

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

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

s

# Department of Computer Science & Engineering

**Subject Name: Java programming**

**Semester: III**

# Subject Code: CSE201 Academic year: 2024-25

Part - 1

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| **2.**  **3.**    **4.**    **5.**        **6.** | Imagine you are developing a simple banking application where you need to display the current balance of a user account. For simplicity, let's say the current balance is $20. Write a java program to store this balance in a variable and then display it to the user.  **PROGRAM CODE:**  class demo  {  public static void main(String a[])  {  int x=20;  System.out.println("The current balance is $"+x);  }  }  **OUTPUT:**    **CONCLUSION:**  The variable is stored in x and the function println() displays its value.  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;  class speed  {  public static void main(String []args)  {  Scanner s = new Scanner(System.in);    System.out.println("Enter the distance: ");  float d=s.nextFloat();    System.out.println("Enter time in hr,min,sec: ");  float hr=s.nextFloat();  float min=s.nextFloat();  float sec=s.nextFloat();    float t=(hr\*60\*60)+(min\*60)+sec;  float speed=d/t;  System.out.println("Speed in m/s is "+speed);    float sk=speed\*(18/5);  System.out.println("Speed in km/h is "+sk);    System.out.println("Speed in mi/h is "+(sk/1.609));    }  }  **OUTPUT:**    **CONCLUSION:**  We take the user’s input for distance (in meters), hours, minutes, and seconds. We calculate the total time in seconds. We compute the speed in meters per second, kilometres per hour, and miles per hour using the given formulas.  Imagine you are developing a budget tracking application. You need to calculate the total expenses for the month. Users will input their daily expenses, and the program should compute the sum of these expenses. Write a Java program to  calculate the sum of elements in an array representing daily expenses.  **PROGRAM CODE:**  import java.util.Scanner;  class budget  {  public static void main(String args[])  {  Scanner s=new Scanner(System.in);  System.out.println("Enter the number of days you want to enter the amount for: ");  int n=s.nextInt();  float a[]=new float [n];  int i,sum=0;  System.out.println("enter the values: ");  for(i=0;i<n;i++)  {  a[i]=s.nextFloat();  sum+=a[i];  }  System.out.println("Sum of the amount is "+sum);    }  }  **OUTPUT:**    **CONCLUSION:**  This program efficiently calculates the total expenses for a month based on daily inputs using an array. It demonstrates basic array handling and iteration in Java.  **Supplementary Experiment:**  You are creating a library management system. The library has two separate lists of books for fiction and non-fiction. The system should merge these lists into a single list for inventory purposes. Write a Java program to merge two  arrays.  **PROGRAM CODE:**  public class supp1  {  public static void main(String[] args)  {  int[] arr1 = { 1, 3, 4, 5, 9};  int[] arr2 = { 2, 4, 6, 8, 7};  int l1 = arr1.length;  int l2 = arr2.length;  int result = l1 + l2;    System.out.println("Array 1: ");  for(int i=0;i<l1;i++)  {  System.out.print(arr1[i]+" ");  }  System.out.println();    System.out.println("Array 2: ");  for(int i=0;i<l2;i++)  {  System.out.print(arr2[i]+" ");  }  System.out.println();  int[] mergearray = new int[result];  for(int i=0;i<l1;i++)  {  mergearray[i]=arr1[i];  mergearray[i+5]=arr2[i];  }    System.out.println("Merged Array:");  for (int i = 0; i < result; i++)  {  System.out.print(mergearray[i] + " ");  }  }  }  **OUTPUT:**    **CONCLUSION:**  Hence, we successfully merged the two arrays.  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.  **Top of Form**  **Bottom of Form**  **PROGRAM CODE:**  import java.util.Scanner;  class app  {  public static void main(String args[])  {  Scanner s=new Scanner(System.in);  System.out.println("Enter the choice of code from the following: ");  System.out.println("1:Motor");  System.out.println("2:Fan");  System.out.println("3:Tube");  System.out.println("4:Wires");  System.out.println("5:Others");  System.out.println("6:Exit");  int ch=s.nextInt();  double price=100;  switch(ch)  {  case 1:  price+=8;  System.out.println("The total amount of motor is "+price);  break;  case 2:  price+=12;  System.out.println("The total amount of fan is "+price);  break;  case 3:  price+=5;  System.out.println("The total amount of tube is "+price);  break;  case 4:  price+=7.5;  System.out.println("The total amount of wires is "+price);  break;  case 5:  price+=3;  System.out.println("The total amount of others is "+price);  break;  default:  break;  }  }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to use a switch statement in Java to calculate a bill based on product codes and prices, applying specific tax rates depending on the product type.    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.lang.\*;  import java.util.Scanner;  class fseries  {  public static void main(String []args)  {  Scanner s = new Scanner(System.in);  System.out.println("Enter the numer of days you want to work out: ");  int n=s.nextInt();  int n1=0,n2=1,i,n3;  System.out.print(n1+" "+n2);  for(i=2;i<n;i++)  {  n3=n1+n2;  System.out.print(" "+n3);  n1=n2;  n2=n3;  }  }  }  **OUTPUT:**    **CONCLUSION:**  This program successfully creates and exercise routine based on the Fibonacci series and displays the formatted output to the user.  **Supplementary Experiment:**  Imagine you are developing a classroom management system. You need to keep track of the grades of students in a class. After collecting the grades, you want to display each student's grade along with a message indicating if they have  passed or failed. Let's assume the passing grade is 50.  **PROGRAM CODE:**  import java.util.Scanner;  public class supp2  {  public static void main(String[] args)  {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter the number of students: ");  int numStudents = scanner.nextInt();  int passingGrade = 50;  for (int i = 1; i <= numStudents; i++)  {  System.out.print("Enter grade for student " + i + ": ");  int studentGrade = scanner.nextInt();  if (studentGrade >= passingGrade)  {  System.out.println("Student " + i + ": Passed");  }  else  {  System.out.println("Student " + i + ": Failed");  }  }  }  }  **OUTPUT:**    **CONCLUSION:**  This Java program effectively manages and displays students' grades along with their pass/fail status based on a predefined passing grade condition.  **Top of Form**  **Bottom of Form** |

