these expenses in an array, computes their sum, and then prints out the total expenses along with identifying information |
| **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.\*;  public class prac5 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          float[] prices = { 100.0f, 200.0f, 300.0f, 400.0f, 0.0f };          String[] products = { "Motor", "Fan", "Tube", "Wires", "" };          System.out.println("code 1 to motor");          System.out.println("code 2 to fan");          System.out.println("code 3 to tube");          System.out.println("code 4 to wires.");          System.out.println("code 5 to All other items");          System.out.print("Enter product code: ");          int productCode = sc.nextInt();          System.out.print("Enter quantity: ");          int quantity = sc.nextInt();          double totalPrice = prices[productCode] \* quantity;          switch (productCode) {              case 1:                  totalPrice = totalPrice + (totalPrice \* 0.08);                  break;              case 2:                  totalPrice = totalPrice + (totalPrice \* 0.12);                  break;              case 3:                  totalPrice = totalPrice + (totalPrice \* 0.05);                  break;              case 4:                  totalPrice = totalPrice + (totalPrice \* 0.075);                  break;              default:                  totalPrice = totalPrice + (totalPrice \* 0.03);          }          System.out.println("Product: " + products[productCode]);          System.out.println("Quantity: " + quantity);          System.out.println("Total Price: " + totalPrice);          System.out.println("23DCS088 RUDRA A. PATEL");      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program calculates the total price of a product based on the product code and quantity input by the user. The total price includes the base price of the product and an additional tax that varies by product type. |
| **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.\*;  public class prac6 {      public static void main(String[] args) {          Scanner sc=new Scanner (System.in);         int n,t1=0,t2=1;         System.out.println("enter the number of days (n) for you want to generate their exercise routine.");          n=sc.nextInt();         for(int i=1;i<=n;++i)         {          System.out.print(t1 + ",");          int sum=t1+t2;          t1=t2;          t2=sum;         }      System.out.println("");      System.out.println("23DCS088 RUDRA A. PATEL");      }    }  **OUTPUT:**    **CONCLUSION:**  This Java program generates an exercise routine for a given number of days based on the Fibonacci sequence, where each day's routine is represented by a term in the sequence. The program outputs the first n terms of the Fibonacci series as the exercise plan. |

**Part – II**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **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.\*;  public class prac7  {      public static void main(String[] args)      {          Scanner sc=new Scanner(System.in);          System.out.print("Enter a word:");          String str=sc.nextLine();          int n;          System.out.print("enter n:");          n=sc.nextInt();          if(str.length()<3)          {              for(int i=0;i<n;i++)              {                 System.out.println(str);              }          }          else          {              for(int i=0;i<n;i++)              {                  System.out.print(str.substring(0, 3));              }          }          System.out.println("");          System.out.println("23DCS088 RUDRA A. PATEL");          sc.close();      }  }  **OUTPUT:**        **CONCLUSION:**  The provided Java code takes a word and a number n as input. If the word has fewer than three characters, it prints the word n times, each on a new line. If the word has three or more characters, it prints the first three characters of the word n times consecutively on a single line. |
| **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.\*;  public class prac8  {      public static void main(String[] args)      {          Scanner sc=new Scanner(System.in);          int n;          System.out.print("Enter the array size:");          n=sc.nextInt();          int []arr=new int[n];          System.out.println("Enter the element of array:");          for(int i=0;i<n;i++)          {              arr[i]=sc.nextInt();          }          System.out.print("The number of 9's in the array is:");          System.out.println(array\_count9(arr));          System.out.println("23DCS088 RUDRA A. PATEL");      }          public static int array\_count9(int[] arr)          {              int count=0;              for(int i=0;i<arr.length;i++)              {                  if(arr[i] == 9 )                  {                      count++;                  }              }              return count;          }  }  **OUTPUT:**          **CONCLUSION:**  The Java code prompts the user to enter the size and elements of an array, then counts and prints the number of times the number 9 appears in the array. This is achieved by iterating through the array and incrementing a counter whenever a 9 is encountered. |
| **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.\*;  public class prac9 {      public static void main(String[] args) {          Scanner sc=new Scanner(System.in);          System.out.print("Enter a word : ");          String str=sc.nextLine();          System.out.println(double\_char(str));          System.out.println("23DCS088 RUDRA A. PATEL");      }      public static String double\_char(String str) {          String result="";          for(int i=0;i<str.length();i++)          {              result += str.charAt(i);              result += str.charAt(i);          }          return result;      }  }  **OUTPUT:**        **CONCLUSION:**  The Java code takes a word as input and doubles each character in the word, then prints the resulting string. This is accomplished by iterating through each character in the input word and appending it twice to a result string. |
