A close-up of a certificate

Description automatically generated

**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**

**SET - 1**

|  |  |
| --- | --- |
| **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  - Class: Blueprint for objects.  - Object: Instance of a class.  - 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 $20. Write a java program to store this balance in a variable and then display it to the user.  **PROGRAM CODE :**  class practical\_2 {  public static void main(String[] args) {  int a = 20;  System.err.print("Current balance is :$");  System.err.println(a);  System.out.println("23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The code effectively demonstrates how to use both System.err and System.out for output in Java. It first outputs a message and a variable to the standard error stream, then prints a student identifier to the standard output stream. This showcases the ability to handle different output streams for different purposes within a simple 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.Scanner;  public class practical\_3 {  public static void main(String[] args) {  Scanner a = new Scanner(System.in);  System.out.print("Enter the Distsnce in meters :");  float Distsnce = a.nextFloat();  System.out.print("Enter the Time in (hour) :");  float hour = a.nextFloat();  System.out.print("Enter the Time in (minutes) :");  float minutes = a.nextFloat();  System.out.print("Enter the Time in (seconds) :");  float seconds = a.nextFloat();  float totaltime = seconds + (60 \* minutes) + (3600 \* hour);  float speed1 = Distsnce / totaltime;  float speed2 = (Distsnce / 1000) / (totaltime / 3600);  float speed3 = (Distsnce / 1609) / (totaltime / 3600);  System.out.println("Speed 1 " + speed1);  System.out.println("Speed 2 " + speed2);  System.out.println("Speed 3 " + speed3);  System.out.println("23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java code calculates the speed of an object based on the distance traveled in meters and the time taken, which is input in hours, minutes, and seconds. It converts the total time into seconds and then calculates the speed in three different units: meters per second, kilometers per hour, and miles per hour. The calculated speeds are printed alongside a student identifier "23DCS092\_Shubh". This program effectively demonstrates basic arithmetic operations, user input handling, and unit conversion in Java. |
| **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 practical\_4 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int n;  int[] ex = new int[100];  float sum = 0;  System.out.println("Enter the number of Days's Expenses: ");  n = sc.nextInt();  for (int i = 0; i < n; i++)  {  ex[i] = sc.nextInt();  sum = sum + ex[i];  }  System.out.println("Total expenses of 30 Days is " + sum);  System.out.println("23DCS092\_Shubh");    }  }  **OUTPUT:**    **CONCLUSION:**  The given Java code is a simple program designed to calculate the total expenses over a specified number of days. The program starts by importing the Scanner class to facilitate user input and defining the main class practical\_4. Within the main method, a Scanner object is instantiated, and an integer array ex of size 100 is declared to store the daily expenses. A float variable sum is initialized to keep a running total of these expenses. |
| **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.Scanner;  class practical\_5{  public static void main(String[] args) {  int a;    int price[] = {200,550,100,170,220};  int itemcode[] = {1, 2, 3, 4, 5};  System.out.println("1-Motor\n2-Fan\n3-Tube\n4-wires\n5-All other item");  System.out.print("Enter your choice :");  Scanner b = new Scanner(System.in);  a = b.nextInt();  switch (a) {  case 1:System.out.println("The price of the Motor is :"+(price[0]+price[0]\*0.08f));  break;  case 2:System.out.println("The price of the Fan is :"+(price[1]+price[1]\*0.12f));  break;  case 3:System.out.println("The price of the Tubes is :"+(price[2]+price[2]\*0.075f));  break;  case 4:System.out.println("The price of the wires is :"+(price[3]+price[3]\*0.03f));  break;  default:  System.out.println("Invalid choice");  }  System.out.println("23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The code demonstrates the use of arrays, the Scanner class for input, and a switch statement for decision-making in Java. It effectively simulates a simple pricing system where the user can choose an item, and the program calculates and displays the final price after adding a specified tax. This code showcases basic programming concepts such as array manipulation, user input handling, and conditional logic. |
| **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.\*;  class practical\_6 {  public static void main(String[] args) {  int a;  int m=0,n=1;  System.out.println("enter the number of days (n) for you want to generate their exercise routine.");  a=sc.nextInt();    for (int i = 0; i < a; i++) {  System.out.print(" "+ m);  int sum = m + n;  m = n;  n = sum;  }  System.out.println("\n23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The code demonstrates how to generate and print the first 10 numbers of the Fibonacci sequence. It initializes the sequence with 0 and 1, then uses a loop to calculate and print each subsequent number. This showcases the use of loops, arithmetic operations, and variable updates in Java to implement a simple algorithm for generating a well-known mathematical sequence. The program effectively demonstrates the basic concept of the Fibonacci sequence and how it can be implemented programmatically. |

