

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

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

Department of Computer Science & Engineering

**Subject Name: JAVA PROGRAMING**

**Semester: 3rd**

**Subject Code: CSE201**

**Academic year: 2024/25**

Part - 1

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| 1. | Demonstration of installation steps of Java, Introduction to Object Oriented Concepts, comparison of Java with other object-oriented programming languages. Introduction to JDK, JRE, JVM, Javadoc, command line argument. Introduction to Eclipse or NetBeans IDE,or BlueJ and Console Programming.   1. **Download JDK (Java Development Kit)**:    * Go to the [Oracle JDK download page](https://www.oracle.com/java/technologies/javase-jdk11-downloads.html) or the [OpenJDK page](https://jdk.java.net/).    * Choose the appropriate version for your operating system (Windows, macOS, or Linux).    * Download the installer. 2. **Install JDK**:    * Run the downloaded installer.    * Follow the installation instructions. The default settings are usually fine.    * During installation, note the installation path. You might need it later. 3. **Set Environment Variables (Windows)**:    * Go to Control Panel > System and Security > System > Advanced system settings.    * Click on Environment Variables.    * Under System variables, click New and add a new variable named JAVA\_HOME with the value of the JDK installation path.    * Find the Path variable in the System variables section, select it, and click Edit.    * Add a new entry with the path to the bin directory inside your JDK installation directory (e.g., C:\Program Files\Java\jdk-11\bin). 4. **Verify Installation**:    * Open a command prompt (or terminal on macOS/Linux).    * Type java -version and javac -version to check if Java and the Java compiler are properly installed.  Introduction to Object-Oriented Concepts  1. **Class**: A blueprint for creating objects. It defines properties (fields) and behaviors (methods). 2. **Object**: An instance of a class. It is created using the new keyword in Java. 3. **Inheritance**: A mechanism where one class inherits the fields and methods of another class. This promotes code reuse. 4. **Encapsulation**: The wrapping of data (fields) and methods into a single unit (class). Access to the data is controlled using access modifiers (private, public, protected). 5. **Polymorphism**: The ability of different classes to respond to the same method call in different ways. It is achieved through method overloading (compile-time) and method overriding (runtime). 6. **Abstraction**: The concept of hiding the complex implementation details and showing only the necessary features of an object.  Comparison of Java with Other Object-Oriented Programming Languages  1. **Java vs. C++**:    * **Memory Management**: Java has automatic garbage collection, while C++ requires manual memory management.    * **Platform Independence**: Java is platform-independent due to the JVM, whereas C++ is platform-dependent.    * **Multiple Inheritance**: Java does not support multiple inheritance directly but can be achieved using interfaces. C++ supports multiple inheritance. 2. **Java vs. Python**:    * **Performance**: Java generally has better performance due to its compiled nature, while Python is interpreted.    * **Syntax**: Java has a more verbose and strict syntax, whereas Python has a more readable and concise syntax.    * **Typing**: Java is statically typed, and Python is dynamically typed. 3. **Java vs. C#**:    * **Platform**: Java is cross-platform, running on any device with a JVM. C# is mainly used on Windows (though it can be cross-platform with .NET Core).    * **Library Support**: Both have extensive libraries, but C# is tightly integrated with the Windows ecosystem.  Introduction to JDK, JRE, JVM, Javadoc, and Command Line Argument  1. **JDK (Java Development Kit)**: A software development environment used for developing Java applications. It includes JRE, an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools. 2. **JRE (Java Runtime Environment)**: Provides libraries, Java Virtual Machine (JVM), and other components to run applications written in Java. It does not include development tools like a compiler or debugger. 3. **JVM (Java Virtual Machine)**: An abstract machine that enables your computer to run a Java program. JVMs are available for many hardware and software platforms, making Java platform-independent. 4. **Javadoc**: A tool for generating API documentation in HTML format from Java source code. It parses the declarations and documentation comments in a set of source files.  