| **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.\*;  public class prac10 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          System.out.print("Enter a String : ");          String str = sc.nextLine();          int length = str.length();          System.out.println("Length of a String is :" + length);          String lower = str.toLowerCase();          System.out.println("Lower case of String is :" + lower);          String upper = str.toUpperCase();          System.out.println("Upper case of String is :" + upper);          String reverse = "";          for (int i = str.length() - 1; i >= 0; i--) {              reverse = reverse + str.charAt(i);          }          System.out.println("Reverse String is :" + reverse);          char[] charArray = str.toCharArray();          Arrays.sort(charArray);          String sortedString = new String(charArray);          System.out.println("Sorted string: " + sortedString);          System.out.println("23DCS088 RUDRA A. PATEL");      }  }  **OUTPUT:**    **CONCLUSION:**  The Java code takes a string input from the user and performs several operations: it calculates and prints the string's length, converts and prints the string in both lowercase and uppercase, reverses and prints the string, and sorts and prints the string characters in alphabetical order. |
| **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 prac11 {      public static void main(String[] args) {          String str = "CHARUSAT UNIVERSITY";          int length = str.length();          System.out.println("Length of a String is :" + length);          String str2 = str.replace('H', 'R');          System.out.println("Modified string is : " + str2);          String lowercaseString = str.toLowerCase();          System.out.println("Lower case of String is :" + lowercaseString);          System.out.println("23DCS088 RUDRA A. PATEL");        }  }  **OUTPUT:**    **CONCLUSION:**  The Java code demonstrates string manipulation by calculating the length of a given string, replacing a character in the string, converting the string to lowercase, and printing the results. The program specifically works with the string "CHARUSAT UNIVERSITY" and showcases these string operations. |

**Part – III**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **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 :**  **(1)**  import java.util.\*;  public class prac12 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  float pound;  System.out.print("Enter Pound :");  pound = Integer.parseInt(args[0]);  float rupees;  rupees = pound \* 100;  System.out.println("rupees :" + rupees);  }  }  **(2)**  import java.util.\*;  public class prac12 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  float pound=10;  float rupees;  rupees = pound \* 100;  System.out.println("rupees :" + rupees);  }  }  **OUTPUT:**  **(1)**    **(2)**    **CONCLUSION:**  This Java program converts Pounds to Rupees using a conversion rate of 1 Pound = 100 Rupees. It accepts the amount either from command-line arguments or interactively from the user. The result is then printed to the console |
| **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 :**  import java.util.\*;  public class prac13 {      public static void main(String args[]) {          Employee e1 = new Employee();          Employee e2 = new Employee();          e1.get();          e2.get();          e1.put();          e2.put();          e1.yearlysalary();          e2.yearlysalary();          e1.raise(10);          e2.raise(10);      }  }  class Employee {      String F\_name;      String L\_name;      double monthly\_salary;      Employee() {          monthly\_salary = 0;      }      public void get() {          Scanner sc = new Scanner(System.in);          System.out.println("Enter your first name : ");          F\_name = sc.nextLine();          System.out.println("Enter your Last name : ");          L\_name = sc.nextLine();          System.out.println("Enter your monthly salary : $");          monthly\_salary = sc.nextDouble();          if (monthly\_salary <= 0) {              monthly\_salary = 0.0;          }      }      public void put() {          System.out.println("Your first name : " + F\_name);          System.out.println("Your last name : " + L\_name);          System.out.println("Your monthly salary :$" + monthly\_salary);      }      public void yearlysalary() {          double Yearlysalary = monthly\_salary \* 12;          System.out.println("Your yearly salary : $" + Yearlysalary);      }      public void raise(int r) {          monthly\_salary += monthly\_salary \* r / 100;          System.out.println("New salary after raise : $" + monthly\_salary);      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program defines an Employee class with methods to calculate yearly salary, give a raise, and display employee details. In the main method, two employees are created, their details and yearly salaries are displayed, and then their salaries are increased by 10%. The updated yearly salaries are printed. |
| **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 :**  import java.util.\*;  public class prac14 {      public static void main(String[] args) {          data d1 = new data(0, 0, 0);          d1.get();          d1.display();          System.out.println("23DCS088 RUDRA PATEL");      }  }  class data {      int day;      int month;      int year;      data(int d, int m, int y) {          day = d;          month = m;          year = y;      }      public void get() {          Scanner sc = new Scanner(System.in);          System.out.print("Enter a Day :");          day = sc.nextInt();          System.out.print("Enter a Month :");          month = sc.nextInt();          System.out.print("Enter a Year :");          year = sc.nextInt();      }      public void display() {          System.out.println("DATE :" + day + "/" + month + "/" + year);      }  }  **OUTPUT:**    **CONCLUSION:**  It defines a Date class with private fields for day, month, and year. There’s also a displayDate() method that prints the date in the format “month/day/year.” The DateTest class creates two instances of Date and displays their dates. |