**SET - 2**

|  |  |
| --- | --- |
| **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 practical\_7 {  public static void main(String[] args) {  int a,b ;  Scanner sc = new Scanner(System.in);  System.out.print("Enter the number of words of String 1 :");  a = sc.nextInt();  String st1 = "Chocolate";  System.out.print("Enter the number of words of string 2:");  b = sc.nextInt();  String st2 = "Abc";  for (int i = 0; i < a;i++) {  System.out.print(st1.substring(0, 3));  }  System.out.println("");  for (int i = 0; i < b;i++) {  System.out.print(st2.substring(0, 3));  }  System.out.println("\n23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java program takes two integers as input, representing the number of times to repeat a substring from two predefined strings, "Chocolate" and "Abc." It then prints the first three characters of "Chocolate" and "Abc" the specified number of times. This program demonstrates basic string manipulation, user input handling, and iterative control structures in Java. |
| **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 :**  public class practical\_8 {  public static int arr\_count9(int[] arr) {  int count = 0;  for (int i = 0; i < arr.length; i++) {  if (arr[i] == 9) {  count++;  }  }  return count;  }  public static void main(String[] args) {  int[] arr1 = {1, 9, 5, 9, 4, 7, 9};  int[] arr2 = {1, 9, 5};  int[] arr3 = {1, 9, 5, 9, 4, 7};  System.out.println("Number 9's in arr1 :" + arr\_count9(arr1));  System.out.println("Number 9's in arr2 :" + arr\_count9(arr2));  System.out.println("Number 9's in arr3 :" + arr\_count9(arr3));  System.out.println("23DCS092\_Shubh”);  }  }  **OUTPUT:**    **CONCLUSION:**  The Java code defines a method arr\_count9 that counts the occurrences of the number 9 in an integer array. The main method creates three different arrays and uses the arr\_count9 method to count and print the number of 9s in each array. The results for each array are printed, demonstrating the functionality of the counting method. |
| **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 :**  public class practical\_9 {  public static void double\_char(String s) {  for (int i = 0; i < s.length(); i++) {  System.out.print(s.charAt(i));  System.out.print(s.charAt(i));  }  }  public static void main(String[] args) {  String st = "The";  String st1 = "AAbb";  String st2 = "Hi-there";  double\_char(st);  System.out.println();  double\_char(st1);  System.out.println();  double\_char(st2);  System.out.println("23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The Java code defines a method double\_char that takes a string and prints each character of the string twice consecutively. The main method calls this function with three different strings: "The", "AAbb", and "Hi-there", demonstrating the functionality of duplicating each character. After processing each string, it prints a newline for clarity. This code exemplifies character manipulation by iterating through each character in the string and printing |
| **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;  public class practical\_10{  public static void main(String[] args) {  String st = "My name is Shubh";  int length = st.length();  System.out.println("Length of the string is: " + length);  String uppercase = st.toUpperCase();  System.out.println("Uppercase of the string is: " + uppercase);  String lowercase = st.toLowerCase();  System.out.println("Lowercase of the string is: " + lowercase);  String reverse = "";  for(int i = st.length() - 1; i >= 0; i--)  {  reverse = reverse + st.charAt(i);  }  System.out.println("Reverse of the string is: " + reverse);  char[] charArray = st.toCharArray();  Arrays.sort(charArray);  String sortedString = new String(charArray);  System.out.println("Sorted string is: " + sortedString);  System.out.println("23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The Java code performs a series of string manipulations on the input "My name is Shubh". It calculates and prints the length of the string, then converts and prints the string in both uppercase and lowercase formats. The code also reverses the string and prints the reversed version. Additionally, it converts the string to a character array, sorts the array, and prints the sorted string. Finally, These operations illustrate various methods of string handling |
| **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 practical\_11{  public static void main(String[] args) {  String st = "CHARUSAT UNIVERSITY";  int length = st.length();  System.out.println("Length of the string is: " + length);  String replace = st.replace('H', 'S');  System.out.println("Replace of the string is: " + replace);  String lowercase = st.toLowerCase();  System.out.println("Lowercase of the string is: " + lowercase);  System.out.println("23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The Java code performs various string manipulations on the input "CHARUSAT UNIVERSITY". It first calculates and prints the length of the string. Then, it replaces the character 'H' with 'S' and prints the modified string. Next, the code converts the entire string to lowercase and prints the result. These operations demonstrate basic string handling techniques in Java. |

