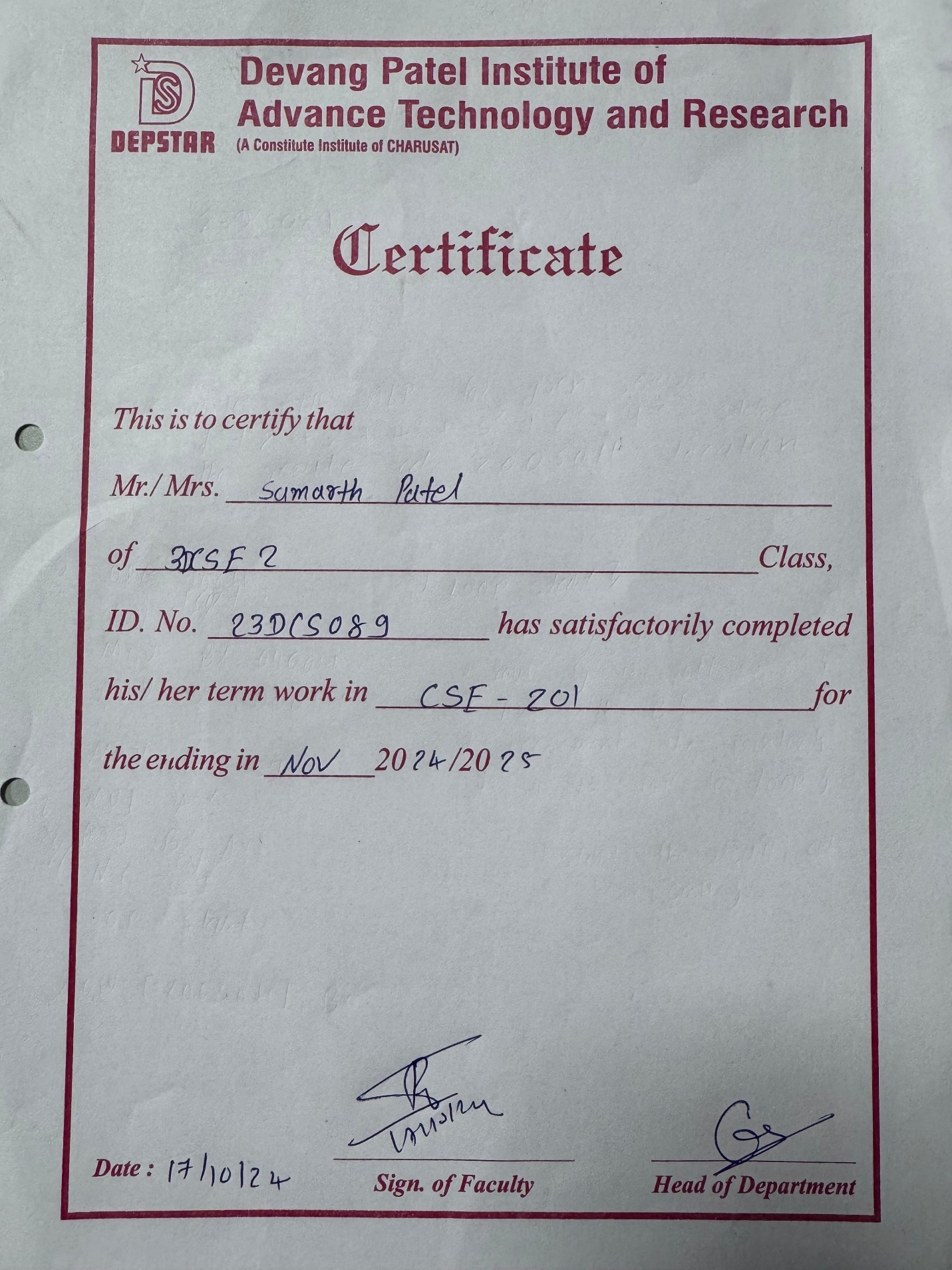
****

**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 - 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.  1. Installation of Java  Steps to install Java Development Kit (JDK):   * Download JDK:   - Go to the Oracle JDK download page: [Oracle JDK Downloads]  (https://www.oracle.com/java/technologies/javase-downloads.html).  - Select the appropriate JDK version for your operating system (Windows, macOS,Linux).  - Download the installer package (.exe for Windows, .dmg for macOS, .tar.gz for Linux).   * Install JDK:   - Windows: Double-click the downloaded .exe file and follow the installation instructions.  - macOS: Double-click the downloaded .dmg file, then drag and drop the JDK package icon to the Applications folder.  - Linux: Extract the downloaded .tar.gz file to a directory and follow the instructions in the README file for installation.   * Set JAVA\_HOME (Optional):   - Windows: Set the JAVA\_HOME environment variable to the JDK installation directory.  - macOS/Linux: Add the JDK bin directory to your PATH and set JAVA\_HOME in your shell profile (e.g., ~/.bash\_profile, ~/.bashrc).   * Verify Installation:   - Open a terminal or command prompt.  - Type `java -version` and `javac -version` to verify that Java runtime and compiler are installed correctly.  2. Introduction to Object-Oriented Concepts  Object-oriented programming (OOP) revolves around the concept of objects, which are instances of classes. Key principles include:  - Classes and Objects: Classes define the blueprint for objects.  - Encapsulation: Bundling data (attributes) and methods (functions) that operate on the data within a single unit (class).  - Inheritance: Mechanism where a new class (derived or child class) is created from an existing class (base or parent class).  - Polymorphism: Ability of different objects to be treated as instances of the same class through method overriding and overloading.  3. Comparison of Java with Other Object-Oriented Programming Languages  Java is often compared with languages like C++, C#, and Python in terms of syntax, features, and application domains. Key points of comparison include:  - Syntax: Java has a C-style syntax with similarities to C++.  - Memory Management: Java uses automatic garbage collection, unlike C++ which requires manual memory management.  - Platform Independence: Java programs are compiled into bytecode, which can run on any JVM, making it platform-independent.  - Libraries: Java has a rich standard library (Java API) comparable to those in C++ and C#.  - Community and Ecosystem: Java has a large developer community and extensive third-party libraries and frameworks.  4. Introduction to JDK, JRE, JVM, Javadoc, Command Line Arguments  - JDK (Java Development Kit): Includes tools for developing and running Java programs, including JRE and development tools such as javac (Java compiler).  - JRE (Java Runtime Environment): Includes JVM (Java Virtual Machine) and libraries required to run Java applications, but does not include development tools.  - JVM (Java Virtual Machine): Executes Java bytecode and provides a runtime environment for Java programs.  - Javadoc: Tool for generating API documentation from Java source code comments.  - Command Line Arguments: Parameters passed to a Java program when it is invoked from the command line.  5. Introduction to Eclipse or NetBeans IDE (Integrated Development Environment)  - Eclipse : A widely used open-source IDE for Java development, also supports other programming languages through plugins. Features include code editing, debugging, and version control integration.  - NetBeans: Another popular open-source IDE primarily for Java development, with features similar to Eclipse.  6. Introduction to BlueJ and Console Programming  - BlueJ : A lightweight IDE specifically designed for teaching and learning Java programming, providing a simplified interface and visualization tools for object-oriented concepts.  - Console Programming : Refers to writing Java programs that interact with users via text-based input and output through the console |
| **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 $20. Write a java program to store this balance in a variable and then display it to the user.  **PROGRAM CODE :**  public class pra\_2{ public static void main(String[] args) { int a = 20; System.err.print("Current Balance is :$"); System.err.println(a); System.err.print("23DSC089\_SAMARTH PATEL"); }}  **OUTPUT:**    **CONCLUSION:**  In this example, we created a basic Java program that simulates a banking application by storing a user's account balance in a variable and then displaying it. This demonstrates the fundamental concept of variable storage and output in Java, which is essential for developing more complex banking systems. |