| **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 :**  import java.util.\*;  public class prac15 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          int l, b;          System.out.print("Enter Length :");          l = sc.nextInt();          System.out.print("Enter Breath :");          b = sc.nextInt();          Area AR = new Area(l, b);          System.out.print("Area of rectangle :");          int a = AR.returnvalue();          System.out.println(a);          System.out.println("23DCS088 RUDRA PATEL");      }  }  class Area {      int length;      int breath;      Area(int length, int breath) {          this.length = length;          this.breath = breath;      }      int returnvalue() {          return this.length \* this.breath;      }  }  **OUTPUT:**    **CONCLUSION:**  The code defines an Area class with private fields for length and breadth. It calculates the area of a rectangle using the formula length \* breadth. The user is prompted to input the length and breadth of the rectangle. The program then creates an instance of the Area class and computes the area. |
| **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.\*;  public class pract16 {       public static void main(String[] args) {                Complex c1 = new Complex(3, 5);              Complex c2 = new Complex(7, 9);              System.out.print("Sum: ");              c1.add(c2);              System.out.print("Difference: ");              c1.subtract(c2);              System.out.print("Product: ");              c1.multiply(c2);       }  }  class Complex{      int real, imaginary;      Complex(int real, int imaginary){          this.real = real;          this.imaginary = imaginary;      }      void add(Complex c){          System.out.println((this.real+c.real)+" + "+(this.imaginary+c.imaginary)+"i");      }      void subtract(Complex c){          System.out.println((this.real-c.real)+" + "+(this.imaginary-c.imaginary)+"i");      }      void multiply(Complex c){          System.out.println((this.real\*c.real - this.imaginary\*c.imaginary)+" + "+(this.real\*c.imaginary + this.imaginary\*c.real)+"i");      }  }  **OUTPUT:**    **CONCLUSION:**  The program defines a Complex class that represents complex numbers. It has methods for calculating the sum, difference, and product of two complex numbers. The user is prompted to input the real and imaginary parts of two complex numbers. The program creates instances of the Complex class for both input numbers and computes the requested operations. |

**Part – IV**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **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 :**  import java.util.\*;  class Parent {      void displayParent() {          System.out.println("This is parent class");      }  }  class Child extends Parent {      void displayChild() {          System.out.println("This is child class");      }  }  public class prac17 {      public static void main(String[] args) {          Parent parentObj = new Parent();          parentObj.displayParent();          Child childObj = new Child();          childObj.displayChild();          childObj.displayParent();      }  }  **OUTPUT:**    **CONCLUSION:**  This code demonstrates basic inheritance in Java, where the ChildClass inherits from the ParentClass. The ChildClass can call both its own method printChild() and the inherited method printParent() from the ParentClass. The code outputs messages from both classes and showcases polymorphism by calling methods of both parent and child classes. |
| **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 :**  import java.util.\*;  public class prac18 {      public static void main(String[] args) {          Employee E = new Employee();          E.name = "Samarth Patel";          E.age = 18;          E.mobile\_number = 1551558625;          E.Address = "maheshana,India";          E.Salary = 50000;          E.Specialization = "Software Development";          Manager M = new Manager();          M.name = "Rudra Patel";          M.age = 18;          M.mobile\_number = 1551558625;          M.Address = "Surat,India";          M.Salary = 50000;          M.Department = "Software Development";          System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");          E.displayEmployeeDetails();          System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");          M.displayManagerDetails();      }  }  class Member {      String name;      int age;      int mobile\_number;      String Address;      double Salary;      void printSalary() {          System.out.println("Salary: " + Salary);      }  }  class Employee extends Member {      String Specialization;      void displayEmployeeDetails() {          System.out.println("Employee Details:");          System.out.println("Name: " + name);          System.out.println("Age: " + age);          System.out.println("Phone Number: " + mobile\_number);          System.out.println("Address: " + Address);          System.out.println("Specialization: " + Specialization);          printSalary();      }  }  class Manager extends Member {      String Department;      void displayManagerDetails() {          System.out.println("Manager Details:");          System.out.println("Name: " + name);          System.out.println("Age: " + age);          System.out.println("Phone Number: " + mobile\_number);          System.out.println("Address: " + Address);          System.out.println("Specialization: " + Department);          printSalary();      }  }  **OUTPUT:**    **CONCLUSION:**  This code demonstrates inheritance in Java where both Employee and Manager classes inherit common attributes and methods from the Member class. It captures and displays details specific to both employees and managers, including specialization for employees and department for managers. The use of inheritance allows for shared functionality, such as printing salaries, while enabling unique attributes for each subclass. |