**SET - 3**

|  |  |
| --- | --- |
| **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:**  import java.util.\*;  public class p12{  public static void main(String[] args){  double pounds;  double rupees = 0;  Scanner sc = new Scanner(System.in);    if(args.length > 0 ){  pounds = Double.parseDouble(args[0]);  rupees = 100\*pounds;  System.out.println("The amount in pound converted into " + rupees " rupees.");  return;    }  else{  System.out.println("Enter the amount in pounds: ");  pounds = sc.nextDouble();  rupees = 100\*pounds;    System.out.println("The amount in rupees is: "+rupees);  }  System.out.println("23DCS092\_Shubh");  }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java program converts an amount in pounds to rupees, using a conversion rate of 100 rupees per pound. If the amount is given as a command-line argument, it directly converts and prints the result. If not, it prompts the user to enter the amount, performs the conversion, and then displays the result. |
| **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 :**  public class practical\_13 {  public static void main(String[] args) {  Employee em1 = new Employee("Shubh", "patel", 50000.0);  Employee em2 = new Employee("Samarth", "patel", 60000.0);  System.out.print("The name of the Employee 1 :");  em1.displayname();  System.out.println("Yearly salary of employee 1: " + em1.getYearlySalary());  System.out.print("The name of the Employee 2 :");  em2.displayname();  System.out.println("Yearly salary of employee 2: " + em2.getYearlySalary());  em1.giveRaise(10);  em2.giveRaise(10);  System.out.println("Yearly salary of employee 1 after raise: " + em1.getYearlySalary());  System.out.println("Yearly salary of employee 2 after raise: " + em2.getYearlySalary());  System.out.println("23DCS092\_Shubh");  }  }  class Employee {  String fname;  String lname;  double salary;  public Employee(String f, String l, double s) {  fname = f;  lname = l;  if (salary < 0) {  salary = 0.0;  } else {  salary = s;  }  }  double getYearlySalary() {  return salary \* 12;  }  void giveRaise(int percent) {  salary = salary + (salary \* percent / 100);  }  void displayname() {  System.out.println(fname + " " + lname);  }  void displaysalary() {  System.out.println("Salary: " + salary);  }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java program defines an Employee class with attributes for first name, last name, and salary, including methods to calculate yearly salary, give a raise, and display the employee's name. The practical\_13 class creates two Employee objects, prints their names and yearly salaries, gives them a 10% raise, and then prints their updated yearly salaries. |
| **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.\*;  class practical\_14 {  public static void main(String[] args) {  int d=0 , m=0 , y = 0;  date d1 = new date(d,m,y);  d1.getdata();  System.err.println("Initial date :");  d1.display();    System.err.println("Enter the New date :");  d1.update();  System.err.println("Updated date :");  d1.display();  System.out.println("23DCS092\_Shubh");  }  }  class date{  int day, month, year;  date(int day, int month, int year){  this.day = day;  this.month = month;  this.year = year;  }  void getdata(){  Scanner sc = new Scanner(System.in);  System.out.println("Enter the date: ");  this.day = sc.nextInt();  System.out.println("Enter the month: ");  this.month = sc.nextInt();  System.out.println("Enter the year: ");  this.year = sc.nextInt();  }  void update(){  Scanner sc = new Scanner(System.in);  System.out.println("Enter the date: ");  this.day = sc.nextInt();  System.out.println("Enter the month: ");  this.month = sc.nextInt();  System.out.println("Enter the year: ");  this.year = sc.nextInt();  }  void display(){  System.out.println("Date: "+day+"/"+month+"/"+year);  }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java program consists of a date class that stores day, month, and year, with methods to get data from the user and display the date. In the practical\_14 class, an instance of date is created with initial values, the user is prompted to enter a date, and the entered date is displayed. |
| **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.Scanner;  public class practical\_15 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("Enter length of rectangle: ");  int l = sc.nextInt();  System.out.println("Enter breadth of rectangle: ");  int b = sc.nextInt();  area a = new area(l,b);  System.out.println("Area of rectangle is: " + a.returnArea());  System.out.println("23DCS092\_Shubh");  }  }  class area {  int length;  int breadth;  public area(int l, int b) {  length = l;  breadth = b;  }  public int returnArea() {  return length \* breadth;  }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java code calculates the area of a rectangle using user input for the length and breadth. It employs a Scanner for input collection and defines a class area to encapsulate the rectangle's dimensions and area computation. The main method initializes an area object with the input values and prints the calculated 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.Scanner;  public class practical\_16 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("Enter the real part :");  int r = sc.nextInt();  System.out.println("Enter the imaginary part :");  int im = sc.nextInt();  complex c1 = new complex(r,im);  c1.putdata();  System.out.println("Enter the real part :");  r = sc.nextInt();  System.out.println("Enter the imaginary part :");  im = sc.nextInt();  complex c2 = new complex(r,im);  c2.putdata();  c1.sum(c2);  c1.difference(c2);  c1.product(c2);  System.out.println("23DCS092\_Shubh");  }  }  class complex {  int real;  int imag;  complex(int r, int im) {  real = r;  imag = im;  }  void putdata()  {  System.out.println(real + " + " + imag + "i");  }  void sum(complex c2)  {  int sumreal = real + c2.real;  int sumimag = imag + c2.imag;  System.out.println("The sum of two Complex number :" + sumreal + " + " + sumimag + "i");  }  void difference(complex c2)  {  int difreal = real - c2.real;  int difimag = imag - c2.imag;  System.out.println("The Difference of two Complex number : (" + difreal + ") + ("+ difimag + ") i");  }  void product(complex c2)  {  int prodreal = (real \* c2.real) - (imag \* c2.imag);  int prodimag = (real \* c2.imag) + (imag \* c2.real);  System.out.println("The Product of two Complex number : (" + prodreal + ") + ("+ prodimag + ") i");  }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java program defines a complex class to represent and manipulate complex numbers. It includes methods to compute and display the sum, difference, and product of two complex numbers. The main method takes user input for two complex numbers and demonstrates these operations, showcasing the functionality of the complex class. |