| **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 Pra3 {      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();          float totalSeconds;          totalSeconds = seconds+(minutes\*60)+(hours\*3600);          float speed1= distance / totalSeconds;          float speed2= (distance / 1000) / (totalSeconds / 3600);          float speed3= (distance / 1609) / (totalSeconds / 3600);          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("23DCS089\_SAMARTH PATEL");      }    }  **OUTPUT:**    **CONCLUSION:**  In this program, we used Java's Scanner class to take user input for distance in meters and time in hours, minutes, and seconds. We then converted the time to total seconds to simplify the calculations. The program calculates and displays the speed in three different units: meters per second, kilometers per hour, and miles per hour. This example demonstrates how to handle user input, perform unit conversions, and output results 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.\*;  public class pra\_4 {      public static void main(String[] args) {          float sum = 0;          int n;          int[] arr = new int[30];          System.out.println("Enter the number of days:");          Scanner b = new Scanner(System.in);          n = b.nextInt();          for (int i = 0; i < n; i++) {              arr[i] = b.nextInt();              sum = sum + arr[i];          }          System.out.println("The total expense is: " + sum);          System.out.println("23DCS089\_SAMARTH PATEL");      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program provides a simple yet effective way to track and calculate monthly expenses based on daily inputs from the user. It demonstrates how to use arrays to store data, take user input through a loop, and perform a summation of array elements. |
| **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;  public class pra5 {      public static void main(String[] args) {          int a;         // int code[] = {1,2,3,4,5};          int price[] = {200,550,100,170,220};          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));              case 2 -> System.out.println("The price of the Fan is :"+(price[1]+price[1]\*0.12f));              case 3 -> System.out.println("The price of the Tubes is :"+(price[2]+price[2]\*0.075f));              case 4 -> System.out.println("The price of the wires is :"+(price[3]+price[3]\*0.03f));              default -> System.out.println("Invalid choice");          }      }  }  **OUTPUT:**    **CONCLUSION:**  This practical exercise reinforced the fundamentals of array handling, control structures, and basic arithmetic operations in Java |
| **6.** | Create a Java program that prompts the user to enter the number of days (n) for which they want to generate their exercise routine. The program should then calculate and display the first n terms of the Fibonacci series, representing the exercise duration for each day.  **PROGRAM CODE :**  import java.util.Scanner;  public class Pr6  {      public static void main(String[] args) {          int j = 1, k, temp = 0, sum = 0;          Scanner sc = new Scanner(System.in);          int n = sc.nextInt();          for (int i = 0; temp <=n; i++) {              System.out.println(temp);              k = j;              j = temp;              temp = j + k;              sum += temp;          }          System.out.println(sum);          sc.close();      }  }  **OUTPUT:**    **CONCLUSION:**  This program provides an easy way to generate an exercise routine based on the Fibonacci series, which is often used to progressively increase workout durations. By following this routine, you can ensure a gradual increase in your exercise time, which can help improve endurance and overall fitness. The Fibonacci series starts with 0 and 1, and each subsequent term is the sum of the previous two terms. This pattern creates a naturally progressive routine, making it a useful tool for designing exercise plans. |

**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 pra\_7 {      public static void main(String[] args) {          String st = "chocolate";          String st1 = "Abc";          int n, m;          Scanner b = new Scanner(System.in);          System.out.println("Enter the number of times you want to print the first 3 characters of the string");          n = b.nextInt();          System.out.println("Enter the number of times you want to print the first 3 characters of the string");          m = b.nextInt();          System.out.println("--------------------");          for (int i = 0; i < n; i++) {              System.out.print(st.substring(0, 3));          }          System.out.println("\n--------------------");          for (int i = 0; i < m; i++) {              System.out.print(st.substring(0, 3));          }          System.out.println("\n--------------------");          for (int i = 0; i < m; i++) {              System.out.print(st1.substring(0, 3));          }          System.out.println("\n23DCS089\_Samarth Patel");      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program prompts the user to enter two integers, n and m, then prints the first 3 characters of the string "chocolate" n times and m times consecutively. Additionally, it prints the first 