CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

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

Department of Computer Science & Engineering

Subject Name:

Semester:

Subject Code:

Academic year:

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.  **PROGRAM CODE:**  public class prac1 {  public static void main(String[] args) {  System.out.println("Hello world");  System.out.println();  System.out.println("23DCS083\_MARK\_PATEL");    }    }  **OUTPUT:**    **DEMONSTRATION:-**  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 (command line interface).  **CONCLUSION:**  In this we learn how to print hello world in java programming. |
| 2. | Imagine you are developing a simple banking application where you need to display the current balance of a user account. For simplicity, let's say the current balance is $20.  Write a java program to store this balance in a variable and then display it to the user.  **PROGRAM CODE:**  public class prac2 {      public static void main(String[] args) {          int money = 20;          System.out.println("$"+money);      }    }  **OUTPUT:**    **CONCLUSION:**  This code teaches us how to use variables in java and how to store different types of values in variables. |
| 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 prac3 {      public static void main(String[] args) {          System.out.println("Enter the Distace in meters");          Scanner sc = new Scanner(System.in);          float distance = sc.nextFloat();          char ch=sc.next();          System.out.println("Enter the time in hour");          float hour = sc.nextFloat();          System.out.println("Enter the time in minutes");          float minutes = sc.nextFloat();          System.out.println("Enter the time in second");          float seconds = sc.nextFloat();          System.out.println("speed in meter per second i.e m/s " + distance / (hour \* 3600 + minutes \* 60 + seconds));          System.out.println("speed in kilometer per hour i.e. km/hr " + distance / (1000 \* (hour + minutes / 60 + seconds / 3600)));          System.out.println("speed in meter per second i.e mile/hr " + distance / (1609 \* (hour + minutes / 60 + seconds / 3600)));      }  }  **OUTPUT:**  **CONCLUSION:**  The Java program calculates speed from user-input distance and time in various units (meters per second, kilometers per hour, and miles per hour). It demonstrates basic input handling, arithmetic operations, and output formatting 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 prac4 {  public static void main(String[] args) {    int total=0;  System.out.println("enter your number of expenses");  Scanner sc=new Scanner(System.in);  int n=sc.nextInt();  int [] arr= new int[n];//you have to dynamically intialise it just like this  for(int i=0;i<n;i++){  System.out.println("Enter your " + (i+1) +" expense ");  arr[i]=sc.nextInt();  }  for(int i=0;i<n;i++){  total+=arr[i];  }  System.out.println("So your tottal Expenses are : " + total);  }    }  **OUTPUT:**    **CONCLUSION:**  So, here we write a program to calculate our monthly expense using arrays and adding of each of them in for loops. |
| 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:**  //package java\_college\_pracs;  import java.util.Scanner;  public class prac5 {  public static void main(String[] args) {  int[] arr={1,2,3,4,5};  float[] arr2={100,200,300,400,500};  Scanner sc = new Scanner(System.in);  System.out.println("enter your choice ");  int ch=sc.nextInt();  switch(ch){  case 1:  System.out.println("The price of motor with tax are "+ (arr2[0]+arr2[0]\*0.08f));  break;    case 2:  System.out.println("The price of fan with tax are "+ (arr2[1]+arr2[1]\*0.12f));  break;  case 3:  System.out.println("The price of tube with tax are "+ (arr2[2]+arr2[2]\*0.05f));  break;  case 4:  System.out.println("The price of wires with tax are "+ (arr2[3]+arr2[3]\*0.075f));  break;  case 5:  System.out.println("The price of other items with tax are "+ (arr2[4]+arr2[4]\*0.03f));  break;  default:  System.out.println("SORRY!");  }  sc.close();  System.out.println("23DCS083\_MARK\_PATEL");  }    }  **OUTPUT:**    **CONCLUSION:**  The Java program simulates an electric appliance shop transaction, allowing users to select items and calculate the total bill based on predefined prices and taxes. It utilizes arrays, loops, and switch-case statements for functionality and demonstrates formatted output for a detailed bill summary. |
| 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:**  public class prac6 {  public static void main(String[] args) {  int j = 1, k = 0, temp = 0;  for (int i = 0; temp <= 50; i++) {  System.out.print(" " +temp);  k = j;  j = temp;  temp = j + k;    }  System.out.println();  System.out.println("23DCS083\_Mark\_Patel");  }  }  **OUTPUT:**  **CONCLUSION:**  This Java program uses basic constructs like loops and variables to generate an exercise routine. Specifically, it calculates and prints the exercise duration for each day based on the Fibonacci sequence for a specified number of days. |