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

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

Department of Computer Science & Engineering

Subject Name: Java Programming

Semester: 3

Subject Code: CSE 201

Academic year: 2024-2025

Part - 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.Scanner;  public class Front\_times {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  System.out.println("Enter string :");  String st=sc.next();  System.out.println("Enter number of times that you want to print the string :");  int n = sc.nextInt();  for(int i=0;i<n;i++){  System.out.print(st.substring(0,3));  }  }  }  **OUTPUT:**      **CONCLUSION:**  The provided Java code takes a string and a non-negative integer as input from the user. It then prints the first three characters of the input string repeatedly for the number of times specified by the user. |
| 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 count {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);  int arr[]=new int[3];  int count=0;  System.out.println("Enter the elements of an array :");  for (int i =0;i<3;i++)  {  arr[i]=sc.nextInt();  }  for (int i=0;i<3;i++)  {  if(arr[i]==9){  count++;  }  }  System.out.println("Number 9 appears "+ count + " times ");  }  }  **OUTPUT:**    **CONCLUSION:**  This Java program reads three integers from the user into an array and counts how many times the number 9 appears in the array. It demonstrates basic array operations, user input handling, and simple counting logic. The program outputs the count of the number 9 after processing the input. |

|  |  |
| --- | --- |
| 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.sql.SQLOutput;  import java.util.\*;  public class practical9 {  static StringBuffer print ()  {  char s1;  Scanner sc = new Scanner(System.in);  System.out.println("Enter string :");  StringBuffer st = new StringBuffer(sc.nextLine());  System.out.println("Enter number of times that you want to print :");  int n = sc.nextInt();  StringBuffer value = new StringBuffer();  for (int i = 0; i<st.length(); i++) {  s1 = st.charAt(i);  for (int j = 0; j < n; j++) {  value = value.append(s1);  }  }  return value;  }  public static void main(String[] args) {  StringBuffer s2 = new StringBuffer(print());  System.out.println(s2);  }  }  **OUTPUT:**      **CONCLUSION:**  This Java program takes a string input from the user and a number specifying how many times each character in the string should be repeated. It then prints the modified string with each character repeated the specified number of times. The program demonstrates the use of StringBuffer for efficient string manipulation and handling user input. |
| 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:**  public class practical10 {  public static void main(String[] args) {  String st = "Tirth";  char[] arr = new char[6];  StringBuffer sc = new StringBuffer(st);  System.out.println(st.length());  System.out.println(st.toLowerCase());  System.out.println(st.toUpperCase());  for (int i = 0; i < 5; i++) {  arr[i] = sc.charAt(i);  }  for (int i = 0; i < 5; i++) {  for (int j = 0; j < 5 - i; j++) {  if (arr[j] > arr[j + 1])  {  char temp = arr[j];  arr[j] = arr[j + 1];  arr[j + 1] = temp;  }  }  }  for (int i = 0; i < 6; i++) {  if(arr[i]!='\0') {  System.out.println(arr[i]);  }  }  }  }  **OUTPUT:**    **CONCLUSION:**  This Java program performs several operations on the string "sakina". It demonstrates string length retrieval, conversion to lowercase and uppercase, and sorting characters in ascending order using a bubble sort algorithm. Finally, it prints the sorted characters, handling the array size slightly beyond the string length. The program showcases basic string manipulation techniques and sorting algorithms 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 practical11 {  public static void main(String[] args) {  String st = "CHARUSAT UNIVERSITY";  char[] arr = new char[6];  StringBuffer sc = new StringBuffer(st);  System.out.println(st.length());  System.out.println(st.replace('H','M'));  System.out.println(st.toLowerCase());  }  }  **OUTPUT:**    **CONCLUSION:**  This Java program illustrates fundamental string manipulations using the string "CHARUSAT UNIVERSITY". It showcases operations like retrieving the string length, replacing characters ('H' with 'M'), and converting the entire string to lowercase. These operations highlight basic string handling capabilities in Java, useful for various text processing tasks in programming. |