| **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 :**  public class practical\_19 {      public static void main(String[] args) {          Rectangle[] rectangles = new Rectangle[2];          rectangles[0] = new Rectangle(4, 11);          rectangles[1] = new Square(6);            for (Rectangle shape : rectangles) {              System.out.println("Area: " + shape.area());              System.out.println("Perimeter: " + shape.perimeter());          }      }  }  class Rectangle {      int length;      int breadth;        Rectangle(int l, int b) {          length = l;          breadth = b;      }      int area() {          return length \* breadth;      }        int perimeter() {          return 2 \* (length + breadth);      }  }  class Square extends Rectangle {      Square(int s) {          super(s, s);      }  }  **OUTPUT:**    **CONCLUSION:**  This code gives us the concept of inheritance and polymorphism in Java. The Square class inherits from the Rectangle class, as a square is a special case of a rectangle where the length and breadth are equal. The array of Rectangle objects includes both rectangles and squares, and through polymorphism, the program calculates and displays the area and perimeter for each shape using their respective implementations. |
| **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 printShape() {          System.out.println("This is a shape.");      }  }  class Rectangle extends Shape {      public void printRectangle() {          System.out.println("This is a rectangular shape.");      }  }  class Circle extends Shape {      public void printCircle() {          System.out.println("This is a circular shape.");      }  }  class Square extends Rectangle {      public void printSquare() {          System.out.println("Square is a rectangle.");      }  }  public class prac20 {      public static void main(String[] args) {          Square mySquare = new Square();          mySquare.printShape();          mySquare.printRectangle();      }  }  **OUTPUT:**    **CONCLUSION:**  This code demonstrates multi-level inheritance in Java. The Square class extends Rectangle, which in turn extends the base class Shape. The Square class inherits methods from both Shape and Rectangle, and it has its own method print4(). The code showcases how different classes represent shapes, with Square being treated as a specific type of rectangle. |
| **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 {      public void getDegree() {          System.out.println("I got a degree.");      }  }  class Undergraduate extends Degree {      public void getDegree() {          System.out.println("I am an Undergraduate.");      }  }  class Postgraduate extends Degree {      public void getDegree() {          System.out.println("I am a Postgraduate.");      }  }  public class prac21 {      public static void main(String[] args) {          Degree degree = new Degree();          Undergraduate undergrad = new Undergraduate();          Postgraduate postgrad = new Postgraduate();          degree.getDegree();          undergrad.getDegree();          postgrad.getDegree();      }  }  **OUTPUT:**    **CONCLUSION:**  This code demonstrates method overriding in Java, where the Undergraduate and Postgraduate classes override the getDegree() method of the Degree class to provide specific outputs. Each class prints a message related to the type of degree it represents. This showcases how subclasses can customize inherited behavior by overriding methods from a parent class. |
| **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 ost 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 prac22 {      public static void main(String[] args) {          MyCalculator my\_calculator = new MyCalculator();          System.out.print("I implemented: ");          int n = 6;          System.out.println(my\_calculator.divisor\_sum(n));      }  }  **OUTPUT:**    **CONCLUSION:**  This code demonstrates the implementation of an interface in Java. The AdvanceArithmetic interface defines a method divisor\_sum(int n), which is implemented by the MyCalculator class. The program calculates and returns the sum of all divisors of a given number, showcasing how interfaces can be used to define a contract for classes to implement specific functionalities. |
| **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 :**  import java.util.\*;  interface Shape {      double area();      default void displayDetails() {          System.out.println("This is a shape.");      }  }  class Circle implements Shape {      private double radius;      private String color;      public Circle(double radius, String color) {          this.radius = radius;          this.color = color;      }      public double area() {          return Math.PI \* radius \* radius;      }      public void displayDetails() {          System.out.println("This is a circle with radius " + radius + " and color " + color);      }  }  class Rectangle implements Shape {      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 area() {          return length \* width;      }      public void displayDetails() {          System.out.println("This is 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 displaySign() {          shape.displayDetails();          System.out.println("Text: " + text);      }  }  public class prac23 {      public static void main(String[] args) {          Circle circle = new Circle(5.0, "Red");          Rectangle rectangle = new Rectangle(4.0, 6.0, "Blue");          Sign circleSign = new Sign(circle, "Hello, World!");          Sign rectangleSign = new Sign(rectangle, "Welcome to the Campus Center!");          circleSign.displaySign();          rectangleSign.displaySign();      }  }  **OUTPUT:**    **CONCLUSION:**  This code demonstrates the use of interfaces, composition, and default methods in Java. The shape interface defines the methods color() and area(), and a default method info(), which are implemented by the Circle and Rectangle classes. The sign class uses composition to associate a shape with text, and its display() method prints both the sign's text and shape information. |