Introduction to Eclipse, NetBeans IDE, or BlueJ and Console Programming  1. **Eclipse**:    * A popular, open-source Integrated Development Environment (IDE) for Java development.    * Download from eclipse.org.    * Features include code completion, debugging, and plugin support. 2. **NetBeans**:    * Another open-source IDE for Java development, also supporting other languages.    * Download from [netbeans.org](https://netbeans.apache.org/download/index.html).    * Features include a powerful editor, debugging, and GUI design tools. 3. **BlueJ**:    * An IDE specifically designed for teaching Java to beginners.    * Download from bluej.org.    * Simple interface with a focus on visualization and interaction. 4. **Console Programming**:    * Writing Java code that runs in the command-line interface (CLI).    * Useful for learning and testing small programs without the overhead of a full IDE.   Example: java Copy code public class HelloWorld  {  public static void main(String[] args)  {  System.out.println("Hello, World!");  }  } |
| 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 Banking  {  public static void main(String[] args)  {  int a;  a=20;  System.out.println("............\nYour Current Balance Is : "+a);  }  }  **OUTPUT:**    **CONCLUSION:**  In This Prectical we Just Print Bank Current balance .  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)  {  float sp,spk,spm;  Scanner a = new Scanner(System.in);  System.out.println("Enter distance (meter) :");  int distance= a.nextInt();  System.out.println("Enter time (hours) :");  int time\_h= a.nextInt();  System.out.println("Enter time (minutes) :");  int time\_m= a.nextInt();  System.out.println("Enter time (second) :");  int time\_s= a.nextInt();  float km=distance/1000;  System.out.println("Distance in km :"+km);  float ml=distance/1609;  System.out.println("Distance in miles :"+ml);  sp=distance/time\_s;  System.out.println("Speed in m/s:" + sp);  spk=km/time\_h;  System.out.println("Speed in km/h:" + spk);  spm=ml/time\_h;  System.out.println("Speed in ml/h:" + spm);  }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to:   * Take user input for distance and time. * Perform unit conversions. * Calculate speed in different units. * Print the results.   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.  **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 :**  import java.util.\*;  class Expence {  public static void main(String[] args) {  Scanner obj=new Scanner(System.in);  System.out.println("Enter the number of days :");  int n=obj.nextInt();  float arr[]=new float[n];  for(int i=0;i<n;i++)  {  System.out.print("Enter the daily expense :"+(i+1)+" ");  arr[i]=obj.nextFloat();  }  float sum=0;  for(int j=0;j<n;j++)  {  sum+=arr[j];    }  System.out.println("The sum of the daily expenses="+sum+" rupees");  }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to:   * Take user input for the number of days and daily expenses. * Calculate the total sum of daily expenses. * Print the total expenses.   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 :**  public class Bill {  public static void main(String[] args) {  int Code[]={1,2,3,4,5};  double tax=0,total=0,item\_bill;  float Price[]={200,2500,500,400,100};  for(int i=0;i<Code.length;i++)  {  int code=Code[i];  float price=Price[i];  switch(code)  {  case 1:  tax=0.08\*price;  break;  case 2:  tax=0.12\*price;  break;  case 3:  tax=0.05\*price;  break;  case 4:  tax=0.075\*price;  break;  case 5:  tax=0.03\*price;  break;  }  item\_bill=price+tax;  System.out.println("Code:"+code+", price of item: "+item\_bill);  total+=item\_bill;  }  System.out.println("The total amount of electronic products is:"+total);  }  }    **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to:   * Use arrays to store item codes and prices. * Use a switch statement to apply different tax rates based on item codes. * Calculate the final price of each item including tax. * Sum up and print the total amount for all items.   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.  **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;  class Fibonacci  {  public static void main(String[] args)  {  int n1=0,n2=1,n3,i,sum=1;  System.out.println("Enter Days :");  Scanner a = new Scanner(System.in);    int N = a.nextInt();    System.out.print(n1+" "+n2);      for(i=2;i<N;++i)  {  n3=n1+n2;  System.out.print(" "+n3);  n1=n2;  n2=n3;  sum=sum+n3;  }  System.out.println("\n"+sum);    }    }    **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to:   * Take user input to determine the number of terms in the Fibonacci sequence. * Generate and print the Fibonacci sequence up to the specified number of terms. * Calculate and print the sum of the Fibonacci sequence terms. |