**SET - 4**

|  |  |
| --- | --- |
| **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:**  public class practical\_17 {      public static void main(String[] args) {          ParentClass parent = new ParentClass();          ChildClass child = new ChildClass();          parent.printParent();          child.printParent();          child.printChild();          System.out.println("23DCS092\_Shubh");      }    }  class ParentClass {      public void printParent() {          System.out.println("This is parent class");      }  }  class ChildClass extends ParentClass{      public void printChild() {          System.out.println("This is child class");      }  }  **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 practical\_18 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          Employee employee = new Employee();          Manager manager = new Manager();            System.out.println("Enter employee details:");          System.out.print("Name: ");          employee.name = sc.nextLine();          System.out.print("Age: ");          employee.age = sc.nextInt();          System.out.print("Phone: ");          employee.phone = sc.nextLong();          sc.nextLine();          System.out.print("Address: ");          employee.address = sc.nextLine();          System.out.print("Salary: ");          employee.salary = sc.nextDouble();          sc.nextLine();          System.out.print("Specialization: ");          employee.specialization = sc.nextLine();          System.out.println("\nEnter manager details:");          System.out.print("Name: ");          manager.name = sc.nextLine();          System.out.print("Age: ");          manager.age = sc.nextInt();          System.out.print("Phone: ");          manager.phone = sc.nextLong();          sc.nextLine();          System.out.print("Address: ");          manager.address = sc.nextLine();          System.out.print("Salary: ");          manager.salary = sc.nextDouble();          sc.nextLine();          System.out.print("Department: ");          manager.department = sc.nextLine();          System.out.println();          System.out.println("Employee Details:");          System.out.println("Name: " + employee.getname());          System.out.println("Age: " + employee.getage());          System.out.println("Phone: " + employee.getphone());          System.out.println("Address: " + employee.getaddress());          employee.printSalary();          System.out.println("Specialization: " + employee.getSpecialization());          System.out.println("\nManager Details:");          System.out.println("Name: " + manager.getname());          System.out.println("Age: " + manager.getage());          System.out.println("Phone: " + manager.getphone());          System.out.println("Address: " + manager.getaddress());          manager.printSalary();          System.out.println("Department: " + manager.getDepartment());          System.out.println("23DCS092\_Shubh");      }    }  class Member{    String name;      int age;      long phone;      String address;      double salary;      void printSalary() {          System.out.println("Salary: " + salary);      }      String getname(){          return name;      }      int getage(){          return age;      }      long getphone(){          return phone;      }      String getaddress(){          return address;      }      double getsalary(){          return salary;      }  }  class Employee extends Member{      String specialization;      String getSpecialization(){          return specialization;      }  }  class Manager extends Member{      String department;      String getDepartment(){          return department;      }  }  **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(5, 10);          rectangles[1] = new Square(5);            for (Rectangle shape : rectangles) {              System.out.println("Area: " + shape.area());              System.out.println("Perimeter: " + shape.perimeter());          }            System.out.println("23DCS092\_Shubh");      }  }  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 :**  public class practical\_20 {      public static void main(String[] args) {          Square s1 = new Square();          s1.print1();          s1.print2();          s1.print4();          System.out.println("23DCS092\_Shubh");      }  }  class Shape {      void print1() {          System.out.println("This is shape");      }  }  class Rectangle extends Shape {      void print2() {          System.out.println("This is rectangular Shape.");      }  }  class Circle extends Shape {      void print3() {          System.out.println("This is circular Shape.");      }  }  class Square extends Rectangle {      void print4() {          System.out.println("Square is a rectangle.");      }  }  **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 :**   public class practical\_21 {      public static void main(String[] args) {          Degree d = new Degree();          d.getDegree();          Undergraduate ug = new Undergraduate();          ug.getDegree();          Postgraduate pg = new Postgraduate();          pg.getDegree();          System.out.println("23DCS092\_Shubh");      }  }  class Degree{      void getDegree(){          System.out.println("I got 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 a Postgraduate");      }  }  **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 most 1000.  **PROGRAM CODE :**  import java.util.Scanner;  public class practical\_22 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          MyCalculator my\_calculator = new MyCalculator();          System.out.print("Enter the number for divisor: ");          int divisor = sc.nextInt();          int x = my\_calculator.divisor\_sum(divisor);          System.out.println("The sum of Divisor :" + x);          System.out.println("23DCS092\_Shubh");      }  }  interface AdvanceAerithmetic {      int divisor\_sum(int n);  }  class MyCalculator implements AdvanceAerithmetic {      public int divisor\_sum(int n) {          int sum = 0;          for (int i = 1; i <= n; i++) {              if (n % i == 0) {                  sum += i;              }          }          return sum;      }  }  **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 :**   public class practical\_23 {      public static void main(String[] args) {          shape circle = new Circle(10, "black");          shape rectangle = new Rectangle(20, 30, "red");          sign cirSign = new sign(circle, "welcome to campus");          sign recSign = new sign(rectangle, "welcome!");          cirSign.display();          recSign.display();          System.out.println("23DCS092\_Shubh");      }  }  interface shape {      String color();      int area();      default String info() {          return "This is a shape with color " + color() + " area " + area();      }  }  class Circle implements shape {      private int radius;      private String color;      public Circle(int radius, String color) {          this.radius = radius;          this.color = color;      }      public String color() {          return color;      }      public int area() {          return (int) (3.14 \* radius \* radius);      }  }  class Rectangle implements shape {      private int length;      private int width;      private String color;      public Rectangle(int length, int width, String color) {          this.length = length;          this.width = width;          this.color = color;      }      public String color() {          return color;      }      public int area() {          return length \* width;      }  }  class sign {      private shape shape;      private String text;      public sign(shape shape, String text) {          this.shape = shape;          this.text = text;      }      void display() {          System.out.println("Sign text: " + text);          System.out.println(shape.info());      }  }  **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. |