**Part – IV**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **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 prac24 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          try {              System.out.print("Enter the value of X :");              int x = sc.nextInt();              System.out.print("Enter the value of Y :");              int y = sc.nextInt();              int result = x / y;              System.out.println("Result : " + result);          } catch (Exception e) {              if (e instanceof ArithmeticException) {                  System.out.println("Error: Divison by 0 is not possible.");              } else {                  System.out.println("Please Enter valid integer.");              }          }      }  }  **OUTPUT:**    **CONCLUSION:**  The program prompts the user for two integers, performs division, and displays the result. If the denominator is zero, it handles the ArithmeticException and provides an appropriate error message. Otherwise, it ensures valid input by catching other exceptions. |
| **25** | Write a Java program that throws an exception and catch it using a try-catch block.  **PROGRAM CODE :**  public class prac25  {      public static void main(String[] args) {          try {              int[] numbers = { 1, 2, 3, 4, 5 };                System.out.println(numbers[5]);          } catch (ArrayIndexOutOfBoundsException e) {              System.out.println("Error: Array index out of bounds!");              System.out.println("Exception Message: " + e.getMessage());          }      }  }  **OUTPUT:**    **CONCLUSION:**  Java program demonstrates handling an ArrayIndexOutOfBoundsException by catching the exception and providing an error message. In just a few lines, it showcases proper exception handling. |
| **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 :**  (1)  class MyException extends Exception {      public MyException() {          System.out.println("Exepction created by user");      }  }  public class prac26\_1 {      static void checkValue(int value) throws MyException {          if (value > 10) {              throw new MyException();          }          System.out.println("Value is acceptable: " + value);      }      public static void main(String[] args) {          try {              checkValue(5);              checkValue(15);          } catch (MyException e) {              System.out.println("Caught exception");          }      }  }  (2)  public class prac26\_2 {      public static void CheckedException() throws Exception {          throw new Exception("This is a checked exception.");      }      public static void UncheckedException() {          int result = 10 / 0;      }      public static void main(String[] args) {          try {              CheckedException();          } catch (Exception e) {              System.out.println("Caught checked exception: " + e.getMessage());          }          try {              UncheckedException();          } catch (ArithmeticException e) {              System.out.println("Caught unchecked exception: " + e.getMessage());          }      }  }  **OUTPUT:**      **CONCLUSION:**  Custom exceptions improve error handling by defining specific conditions, while checked and unchecked exceptions handle compile-time and runtime errors. Proper use of try-catch blocks ensures robust and fault-tolerant programs. |

**Part – VI**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **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.\*;  public class prac27 {      public static void main(String[] args) throws Exception {          if (args.length == 0) {              System.out.println("No file Found!");          } else {              for (int i = 0; i < args.length; i++) {                  try {                      BufferedReader f = new BufferedReader(new FileReader(args[i]));                      String j;                      int count = 0;                      while ((j = f.readLine()) != null) {                          count++;                      }                      System.out.println("File name is : " + args[i] + " and Number of lines are : " + count);                  } catch (Exception e) {                      System.out.println(e);                  }              }          }      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program reads several files named by the command line arguments and counts the number of lines in each. If no files are provided as command-line arguments, it will print out the appropriate message. Exception handling ensures graceful error management during file reading, thus a stable program. |
| **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 P28{      public static void main(String[] args) {          if (args.length < 2) {              System.out.println("Usage: java P28 <character> <filename>");              return;          }          char targetChar = args[0].charAt(0);          String fileName = args[1];          int count = 0;          try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {              int ch;              while ((ch = reader.read()) != -1) {                  if (ch == targetChar) {                      count++;                  }              }              System.out.println("The character '" + targetChar + "' appears " + count + " times in " + fileName);          } catch (IOException e) {              System.out.println("Error reading " + fileName + ": " + e.getMessage());          }      }  }  **OUTPUT:**    **CONCLUSION:**  The Java program successfully counts the occurrences of a specified character in a given file, providing the result in a clear format. It handles file read errors gracefully, ensuring robust performance even if issues arise during file access. |