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

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

**Department of Computer Science & Engineering**

**Subject Name: JAVA PROGRAMMING**

**Semester: 3**

**Subject Code: CSE201**

**Academic year:2024-25**

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

|  |  |
| --- | --- |
| **12.** | Imagine you are developing a currency conversion tool for a travel agency. This tool should be able to convert an amount in Pounds to Rupees. For simplicity, we assume the conversion rate is fixed: 1 Pound = 100 Rupees. The tool should be able to take input both from command-line arguments and interactively from the user.  **PROGRAM CODE:**  import java.util.\*;  public class Pra {  public static void main(String[] args) {  if (args.length == 1) {  // Command-line argument  double amount = Double.parseDouble(args[0]);  double rupees = amount\*100;  System.out.println(amount + " Pounds = " + rupees + " Rupees");  }  else {  // Interactive mode  Scanner scanner = new Scanner(System.in);  System.out.print("Enter amount in Pounds: ");  double amount = scanner.nextDouble();  double rupees = amount\*100;  System.out.println(amount + " Pounds = " + rupees + " Rupees");  }  }  }  **OUTPUT:**    **CONCLUSION:**  The Java program converts Pounds to Rupees at a fixed rate of 1 Pound = 100 Rupees, handling both command-line arguments and interactive user inputs effectively, ensuring user-friendly currency conversion. |
| **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.\*;  class Employee{  Scanner sc=new Scanner(System.in);  String fs=" ";  String ls=" ";  double sal;  Employee(String f,String l,double s){  fs=f;  ls=l;  sal=s;  }  void setfs(){  fs=sc.nextLine();  }  void setls(){  ls=sc.nextLine();  }  void setsal(){  sal=sc.nextDouble();  if(sal<0){  sal=0.0;  }  else{  sal=sal+(sal\*0.10);  }  }  String getfs(){  return fs;  }  String getls(){  return ls;  }  double getsal(){  return sal;  }  }  class EmpT{  public static void main(String args[]){  Employee E1=new Employee("","",0.0);  System.out.println("Enter First name:");  E1.setfs();  System.out.println("enter last name:");  E1.setls();  System.out.println("enter salary:");  E1.setsal();  System.out.println("Employee name:"+E1.getfs()+" "+E1.getls());  System.out.println("Salary:"+E1.getsal());  }  }  **OUTPUT:**    **CONCLUSION:**  The Employee class initializes and updates employee data through setters and getters. The EmpT class demonstrates capturing employee details and adjusting salaries, showcasing the class's functionality effectively. |
| **14** | Create a class called Date that includes three pieces of information as instance variables—a month (type int), a day (type int) and a year (type int). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test application named DateTest that demonstrates class Date’s capabilities.  **PROGRAM CODE:**  import java.util.\*;  class Date{  Scanner sc=new Scanner(System.in);  int d;  int m;  int y;  Date(int day, int month ,int year){  d=day;  m=month;  y=year;  }  void setd(){  d=sc.nextInt();  if(d>31){  System.out.println("enter right day");  }  }  void setm(){  m=sc.nextInt();  if(m>12){  System.out.println("enter right month");  }  }  void sety(){  y=sc.nextInt();  }  int getd(){  return d;  }  int getlm(){  return m;  }  int gety(){  return y;  }  void displayDate(){  System.out.println("\nDate:"+d+"/"+m+"/"+y);  }  }  class DateTest{  public static void main(String args[]){  Date D=new Date(0,0,0);  System.out.println("Enter day:");  D.setd();  System.out.println("enter month:");  D.setm();  System.out.println("enter year:");  D.sety();  D.displayDate();  }  }  **OUTPUT:**    **CONCLUSION:**  The Date class successfully manages date information with proper setters, getters, and a display method. The DateTest application effectively demonstrates the class's ability to handle and display date information. |
| **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.\*;  class Area{  int l, b;  Scanner sc=new Scanner(System.in);  Area(){  }  Area(int length, int breadth){  b=breadth;  l=length;  }  void setl(){  l=sc.nextInt();  }  void setb(){  b=sc.nextInt();  }  void returnArea(){  System.out.println(l\*b);  }  }  class AreaT{  public static void main(String args[]){  Area a=new Area(0,0);  System.out.print("Enter the length:");  a.setl();  System.out.print("Enter the Breadth:");  a.setb();  System.out.print("Area of rectangle:");  a.returnArea();  System.out.println("23DCS027 Tirth Ganatra ");  }  }  **OUTPUT:**    **CONCLUSION:**  The program effectively calculates and prints the area of a rectangle using the Area class, which initializes length and breadth via constructor parameters and provides a method to return the calculated area. |
| **16** | Print the sum, difference and product of two complex numbers by creating a class named ‘Complex’ with separate methods for each operation whose real and imaginary parts are entered by user.  **PROGRAM CODE:**  import java.util.Scanner;  class Complex {  private double real;  private double imag;  public Complex(double real, double imag) {  this.real = real;  this.imag = imag;  }  public Complex add(Complex other) {  double real = this.real + other.real;  double imag = this.imag + other.imag;  return new Complex(real, imag);  }  public Complex subtract(Complex other) {  double real = this.real - other.real;  double imag = this.imag - other.imag;  return new Complex(real, imag);  }  public Complex multiply(Complex other) {  double real = this.real \* other.real - this.imag \* other.imag;  double imag = this.real \* other.imag + this.imag \* other.real;  return new Complex(real, imag);  }  public String toString() {  return real + " + " + imag + "i";  }  }  public class ComplexI {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter real part of first complex number: ");  double real1 = scanner.nextDouble();  System.out.print("Enter imaginary part of first complex number: ");  double imag1 = scanner.nextDouble();  System.out.print("Enter real part of second complex number: ");  double real2 = scanner.nextDouble();  System.out.print("Enter imaginary part of second complex number: ");  double imag2 = scanner.nextDouble();  Complex num1 = new Complex(real1, imag1);  Complex num2 = new Complex(real2, imag2);  Complex sum = num1.add(num2);  Complex difference = num1.subtract(num2);  Complex product = num1.multiply(num2);  System.out.println("Sum: " + sum);  System.out.println("Difference: " + difference);  System.out.println("Product: " + product);  }  }  **OUTPUT:**  **CONCLUSION:**  The Complex class correctly performs addition, subtraction, and multiplication of two complex numbers, with methods for each operation. User input for real and imaginary parts demonstrates the class's functionality. |