3 characters of the string "Abc" m times. The program separates different outputs with dashes and concludes by displaying a footer message. The use of loops ensures repetitive printing based on user input, demonstrating basic input-output operations and string manipulation 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 Pra\_8 {      public static int array\_count9(int[] nums) {          int count = 0;          for (int num : nums) {              if (num == 9) {                  count++;              }          }          return count;      }      public static void main(String[] args) {          int[] nums1 = {1, 2, 9};          int[] nums2 = {1, 9, 9};          int[] nums3 = {1, 9, 9, 3, 9};          System.out.println("Number of 9's in nums1: " + array\_count9(nums1));          System.out.println("Number of 9's in nums2: " + array\_count9(nums2));          System.out.println("Number of 9's in nums3: " + array\_count9(nums3));      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program defines a method `array\_count9` that calculates and returns the number of times the integer `9` appears in an input integer array. It then demonstrates the functionality by applying the method to three different arrays (`nums1`, `nums2`, `nums3`) and prints the results. Finally, it includes a footer message `"23DCS089\_Samarth Patel"` for identification. The program effectively showcases basic array manipulation and function usage in Java. |
| **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.lang.\*;  public class Pr9 {      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 s1 = "The";          String s2 = "AAbb";          String s3 = "Hi-There";          double\_char(s1);          double\_char(s2);          double\_char(s3);          System.out.println("23DCS089\_Samarth Patel");      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program defines a method `doubleChar` within the class `pra\_9`, which takes a string `str` as input and doubles each character in the string. It uses a `StringBuilder` to efficiently build the resultant string by appending each character twice in a loop. The `main` method demonstrates the functionality of `doubleChar` by creating an instance of `pra\_9`, calling the method with different input strings ("The", "AAbb", "Hi-There"), and printing the transformed strings. The program concludes with a footer. |
| **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 pra\_10 {      public static void main(String[] args) {          String str = "Hello World!";            int length = str.length();          System.out.println("Length of the string: " + length);          String lowercase = str.toLowerCase();          System.out.println("Lowercase string: " + lowercase);            String uppercase = str.toUpperCase();          System.out.println("Uppercase string: " + uppercase);            String reverse = "";          for (int i = length - 1; i >= 0; i--) {              reverse = reverse + str.charAt(i);          }          System.out.println("Reversed string: " + reverse);          char[] charArray = str.toCharArray();          Arrays.sort(charArray);          String sortedString = new String(charArray);          System.out.println("Sorted string: " + sortedString);          System.out.println("23DCS089\_Samarth Patel");      }  }  **OUTPUT:**    **CONCLUSION:**  The Java program provided demonstrates several string manipulation techniques. It calculates the length of a string, converts it to lowercase and uppercase, reverses the string, and sorts its characters alphabetically. These operations showcase fundamental string handling capabilities in Java using built-in methods and loops. Additionally, the program concludes by printing a fixed message. Overall, it serves as a basic example of how to perform common string operations and utilize array manipulation functions in Java. |
| **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 pra\_11 {      public static void main(String[] args) {          String str = "CHARUSAT UNIVERSITY";          int length = str.length();          System.out.println("Length: " + length);          String replacedStr = str.replace('H', 'S');          System.out.println("Replaced String: " + replacedStr);          String lowercaseStr = str.toLowerCase();          System.out.println("Lowercase String: " + lowercaseStr);          System.out.println("23DCS089\_Samarth Patel");      }  }  **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.Scanner;  public class pra\_12 {  public static void main(String[] args) {    if (args.length > 0) {  double pounds = Double.parseDouble(args[0]);  double rupees = pounds \* 100.0;  System.out.println(pounds + " Pounds is equal to " + rupees + " Rupees");  } else {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter the amount in Pounds: ");  double pounds = scanner.nextDouble();  double rupees = pounds \* 100.0;  System.out.println(pounds + " Pounds is equal to " + rupees + " Rupees");  }  }  }  **OUTPUT:**    **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:**  