| **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;  public class P29 {  public static void main(String[] args) {  if (args.length < 2) {  System.out.println("Usage: java P29 <word> <filename>");  return;  }  String searchWord = args[0];  String fileName = args[1];  Integer count = 0;  try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  String line;  while ((line = reader.readLine()) != null) {  String[] words = line.split("\\W+");  for (String word : words) {  if (word.equalsIgnoreCase(searchWord)) {  count++;  }  }  }  System.out.println("The word '" + searchWord + "' appears " + count + " times in " + fileName);  } catch (IOException e) {  System.out.println("Error reading " + fileName + ": " + e.getMessage());  }  }  }  **OUTPUT:**    **CONCLUSION:**  This Java program effectively searches for a specified word in a given file and counts its occurrences. It demonstrates the use of the Integer wrapper class to manage the count, showcasing how wrapper classes can be used for object manipulation in Java. |
| **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.FileReader;  import java.io.FileWriter;  import java.io.IOException;  public class P30 {      public static void main(String[] args) {          if (args.length < 2) {              System.out.println("Usage: java P30 <source file> <destination file>");              return;          }          String sourceFile = args[0];          String destinationFile = args[1];          try (FileReader fr = new FileReader(sourceFile);                  FileWriter fw = new FileWriter(destinationFile)) {              int ch;              while ((ch = fr.read()) != -1) {                  fw.write(ch);              }              System.out.println("Data copied from " + sourceFile + " to " + destinationFile);          } catch (IOException e) {              System.out.println("Error: " + e.getMessage());          }      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program efficiently copies data from a source file to a destination file, automatically creating the destination file if it does not already exist. It handles any potential I/O exceptions during the process, ensuring robust performance. |
| **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.\*;  public class P31 {      public static void main(String[] args) {          BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));          String fileName = "output.txt";          try (BufferedWriter fileWriter = new BufferedWriter(new FileWriter(fileName))) {              System.out.println("Enter text (type 'exit' to finish):");              String input;              while (!(input = consoleReader.readLine()).equalsIgnoreCase("exit")) {                  fileWriter.write(input);                  fileWriter.newLine();              }              System.out.println("Data written to " + fileName);          } catch (IOException e) {              System.out.println("Error: " + e.getMessage());          }      }  }  **OUTPUT:**    **CONCLUSION:**  This program effectively demonstrates the use of character streams via BufferedReader and BufferedWriter for reading console input and writing it to a file. It showcases how to handle text data efficiently while managing resources properly with try-with-resources. |

**Part – VII**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **32.** | Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.  **PROGRAM CODE :**  public class pract32 {      public static void main(String[] args) {          Thread1 t1 = new Thread1();          t1.start();          Thread t2 = new Thread(new Thread2());          t2.start();      }  }  class Thread2 implements Runnable {      public void run() {          System.out.println("Hello World");      }  }  class Thread1 extends Thread {      public void run() {          System.out.println("Hello World");      }  }  **OUTPUT:**    **CONCLUSION:**  The Java code demonstrates two approaches to creating threads: extending the Thread class and implementing the Runnable interface. It starts a thread that prints "Hello world" and another that prints "hello world runnable interface." This example illustrates the flexibility of thread creation in Java and showcases basic multithreading concepts. |
| **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 :**  import java.util.\*;  public class prac33 {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);            System.out.print("Enter the number N (total numbers to sum): ");          int N = scanner.nextInt();            System.out.print("Enter the number of threads: ");          int numThreads = scanner.nextInt();            int sum = 0;          Thread[] threads = new Thread[numThreads];          Summation.sum = new int[numThreads];            for (int i = 0; i < numThreads; i++) {              threads[i] = new Thread(new Summation(N, i, numThreads));              threads[i].start();          }          for (int i = 0; i < numThreads; i++) {              try {                  threads[i].join();              } catch (InterruptedException e) {                  e.printStackTrace();              }          }            for (int i = 0; i < numThreads; i++) {              sum += Summation.sum[i];          }            System.out.println("Sum: " + sum);      }  }  class Summation implements Runnable {      static int[] sum;      int N, start, numThreads;        Summation(int N, int start, int numThreads) {          this.N = N;          this.start = start;          this.numThreads = numThreads;      }      public void run() {          for (int i = start + 1; i <= N; i += numThreads) {              sum[start] += i;          }      }  }  **OUTPUT:**    **CONCLUSION:**  The Java code efficiently calculates the sum of the first N natural numbers using multiple threads. Each thread contributes to the total by summing a portion of the range, and the results are aggregated at the end. This example highlights the use of multithreading for parallel computation and demonstrates thread management in Java. |