**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.\*;  import javax.swing.\*;  public class prac7\_1 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);    System.out.println("enter the string");  String st = sc.nextLine();  System.out.println("enter the number");  int n = sc.nextInt();  front\_times(st, n);  System.out.println();  System.out.println("23DCS083\_Mark\_Patel");  }  public static void front\_times(String st, int n) {  for (int i = 0; i < n; i++) {  System.out.print(st.substring(0, 3));  }  }  }  **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 :**  import java.util.Scanner;  public class prac8 {  public static void main(String[] args) {  Scanner sc=new Scanner(System.in);  System.out.println("Enter the total no. of elements ");  int n=sc.nextInt();    int [] arr=new int[n];  for(int i=0;i<n;i++){  System.out.println("Enter the elenemt no " +(i+1) +" :");  arr[i]=sc.nextInt();  }  array\_count9(arr);  sc.close();  System.out.println("23DCS083\_Mark\_patel");  }  public static void array\_count9(int []arr){  int count=0;  //int c =arr.length();  for(int i=0;i<arr.length;i++){  if(arr[i]==9){  count++;  }    }  System.out.println("The total count of 9's are " +count);  }  }  **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.util.Scanner;  public class prac9 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("enter the string");  String st = sc.nextLine();  double\_char(st);  System.out.println();  System.out.println("23DCS083\_MARK\_PATEL");  }  public static void double\_char(String st){  for(int i=0;i<st.length();i++){  System.out.print(st.charAt(i));  System.out.print(st.charAt(i));  }  }  }  **OUTPUT:**    **CONCLUSION:**  This Java program takes a user-inputted string and calculate the length of the string, duplicates each character using function in that string, and then prints the double char of that string to the output using for loop. |
| **10.** | Perform following functionalities of the string:  ● Find Length of the String  ● Lowercase of the String  ● Uppercase of the String  ● Reverse String  Sort the string  **PROGRAM CODE :**  import java.util.\*;  public class prac10 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("Enter your String ");  String st=sc.nextLine();  StringBuffer s=new StringBuffer(st);  System.out.println("length of String is " +st.length());  System.out.println("Lowercase of String is " +st.toLowerCase());  System.out.println("Uppercase of String is " +st.toUpperCase());  System.out.println("Reverse of String is " +s.reverse());  char [] arr = st.toCharArray();  Arrays.sort(arr);  String temp = new String (arr);  System.out.println("Sorted string is " + temp);  System.out.println("\n");  System.out.println("23DCS083\_MARK\_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 prac11 {  public static void main(String[] args) {  StringBuffer st = new StringBuffer("CHARUSAT UNIVERSITY");  System.out.println("length of String is :" +st.length());  st.setCharAt(1, 'M');  System.out.println("replacing my name character :" +st);  String s = new String(st);    System.out.println("to lowercase : " +s.toLowerCase());  System.out.println();  System.out.println("23DCS083\_MARK\_PATEL");  }  }  **OUTPUT:**    **CONCLUSION:**  In this java program we learn how to find the length of the given String and then replacing its character with another character using setcharat method and then in last we use tolowercase to convert it into lower case. |

**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:**  Simple :-  import java.util.\*;  public class prac12{  public static void main(String[] args){  Scanner sc = new Scanner(System.in);  System.out.println("Enter tota pounds");  int n = sc.nextInt();  int tot=torupees(n);  System.out.println("the total rupees are " +tot);  System.out.println("23DCS083\_MARK\_PATEL");  }  public static int torupees(int x){  return x\*100;  }  }  Using command line argument:-  public class prac12\_1 {  public static void main(String[] args) {  float rup = Float.parseFloat(args[0]);  double pound = rup\*100;  System.out.println("rupees is " +rup);  System.out.println("pounds is " +pound);  System.out.println("23DCS083\_MARK\_PATEL");  }  }  **OUTPUT:**      **CONCLUSION:**  The currency conversion tool efficiently converts Pounds to Rupees at a fixed rate of 1 Pound = 100 Rupees. It supports both command-line input for automation and interactive input for user convenience, ensuring a versatile and user-friendly experience for travelers. |
| **13.** | Create a class called Employee that includes three pieces of information as instance variables—a first name (type String), a last name (type String) and a monthly salary  (double). Your class should have a constructor that initializes the three instance variables. Provide a set and a get method for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named EmployeeTest that demonstrates class Employee’s capabilities. Create two Employee objects and display each object’s yearly salary. Then give each Employee a 10% raise and display each Employee’s yearly salary again.  **PROGRAM CODE:**  import java.util.Scanner;  class Employees {  String firstname;  String lastname;  double salary;  Scanner sc = new Scanner(System.in);  Employees() {  firstname = "HELLO";  lastname = "hi";  salary = 0;  }  void setdata() {  System.out.println("Enter your first name:");  firstname = sc.nextLine();  System.out.println("Enter your last name:");  lastname = sc.nextLine();  System.out.println("Enter your monthly salary:");  salary = sc.nextDouble();  sc.nextLine(); // consume the leftover newline  if (salary < 0) {  salary = 0.0;  }  }  void display() {  double realsal = setSalary();  System.out.println("The salary of " + firstname + " " + lastname + " with 10% raise is " + realsal);  }  double setSalary() {  return ((salary \* 12) + (salary \* 0.10 \* 12));  }  }  public class prac13 {  public static void main(String[] args) {  int number;  Scanner s = new Scanner(System.in);  System.out.println("Enter total number of Employees:");  number = s.nextInt();  s.nextLine(); // consume the leftover newline  Employees[] emp = new Employees[number];  for (int i = 0; i < number; i++) {  emp[i] = new Employees(); // instantiate each Employees object  emp[i].setdata();  }  for (int i = 0; i < number; i++) {  emp[i].display();  }  System.out.println("23DCS083\_MARK\_PATEL");  s.close(); // close the Scanner object  }  }  **OUTPUT:**    **CONCLUSION:**  The Employee class efficiently manages employee details with proper initialization and validation for monthly salary. The EmployeeTest application demonstrates its capabilities by displaying yearly salaries for two employees, applying a 10% raise, and showing the updated salaries, highlighting the class's practical utility. |
| **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.Scanner;  class Date{  int day;  int month;  int year;  Date(){  day=4;  month = 9;  year=2005;  }  Date(int x,int y,int z){  day=x;  month=y;  year=z;  }  void setdata(){  Scanner sc = new Scanner(System.in);  System.out.println("Enter the day");  day=sc.nextInt();  System.out.println("Enter the month");  month=sc.nextInt();  System.out.println("Enter the year");  year=sc.nextInt();  }  void display(){  System.out.println( day +"/"+ month + "/"+year);  }  }  public class prac14 {  public static void main(String[] args) {  Date obj = new Date();  obj.display();    Date obj1 = new Date(13,12,2024);  obj1.display();    Date obj2 = new Date();  obj2.setdata();  obj2.display();  System.out.println();  System.out.println("23DCS083\_MARK\_PATEL");  }  }  **OUTPUT:**    **CONCLUSION:**  The Date class effectively handles date information with instance variables for month, day, and year, initialized via a constructor. Through setter and getter methods, it ensures flexible manipulation and retrieval of date components. The DateTest application demonstrates the class's functionality by displaying dates formatted with forward slashes, showcasing the class's practical utility. |
| **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;  class Area{  int length;  int breadth;  Area(){    }  Area(int x,int y){  length=x;  breadth=y;  }  int return\_area(){  return length\*breadth;  }  }  public class prac15 {  public static void main(String[] args) {  int len,bre;  Scanner sc = new Scanner(System.in);  System.out.println("Enter the length:");  len=sc.nextInt();  System.out.println("Enter the breadth:");  bre=sc.nextInt();  Area ob = new Area(len,bre);  int res = ob.return\_area();  System.out.println("rectangle area is " +res);  System.out.println();  System.out.println("23DCS083\_MARK\_PATEL");  }  }  **OUTPUT:**    **CONCLUSION:**  The Area class efficiently calculates the area of a rectangle using length and breadth provided via its constructor. With the returnArea method, it simplifies area computation. The program's keyboard input functionality and method demonstration underscore its practical application in geometry calculations. |
| **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.\*;  class Complex{  int realpart;  int imaginarypart;      Complex(){  realpart=2;  imaginarypart=3;  }  void setdata(int x,int y){  realpart=x;  imaginarypart=y;  }    Complex add(Complex o1,Complex o2){  Complex demo = new Complex();  demo.realpart=o1.realpart+o2.realpart;  demo.imaginarypart=o1.imaginarypart+o2.imaginarypart;  return demo;    }  Complex sub(Complex o1,Complex o2){  Complex demo = new Complex();  demo.realpart=o1.realpart-o2.realpart;  demo.imaginarypart=o1.imaginarypart-o2.imaginarypart;  return demo;    }  Complex multi(Complex o1, Complex o2){  Complex demo = new Complex();  demo.realpart = (o1.realpart \* o2.realpart) - (o1.imaginarypart \* o2.imaginarypart);  demo.imaginarypart = (o1.realpart \* o2.imaginarypart) + (o2.realpart \* o1.imaginarypart);  return demo;  }    void display(){  System.out.println("The complex number is "+realpart+" + "+imaginarypart+"i");  System.out.println();  System.out.println("23DCS083\_MARK\_PATEL");  }      };  public class prac16 {    public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  Complex obj1 = new Complex();  int r,i;  System.out.println("Enter the real part ");  r=sc.nextInt();  System.out.println("Enter the imaginary part ");  i=sc.nextInt();  Complex obj2 = new Complex();  obj2.setdata(r, i);  System.out.println("Enter the opertaion