Java Programming [CSE201] Enrolment No.: 23DCS027

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

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

Department of Computer Science & Engineering

Subject Name: Java Programming

Semester: 3

Subject Code: 203

Academic year: 2024-2025

Part - 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 {      public void displayparent() {          System.out.println("this is parent class");      }  }  class child extends parent {      public void displaychild() {          System.out.println("this is child class");  }  }  public class Practical17 {      public static void main(String[] args) {          parent p1=new parent();          child c1= new child();          System.out.println("calling parent class method");          p1.displayparent();          System.out.println("calling inherited method from child class");          c1.displayparent();          System.out.println("calling child class method");          c1.displaychild();      }  }  **OUTPUT:**    **CONCLUSION:**  By performing this experiment we learnt how to use the concept of inheritance in java language. It uses the extends keyword to inherit a base class into a derived class, and the object of the derived class can access all the methods and data of the base class provided that they are inherited publicly or protected. |
| 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:**  import java.util.\*;  class Member {  Scanner sc = new Scanner(System.in);  int age;  int phone;  String name;  String address;  int salary;  public void printSalary() {  System.out.println("The salary of the member is" + this.salary);  }  public void getdata() {  System.out.println("enter the age of member");  this.age=sc.nextInt();  System.out.println("enter the Name of member");  this.name=sc.next();  System.out.println("enter the phone number of member");  this.phone=sc.nextInt();  System.out.println("enter the address of member");  this.address=sc.next();  System.out.println("enter the salary of member");  this.salary=sc.nextInt();  }  public void printdata() {  System.out.println("The name of the member is " + this.name);  System.out.println("The age of the member is " + this.age);  System.out.println("The phone number of the member is " + this.phone);  System.out.println("The address of the member is " + this.address);  System.out.println("The salary of the member is "+this.salary);  }  }  class employee extends Member {  String specialization;  }  class manager extends Member {  // String specialization;  String department;  }  public class Practical18 {  public static void main(String[] args) {  employee e1 = new employee();  manager m1 = new manager();  e1.getdata();  e1.printdata();  m1.getdata();  m1.printdata();  }  }  **OUTPUT:**    **CONCLUSION:**  By performing this experiment we were able to create constructors and methods to input data for data members of derived class and base class. And further we can extend by creating a method which can display the data. |
| 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:**  class Rectangle {      int length, breadth;      public Rectangle(int l, int b) {          this.length = l;          this.breadth = b;      }      public void area() {          int area = this.length \* this.breadth;          System.out.println("the area is " + area);      }      public void perimeter() {          int perimeter = 2 \* (this.length + this.breadth);          System.out.println("the perimeter is " + perimeter);      }  }  class Square extends Rectangle {      public Square(int x) {          super(x, x);      }  }    public class Practical19 {      public static void main(String[] args) {          Square[] s1 = new Square[2];          s1[0] = new Square(10);            s1[1] = new Square(5);          s1[0].area();          s1[0].perimeter();          s1[1].area();          s1[1].perimeter();      }    }  **OUTPUT:**    **CONCLUSION:**  By this experiment we learned how constructors work in inheritance and how they are called. They are called by using the super keyword and passing the parameter given to the constructor of the derived class to the constructor of the base class. |
| 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:**  class Shape {      public void displayshape(){          System.out.println("This is Shape");      }  }  class Rectangle extends Shape {      public void displayRectangle() {          System.out.println("this is Rectangular shape");      }    }  class Circle extends Shape {      public void displaycircle() {          System.out.println("this is circular shape");      }  }  class Square extends Rectangle {      public void displaysquare() {          System.out.println("Square is rectangle");      }  }    public class Practical20 {      public static void main(String[] args) {          Square s1 = new Square();          s1.displayRectangle();          s1.displayshape();      }  }  **OUTPUT:**    **CONCLUSION:**  By performing this program we learnt how to perform hierarchal inheritance in java. |
| 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:**  class Degree {      public void getDegree() {          System.out.println("i got a degree");      }  }  class Undergraduate extends Degree {      public void getDegree() {          System.out.println("i am undergraduate");      }  }  class Postgraduate extends Degree {      public void getDegree() {          System.out.println("i am postgraduate");      }  }  public class Practical21 {      public static void main(String[] args) {          Degree d1 = new Degree();          Undergraduate u1 = new Undergraduate();          Postgraduate p1 = new Postgraduate();          d1.getDegree();          u1.getDegree();          p1.getDegree();        }  }  **OUTPUT:**    **CONCLUSION:**  In this practical we see another example of inheritance. |
| 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:**  import java.util.\*;  import java.io.\*;  interface AdvancedArithmetic{  int divisor\_sum(int n);  }  class calledMyCalculator implements AdvancedArithmetic{  public int divisor\_sum(int n){  int sum = 0;  for(int i = 1; i <= n; i++){  if(n % i == 0){  sum += i;  }  }  return sum;    }  }  public class Practical22 {  public static void main(String[] args){  Scanner sc = new Scanner(System.in);  calledMyCalculator s = new calledMyCalculator();  System.out.println("Enter the number: ");  int n = sc.nextInt();  System.out.println("The sum of the divisors of the number is: " + s.divisor\_sum(n));        }    }  **OUTPUT:**  **CONCLUSION:**  By this experiment I learnt how to use | |
| 23 | Assume you want to capture shapes, which can be eithercircles (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:**  // Shape.java  interface Shape {  String getColor();  double getArea();    default void printDetails() {  System.out.println("Color: " + getColor());  System.out.println("Area: " + getArea());  }  }  // Circle.java  class Circle implements Shape {  private double radius;  private String color;  public Circle(double radius, String color) {  this.radius = radius;  this.color = color;  }  @Override  public String getColor() {  return color;  }  @Override  public double getArea() {  return Math.PI \* radius \* radius;  }  }  // Rectangle.java  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;  }  @Override  public String getColor() {  return color;  }  @Override  public double getArea() {  return length \* width;  }  }  // Sign.java  class Sign {  private Shape backgroundShape;  private String text;  public Sign(Shape backgroundShape, String text) {  this.backgroundShape = backgroundShape;  this.text = text;  }  public void display() {  System.out.println("Sign Text: " + text);  System.out.println("Background Shape Details:");  backgroundShape.printDetails();  }  }  // Main.java  public class P23 {  public static void main(String[] args) {  Shape circle = new Circle(5, "Red");  Shape rectangle = new Rectangle(4, 6, "Blue");    Sign sign1 = new Sign(circle, "Welcome to the Campus!");  Sign sign2 = new Sign(rectangle, "Library Entrance");  System.out.println("Sign 1:");  sign1.display();  System.out.println("\nSign 2:");  sign2.display();  }  }  **OUTPUT:**    **CONCLUSION:**  This program showcases the use of interfaces and polymorphism in Java by defining a Shape interface with methods to get color and calculate area. It demonstrates how different shapes like Circle and Rectangle implement this interface. |