**SET - 5**

|  |  |
| --- | --- |
| **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 practical\_24 {      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.");              }          }          System.out.println("23DCS092\_Shubh");      }  }  **OUTPUT:**        **CONCLUSION:**  This program prompts the user to input two integers and attempts to divide the first integer by the second. It uses exception handling to catch errors such as division by zero or invalid inputs. If a ArithmeticException occurs (e.g., division by zero), it informs the user, and if any other exception arises (like entering non-integer values), it prompts the user to enter valid integers. Finally, it prints a message with the user's roll number or identifier. |
| **25.** | Write a Java program that throws an exception and catch it using a try-catch block.  **PROGRAM CODE :**  public class practical\_25  {      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());          }          System.out.println("23DCS092\_Shubh");      }  }  **OUTPUT:**    **CONCLUSION:** The code attempts to access an array element out of its bounds, which causes an ArrayIndexOutOfBoundsException. The exception is caught, and an error message is displayed. |
| **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.  public class practical\_26\_1 {      static void checkage(int age) throws MyException {          if (age > 100) {              throw new MyException();          }          System.out.println("Age is acceptable: " + age);      }      public static void main(String[] args) {          try {              checkage(5);              checkage(15);              checkage(105);          } catch (MyException e) {              System.out.println("Caught exception: ");          }          System.out.println("23DCS092\_Shubh");      }  }  class MyException extends Exception {      public MyException() {          System.out.println("Exepction Caught");      }  }  2.  public class practical\_26\_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());          }          System.out.println("23DCS092\_Shubh ");      }  }  **OUTPUT:**  1.    2.    **CONCLUSION:**  The code demonstrates handling both checked and unchecked exceptions in Java. The CheckedException() method throws a checked exception, which is caught and handled in the main method. The UncheckedException() method causes an ArithmeticException (division by zero), also caught in the main method. This illustrates proper exception handling using try-catch blocks for different exception types. |

**SET-6**

|  |  |
| --- | --- |
| 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 P27 {  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);  } } }  System.out.println("ID :23DCS092\_Shubh ")  } }  **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());  }  System.out.println("ID : 23DCS092\_Shubh ");  }}  **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());  }  System.out.println("ID : 23DCS092\_Shubh ");  } }  **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());  }  System.out.println("ID : 23DCS092\_Shubh");  } }  **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());  }  System.out.println("ID : 23DCS092\_Shubh");  } }  **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. |