you want to perform +,-,\*");  char choice = sc.next().charAt(0);//char at string index 0    switch (choice) {  case '+':  Complex temp = new Complex();  Complex obj3=temp.add(obj1, obj2);  obj3.display();    break;    case '-':  Complex temp1 = new Complex();  Complex obj4=temp1.sub(obj1, obj2);  obj4.display();    break;    case '\*':  Complex temp2 = new Complex();  Complex obj5=temp2.multi(obj1, obj2);  obj5.display();    break;    default:  break;  }    }  }  **OUTPUT:**        **CONCLUSION:**  The Complex class effectively handles arithmetic operations on complex numbers with separate methods for sum, difference, and product. By allowing user input for the real and imaginary parts, the class demonstrates its practicality and versatility in performing complex number calculations. |

**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 parent{        void display(){          System.out.println("This is a parent class");      }  }  class child1 extends parent{      void display1(){          System.out.println("This is a child class");      }  }  public class prac17 {      public static void main(String[] args) {          child1 obj = new child1();            obj.display();          obj.display1();          System.out.println();          System.out.println("23DCS083\_MARK\_PATEL");      }  }  **OUTPUT:**    **CONCLUSION:**  This practical demonstrates the concept of inheritance in Java, where the child1 class inherits the properties and methods of the parent class. By using inheritance, the child1 object can access both its own methods and those of its parent class, showcasing how inheritance promotes code reuse and establishes a hierarchical relationship between 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);          printsalary();      }  }  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);          printsalary();      }  }  public class prac18 {      public static void main(String[] args) {          Employee o1 = new Employee();          o1.setdata();          o1.display();          Manager o2 = new Manager();          o2.setdata();          o2.display();          System.out.println();          System.out.println("23DCS083\_MARK");      }  }  **OUTPUT:**    **CONCLUSION:**  This practical illustrates the concept of inheritance in Java, where common attributes and methods are encapsulated in a base class (Member), and specialized classes (Employee and Manager) extend this base class to include additional functionality. This approach promotes code reusability and better organization, making it easier to manage and extend the program as requirements evolve. |
| **19.** | Create a class named 'Rectangle' with two data members 'length' and 'breadth' and two methods to print the area and perimeter of the rectangle respectively. Its constructor having parameters for length and breadth is used to initialize length and breadth of the rectangle. Let class 'Square' inherit the 'Rectangle' class with its constructor having a parameter for its side (suppose s) calling the constructor of its parent class as 'super(s,s)'. Print the area and perimeter of a rectangle and a square. Also use array of objects.  **PROGRAM CODE:**  import java.util.\*;  class Rectangle  {      float Length;      float Breadth;      public Rectangle()      {          Length = 0;          Breadth = 0;      }      public Rectangle(float Length,float Breadth)      {          this.Length = Length;          this.Breadth = Breadth;      }      void Perimeter()      {          System.out.println("\nPerimeter of Rectangle is : " + 2\*(Length+Breadth));          System.out.println("Perimeter of Square is : " +  4\*Length);      }      void Area()      {          System.out.println("\nArea of Rectangle is : " + Length\*Breadth);          System.out.println("Area of Square is : " + Length\*Length);      }  }  class Square extends Rectangle  {      public Square(float S)      {          super(S,S);      }  }  class prac19  {      public static void main(String[] args)      {          Scanner sc = new Scanner(System.in);          int no\_of\_object;          float L;          System.out.println(" ");          System.out.print("Enter the number of Object's that you want to create : ");          no\_of\_object = sc.nextInt();          Square[] S = new Square[no\_of\_object];          for(int i=0;i<no\_of\_object;i++)          {              System.out.print("\nEnter the Value of Length : ");              L = sc.nextFloat();              S[i] = new Square(L);              S[i].Perimeter();              S[i].Area();          }          sc.close();          System.out.println(" ");          System.out.println(" ");          System.out.println("23DCS083\_MARK");      }  }  **OUTPUT:**    **CONCLUSION:**  This practical demonstrates the use of inheritance in Java to create specialized classes, such as Square, from a general class like Rectangle. The use of the super keyword to call the parent class's constructor shows how derived classes can inherit and extend functionality. |