13

**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: 2024-2025**

**Part – 5[Exception Handling]**

|  |  |
| --- | --- |
| **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.lang.\*;  import java.util.Scanner;  class prc24  {  public static void main(String args[])  {  System.out.println("Enter two numbers: ");  Scanner s=new Scanner(System.in);  int a=s.nextInt();  int b=s.nextInt();  if(b==0)  {  try  {  int c=a/b;  System.out.println(c);  }  catch(ArithmeticException ae)  {  System.out.println("Exception handled. "+ae.toString());  }  }  else  {  int c=a/b;  System.out.println(c);  }    }  }  **OUTPUT:**    **CONCLUSION:**  By this code, we can conclude that arithmetic exception is implicitly thrown by the statement and is handled by the try-catch block so that the program doesn’t terminate abruptly. |
| **25.** | Write a Java program that throws an exception and catch it using a try-catch block.  **PROGRAM CODE :**  public class prc25  {  public static void main(String[] args)  {  try  {  throwMethod();  }  catch(Exception e)  {  System.out.println(e);  }  }  static void throwMethod()  {  throw new ArithmeticException("User Thrown Exception");  }  }  **OUTPUT:**    **CONCLUSION:**  By this experiment, we learn to catch an exception using the try-catch block. |
| **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.\*;  class CheckedException extends Exception{  CheckedException(String Message){  super(Message);  }  CheckedException(){  System.out.println("Default Constructor of Checked Exception.");  }  }  class UncheckedException extends RuntimeException{  UncheckedException(String Message){  super(Message);  }  UncheckedException(){  System.out.println("Default Constructor of Unchecked Exception.");  }  }  public class prc26{  public static void main(String[] args)  {  try  {  int x,y,z;  x=y=0;  z=x/y;  }  catch(Exception e)  {  System.out.println(e);  }  try  {  throwCNFException();  }  catch (ClassNotFoundException e)  {  System.out.println(e);  }  try  {  int[] a={1,2,3};  System.out.println(a[10]);  }  catch(Exception e)  {  System.out.println(e);  }  try  {  throwFNFException();  }  catch (FileNotFoundException e)  {  System.out.println(e+"\n");  }  try  {  throwCheckedException("User-Defined Checked Exception - 1");  }  catch(Exception e)  {  System.out.println(e);  }  try  {  throwCheckedException("User-Defined Checked Exception - 2");  }  catch(Exception e)  {  System.out.println(e);  }  try  {  throw new CheckedException();  }  catch(Exception e)  {  e.getMessage();  }  try  {  throwUncheckedException("User-Defined Unchecked Exception - 1");  }  catch(Exception e)  {  System.out.println(e);  }  try  {  throwUncheckedException("User-Defined Unchecked Exception - 2");  }  catch(Exception e)  {  System.out.println(e);  }  }  static void throwCNFException() throws ClassNotFoundException{  throw new ClassNotFoundException("Pre-Defined Checked Exception : Class Not Found Exception");  }  static void throwFNFException() throws FileNotFoundException{  throw new FileNotFoundException("Pre-Defined Checked Exception : File Not Found Exception");  }  static void throwCheckedException(String message) throws CheckedException{  throw new CheckedException(message);  }  static void throwUncheckedException(String message) {  throw new UncheckedException(message);  }  }    **OUTPUT:**    **CONCLUSION:**  Hereby, we learnt about checked and unchecked exceptions.  **Supplementary experiment:**  **PROGRAM CODE:**  import java.util.Scanner;  class DuplicateNumberException extends Exception {  DuplicateNumberException(String message) {  super(message);  }  }  public class supp5 {  public static void checkForDuplicates(int[] numbers) throws DuplicateNumberException {  for (int i = 0; i < numbers.length; i++) {  for (int j = i + 1; j < numbers.length; j++) {  if (numbers[i] == numbers[j]) {  throw new DuplicateNumberException("Duplicate number found: " + numbers[i]);  }  }  }  }  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter the number of integers: ");  int count = scanner.nextInt();  int[] numbers = new int[count];  System.out.println("Enter the integers:");  for (int i = 0; i < count; i++) {  numbers[i] = scanner.nextInt();  }  try {  checkForDuplicates(numbers);  System.out.println("All numbers are unique.");  } catch (DuplicateNumberException e) {  System.out.println(e.getMessage());  }  }  }  **OUTPUT:** |
| **1.**  **2.** | **EXTRA QUESTIONS**  **CODE:**  import java.util.Scanner;  class InsufficientBalanceException extends Exception {  InsufficientBalanceException(String message) {  super(message);  }  }  public class insufficientbank {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);    double balance;  System.out.print("Enter your account balance: ");  balance = scanner.nextDouble();  System.out.print("Enter the amount to withdraw: ");  double amount = scanner.nextDouble();  try {  if (balance < amount) {  throw new InsufficientBalanceException("Insufficient balance in your account");  }  balance -= amount;  System.out.println("Withdrawal successful. New balance: " + balance);    } catch (InsufficientBalanceException e) {  System.out.println("Error: " + e.getMessage());  }  }  }  **OUTPUT:**    **CODE:**  mport java.util.\*;  class InvalidAgeException extends Exception  {  InvalidAgeException(String s)  {  super(s);  }  }  class extra2  {  public static void main(String []args)  {  Scanner sc=new Scanner(System.in);  System.out.println("Enter your age:");  int age=sc.nextInt();  if(age<18)  {  try  {  throw new InvalidAgeException("Not eligible for voting in 2024");  }  catch(InvalidAgeException e)  {  System.out.println(e.getMessage());  System.out.println("Exception caught successfully");  }  }  else  {  System.out.println("Eligible for voting in 2024");  }  }  }  **OUTPUT:** |

# CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

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

Department of Computer Science Engineering

# Subject Name: Java Programming Semester: III

**Subject Code: CSE201 Academic year: 2024-25**

Part - 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.BufferedReader; import java.io.FileReader; import java.io.IOException;  public class pra27 {  public static void main(String[] args) { if (args.length == 0) {  args = new String[]{"hello.txt"}; |

|  |  |
| --- | --- |
|  | }  for (String fileName : args) {  try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  int lineCount = 0;  while (reader.readLine() != null) { lineCount++;  }  System.out.println(fileName + ": "  + lineCount + " lines");  } catch (IOException e) {  System.err.println("Error reading file " + fileName + ": " + e.getMessage());  }  }  }  }  **OUTPUT:** |

|  |  |
| --- | --- |
|  | **CONCLUSION:**  This program counts the number of lines in a file using Java. It reads each file specified in the command-line arguments or defaults to hello.txt if no arguments are provided. The program uses BufferedReader to read each line and increments a counter for each line read. It handles file reading errors gracefully using a try-with-resources block. The program prints the number of lines for each file processed. This showcases efficient file handling and error management in Java. |
| **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 Prac\_28 {  public static void main(String[] args) { if (args.length != 2) {  System.out.println("Usage: java CharCount <file> <character>"); return;  }  String fileName = args[0];  char targetChar = args[1].charAt(0); |

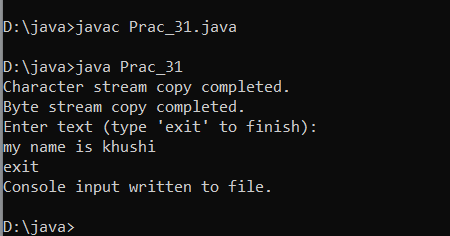
|  |  |
| --- | --- |
|  | try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) { int charCount = 0;  int c;  while ((c = reader.read()) != -1) { if (c == targetChar) {  charCount++;  }  }  System.out.println("The character '" + targetChar + "' appears " + charCount + " times in the file " + fileName);  } catch (IOException e) {  System.err.println("Error reading file " + fileName + ": " + e.getMessage());  }  }  }  **OUTPUT:**    **CONCLUSION:**  This program counts the occurrences of a specific character in a file using Java. It reads the file character by character with BufferedReader and compares each character to the target character. If they match, it increments a counter. The program handles file reading errors using a try-with-resources block to ensure the reader is closed properly. It also provides usage instructions if the required command-line arguments are not provided.  This showcases efficient character processing and error management in Java. |
| **29** | This program counts the occurrences of a specific character in a file using Java. It reads the file character by character with BufferedReader and compares each character to the target character. If they match, it increments a counter. The program handles file reading errors using a try-with-resources block to ensure the reader is closed properly. It also provides usage instructions if the required command-line arguments are not provided.  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 Prac\_29 {  public static void main(String[] args) { if (args.length != 2) {  System.out.println("Usage: java Prac\_29 <file> <word>"); return;  }  String fileName = args[0]; String targetWord = args[1];  try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) { int wordCount = 0;  String line;  while ((line = reader.readLine()) != null) { String[] words = line.split("\\s+");  for (String word : words) {  if (word.equals(targetWord)) { wordCount++;  }  }  }  System.out.println("The word '" + targetWord + "' appears " + wordCount + " times in the file " + fileName);  } catch (IOException e) {  System.err.println("Error reading file " + fileName + ": " + e.getMessage());  }  // Wrapper Class Example  Integer wrapperInt = Integer.valueOf(10); // Using Integer wrapper class int primitiveInt = wrapperInt.intValue(); // Converting back to primitive int |
|  | System.out.println("Wrapper Class Example: Integer value is " + wrapperInt + " and primitive int value is " + primitiveInt);  }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to count the occurrences of a specific word in a file using Java. It reads the file line by line with BufferedReader and splits each line into words. It then compares each word to the target word and increments a counter if they match. The program handles file reading errors gracefully using a try-with-resources block. It also provides usage instructions if the required command-line arguments are not provided. This showcases efficient text processing and error management 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.FileInputStream; import java.io.FileOutputStream; import java.io.IOException;  public class Prac\_30 {  public static void main(String[] args) { if (args.length != 2) {  System.out.println("Usage: java Prac\_30 <source file> <destination file>"); return;  }  String sourceFile = args[0]; String destinationFile = args[1];  try (FileInputStream fis = new FileInputStream(sourceFile); FileOutputStream fos = new FileOutputStream(destinationFile)) {  byte[] buffer = new byte[1024]; int bytesRead;  while ((bytesRead = fis.read(buffer)) != -1) { fos.write(buffer, 0, bytesRead);  }  System.out.println("File copied successfully from " + sourceFile + " to " + |