| **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 :**  public class prac34 {      public static void main(String[] args) {          Thread1 t1 = new Thread1();          t1.start();      }  }  class Thread1 extends Thread {      public void run() {          while (true) {              int n = (int) (Math.random() \* 100);              System.out.println("Generated number: " + n);              if (n % 2 == 0) {                    new Thread2(n).start();              } else {                    new Thread3(n).start();              }              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  e.printStackTrace();              }          }      }  }  class Thread2 extends Thread {      private int n;      Thread2(int n) {          this.n = n;          setName("EvenThread");      }      public void run() {          System.out.println(getName() + ": Square of " + n + " is " + (n \* n));      }  }  class Thread3 extends Thread {      private int n;      Thread3(int n) {          this.n = n;          setName("OddThread");      }      public void run() {          System.out.println(getName() + ": Cube of " + n + " is " + (n \* n \* n));      }  }  **OUTPUT:**    **CONCLUSION:**  The Java code implements a multithreaded application that generates random numbers and spawns new threads based on whether the number is even or odd. Even numbers trigger the creation of a thread that calculates and prints their square, while odd numbers create a thread that computes their cube. This example effectively demonstrates dynamic thread creation and the handling of different tasks using Java's threading capabilities. |
| **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 :**  public class prac35 {      public static void main(String[] args) {          Thread1 t1 = new Thread1();          t1.start();      }  }  class Thread1 extends Thread {      public void run() {          int n = 0;          while (true) {              n++;              System.out.println(n);              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                   System.out.println("Thread interrupted.");              }          }      }  }  **OUTPUT:**    **CONCLUSION:**  The Java code creates a simple multithreaded application that continuously increments and displays a variable every second. By extending the Thread class, it demonstrates basic thread functionality, including looping and sleeping. This example effectively illustrates the fundamental principles of thread execution and timing in Java. |
| **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 :**  public class prac36 {      public static void main(String[] args) {          Thread first = new Thread(new Runnable() {              public void run() {                  System.out.println(Thread.currentThread().getName() + " Priority: " + Thread.currentThread().getPriority());              }          });            Thread second = new Thread(new Runnable() {              public void run() {                  System.out.println(Thread.currentThread().getName() + " Priority: " + Thread.currentThread().getPriority());              }          });            Thread third = new Thread(new Runnable() {              public void run() {                  System.out.println(Thread.currentThread().getName() + " Priority: " + Thread.currentThread().getPriority());              }          });          first.setPriority(3);          second.setPriority(5);          third.setPriority(7);          first.setName("FIRST");          second.setName("SECOND");          third.setName("THIRD");          first.start();          second.start();          third.start();      }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java code demonstrates the creation and management of threads by extending the Thread class. It showcases setting custom priorities for three threads and starts their execution, which will display their names in the console. This example highlights basic threading concepts, including priority management in Java. |
| **37.** | Write a program to solve producer-consumer problem using thread synchronization  **PROGRAM CODE :**  public class prac37 {      public static void main(String[] args) {          Buffer buffer = new Buffer();          Producer producer = new Producer(buffer);          Consumer consumer = new Consumer(buffer);          producer.start();          consumer.start();      }  }  class Buffer {      private int[] buffer;      private int size;      private int count;      Buffer() {          size = 5;          buffer = new int[size];          count = 0;      }      public synchronized void produce(int item) {          while (count == size) {              try {                  wait();              } catch (InterruptedException e) {                  e.printStackTrace();              }          }          buffer[count] = item;          count++;          System.out.println("Produced: " + item);          notify();      }      public synchronized int consume() {          while (count == 0) {              try {                  wait();              } catch (InterruptedException e) {                  e.printStackTrace();              }          }          int item = buffer[count - 1];          count--;          System.out.println("Consumed: " + item);          notify();          return item;      }  }  class Producer extends Thread {      private Buffer buffer;      Producer(Buffer buffer) {          this.buffer = buffer;      }      public void run() {          for (int i = 0; i < 10; i++) {              buffer.produce(i);          }      }  }  class Consumer extends Thread {      private Buffer buffer;      Consumer(Buffer buffer) {          this.buffer = buffer;      }      public void run() {          for (int i = 0; i < 10; i++) {              buffer.consume();          }      }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates the classic Producer-Consumer problem using multithreading and synchronization in Java. The producer generates items and adds them to a shared buffer, while the consumer removes and processes them. Synchronization ensures proper coordination between the  threads. |