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

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

Department of Computer Science & Engineering

**Subject Name: Java Programing**

**Semester: 3rd**

**Subject Code: CSE201**

**Academic year: 2024/25**

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.\*;  public class prac7 {  static String front\_times(String a,int b){  String newstr="";  if(a.length()<=3)  {  for (int i = 0; i < b; i++) {  newstr+=a;  }  }  else{  for (int i = 0; i < b; i++) {  newstr+=a.substring(0, 3);  }  }  return newstr;  }  public static void main(String[] args) {  String a;  Scanner sc =new Scanner(System.in);  System.out.println("Enter the string :");  a=sc.nextLine();  System.out.println("Enter the number of time you want to repeat the first three character :");  int n=sc.nextInt();  String newstr=front\_times(a,n);  System.out.println("The new string is \n"+newstr);    }    }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to:   * Take user input for a string and an integer. * Repeat the first three characters of the string (or the entire string if it is less than or equal to three characters) a specified number of times. * Use a method to perform the string repetition logic and return the new string. * Print the resulting new string. |
| 8  9.  10.  11. | 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  **Supplementary Experiment:**  1. Write a Java program to replace each substring of a given  string that matches the given regular expression with the  given replacement.  Sample string : "The quick brown fox jumps over the lazy  dog."  **In the above string replace all the fox with cat.**  **PROGRAM CODE :**  import java.util.Scanner;  public class Prec8  {  static int count9(int a[]) {  int count = 0;  for (int i = 0; i < a.length; i++) {  if (a[i] == 9)  count++;  }  return count;  }  public static void main(String[] args) {  int size;  Scanner sc=new Scanner(System.in);  System.out.println("Enter the size of array :");  size=sc.nextInt();  int array[]=new int[size];    for (int i=0;i<size;i++) {  System.out.println("a["+i+"]");  array[i]=sc.nextInt();  }  System.out.println("Number of 9 in given array is :"+count9(array));  sc.close();  }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to:   * Take user input for the size and elements of an array. * Count the occurrences of a specific number (9) in the array. * Print the count of the specified number.   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 {  static String double\_char(String str)  {  String newstr="";  for (int i = 0; i < str.length(); i++) {  newstr+= str.charAt(i);  newstr+= str.charAt(i);  }  return newstr;  }  public static void main(String[] args) {  String str;  Scanner sc=new Scanner(System.in);  System.out.println("Enter the string :");  str=sc.nextLine();  System.out.println("New String is :\n"+double\_char(str));  sc.close();  }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates how to:   * Take user input for a string. * Iterate through each character of the string and double it. * Construct and print a new string with each character repeated twice.   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 prac10{  public static void main(String[] args)  {  String str="CHARUSAT UNIVERSITY", nstr="";  char ch;  System.out.println("Lenght of string :"+str.length());  System.out.println("Convert to lowercase:"+str.toLowerCase());  System.out.println("Convert to Uppercase :"+str.toUpperCase());  for (int i=0; i<str.length(); i++)  {  ch= str.charAt(i);  nstr= ch+nstr;  }  System.out.println("Reverse the String :"+nstr);    }  }  **OUTPUT:**    **CONCLUSION:**  **Length of String:**   * Prints the length of the string "CHARUSAT UNIVERSITY".   **Convert to Lowercase:**   * Converts the string to lowercase and prints it.   **Convert to Uppercase:**   * Converts the string to uppercase and prints it.   **Reverse the String:**   * Reverses the string character by character and prints the reversed string.   Perform following Functionalities of the string:  “CHARUSAT UNIVERSITY”  ● Find length  ● Replace ‘H’ by ‘FIRST LATTER OF YOUR NAME’  ● Convert all character in lowercase  Supplementary Experiment:  1. Write a Java program to count and print all duplicates in  the input string.  Sample Output:  The given string is: resource  The duplicate characters and counts are:  e appears 2 times  r appears 2 times  **PROGRAM CODE :**  public class Prec11    {  public static void main(String[] args) {  String str="CHARUSAT UNIVERSITY";  System.out.println("Lenght of string : "+str.length());  System.out.println("Replace H by V : "+(str=str.replace('H','V')));  System.out.println("Convert To Lowercase : "+str.toLowerCase());  }  }  **OUTPUT:**    **CONCLUSION:**  **Length of String:**   * Prints the length of the string "CHARUSAT UNIVERSITY".   **Replace 'H' by 'V':**   * Replaces all occurrences of the character 'H' with 'V' in the string str. * Updates str with the modified string and prints it.   **Convert to Lowercase:**   * Converts the string str (which now has 'H' replaced by 'V') to lowercase. * Prints the lowercase version of the string. |