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() {          // getYearlySalary();          System.out.println("Salary: " + salary);      }  }  public class pra\_13 {      public static void main(String[] args) {          Employee em1 = new Employee("Shubh", "patel", 4000.0);          Employee em2 = new Employee("Mark", "patel", 5000.0);          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("23DCS089\_Samarth");      }  }  **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:**  class date{      private int day;      private int month;      private int year;      date(int d, int m, int y){          day = d;          month = m;          year = y;      }      void display(){          System.out.println(month + "/" + day + "/" + year);      }  }  public class pra\_14 {      public static void main(String[] args) {          date d1 = new date(10, 10, 2021);          date d2 = new date(11, 7, 2022);          d1.display();          System.out.println("New date :");          d2.display();          System.out.println("23DCS089\_Samarth Patel");      }  }  **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:**  import java.util.\*;  class Area {      private int length;      private int breadth;      Area(int l, int b) {          length = l;          breadth = b;      }      int returnArea() {          return length \* breadth;      }  }  public class pra\_15 {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          System.out.print("Enter the length of the rectangle: ");          int length = scanner.nextInt();          System.out.print("Enter the breadth of the rectangle: ");          int breadth = scanner.nextInt();          Area rectangle = new Area(length, breadth);          System.out.println("Area of the rectangle: " + rectangle.returnArea());          System.out.println("23DCS089\_Samarth Patel");      }  }  **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:**  import java.util.Scanner;  class Complex {      private double real;      private double imaginary;          Complex(double r, double i) {          real = r;          imaginary = i;      }          void Sum(Complex other) {          double sumReal = real + other.real;          double sumImaginary = imaginary + other.imaginary;          System.out.println("Sum: " + sumReal + " + " + sumImaginary + "i");      }          void Difference(Complex other) {          double diffReal = real - other.real;          double diffImaginary = imaginary - other.imaginary;          System.out.println("Difference: " + diffReal + " + " + diffImaginary + "i");      }          void Product(Complex other) {          double productReal = real \* other.real - imaginary \* other.imaginary;          double productImaginary = real \* other.imaginary + imaginary \* other.real;          System.out.println("Product: " + productReal + " + " + productImaginary + "i");      }  }  public class pra\_16 {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);            System.out.print("Enter the real part of the first complex number: ");          double real1 = scanner.nextDouble();          System.out.print("Enter the imaginary part of the first complex number: ");          double imaginary1 = scanner.nextDouble();            System.out.print("Enter the real part of the second complex number: ");          double real2 = scanner.nextDouble();          System.out.print("Enter the imaginary part of the second complex number: ");          double imaginary2 = scanner.nextDouble();            Complex complex1 = new Complex(real1, imaginary1);          Complex complex2 = new Complex(real2, imaginary2);          complex1.Sum(complex2);          complex1.Difference(complex2);          complex1.Product(complex2);          System.out.println("23DCS089\_Samarth Patel");      }  }  **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. |

**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:**  class Parant{      public void printp(){          System.out.println("Parent");      }  }  class Child extends  Parant{      public void printc(){          System.out.println("Child");      }  }  public class pra\_17{      public static void main(String[] args){            Parant p = new Parant();          Child c = new Child();          p.printp();          c.printc();          c.printp();      }  }  **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.Scanner;  class Member{      String Name;      int Age;      long phone\_number;      String address;      double salary;        void printsalary(){          System.out.println("your salary is :" +salary);      }    }  class Employee extends Member{      String specialization;      void setdata(){          Scanner sc = new Scanner(System.in);          System.out.print("Enter your name:");          Name=sc.nextLine();          System.out.print("Enter your age:");          Age=sc.nextInt();          System.out.print("Enter your phone number:");          phone\_number=sc.nextLong();          sc.nextLine();          System.out.print("Enter your address:");          