**SET - 7**

|  |  |
| --- | --- |
| **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 practical\_32{      static class HelloThread extends Thread {          @Override          public void run() {              System.out.println("Hello World from thread class!");          }      }      static class HelloRunnable implements Runnable{          @Override          public void run() {              System.out.println("Hello World Runnable interface!");          }      }      public static void main(String[] args) {          HelloThread helloThread = new HelloThread();          HelloRunnable helloRunnable = new HelloRunnable();          helloThread.start();          Thread thread = new Thread(helloRunnable);          thread.start();          System.out.println("\n23DCS092\_Shubh");      }  }  **OUTPUT:**    **CONCLUSION:**  The code demonstrates two ways to create a thread: by extending the Thread class and by implementing the Runnable interface. Extending Thread directly ties the class to thread behavior, while using Runnable allows more flexibility since it separates task logic from thread management. The use of thread1.run() calls the method on the main thread, but thread1.start() should be used to run it in a new thread. |
| **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 practical\_33 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          System.out.print("Enter the start and end value: ");          int x = sc.nextInt();          int y = sc.nextInt();          sumthread st = new sumthread(x,y);          st.start();      }  }  class sumthread extends Thread {      private int start, end;      int sum = 0;      public sumthread(int a, int b) {          start = a;          end = b;      }      public void run() {          for (int i = start; i <= end; i++) {              sum += i;          }          System.out.println("Sum of start to end is: " + sum);      }  }  **OUTPUT:**    **CONCLUSION:**  The program calculates the sum of the first N numbers using multiple threads, each handling a specific range. It efficiently divides the task, synchronizes thread completion using join(), and combines the partial results to get the final sum. This showcases the use of multithreading for improved performance. |
| **34.** | 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 :**  public class practical\_34 {      public static void main(String[] args) {          RandomThread randomThread = new RandomThread();          randomThread.start();          System.out.println("23DCS092\_Shubh");      }  }  class RandomThread extends Thread {      public int randomNum;      public void run() {          for (int i = 0; i < 10; i++) {              randomNum = (int) (Math.random() \* 100);              System.out.println("Generated Random Number: " + randomNum);              if (randomNum % 2 == 0)  {                  EvenintegerThread evenThread = new EvenintegerThread();                  evenThread.number = randomNum;                  evenThread.start();              }  else  {                  OddintegerThread oddThread = new OddintegerThread();                  oddThread.number = randomNum;                  oddThread.start();              }              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  System.out.println(e);              }          }      }  }  class EvenintegerThread extends Thread {      public int number;      public void run() {          if (number % 2 == 0) {              int square = number \* number;              System.out.println("Square of " + number + " is: " + square);          }      }  }  class OddintegerThread extends Thread {      public int number;      public void run() {          if (number % 2 != 0) {              int cube = number \* number \* number;              System.out.println("Cube of " + number + " is: " + cube);          }      }  }  **OUTPUT:**    **CONCLUSION:**  The program generates random numbers and processes them in separate threads based on whether they are even or odd. For even numbers, a thread calculates their square, and for odd numbers, another thread calculates their cube. This demonstrates the use of multithreading to handle different tasks concurrently, improving efficiency and organizing logic based on specific conditions. |