| **20.** | Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' having a method to print "Square is a rectangle". Now call the method of 'Shape' and 'Rectangle' class by the object of 'Square' class.  **PROGRAM CODE:**  class Shape{       public void print1(){          System.out.println("This is shape");      }  }  class Rectangle extends Shape{      public void print2(){          System.out.println("This is rectagle shape");      }  }  class Circle extends Shape{      public void print3(){      System.out.println("This is circular shape");      }  }  class Square extends Rectangle{      public void print4(){          System.out.println("Square is rectangle");      }  }  public class prac20 {      public static void main(String[] args) {          Square obj = new Square();          obj.print1();          obj.print2();          obj.print4();   System.out.println();          System.out.println("23DCS083\_MARK");      }  }  **OUTPUT:**    **CONCLUSION:**  This practical demonstrates how inheritance allows a class to extend another, inheriting its properties and behaviors. By creating a hierarchy of classes, we observe how method overriding can provide specialized behavior in derived classes. This enhances the design of modular and reusable code in object-oriented programming. |
| **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 print(){          System.out.println("I got a degree");      }  }  class Undergraduate extends Degree{      @Override      public void print(){          System.out.println("I am  an Undergraduate");      }  }  class Postgraduate extends Degree{      @Override      public void print(){          System.out.println("I am  an Postgraduate");      }  }  public class prac21 {      public static void main(String[] args) {          Degree d1 = new Degree();          d1.print();          Undergraduate d2 = new Undergraduate();          d2.print();          Postgraduate d3 = new Postgraduate();          d3.print();          System.out.println();          System.out.println("23DCS083\_MARK");      }  }  **OUTPUT:**    **CONCLUSION:**  This practical demonstrates the concept of inheritance and method overriding in Java. By creating subclasses that override the “print” method, we achieve polymorphism, allowing the method to behave differently depending on the object type. This enhances code reusability and flexibility**.** |

**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.\*;  public class prac24 {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          try{              System.out.print("Enter first number:");              int a = sc.nextInt();              System.out.print("Enter Second number:");              int b = sc.nextInt();              int data = a/b;              System.out.println(data);          }          catch(InputMismatchException e){              System.out.println("the number is non-integer");          }          catch(Exception e){              System.out.println("Any number is not divisible by 0");          }          finally{              sc.close();              System.out.println("23DCS083\_MARK");          }      }  }  **OUTPUT:**    **CONCLUSION:**  This program we learn how to deal with basic exception handling in Java. By catching exceptions such as Arithmetic-Exception for division by zero and general exceptions for invalid input types, it ensures that errors are handled gracefully. The try-catch-finally block provides a structured way to manage errors and ensures the program will run easily and without any exception. |
| **25.** | Write a Java program that throws an exception and catch it using a try-catch block.  **PROGRAM CODE:**  import java.util.\*;  public class prac25 {      public static void main(String[] args) {          int balance = 5000;          Scanner sc = new Scanner(System.in);          System.out.println("Enter Amount to withdraw");          int withdraw = sc.nextInt();          try {              if (balance - withdraw < 1000) {                  throw new Exception("Your WithDraw amount is Greater than your Balance.");              }              else {                  balance = balance - withdraw;              }          }          catch (Exception e) {              e.printStackTrace();            }          finally{              System.out.println("Available balance :"+balance);              System.out.println("23DCS083\_MARK");          }      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program effectively demonstrates exception handling in a banking scenario, where it checks if a withdrawal would leave the account balance below a minimum threshold. If the condition is violated, an exception is thrown to alert the user. The use of a try-catch block ensures that the error is caught and displayed without terminating the program abruptly. Finally, the remaining balance is displayed regardless of the outcome, ensuring important details are always conveyed. |