**Part – VIII**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **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.ArrayList;  class MyStack {  private ArrayList<Object> list = new ArrayList<>();  public boolean isEmpty() {  return list.isEmpty();  }  public int getSize() {  return list.size();  }  public Object peek() {  if (isEmpty()) {  return "Stack is empty";  }  return list.get(list.size() - 1);  }  public Object pop() {  if (isEmpty()) {  return "Stack is empty";  }  return list.remove(list.size() - 1);  }  public void push(Object o) {  list.add(o);  } }  public class P38 {  public static void main(String[] args) {  MyStack stack = new MyStack();  stack.push(10);  stack.push(20);  stack.push(30);  System.out.println("Top element is: " + stack.peek());  System.out.println("Popped element: " + stack.pop());  System.out.println("Popped element: " + stack.pop());  System.out.println("Is stack empty ? " + stack.isEmpty());  System.out.println("Current stack size: " + stack.getSize());  System.out.println("Top element now: " + stack.peek());  } }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates the implementation of a custom stack using the ArrayList class in Java. It provides functionalities to push, pop, peek, check if the stack is empty, and get the current size of the stack. The program effectively showcases how to manage a dynamic collection of elements while adhering to stack principles. |
| **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 :**  import java.util.Arrays;  public class P39 {  public static <T extends Comparable<T>> void sortArray(T[] array) {  Arrays.sort(array);  }  public static void main(String[] args) {  Integer[] numbers = {5, 3, 9, 1, 7};  System.out.println("Before sorting (Integers): " + Arrays.toString(numbers));  sortArray(numbers);  System.out.println("After sorting (Integers): " + Arrays.toString(numbers));  String[] names = {"John", "Alice", "Bob", "David"};  System.out.println("\nBefore sorting (Strings): " + Arrays.toString(names));  sortArray(names);  System.out.println("After sorting (Strings): " + Arrays.toString(names));  Product[] products = {  new Product("Laptop", 1000),  new Product("Phone", 800),  new Product("Tablet", 600),  new Product("Smartwatch", 200)  };  System.out.println("\nBefore sorting (Products by price): ");  for (Product p : products) {  System.out.println(p);  }  sortArray(products);  System.out.println("\nAfter sorting (Products by price): ");  for (Product p : products) {  System.out.println(p);  } } }  class Product implements Comparable<Product> {  private String name;  private int price;  public Product(String name, int price) {  this.name = name;  this.price = price;  }  @Override  public int compareTo(Product other) {  return this.price - other.price;  }  @Override  public String toString() {  return name + ": $" + price;  } }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates the use of generics in Java to create a versatile sorting method for arrays of different types. By implementing the Comparable interface in the Product class, it enables sorting of custom objects based on specific criteria, such as price. The output shows the effective sorting of integers, strings, and products, highlighting the flexibility and reusability of the generic sorting method. |
| **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 P40 {  public static void main(String[] args) {  Map<String, Integer> wordMap = new TreeMap<>();  Scanner scanner = new Scanner(System.in);  System.out.println("Enter a text:");  String text = scanner.nextLine();  String[] words = text.toLowerCase().split("\\W+");  for (String word : words) {  if (!word.isEmpty()) {  wordMap.put(word, wordMap.getOrDefault(word, 0) + 1);  } }  System.out.println("\nWord Occurrences (in alphabetical order):");  Set<Map.Entry<String, Integer>> entrySet = wordMap.entrySet();  for (Map.Entry<String, Integer> entry : entrySet) {  System.out.println(entry.getKey() + ": " + entry.getValue());  } } }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to count and display the occurrences of words in a given text using Java's Map and Set classes. The words are stored in a TreeMap, ensuring that they are presented in alphabetical order. The use of getOrDefault() simplifies the counting process, showcasing efficient word frequency analysis. |
| **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.\*;   import java.util.\*;   public class P41 {  private static final HashSet<String> keywords = new HashSet<>();  static {  String[] keywordArray = {  "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class",  "const", "continue", "default", "do", "double", "else", "enum", "extends", "final",  "finally", "float", "for", "goto", "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",  };  for (String keyword : keywordArray) {  keywords.add(keyword);  } }  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter the path of the Java source file: ");  String filePath = scanner.nextLine();  try {  File file = new File(filePath);  Scanner fileScanner = new Scanner(file);  int keywordCount = 0;  while (fileScanner.hasNext()) {  String word = fileScanner.next();  if (keywords.contains(word)) {  keywordCount++;  } }  System.out.println("Number of Java keywords in the file: " + keywordCount);  fileScanner.close();  } catch (FileNotFoundException e) {  System.out.println("File not found: " + filePath);  } } }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates the use of a HashSet to efficiently count Java keywords in a source file. By reading each word from the file and checking for its presence in the set of keywords, it showcases how to utilize collections for rapid lookups. The result is the total number of keywords, providing a simple yet effective tool for analyzing Java code. |