| **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 :**  import java.util.Scanner;  public class practical\_35 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          System.out.print("Enter the number: ");          int x = sc.nextInt();          numberthread nt = new numberthread(x);          nt.start();          System.out.println("23DCS092\_Shubh");      }  }  class numberthread extends Thread{      public int number;      numberthread (int n){          number = n;      }      public void run(){          for(int i = 0; i < 10; i++){              number++;              System.out.println("Number: " + number);              try {                  Thread.sleep(1000);              } catch (InterruptedException e)  {                  System.out.println(e);              }          }      }  }  **OUTPUT:**    **CONCLUSION:**  The program creates a thread that prints numbers from 1 to 7, pausing for 1 second between each. The Thread.sleep() method is used to introduce the delay, and any interruption is handled by printing a message. This example demonstrates the basic concept of multithreading, where a thread runs concurrently with the main program, allowing for independent 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 :**   public class practical\_36 {      public static void main(String[] args) {          customthread t1 = new customthread("First Thread");          customthread t2 = new customthread("Second Thread");          customthread t3 = new customthread("Third Thread");          t1.setPriority(3);          t3.setPriority(7);          t1.start();          t2.start();          t3.start();          System.out.println("23DCS092\_Shubh");      }  }  class customthread extends Thread {      public customthread(String name) {          super(name);      }      @Override      public void run() {          System.out.println("Running thread :" + getName() + " with priority :" + getPriority());      }  }  **OUTPUT:**    **CONCLUSION:**  The program demonstrates thread prioritization in Java. It creates three threads (FIRST, SECOND, and THIRD), displays their default priorities, and then updates them. The setPriority() method is used to assign different priorities to the threads, and they are started in descending order of priority. The program showcases how thread priorities can influence the scheduling and execution order, although thread execution is ultimately managed by the JVM and may not strictly follow the set priorities. |
| **37.** | Write a program to solve producer-consumer problem using thread synchronization.  **PROGRAM CODE :**  class SharedBuffer {      int item; // A shared place for the item      boolean isProduced = false; // Whether the item is produced or not      public synchronized void produce() throws InterruptedException {          if (isProduced) {              return;  // If an item is already produced, do nothing          }          item = (int) (Math.random() \* 100); // Produce a random item          System.out.println("Produced: " + item);          isProduced = true; // Mark the item as produced          notify(); // Notify the consumer that the item is ready      }      public synchronized void consume() throws InterruptedException {          if (!isProduced) {              return;  // If no item is produced, do nothing          }          System.out.println("Consumed: " + item); // Consume the item          isProduced = false; // Mark that the item has been consumed          notify(); // Notify the producer that the buffer is now empty      }  }  class Producer extends Thread {      SharedBuffer buffer;      public Producer(SharedBuffer buffer) {          this.buffer = buffer;      }      @Override      public void run() {          try {              for (int i = 0; i < 10; i++) {                  buffer.produce(); // Produce an item                  Thread.sleep(1000); // Simulate some delay              }          } catch (InterruptedException e) {              e.printStackTrace();          }      }  }  class Consumer extends Thread {      SharedBuffer buffer;      public Consumer(SharedBuffer buffer) {          this.buffer = buffer;      }      @Override      public void run() {          try {              for (int i = 0; i < 10; i++) {                  buffer.consume(); // Consume an item                  Thread.sleep(1000); // Simulate some delay              }          } catch (InterruptedException e) {              e.printStackTrace();          }      }  }  public class practical\_37 {      public static void main(String[] args) throws InterruptedException {          SharedBuffer buffer = new SharedBuffer(); // Shared buffer          // Create producer and consumer threads by extending Thread          Producer producerThread = new Producer(buffer);          Consumer consumerThread = new Consumer(buffer);          // Start the threads          producerThread.start();          consumerThread.start();          // Wait for both threads to complete          producerThread.join();          consumerThread.join();          System.out.println("Producer and Consumer have finished execution.");          System.out.println(" ");          System.out.println("23DCS092\_Shubh");      }  }  **OUTPUT:**    **CONCLUSION:**  The program implements a producer-consumer scenario using a shared buffer. The Producer thread generates random numbers and the Consumer thread retrieves them, with synchronization ensuring safe access to the shared resource. The use of notify() facilitates communication between the producer and consumer, preventing race conditions. This example effectively demonstrates inter-thread coordination in Java. |