| **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 prac26 {      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 = 7 / 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("abc3.txt");          } catch (IOException e) {              System.out.println("Checked Exception caught: " + e);          }          try {              Class.forName("prac216");             // System.out.println("Class found");          } catch (ClassNotFoundException e) {              System.out.println("Checked Exception caught: " + e);          }          finally{              System.out.println("23DCS083\_MARK");          }      }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to handle different types of errors in Java using try-catch blocks. It catches unchecked exceptions like dividing by zero and accessing null values, as well as checked exceptions like file handling errors and missing classes. This ensures the program runs smoothly without crashing, even when errors occur, and always executes the final statement. |

**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 prac27 {      public static void main(String[] args) throws Exception {          if (args.length == 0) {              System.out.println("No file Found!");          } else {              for (int i = 0; i < args.length; i++) {                  try {                      BufferedReader f = new BufferedReader(new FileReader(args[i]));                      String j;                      int count = 0;                      while ((j = f.readLine()) != null) {                          count++;                      }                      System.out.println("File name is : " + args[i] + " and Number of lines are : " + count);                  } catch (Exception e) {                      System.out.println(e);                  }              }          }          System.out.println("23DCS083\_MARK");      }  }  **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 prac28{  public static void main(String[] args) {  if (args.length < 2) {  System.out.println("Usage: java prac28 <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("23DCS083\_MARK");  }}  **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 prac29 {  public static void main(String[] args) {  if (args.length < 2) {  System.out.println("Usage: java prac29 <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("23DCS083\_MARK");  } }  **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.\*;  public class prac30 {      public static void main(String[] args) {          // Specify the source and destination file paths          String sourceFilePath = "source.txt";          String destinationFilePath = "destination.txt";          // Use try-with-resources to ensure resources are closed automatically          try (              FileInputStream fis = new FileInputStream(sourceFilePath);              FileOutputStream fos = new FileOutputStream(destinationFilePath)          ) {              int byteContent;              // Read from source and write to destination file byte by byte              while ((byteContent = fis.read()) != -1) {                  fos.write(byteContent);              }              System.out.println("File copied successfully.");          } catch (FileNotFoundException e) {              System.out.println("File not found: " + e.getMessage());          } catch (IOException e) {              System.out.println("Error occurred while copying the file: " + e.getMessage());          }      }  }  **OUTPUT:**      **CONCLUSION:**  This program efficiently copies data from a source file to a destination file in Java, creating the destination file automatically if it doesn't exist. It uses file input and output streams to handle byte-by-byte reading and writing, ensuring proper resource management with try-with-resources. |
| **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 prac31 {      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("23DCS083\_MARK");      }  }  **OUTPUT:**    **CONCLUSION:**  The program reads user input and writes it to a file called "output.txt." It uses BufferedReader and BufferedWriter for efficient input and output handling. The process stops when the user types "exit." This demonstrates simple file handling in Java |

**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:**  class Myclass extends Thread{      public void run(){          System.out.println("Hello world");      }  }  class Myclass1 implements Runnable{      public void run(){          System.out.println("Hello world 2");      }  }  public class prac32 {      public static void main(String[] args) {          Myclass thread = new Myclass();          thread.start();          Myclass1 runnable = new Myclass1();          Thread thread1 = new Thread(runnable);          thread1.start();          System.out.println("23DCS083\_MARK");        }  }  **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.