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

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

Department of Computer Science & Engineering

**Subject Name: Java Programing**

**Semester: 3rd**

**Subject Code: CSE201**

**Academic year: 2024/25**

Part - 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 :**  public class Prec12  {  public static void main(String[] args)  {  for(String str:args){  float argument=Float.parseFloat(str);  float r=argument\*100;  System.out.println("Pounds="+argument+"And Rupees="+r);  }  }  }  **OUTPUT:**    **CONCLUSION:**  The provided Java code reads command-line arguments as floating-point numbers representing amounts in pounds.  For each argument, it calculates the equivalent amount in rupees by multiplying the pound value by 100. It then prints both the original amount in pounds and the calculated amount in rupees. This program demonstrates a simple currency conversion from pounds to rupees using a fixed conversion rate. |
| 13.  14.  15.  16. | 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 :**  class Employee{  String fn,ln;  double s=0;  Employee(String a,String b,double c)  {  fn=a;  ln=b;  if(c<0)  {  s=0.0;  }  else  s=c;  }  public void set(String f\_name,String l\_name,double sal)  {  fn=f\_name;  ln=l\_name;  if(s<0)  {  s=0.0;  }  else  s=sal;  }  public void get()  {  System.out.println("First Name: "+fn);  System.out.println("Last Name: "+ln);  System.out.println("Yearly Salary: "+s\*12);  }  public void raise()  {  s+=s\*(10.0/100);  }  }  class Prac13{  public static void main(String[] args)  {  Employee obj1 = new Employee("Promish","Rakholiya",500000);  Employee obj2 = new Employee("Nihir","Patel",-4000.0);  obj1.get();  obj2.get();  obj2 = new Employee("Nihir","Patel",4000.0);  obj1.raise();  obj2.raise();  obj1.get();  obj2.get();  }  }  **OUTPUT:**    **CONCLUSION:**  The Java code defines an Employee class that includes a constructor to initialize the employee's name and salary, setting the salary to 0 if it's negative. The class has methods to update details, display annual salary, and apply a 10% salary raise.  In the Prac13 class, two Employee objects are created—one with a positive salary and another with a negative salary (set to 0). After displaying their details, the negative salary employee is reinitialized with a positive salary. The raise method is then applied to both employees, and their updated details are displayed.  Create a class called Date that includes three pieces of information as instance variables a month (type int), a day (type int) and a year (type int). Your class should have a constructor that initializes the three instance variables and assumes that the values provided are correct. Provide a set and a get method for each instance variable. Provide a method displayDate that displays the month, day and year separated by forward slashes (/). Write a test application named DateTest that demonstrates class Date’s capabilities.  **PROGRAM CODE :**  import java.util.\*;  public class Prac14  {  int month,day,year,d,m,y;  Prac14(int m,int d,int y)  {  month=m;  day=d;  year=y;  }  public void get()  {  System.out.println("Enter day:");  Scanner sc=new Scanner(System.in);  d=sc.nextInt();  System.out.println("Enter month:");  m=sc.nextInt();  System.out.println("Enter year:");  y=sc.nextInt();  }  public int getmonth()  {  return month;  }  public int getday()  {  return day;  }  public int getyear()  {  return year;  }  public void display()  {  System.out.println("The date using constructor is: ");  System.out.println(day+"/"+month+"/"+year); System.out.println("The date using method is: ");  System.out.println(d+"/"+m+"/"+y);  }  public static void main(String[] args) {  Prac14 obj=new Prac14(11,21,2005);  int p,q,r;  int a=obj.getday();  System.out.println("Day: "+a);  int b=obj.getmonth();  System.out.println("Month: "+b);  int c=obj.getyear();  System.out.println("Year: "+c);  obj.get();  obj.display();  }  }  **OUTPUT:**    **CONCLUSION:**  The Java code defines a Prac14 class to manage and display date information. It initializes the date using a constructor and allows updating and displaying the date through methods.   * The Prac14 class has attributes for month, day, and year, and methods to get these values, update them via user input, and display the date. * In the main method, an instance is created with a specific date, and the class methods are used to retrieve and display date values both from the constructor and from user input.   Overall, the code demonstrates basic date handling and user interaction with a class in Java.  Write a program to print the area of a rectangle by creating a class named 'Area' taking the values of its length and breadth as parameters of its constructor and having a method named 'returnArea' which returns the area of the rectangle. Length and breadth of rectangle are entered through keyboard.  **PROGRAM CODE :**  import java.util.\*;  public class Prac15 {  int length,breadth;  Prac15(int l,int b)  {  length=l;  breadth=b;  System.out.println("Area using constructor= "+(length\*breadth));  }  public float returnArea(int l,int b)  {  return l\*b;  }  public static void main(String[] args) {  Scanner sc=new Scanner(System.in);  System.out.println("Enetr the length of the rectangle: ");  int a=sc.nextInt();  System.out.println("Enetr the breadth of the rectangle: ");  int b=sc.nextInt();  Prac15 obj=new Prac15(a,b);  float ans=obj.returnArea(a,b);  System.out.println("Area of rectangle = "+ans);  }  }  **OUTPUT:**    **CONCLUSION:**  The Java code defines a Prac15 class to calculate and display the area of a rectangle.   * The class has a constructor that initializes the length and breadth of the rectangle, and calculates the area, which is displayed immediately. * The returnArea method calculates the area of a rectangle given its length and breadth. * In the main method, the program prompts the user to enter the length and breadth of the rectangle, creates an instance of Prac15 to display the area using the constructor, and then calculates and prints the area using the returnArea method.   Print the sum, difference and product of two complex numbers by creating a class named ‘Complex’ with separate methods for each operation whose real and imaginary parts  are entered by user.  **PROGRAM CODE :**  import java.util.\*;  public class Prac16 {  double real;  double img;  public void add(int r1,int im1,int r2,int im2) {  int r,im;  r= r1+r2;  im= im1+im2;  System.out.println("Sum= "+r+"+"+im+"i");  }  public void sub(int r1,int im1,int r2,int im2) {  int r,im;  r= r1-r2;  im= im1-im2;  if(im<0)  System.out.println("Subtraction= "+r+im+"i");  else  System.out.println("Subtraction= "+r+"+"+im+"i");  }  public void mul(int r1,int im1,int r2,int im2) {  int r,im;  r= r1\*r2-im1\*im2;  im= r1\*im2+im1\*r2;  System.out.println("Multiplication= "+r+"+"+im+"i");  }  public static void main(String[] args) {  Scanner sc=new Scanner(System.in);  System.out.println("Enter real part 1: ");  int r1=sc.nextInt();  System.out.println("Enter imaginary part 1: ");  int im1=sc.nextInt();  System.out.println("Enter real part 2: ");  int r2=sc.nextInt();  System.out.println("Enter imaginary part 2: ");  int im2=sc.nextInt();  Prac16 obj1=new Prac16();  obj1.add(r1,im1,r2,im2);  obj1.sub(r1,im1,r2,im2);  obj1.mul(r1,im1,r2,im2);  }  }  **OUTPUT:**  **CONCLUSION:**  The Java code defines a Prac16 class to perform arithmetic operations on complex numbers.   * The class includes methods to add, subtract, and multiply two complex numbers, where each number has a real and imaginary part. * The add method calculates and prints the sum of two complex numbers. * The sub method calculates and prints the difference, handling the sign of the imaginary part correctly. * The mul method calculates and prints the product of two complex numbers. * In the main method, the program prompts the user to input the real and imaginary parts of two complex numbers, then creates an instance of Prac16 and uses it to perform and display the results of the arithmetic operations. |