**SET - 8**

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **38.** | Design a Custom Stack using ArrayList class, which mplements following functionalities of stack. My Stack -list ArrayList.  **PROGRAM CODE:**  import java.awt.DisplayMode;  import java.util.\*;  class Stack {      ArrayList<Integer> Stack = new ArrayList<Integer>();      public void push(int data) {          Stack.add(data);          System.out.println("Data added succesfully !");      }      public void pop() {          if (!isempty()) {              Stack.remove(Stack.size() - 1);          } else {              System.out.println("Stack is empty !");          }      }      public int getSize() {          return Stack.size();      }      public int peek() {          if (!isempty()) {              return Stack.get(Stack.size() - 1);          } else {              System.out.println("Stack is empty !");              return 0;          }      }      public boolean isempty() {          return Stack.isEmpty();      }      public void display() {          for (int i = Stack.size() - 1; i >= 0; i--) {              System.out.println(Stack.get(i));          }      }  }  public class practical\_38 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          Stack stack = new Stack();          System.out.println("1.Push Element ");          System.out.println("2.Pop Element ");          System.out.println("3.Get size of Stack ");          System.out.println("4.Peek ");          System.out.println("5.isEmpty ?  ");          System.out.println("6.Displays");          System.out.println("7.exit");          while (true) {              System.out.print("Enter Choice: ");              int ch = sc.nextInt();              switch (ch) {                  case 1:                      System.out.print("Enter Data: ");                      int data = sc.nextInt();                      stack.push(data);                      break;                  case 2:                      stack.pop();                      break;                  case 3:                      System.out.println("Size of the stack is :" + stack.getSize());                      break;                  case 4:                      int d = stack.peek();                      System.out.println("top element is : " + d);                      break;                  case 5:                      boolean flag = stack.isempty();                      if (flag) {                          System.out.println("The Stack is empty");                      } else {                          System.out.println("Stack is not empty.");                      }                      break;                  case 6:                      stack.display();                  case 7:                      break;              }              System.out.println("");              if (ch == 7)                  break;          }          System.out.println("23DCS092\_Shubh");      }  }  **OUTPUT:**    **CONCLUSION:**  The code demonstrates two ways to create a thread: by extending the Thread class and by implementing the Runnable interface. Extending Thread directly ties the class to thread behavior, while using Runnable allows more flexibility since it separates task logic from thread management. The use of thread1.run() calls the method on the main thread, but thread1.start() should be used to run it in a new thread. |
| **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 practical\_39 {      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 : " + Arrays.toString(numbers));          sortArray(numbers);          System.out.println("After sorting : " + Arrays.toString(numbers));          String[] names = {"John", "Alice", "Bob", "David"};          System.out.println("\n Strings Before sorting : " + Arrays.toString(names));          sortArray(names);          System.out.println("Strings After sorting : " + 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);          }          System.out.println("23DCS092\_Shubh");      }  }  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:**  The program calculates the sum of the first N numbers using multiple threads, each handling a specific range. It efficiently divides the task, synchronizes thread completion using join(), and combines the partial results to get the final sum. This showcases the use of multithreading for improved performance. |
| **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 practical\_40 {      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());          }          System.out.println("23DCS092\_Shubh");      }  }  **OUTPUT:**    **CONCLUSION:**  The program generates random numbers and processes them in separate threads based on whether they are even or odd. For even numbers, a thread calculates their square, and for odd numbers, another thread calculates their cube. This demonstrates the use of multithreading to handle different tasks concurrently, improving efficiency and organizing logic based on specific conditions. |
| **41.** | 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.File;  import java.io.FileNotFoundException;  import java.util.HashSet;  import java.util.Scanner;  public class practical\_41 {      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", "while"          };          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); // Corrected: Using File object to read the file              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);          }          System.out.println("23DCS092\_Shubh");      }  }  **OUTPUT:**      **CONCLUSION:**  The program creates a thread that prints numbers from 1 to 7, pausing for 1 second between each. The Thread.sleep() method is used to introduce the delay, and any interruption is handled by printing a message. This example demonstrates the basic concept of multithreading, where a thread runs concurrently with the main program, allowing for independent execution. |