\*;  class SumTask implements Runnable {      private int start;      private int end;      private int[] result;      private int index;      public SumTask(int start, int end, int[] result, int index) {          this.start = start;          this.end = end;          this.result = result;          this.index = index;      }      @Override      public void run() {          int sum = 0;          for (int i = start; i <= end; i++) {              sum += i;          }          result[index] = sum;      }  }  public class prac33 {      public static void main(String[] args) {          if (args.length < 2) {              System.out.println("Please provide two arguments: N and the number of threads.");              return;          }          int N = Integer.parseInt(args[0]);          int numThreads = Integer.parseInt(args[1]);          int[] result = new int[numThreads];          int range = N / numThreads;          int remainder = N % numThreads;          Thread[] threads = new Thread[numThreads];          int start = 1;          for (int i = 0; i < numThreads; i++) {              int end = start + range - 1;              if (i == numThreads - 1) {                  end += remainder;              }              threads[i] = new Thread(new SumTask(start, end, result, i));              threads[i].start();              start = end + 1;          }          try {              for (Thread thread : threads) {                  thread.join();              }          } catch (InterruptedException e) {              System.out.println("Thread interrupted: " + e.getMessage());          }          int finalSum = 0;          for (int sum : result) {              finalSum += sum;          }          System.out.println("The sum of the first " + N + " numbers is: " + finalSum);          System.out.println(" ");          System.out.println("23DCS083\_MARK");      }  }  **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 java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.  **PROGRAM CODE:**  class Thread1 extends Thread {      public void run() {          int i=0;          while (i<4) {              int n = (int) (Math.random() \* 100);              System.out.println("Generated number: " + n);              if (n % 2 == 0) {                  // Create and start Thread2 for even number                  new Thread2(n).start();              } else {                  // Create and start Thread3 for odd number                  new Thread3(n).start();              }              try {                  Thread.sleep(1000); // Sleep for 1 second              } catch (InterruptedException e) {                  e.printStackTrace();              }              i++;          }      }  }  class Thread2 extends Thread {      private int n;      Thread2(int n) {          this.n = n;          setName("EvenThread"); // Set a meaningful name      }      public void run() {          System.out.println(getName() + ": Square of " + n + " is " + (n \* n));      }  }  class Thread3 extends Thread {      private int n;      Thread3(int n) {          this.n = n;          setName("OddThread"); // Set a meaningful name      }      public void run() {          System.out.println(getName() + ": Cube of " + n + " is " + (n \* n \* n));      }  }  public class prac34 {      public static void main(String[] args) {          Thread1 t1 = new Thread1();          t1.start(); // Start the first thread          System.out.println("23DCS083\_MARK");      }  }  **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:**  class Thread1 extends Thread {      public void run() {          int i=0,n = 0;          while (i<7) {              n++;              System.out.println(n);              try {                  Thread.sleep(1000); // Sleep for 1 second              } catch (InterruptedException e) {                    System.out.println("Thread interrupted.");              }              i++;          }      }  }  public class prac35 {      public static void main(String[] args) {          Thread1 t1 = new Thread1();          t1.start(); // Start the thread          System.out.println("23DCS083\_MARK");      }  }  **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 T extends Thread {      public T(String s) {          super(s);      }      public void run() {          System.out.println("This is run method :" + getName());      }  }  public class prac36 {      public static void main(String[] args) {          T FIRST = new T("FIRST");          T SECOND = new T("SECOND");          T THIRD = new T("THIRD");          System.out.println("defualt priority:");          System.out.println("FIRST:" + FIRST.getPriority());          System.out.println("SECOND:" + SECOND.getPriority());          System.out.println("THIRD:" + THIRD.getPriority());          FIRST.setPriority(3);          SECOND.setPriority(5);          THIRD.setPriority(7);          System.out.println("updated priority:");          System.out.println("FIRST:" + FIRST.getPriority());          System.out.println("SECOND:" + SECOND.getPriority());          System.out.println("THIRD:" + THIRD.getPriority());          THIRD.start();          SECOND.start();          FIRST.start();          System.out.println("23DCS083\_MARK");      }  }  **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("23DCS083\_MARK");      }  }  **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 implements following functionalities of stack. My Stack  -list ArrayList<Object>: A list to store elements.  +isEmpty: boolean: Returns true if this stack is empty.  +getSize(): int: Returns number of elements in this stack.  +peek(): Object: Returns top element in this stack without  removing it.  +pop(): Object: Returns and Removes the top elements in  this stack.  +push(o: object): Adds new element to the top of this  stack.  **PROGRAM CODE:**  import java.util.\*;  class StackList{      private ArrayList<Integer> STL;      public StackList(){          STL = new ArrayList<>();      }      public boolean isEmpty() {          return STL.isEmpty();      }      public void push(Integer value) {            STL.add(value);          //updateFile();      }      public void pop(){          if(STL.isEmpty()){              System.out.println("Stack is empty");              //return null;          }          else{               STL.remove(STL.size()-1);          }      }      public Integer peek(){          if(STL.isEmpty()){              System.out.println("Stack is empty");              return null;          }          else{              return STL.get(STL.size()-1);          }      }      public void display(){          System.out.println(STL);      }      public void clr(){          STL.clear();      }      public void size1(){          System.out.println(STL.size());      }     public void updatafile(){       }  }  public class prac38 {      public static void main(String[] args) {          StackList stack = new StackList();          stack.push(10);          stack.push(20);          stack.push(70);          stack.push(40);          stack.push(30);          stack.display();          System.out.println("Top element: " + stack.peek()); // Outputs: 30          stack.pop();          stack.display();            // System.out.println("Popped element: " + stack.pop()); // Outputs: 30          System.out.println("Top element after pop: " + stack.peek()); // Outputs: 20          System.out.println("23DCS083\_MARK");      }  }  **OUTPUT:**    **CONCLUSION:**  This program implements a stack using an ArrayList, providing basic stack operations like push, pop, peek, and clear. It allows adding and removing elements from the stack and displays the current stack contents. The program efficiently handles stack operations and checks for underflow when the stack is empty. |
| **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;  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;      }  }  public class prac39 {      public static <T extends Comparable<T>> void sortArray(T[] array) {          Arrays.sort(array);      }  public static void main(String[] args) {          Integer[] numbers = { 8, 3, 19, 13, 7 ,2};          System.out.println("Before sorting (Integers): " + Arrays.toString(numbers));          sortArray(numbers);          System.out.println("After sorting (Integers): " + Arrays.toString(numbers));          String[] names = { "Cristiano", "Alice", "Marco", "Messi" };          System.out.println("\nBefore sorting (Strings): " + Arrays.toString(names));          sortArray(names);          System.out.println("After sorting (Strings): " + Arrays.toString(names));          Product[] products = {                  new Product("Laptop", 700),                  new Product("Phone", 550),                  new Product("Tablet", 540),                  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("23DCS083\_MARK");      }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates generic sorting by using Java's Comparable interface. It sorts arrays of integers, strings, and custom Product objects based on price in ascending order. By leveraging the Arrays.sort() method, it efficiently arranges elements and displays the sorted results. It provides a versatile approach to sorting different types of objects. |
| **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 prac40 {  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("23DSC083\_MARK");  } } }  **OUTPUT:**    **CONCLUSION:**  This program takes a text input from the user, counts the occurrences of each word, and displays the results in alphabetical order. It uses a TreeMap to store words, ensuring automatic sorting by key. The program efficiently processes text by splitting it into words and counting their frequency. |
| **41.** | Write a code which counts the number of the keywords in a Java source file. Store all the keywords in a HashSet and use the contains () method to test if a word is in the keyword set.  **PROGRAM CODE:**  import java.io.\*;  import java.util.\*;  public class P41 {  private static final HashSet<String> keywords = new HashSet<>();  static {  String[] keywordArray = {  "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class",  "const", "continue", "default", "do", "double", "else", "enum", "extends", "final",  "finally", "float", "for", "goto", "if", "implements", "import", "instanceof", "int",  "interface", "long", "native", "new", "package", "private", "protected", "public",  "return", "short", "static", "strictfp", "super", "switch", "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);  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("23DCS083\_MARK");  } }  **OUTPUT:**    **CONCLUSION:**  This program reads a Java source file and counts the number of Java keywords it contains. By utilizing a predefined set of keywords, it efficiently scans through the file and outputs the total count. The program also handles file not found errors gracefully. |