|  |  |  |
| --- | --- | --- |
|  | destinationFile);  } catch (IOException e) {  System.err.println("Error copying file: " + e.getMessage());  }  }  }  **OUTPUT:** |  |
| **CONCLUSION:**  This program demonstrates how to copy data from one file to another using byte streams in Java. It reads from a source file and writes to a destination file, creating the destination file if it does not exist. The program uses FileInputStream to read bytes and FileOutputStream to write bytes. It handles errors using a try-with-resources block to ensure streams are closed properly. The program also provides usage instructions if the required command-line arguments are not provided. This showcases efficient file handling and error management in Java. | |
| **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.\*;  class Prac\_31 {  public static void main(String[] args) {  // Demonstrate character stream  try (FileReader fr = new FileReader("input.txt"); | |

|  |  |
| --- | --- |
|  | FileWriter fw = new FileWriter("output\_char.txt")) { int c;  while ((c = fr.read()) != -1) { fw.write(c);  }  System.out.println("Character stream copy completed.");  } catch (IOException e) {  System.err.println("Error with character stream: " + e.getMessage());  }  // Demonstrate byte stream  try (FileInputStream fis = new FileInputStream("input.txt"); FileOutputStream fos = new FileOutputStream("output\_byte.txt")) { byte[] buffer = new byte[1024];  int bytesRead;  while ((bytesRead = fis.read(buffer)) != -1) { fos.write(buffer, 0, bytesRead);  }  System.out.println("Byte stream copy completed.");  } catch (IOException e) {  System.err.println("Error with byte stream: " + e.getMessage());  } |

// Use BufferedReader and BufferedWriter to read from console and write to a file try (BufferedReader br = new BufferedReader(new InputStreamReader(System.in));



BufferedWriter bw = new BufferedWriter(new FileWriter("console\_output.txt")))

{

System.out.println("Enter text (type 'exit' to finish):"); String line;

while (!(line = br.readLine()).equalsIgnoreCase("exit")) { bw.write(line);

bw.newLine();

}

System.out.println("Console input written to file.");

} catch (IOException e) {

System.err.println("Error with BufferedReader/BufferedWriter: " + e.getMessage());

}

}

}

**OUTPUT:**

|  |  |
| --- | --- |
|  | **CONCLUSION:**  This program demonstrates the use of character and byte streams in Java. It reads from input.txt and writes to output\_char.txt using character streams, and to output\_byte.txt using byte streams. Additionally, it uses BufferedReader to read console input and BufferedWriter to write the input to console\_output.txt. The program continues to read from the console until the user types "exit". This showcases efficient file handling and console interaction in Java. |