address=sc.nextLine();          // sc.nextLine();          System.out.print("Enter your salary:");          salary=sc.nextDouble();          sc.nextLine();          System.out.print("Enter your specialization:");          specialization=sc.nextLine();        }      void display(){          System.out.println("Name           :" +Name);          System.out.println("Age            :" +Age);          System.out.println("phone          :" +phone\_number);          System.out.println("address        :" +address);          System.out.println("specialization :" +specialization);      }  }  class Manager extends Member{      String department;      void setdata(){          Scanner sc = new Scanner(System.in);      System.out.print("Enter your name:");          Name=sc.nextLine();          System.out.print("Enter your age:");          Age=sc.nextInt();          System.out.print("Enter your phone number:");          phone\_number=sc.nextLong();          sc.nextLine();          System.out.print("Enter your address:");          address=sc.nextLine();          // sc.nextLine();          System.out.print("Enter your salary:");          salary=sc.nextDouble();          sc.nextLine();          System.out.print("Enter your department:");          department=sc.nextLine();          sc.close();      }      void display(){          System.out.println("Name           :" +Name);          System.out.println("Age            :" +Age);          System.out.println("phone          :" +phone\_number);          System.out.println("address        :" +address);          System.out.println("department     :" +department);      }  }  public class pra\_18 {      public static void main(String[] args) {          Employee o1 = new Employee();          Manager o2 = new Manager();          o1.setdata();          o2.setdata();          o1.display();          o2.display();      }  }  **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 :**  class Rectangle {      int width;      int height;      Rectangle(int w, int h) {          width = w;          height = h;      }      int getArea() {          return width \* height;      }      int getPerimeter() {          return 2 \* (width + height);      }  }  class Square extends Rectangle {      Square(int s) {          super(s, s);      }  }  public class pra\_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 r : rectangles) {              System.out.println("Area: " + r.getArea());              System.out.println("Perimeter: " + r.getPerimeter());              System.out.println();          }          System.out.println("23DCS089\_Samarth");      }  }  **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{      void print(){          System.out.println("This is shape");      }  }  class Rectangle extends shape{      void print1(){          System.out.println("This is Rectangle");      }  }  class Circle extends shape{      void print2(){          System.out.println("This is Circle");      }  }  class Squre extends Rectangle{      void print3(){          System.out.println("Squre is rectangle");      }  }  public class pra\_20 {      public static void main(String[] args) {          Squre s = new Squre();          s.print();          s.print1();          s.print3();          System.out.println("23DCS089\_Samarth");      }  }  **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 pra\_21 {      public static void main(String[] args) {          Degree degree = new Degree();          Undergraduate undergraduate = new Undergraduate();          Postgraduate postgraduate = new Postgraduate();          degree.getDegree();          undergraduate.getDegree();          postgraduate.getDegree();          System.out.println("23DCS089\_Samarth");      }  }  **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 :**  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 pra\_22 {      public static void main(String[] args) {          MyCalculator my\_calculator = new MyCalculator();          System.out.print("I implemented: ");          int n = 12;          System.out.println(my\_calculator.divisor\_sum(n));          System.out.println("23DCS089\_Samarth");      }  }  **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 :**    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 pra\_23 {      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();          System.out.println("23DCS089\_Samarth");      }  }  **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 pra\_24 {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);            try {              System.out.print("Enter the first integer (x): ");              int x = scanner.nextInt();                System.out.print("Enter the second integer (y): ");              int y = scanner.nextInt();                int result = x / y;              System.out.println("Result: " + result);            } catch (Exception e) {              if (e instanceof ArithmeticException) {                  System.out.println("Error: Division by zero is not allowed.");              } else {                  System.out.println("Error: Please enter valid integers.");              }          }      }  }  **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 pra\_25 {      public static void main(String[] args) {          try {              int[] numbers = {1, 2, 3};              System.out.println("Accessing element at index 5: " + numbers[1]);            } catch (ArrayIndexOutOfBoundsException e) {                System.out.println("Exception caught: Array index is out of bounds.");          }          System.out.println("23DCS089\_Samarth");      }  }  **OUTPUT:**    **CONCLUSION:**  n this program, an attempt is made to access an array element within its valid bounds, so no exception occurs. The try-catch block is designed to handle cases where an invalid array index might be accessed, which would trigger an ArrayIndexOutOfBoundsException. |
| **26.** | Write a java program to generate user defined exception using “throw” and “throws” keyword. Also Write a java that differentiates checked and  unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).  **PROGRAM CODE :**  import java.io.\*;  public class P26 {  public static void readFile(String filePath) throws IOException {  FileReader file = new FileReader(filePath);  BufferedReader br = new BufferedReader(file);  System.out.println(br.readLine());  br.close();  }  public static void main(String[] args) {  try {  int a = 10 / 0;  } catch (ArithmeticException e) {  System.out.println("Unchecked Exception caught: " + e);  }  try {  String str = null;  System.out.println(str.length());  } catch (NullPointerException e) {  System.out.println("Unchecked Exception caught: " + e);  }  try {  readFile("nonexistentfile.txt");  } catch (IOException e) {  System.out.println("Checked Exception caught: " + e);  }  try {  Class.forName("UnknownClass");  } catch (ClassNotFoundException e) {  System.out.println("Checked Exception caught: " + e);  }  } }  **OUTPUT:**    **CONCLUSION:**  This Java program manages unchecked as well as checked exceptions. Arithmetic, null pointer file not found, and class not found exceptions are caught using the try-catch blocks, therefore the program runs quite seamlessly. Exception handling is quite important in building robust applications that can handle errors with grace. |

**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 pra\_27 {      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("23DCS089 Samarth");        }  }  **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.** | **Aim: 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:**  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  public class Pr28{  public static void main(String[] args) {  if (args.length < 2) {  System.out.println("Usage: java PrW28 <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("23DCS089 Samarth");  }}  **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.** | **AIM: Write a Java Program to Search for a given word in a File. Also show use of Wrapper Class with an example.**  **Program:**  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  public class Pr29 {      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("23DCS089 Samarth");      }  }  **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.** | **AIM: Write a program to copy data from one file to another file. If the destination file does not exist, it is created automatically.**  **Program:**  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  public class Pr30 {      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("23DCS089\_Samarth ");      }  }  **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.** | **AIM: 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:**  import java.io.\*;  public class pra\_31 {      public static void main(String[] args) {          BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));          String fileName = "file1.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("23DCS089\_Samarth");      }  }  **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 pra\_32 {      static class HelloWorldThread extends Thread {          @Override          public void run() {              System.out.println("Hello World from Thread class");          }      }      static class HelloWorldRunnable implements Runnable {          @Override          public void run() {              System.out.println("Hello World from Runnable interface");          }      }      public static void main(String[] args) {          HelloWorldThread thread = new HelloWorldThread();          thread.start();          HelloWorldRunnable runnable = new HelloWorldRunnable();          Thread thread1 = new Thread(runnable);          thread1.start();          System.out.println("23DCS089\_Samarth");      }  }  **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:**  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);      }  }  public class pra\_33 {      public static void main(String[] args) {          sumthread st = new sumthread(2,6);          st.start();          System.out.println("23DCS089\_Samarth");      }  }  **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:**  class RandomNumberGenerate extends Thread {      public void run() {          int i = 0;          while (i <= 10) {              int randomNum = (int) (Math.random() \* 100);              System.out.println("Generated Random Number: " + randomNum);              if (randomNum % 2 == 0) {                  new SquareThread(randomNum).start();              } else {                  new CubeThread(randomNum).start();              }              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  e.printStackTrace();              }              i++;          }      }  }  class SquareThread extends Thread {      private int number;      SquareThread(int n) {          number = n;      }      public void run() {          int square = number \* number;          System.out.println("Square of " + number + " is: " + square);      }  }  class CubeThread extends Thread {      private int number;      CubeThread(int n) {          number = n;      }      public void run() {          int cube = number \* number \* number;          System.out.println("Cube of " + number + " is: " + cube);      }  }  public class pra\_34 {      public static void main(String[] args) {          RandomNumberGenerate rng = new RandomNumberGenerate();          rng.start();          System.out.println("23DCS089\_Samarth");      }  }  **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;  class IncrementThread extends Thread {      int number;      IncrementThread(int n) {          number = n;      }      public void run() {          int i = 0;          while (i < 5) {              number++;              System.out.println("Incremented Value: " + number);              try {                  Thread.sleep(1000);              } catch (InterruptedException e) {                  e.printStackTrace();              }              i++;          }      }  }  public class pra\_35 {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          System.out.print("Enter the starting number: ");          int n = scanner.nextInt();          scanner.close();          IncrementThread it = new IncrementThread(n);          it.start();          System.out.println("23DCS089\_Samarth");      }  }  **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:**  class CustomThread extends Thread {      public CustomThread(String name) {          super(name);      }      @Override      public void run() {          System.out.println("Running thread: " + getName() + " with priority: " + getPriority());      }  }  public class pra\_36 {      public static void main(String[] args) {          CustomThread firstThread = new CustomThread("FIRST");          CustomThread secondThread = new CustomThread("SECOND");          CustomThread thirdThread = new CustomThread("THIRD");          firstThread.setPriority(3);          // secondThread.setPriority(Thread.NORM\_PRIORITY);          thirdThread.setPriority(7);          firstThread.start();          secondThread.start();          thirdThread.start();      }  }  **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 prac37 {      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("23DCS089\_Samarth");      }  }  **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**

|  |  |
| --- | --- |
| **38** | **Aim: Design a Custom Stack using ArrayList class, which mplements following functionalities of stack. My Stack -list ArrayList.**  **Program:**  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 pra\_38 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);  Stack stack=new Stack();          while (true) {              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");              System.out.print("Enter Choice: ");              int ch = sc.nextInt();              switch (ch) {                  case 1:                      System.out.println("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("23DCS089\_ Samarth");      }  }  **Output:**    **Conclusion:**  In this java program Stack is successfully implemented using arraylist and fuctionallites of stack like push,pop,isempty?, peek,getsize are added using arraylist`s functions like add(),get(),remove(),isempty(). . |
| **39.** | **Aim:** **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:**  import java.util.Arrays;  public class pra\_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("23DCS089\_ Samarth");      }  }  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.** | **Aim:** **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:**  import java.util.\*;  public class pra\_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("23DCS089\_Samarth");      }  }  **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.** | **Aim:** **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:**  import java.io.File;  import java.io.FileNotFoundException;  import java.util.HashSet;  import java.util.Scanner;  public class pra\_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("23DCS089\_Samarth");      }  }  **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. |