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

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

Department of Computer Science & Engineering

**Subject Name: Java Programing**

**Semester: 3rd**

**Subject Code: CSE201**

**Academic year: 2024/25**

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{  void base()  {  System.out.println("This is parent class");  }  }  class child extends parent{  void derived()  {  System.out.println("This is child class");  }  }  public class Prec17 {  public static void main(String[] args) {  parent obj1=new parent();  child obj2=new child();  obj1.base();  }  }  **OUTPUT:**  **CONCLUSION:**  The code demonstrates the concept of inheritance in Java, where the child class inherits from the parent class. The main() method shows how to instantiate objects of both the parent and child classes and call the method from the parent class. The child class also has its own method, but it is not called in this example. Inheritance allows the child class to reuse code from the parent class, promoting reusability and reducing redundancy. |
| 18.  19.  20.  21.  22.  23. | 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.\*;  class Member{  String name,address;  int age;  long num;  double salary;  Scanner sc=new Scanner(System.in);  void printSalary()  {  System.out.println(salary);  }  void getdata()  {  System.out.println("Enter the Name : ");  name=sc.nextLine();  System.out.println("Enter the Age : ");  age=sc.nextInt();  sc.nextLine();  System.out.println("Enter the Address : ");  address=sc.nextLine();  System.out.println("Enter the Phone number : ");  num=sc.nextLong();  System.out.println("Enter the Salary : ");  salary=sc.nextDouble();  }  void print()  {  System.out.println("Name : "+name);  System.out.println("Age : "+age);  System.out.println("Phone Number : "+num);  System.out.println("Address : "+address);  System.out.print("Salary : ");  printSalary();  }  }  class Employee extends Member{  String specialization;  @Override  void getdata() {  super.getdata();  sc.nextLine();  System.out.println("Enter the Designation of Employee : ");  specialization=sc.nextLine();  }  @Override  void print()  {  super.print();  System.out.println("Specialization : "+specialization);  }  }  class Manager extends Member{  String department;  @Override  void getdata() {  super.getdata();  sc.nextLine();  System.out.println("Enter the Department : ");  department=sc.nextLine();  }  @Override  void print()  {  super.print();  System.out.println("Department : "+department);  }  }  public class pra18 {  public static void main(String[] args) {  Employee obj1=new Employee();  Manager obj2=new Manager();  obj1.getdata(); System.out.println("\*\*\*\*\*\*\*\*\*EMPLOYEE\*\*\*\*\*\*\*\*\*");  obj1.print();  obj2.getdata(); System.out.println("\*\*\*\*\*\*\*\*\*MANAGER\*\*\*\*\*\*\*\*\*");  obj2.print();  }  }  **OUTPUT:**    **CONCLUSION:**  The code effectively demonstrates the principles of object-oriented programming in Java, particularly inheritance and method overriding. The Member class serves as a base class for common data and behavior, while the Employee and Manager classes extend this functionality with additional attributes and specialized methods. The getdata() and print() methods in the derived classes override those in the base class to include specific details relevant to employees and managers.  This structure allows for code reusability and easy maintenance, as common attributes and methods are centralized in the Member class, while specific details for employees and managers are handled in their respective classes. The program also demonstrates how user input can be gathered and displayed, making it interactive and dynamic.  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{  double length,width;  Rectangle(double length,double width)  {  this.length=length;  this.width=width;  }  double area()  {  return length\*width;  }  double perimeter()  {  return 2\*(length+width);  }  }  class Square extends Rectangle{  Square(double s){  super(s,s); //calling the base constructor  }  }  public class Prec19{  public static void main(String[] args)  {  System.out.print("Enter the size of the array : ");  Scanner sc=new Scanner(System.in);  int n=sc.nextInt();  Rectangle arr[]=new Rectangle[n];  System.out.println("Enter the Rectangle's or Square's Parameters : ");  for(int i=0;i<n;i++){  System.out.println("Enter R for Rectangle and S for Square : ");  char c=sc.next().charAt(0);  if(c=='r' || c=='R')  {  System.out.println("Enter Length : ");  double l=sc.nextDouble();  System.out.println("Enter Breadth : ");  double b=sc.nextDouble();  arr[i]=new Rectangle(l, b);  }  else if(c=='s' || c=='S'){  System.out.println("Enter the side of the square : ");  double s=sc.nextDouble();  arr[i]=new Square(s);  }  else{  System.out.println("Enter correct character : ");  i--;  }  }  for(Rectangle i:arr){  System.out.println(i.area());  System.out.println(i.perimeter());  }  }  }  **OUTPUT:**    **CONCLUSION:**  This code effectively demonstrates how inheritance and polymorphism can be used to model different shapes in Java. The Square class inherits from the Rectangle class, leveraging its existing methods and structure, while overriding the constructor to enforce the square's property of equal sides.  The use of an array of type Rectangle to store both Rectangle and Square objects highlights the power of polymorphism, allowing the program to treat squares as specialized rectangles. This design pattern promotes code reusability and flexibility, as the same methods (area() and perimeter()) can be used for both rectangles and squares without modification.  Create a class named 'Shape' with a method to print "This is This is shape". Then create two other classes named 'Rectangle', 'Circle' inheriting the Shape class, both having a method to print "This is rectangular shape" and "This is circular shape" respectively. Create a subclass 'Square' of 'Rectangle' having a method to print "Square is a rectangle". Now call the method of 'Shape' and 'Rectangle' class by the object of 'Square' class.  **PROGRAM CODE :**  class Shape{      void print\_shape(){          System.out.println("This is Shape.");      }  }  class Rectangle extends Shape{      void print\_rect(){      System.out.println("This is Rectangular Shape.");      }  }  class Circle extends Shape{      void print\_circle(){      System.out.println("This is Circluar Shape.");      }  }  class Square extends Rectangle{      void print\_square(){          System.out.println("Square is Rectangle.");      }  }  public class Prac20 {      public static void main(String[] args) {          Square sq=new Square();          sq.print\_shape();          sq.print\_rect();          sq.print\_square();      }  }  **OUTPUT:**    **CONCLUSION:**  The Java code defines a hierarchy of shape classes to illustrate inheritance:   1. **Shape Class**: The base class with a method print\_shape() that prints a general message about shapes. 2. **Rectangle Class**: A subclass of Shape that introduces a method print\_rect() to specify that it is a rectangular shape. 3. **Circle Class**: Another subclass of Shape with a method print\_circle() indicating that it is a circular shape. 4. **Square Class**: A subclass of Rectangle, inheriting its properties and methods. It adds a method print\_square() to specify that a square is a type of rectangle.   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{      void getDegree(){          System.out.println("I got Degree");      }  }  class Undergraduate extends Degree{      void getDegree(){          System.out.println("I am an Undergraduate");      }  }  class Postgraduate extends Degree{      void getDegree(){          System.out.println("I am a Postgraduate");      }  }  public class Prac21 {      public static void main(String[] args) {          Undergraduate std1=new Undergraduate();          Postgraduate std2=new Postgraduate();          std1.getDegree();          std2.getDegree();      }  }  **OUTPUT:**    **CONCLUSION:**  The Java code illustrates method overriding with a simple inheritance structure involving degree levels:   1. **Degree Class**: This is the base class with a method getDegree() that prints a general message indicating the attainment of a degree. 2. **Undergraduate Class**: A subclass of Degree that overrides the getDegree() method to specify that it represents an undergraduate degree. 3. **Postgraduate Class**: Another subclass of Degree that overrides the getDegree() method to denote a postgraduate degree.   Write a java that implements an interface AdvancedArithmetic which contains amethod signature int divisor\_sum(int n). You need to write a class calledMyCalculator which implements the interface. divisorSum function just takes an integer as input and return the sum of all its divisors. For example, divisors of 6 are 1, 2, 3 and 6, so divisor\_sum should return 12. The value of n will be at most 1000.  **PROGRAM CODE :**  import java.util.\*;  interface AdvancedArithmetic{      int divisor\_sum(int n);  }  class calledMyCalculator implements AdvancedArithmetic{      int sum;      public int divisor\_sum(int n){          for(int i=1;i<=n;i++){              if(n%i==0){                  sum+=i;              }          }  return sum;      }  }  public class Prec22 {      public static void main(String[] args) {          Scanner sc=new Scanner(System.in);          System.out.println("Enter The Number : ");          int n=sc.nextInt();          calledMyCalculator obj=new calledMyCalculator();          System.out.println("Sum of All Divisors = "+obj.divisor\_sum(n));      }  }  **OUTPUT:**    **CONCLUSION:**   **Interface Implementation**: How a class implements an interface and provides the specific functionality required by the interface.   **Basic Loop Operations**: How to use a loop to find and sum the divisors of an integer.   **User Input Handling**: How to take input from the user and process it to produce the desired output.  Assume you want to capture shapes, which can be either circles (with a radiusand a color) or rectangles (with a length, width, and color). You also want to be able to create signs (to post in the campus center, for example), each of which has a shape (for the background of the sign) and the text (a String) to put on the sign. Create classes and interfaces for circles, rectangles, shapes, and signs. Write a program that illustrates the significance of interface default method.  **PROGRAM CODE :**  interface Shape  {      String getColor();  }  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;      }  }  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;      }  }  class Sign {      private Shape shape;      private String text;      public Sign(Shape shape, String text) {          this.shape = shape;          this.text = text;      }      public void display() {          System.out.println("Sign Details:");          System.out.println("Shape: " + shape.getClass().getSimpleName());          System.out.println("Color: " + shape.getColor());          System.out.println("Text: " + text);      }  }  public class Prec23{      public static void main(String[] args) {          Shape circle = new Circle(5, "Red");          Shape rectangle = new Rectangle(4, 6, "Blue");          Sign circleSign = new Sign(circle, "Welcome to the Campus");          Sign rectangleSign = new Sign(rectangle, "Library Entrance");          circleSign.display();          System.out.println();          rectangleSign.display();      }  }  **OUTPUT:**    **CONCLUSION:**  The program effectively demonstrates:   * **Interface Implementation**: How classes (Circle and Rectangle) implement a common interface (Shape) to ensure they provide specific functionality. * **Polymorphism**: How the Sign class uses the Shape interface to handle different shape types without needing to know their specific implementations. * **Composition**: How the Sign class contains and uses a Shape object, showcasing the concept of building complex objects from simpler ones |

**CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY**

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

Department of Computer Science & Engineering

**Subject Name: Java Programing**

**Semester: 3rd**

**Subject Code: CSE201**

**Academic year: 2024/25**

Part - 5

|  |  |
| --- | --- |
| **No** | **Aim of the Practical** |
| 24.  25. | 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 Prec24 {      public static void main(String[] args) {          int x, y;          float ans;          try {              System.out.println("Enter the value of X : ");              Scanner sc = new Scanner(System.in);              x = sc.nextInt();              System.out.println("Enter the value of Y : ");              y = sc.nextInt();              ans = x / y;              System.out.println(ans);          } catch (ArithmeticException e) {              System.out.println("The Value of Y can't be ZERO!!!!");          } catch (InputMismatchException e2) {              System.out.println(e2);          } catch (Exception e3) {              System.out.println(e3);          }      }  }  **OUTPUT:**    **CONCLUSION:**  The Java program demonstrates the usage of exception handling through a try-catch mechanism. In this program, the user is prompted to input two integers, x and y, and then the program attempts to divide x by y.   1. **Exception Handling:**    * **ArithmeticException:** If the user enters 0 for y, an ArithmeticException will be triggered due to division by zero, and the program will print the message "The Value of Y can't be ZERO!!!!".    * **InputMismatchException:** If the user enters a non-integer value (e.g., a string or a floating-point number), an InputMismatchException is thrown, and the program will print the exception details.    * **General Exception:** Any other unforeseen exceptions are caught by the general Exception block, ensuring the program doesn't crash unexpectedly. 2. **Code Structure:**    * The try block contains the critical code that may throw exceptions: accepting user input and performing the division operation.    * The program contains three catch blocks, each designed to handle specific types of exceptions, providing more meaningful error messages and allowing graceful handling of various errors. 3. **User Experience:**    * The program ensures that invalid inputs, such as zero for y or incorrect data types, are handled gracefully with proper messages rather than terminating the program abruptly.   Write a Java program that throws an exception and catch it using a try-catch block.  **PROGRAM CODE :**    import java.util.\*;  public class Prec25 {      public static void main(String[] args) {          int array[]=new int[5];          Scanner sc=new Scanner(System.in);          System.out.println("Enter the array value");          try {              for (int i = 0; i < array.length; i++) {                  array[i]=sc.nextInt();              }              for (int i = 3; i <10; i++) {                  System.out.println(array[i]);              }          } catch(InputMismatchException e){              System.out.println(e);          } catch (ArrayIndexOutOfBoundsException e1) {              System.out.println(e1);          }      }  }  **OUTPUT:**        **CONCLUSION:**   * **Array Boundaries**: The code demonstrates what happens when array elements are accessed outside of valid boundaries. * **Input Validation**: It also showcases how to handle user input errors like entering the wrong data type. * **Error Handling**: By using try-catch, the program prevents crashes and informs the user of any input-related issues or logical errors like accessing invalid array indices.   This is a basic example of how to manage array access errors and incorrect input in Java programs using exception handling. |
| 26. | Write a java program to generate user defined exception using “throw” and “throws” keyword. Also Write a java that differentiates checked and unchecked exceptions. (Mention at least two checked and two unchecked exceptions in program).  **PROGRAM CODE :**  import java.util.Scanner;  class UserDefinedException extends Exception {      public UserDefinedException(String message) {          super(message);      }  }  public class Prec26 {      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          try {              System.out.print("Enter a number: ");              int number = scanner.nextInt();              if (number < 0) {                  throw new UserDefinedException("Number cannot be negative");              }              System.out.println("You entered: " + number);          } catch (UserDefinedException e) {              System.out.println("Caught UserDefinedException: " + e.getMessage());          } catch (Exception e) {              System.out.println("Caught Exception: " + e.getMessage());          }          try {              int[] array = {1, 2, 3};              System.out.println(array[4]);              int result = 10 / 0;          } catch (ArrayIndexOutOfBoundsException e) {              System.out.println("Caught ArrayIndexOutOfBoundsException: " + e.getMessage());          } catch (ArithmeticException e) {              System.out.println("Caught ArithmeticException: " + e.getMessage());          } finally {              scanner.close();          }      }  }  **OUTPUT:**    **CONCLUSION:**   * The code demonstrates how custom and built-in exceptions can be handled in Java. * It highlights the importance of using try-catch blocks to manage exceptions and ensure graceful program execution. * The finally block is used for resource cleanup, ensuring that resources like Scanner are closed properly to prevent memory leaks or resource locks.   This example is useful for understanding the concept of exception handling, including custom exception creation, handling specific exceptions, and cleaning up resources. |