**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: 2024-2025**

**Part- 7[Multithreading]**

|  |  |
| --- | --- |
| **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 :**   1. **BY EXTENDING THREAD CLASS:**   import java.util.\*;  class prc32\_1 extends Thread  {  public void run()  {  System.out.println("Hello, World.(using Thread class)");  }    public static void main(String args[])  {  prc32\_1 t=new prc32\_1();  t.start();  }  }  **OUTPUT:**     1. **By using runnable interface:**   import java.util.\*;  class prc32\_2 implements Runnable  {  public void run()  {  System.out.println("Hello, World.(using runnable thread)");  }    public static void main(String[] args)  {  prc32\_2 m1 = new prc32\_2();  Thread t1=new Thread(m1);  t1.start();  }  }  **OUTPUT:**    **CONCLUSION:**  By this code, we can conclude that threads can be created by extending the in-built thread class as well as using the runnable interface. |
| **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.Scanner;  // Thread class for calculating sum of a portion of numbers  class SumThread extends Thread {  private int start;  private int end;  private int partialSum;  // Constructor to define range of numbers this thread will handle  public SumThread(int start, int end) {  this.start = start;  this.end = end;  }  @Override  public void run() {  partialSum = 0;  for (int i = start; i <= end; i++) {  partialSum += i;  }  }  // Method to return the partial sum calculated by this thread  public int getPartialSum() {  return partialSum;  }  }  public class prc33\_1 {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);    // Input N and number of threads  System.out.print("Enter the value of N (sum numbers from 1 to N): ");  int N = scanner.nextInt();    System.out.print("Enter the number of threads: ");  int numThreads = scanner.nextInt();  // Create an array to hold threads  SumThread[] threads = new SumThread[numThreads];  // Calculate the range of numbers each thread should handle  int range = N / numThreads;  int start = 1;  // Create and start threads  for (int i = 0; i < numThreads; i++) {  int end = (i == numThreads - 1) ? N : (start + range - 1); // Last thread takes the remaining range  threads[i] = new SumThread(start, end);  threads[i].start();  start = end + 1;  }  // Wait for all threads to finish and collect results  int totalSum = 0;  for (int i = 0; i < numThreads; i++) {  try {  threads[i].join(); // Wait for the thread to finish  totalSum += threads[i].getPartialSum(); // Add each thread's partial sum to total sum  } catch (InterruptedException e) {  System.out.println("Thread interrupted: " + e.getMessage());  }  }  // Display the final result  System.out.println("The sum of numbers from 1 to " + N + " is: " + totalSum);  }  }    **OUTPUT:**    **CONCLUSION:**  By this code, we learn about separating tasks and dividing them among threads running simultaneously to make the process quicker and generate the sum accordingly. |
| **34.** | Write a java program that implements a multi-thread application that has three threads. First thread generates random integer every 1 second and if the value is even, second thread computes the square of the number and prints. If the value is odd, the third thread will print the value of cube of the number.  **PROGRAM CODE:**  import java.util.\*;  class GeneratorThread extends Thread  {  public void run()  {  for(int number=0;number<=10;number++)  {  System.out.println("Generated: " + number);  if (number % 2 == 0)  {  SquareThread.square(number);  }  else  {  CubeThread.cube(number);  }    try  {  Thread.sleep(1000);  }  catch (InterruptedException e)  {  e.printStackTrace();  }  }  }  }  class SquareThread extends Thread  {  public static void square(int number)  {  int square = number \* number;  System.out.println("Square: " + square);  }  }  class CubeThread extends Thread  {  public static void cube(int number)  {  int cube = number \* number \* number;  System.out.println("Cube: " + cube);  }  }  public class prc34\_1  {  public static void main(String[] args)  {  GeneratorThread generatorThread = new GeneratorThread();  generatorThread.start();  SquareThread squareThread = new SquareThread();  squareThread.start();  CubeThread cubeThread = new CubeThread();  cubeThread.start();  }  }  **OUTPUT:**    **CONCLUSION:**  By this problem, we learn that multiple threads can work together concurrently, generating numbers, checking if they are even or odd and give the output accordingly. |
| **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.\*;  class MyThread extends Thread  {  public void run()  {  for(int i=0;i<=10;i++)  {  System.out.println(i);  try  {  sleep(1000);  }  catch(InterruptedException e)  {  e.printStackTrace();  }  }  }  }  class prc35  {  public static void main(String args[])  {  MyThread t=new MyThread();  t.start();  }  }  **OUTPUT:**    **CONCLUSION:**  By this program, we learn to use the sleep method which delays the action of the code to manage the thread tasks as per our convenience. |
| **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:**  import java.util.\*;  class first extends Thread  {  public void run()  {  System.out.println(first.class);  }  }  class second extends Thread  {  public void run()  {  System.out.println(second.class);  }  }  class third extends Thread  {  public void run()  {  System.out.println(third.class);  }  }  class prc36 extends Thread  {  public static void main(String args[])  {  first f1=new first();  second s1=new second();  third t1=new third();  f1.setPriority(3);  s1.setPriority(5);  t1.setPriority(7);  f1.start();  s1.start();  t1.start();  }  }  **OUTPUT:**    **CONCLUSION:**  By this program, we learn about setting the priorities of threads as per our requirements. |
| **37.** | Write a program to solve producer-consumer problem using thread synchronization.  **PROGRAM CODE:**  import java.util.LinkedList;  import java.util.Queue;  import java.util.Scanner;  // Shared buffer class with synchronization  class SharedBuffer {  private Queue<Integer> buffer = new LinkedList<>();  private int capacity;  public SharedBuffer(int capacity) {  this.capacity = capacity;  }  // Method for the producer to add items to the buffer  public synchronized void produce(int item) throws InterruptedException {  while (buffer.size() == capacity) {  wait(); // Wait if the buffer is full  }  buffer.add(item);  System.out.println("Produced: " + item);  notifyAll(); // Notify the consumer that an item has been produced  }  // Method for the consumer to take items from the buffer  public synchronized int consume() throws InterruptedException {  while (buffer.isEmpty()) {  wait(); // Wait if the buffer is empty  }  int item = buffer.poll();  System.out.println("Consumed: " + item);  notifyAll(); // Notify the producer that space is available in the buffer  return item;  }  }  // Producer thread class  class Producer extends Thread {  private SharedBuffer buffer;  private int itemsToProduce;  public Producer(SharedBuffer buffer, int itemsToProduce) {  this.buffer = buffer;  this.itemsToProduce = itemsToProduce;  }  @Override  public void run() {  try {  for (int i = 0; i < itemsToProduce; i++) {  buffer.produce(i);  Thread.sleep(500); // Simulate time taken to produce an item  }  } catch (InterruptedException e) {  System.out.println("Producer interrupted.");  }  }  }  // Consumer thread class  class Consumer extends Thread {  private SharedBuffer buffer;  private int itemsToConsume;  public Consumer(SharedBuffer buffer, int itemsToConsume) {  this.buffer = buffer;  this.itemsToConsume = itemsToConsume;  }  @Override  public void run() {  try {  for (int i = 0; i < itemsToConsume; i++) {  buffer.consume();  Thread.sleep(1000); // Simulate time taken to consume an item  }  } catch (InterruptedException e) {  System.out.println("Consumer interrupted.");  }  }  }  public class prac\_37 {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  // Input for buffer capacity  System.out.print("Enter the buffer capacity: ");  int bufferCapacity = scanner.nextInt();  // Input for the number of items to produce and consume  System.out.print("Enter the number of items to produce: ");  int itemsToProduce = scanner.nextInt();  System.out.print("Enter the number of items to consume: ");  int itemsToConsume = scanner.nextInt();  // Create shared buffer  SharedBuffer sharedBuffer = new SharedBuffer(bufferCapacity);  // Create and start producer and consumer threads  Producer producer = new Producer(sharedBuffer, itemsToProduce);  Consumer consumer = new Consumer(sharedBuffer, itemsToConsume);  producer.start();  consumer.start();  try {  // Wait for both threads to complete execution  producer.join();  consumer.join();  } catch (InterruptedException e) {  System.out.println("Main thread interrupted.");  }  System.out.println("Producer and Consumer execution completed.");  }  }  **OUTPUT:**    **CONCLUSION:**  Here, we learn about using the synchronise keyword as well as join and notifyall methods. |

CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

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

Department of Computer Science Engineering

Subject Name: Java Programming Semester: III

Subject Code: CSE201 Academic year: 2024-25

Part - 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 :**  **i**mport java.util.\*; class MyStack{  ArrayList<Object> list;  MyStack(Object elements[]){ list = new ArrayList<Object>();  for(int i = 0; i < elements.length; i++){ list.add( elements[i] );  }  }  MyStack(){ |

|  |  |
| --- | --- |
|  | list = new ArrayList<Object>();  }  boolean isEmpty(){ return (list.size() == 0);  }  Object peek(){  return list.get( list.size()-1 );  }  Object pop(){  Object ob = list.get( list.size()-1 ); list.remove( list.size()- 1 );  return ob;  }  void push(Object o){ list.add(o);  }  }  public class Prac\_38{  public static void main(String[] args){ Integer arr[] = new Integer[]{1,2,3,4}; MyStack s = new MyStack( arr );  System.out.println("Current top = " + s.peek()); System.out.println("Pushing 7,8,9 in the stack"); s.push(7);  s.push(8);  s.push(9);  s.pop();  System.out.println("Elements in the stack are: "); while(!s.isEmpty()){ System.out.println(s.pop());  }  }  } |

|  |  |  |
| --- | --- | --- |
|  | **OUTPUT:** |  |
| **CONCLUSION:**  From this practical, I learned how to create a custom stack using the ArrayList class in Java. I implemented basic stack functionalities like checking if the stack is empty, getting the size, viewing the top element, and performing push and pop operations. This exercise helped me understand how to use an ArrayList to dynamically store elements and simulate a stack structure. | |
| **39** | Imagine you are developing an e-commerce application. The platform needs to sort lists of products based on different criteria, such as price, rating, or name. Each product object implements the Comparable interface to define the natural ordering. To ensure flexibility and reusability, you need a generic method that can sort any array of Comparable objects. Create a generic method in Java that sorts an array of Comparable objects. This method should be versatile enough to sort arrays of different types of objects (such as products, customers, or orders) as long as they implement the Comparable interface.  **PROGRAM CODE :**  public class Prac\_39 {  public static <T extends Comparable<T>> void sortArray(T[] array) { | |

|  |  |
| --- | --- |
|  | int n = array.length; boolean swapped;  for (int i = 0; i < n - 1; i++) { swapped = false;  for (int j = 0; j < n - 1 - i; j++) {  if (array[j].compareTo(array[j + 1]) > 0) { T temp = array[j];  array[j] = array[j + 1]; array[j + 1] = temp; swapped = true;  }  }  if (!swapped) { break;  }  }  }  public static void main(String[] args) { Product[] products = {  new Product("Laptop", 1200, 4.5),  new Product("Phone", 800, 4.3),  new Product("Headphones", 150, 4.7),  new Product("Monitor", 300, 4.4)  };  sortArray(products);  for (Product p : products) { System.out.println(p);  }  }  } |

|  |  |  |
| --- | --- | --- |
|  | class Product implements Comparable<Product> { String name;  double price; double rating;  public Product(String name, double price, double rating) { this.name = name;  this.price = price; this.rating = rating;  }  @Override  public int compareTo(Product other) {  return Double.compare(this.price, other.price);  }  @Override  public String toString() {  return name + " - $" + price + " - Rating: " + rating;  }  }  **OUTPUT:** | |
|  |  |

|  |  |
| --- | --- |
|  | **CONCLUSION:**  Through this practical, I gained insights into implementing a generic method in Java to sort arrays of objects that implement the Comparable interface. I learned how to ensure flexibility and reusability by enabling the method to sort various types of objects, such as products, customers, and orders, based on their natural ordering. |
| **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 Prac\_40 {  public static void main(String[] args) { Scanner sc = new Scanner(System.in); System.out.print("Enter the text: "); String inputText = sc.nextLine();  inputText = inputText.toLowerCase();  HashMap<String, Integer> wordCountMap = new HashMap<>(); StringBuilder currentWord = new StringBuilder();  for (int i = 0; i < inputText.length(); i++) { char c = inputText.charAt(i);  if (Character.isLetter(c) || Character.isDigit(c)) { currentWord.append(c);  } else {  if (currentWord.length() > 0) {  String word = currentWord.toString();  wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1); currentWord.setLength(0);  } |

|  |  |
| --- | --- |
|  | } |
| } |
| if (currentWord.length() > 0) { |
| String word = currentWord.toString(); |
| wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1); |
| } |
| TreeSet<String> sortedWords = new TreeSet<>(wordCountMap.keySet()); |
| System.out.println("Word occurrences:"); |
| for (String word : sortedWords) { |
| System.out.println(word + ": " + wordCountMap.get(word)); |
| } |
| sc.close(); |
| } |
| } |
| **OUTPUT:** |
|  |
| **CONCLUSION:** |
| In this practical, I learned how to use Java's Map and Set classes to count and display the |
| occurrences of words in a given text. I implemented a method that not only counts the |
| occurrences but also sorts the words in alphabetical order. This exercise enhanced my |
| understanding of utilizing collections to efficiently manage and manipulate data. |

|  |  |
| --- | --- |
| **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.util.\*; import java.io.\*;  public class Prac\_41{  public static void main(String[] args) throws IOException{ Scanner sc = new Scanner(System.in); System.out.print("Enter the file name you want to scan : "); String f = sc.nextLine();  File file = new File(f);  FileReader br = new FileReader(file); BufferedReader fr = new BufferedReader(br);  String keywords[] = new String[]{"abstract","assert ","boolean","break","byte","case","catch","char","class",  "continue","default","do","double","else","enum ","extends","final","finally",  "float","for","if","implements","import","instanceof","int","interface","long",  "native","new","package","private","protected","public","return","short","static",  "strictfp","super","switch","synchronized","this","throw","throws","transient","try", "void","volatile","while"};  HashSet<String> set = new HashSet<String>(); for(int i =0;i < keywords.length; ++i){  set.add( keywords[i] );  }  String st;  int count =0 ;  while ((st = fr.readLine()) != null){  StringTokenizer str = new StringTokenizer( st, " +-/\*%<>;:=&|!~()");  while(str.hasMoreTokens()){ String swre = str.nextToken(); |

|  |  |  |
| --- | --- | --- |
|  | if(set.contains(swre )){ |  |
| count++; |
| } |
| } |
| } |
| System.out.println("Total keywords are : " + count); |
| fr.close(); |
| sc.close(); |
| } |
| } |
| **OUTPUT:** |
|  |
| **CONCLUSION:** | |
| From this practical, I learned how to count the occurrences of Java keywords in a source file | |
| by storing all the keywords in a HashSet. By using the contains() method, I was able | |
| to check whether a word is a keyword or not. This practical improved my skills in working | |
| with Java's collection framework, particularly using sets for fast lookups. | |