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: 2024-25

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 LineCounts {  public static void main(String[] args) {  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.out.println("Error reading " + fileName + ": " + e.getMessage());  }  }  }  }  **OUTPUT:**    **CONCLUSION:**  The Java program counts the lines in each text file given as command-line arguments, handling any read errors along the way. It outputs the results for each file, ensuring that it continues processing even if some files can't be read. |
| 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.File;  import java.io.FileNotFoundException;  import java.util.Scanner;  public class Prac28 {      public static void main(String[] args) {            if (args.length != 2) {              System.out.println("Usage: java CharacterCountInFileScanner <file-path> <character>");              return;          }          String fileName = args[0];          char targetChar = args[1].charAt(0);          int count = 0;          try {                File file = new File(fileName);              Scanner scanner = new Scanner(file);                while (scanner.hasNextLine()) {                  String line = scanner.nextLine();                    for (char c : line.toCharArray()) {                      if (c == targetChar) {                          count++;                      }                  }              }              scanner.close();          } catch (FileNotFoundException e) {              System.out.println("File not found: " + fileName);            }          System.out.println("Character '" + targetChar + "' appears " + count + " times in the file.");      }  }  **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 WordSearch {  public static void main(String[] args) {  if (args.length < 2) {  System.out.println("Usage: java WordSearch <word> <filename>");  return;  }  String searchWord = args[0];  String fileName = args[1];  Integer count = 0;  try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  String line;  while ((line = reader.readLine()) != null) {  String[] words = line.split("\\W+");  for (String word : words) {  if (word.equalsIgnoreCase(searchWord)) {  count++;  }  }  }  System.out.println("The word '" + searchWord + "' appears " + count + " times in " + fileName);  } catch (IOException e) {  System.out.println("Error reading " + fileName + ": " + e.getMessage());  }  }  }    **OUTPUT:**    **CONCLUSION:**  This Java program effectively searches for a specified word in a given file and counts its occurrences. It demonstrates the use of the Integer wrapper class to manage the count, showcasing how wrapper classes can be used for object manipulation in Java. |
| 30. | Write a program to copy data from one file to another file. If the destination file does not exist, it is created automatically.  **PROGRAM CODE:**  import java.io.FileReader;  import java.io.FileWriter;  import java.io.IOException;  public class FileCopy {  public static void main(String[] args) {  if (args.length < 2) {  System.out.println("Usage: java FileCopy <source file> <destination file>");  return;  }  String sourceFile = args[0];  String destinationFile = args[1];  try (FileReader fr = new FileReader(sourceFile);  FileWriter fw = new FileWriter(destinationFile)) {    int ch;  while ((ch = fr.read()) != -1) {  fw.write(ch);  }  System.out.println("Data copied from " + sourceFile + " to " + destinationFile);  } catch (IOException e) {  System.out.println("Error: " + e.getMessage());  }  }  }  **OUTPUT:**    **CONCLUSION:**  This Java program efficiently copies data from a source file to a destination file, automatically creating the destination file if it does not already exist. It handles any potential I/O exceptions during the process, ensuring robust performance. |
| 31. | Write a program to show use of character and byte stream. Also show use of  BufferedReader/BufferedWriter to read console input and write them into a file.  **PROGRAM CODE:**  import java.io.BufferedReader;  import java.io.BufferedWriter;  import java.io.FileWriter;  import java.io.IOException;  import java.io.InputStreamReader;  public class ConsoleToFile {  public static void main(String[] args) {  BufferedReader consoleReader = new BufferedReader(new InputStreamReader(System.in));  String fileName = "output.txt";  try (BufferedWriter fileWriter = new BufferedWriter(new FileWriter(fileName))) {  System.out.println("Enter text (type 'exit' to finish):");  String input;  while (!(input = consoleReader.readLine()).equalsIgnoreCase("exit")) {  fileWriter.write(input);  fileWriter.newLine();  }  System.out.println("Data written to " + fileName);  } catch (IOException e) {  System.out.println("Error: " + e.getMessage());  }  }  }  **OUTPUT:**    **CONCLUSION:**  This program effectively demonstrates the use of character streams via BufferedReader and BufferedWriter for reading console input and writing it to a file. It showcases how to handle text data efficiently while managing resources properly with try-with-resources. |

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: 2024-25

Part - 7

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| 32. | Write a program to create thread which display “Hello World” message. A. by extending Thread class B. by using Runnable interface.  **PROGRAM CODE:**  public class  Prac32 {  static class HelloWorldThread extends Thread {  public void run() {  System.out.println("Hello World");  }  }  static class HelloWorldRunnable implements Runnable {  public void run() {  System.out.println("Hello World");  }  }  public static void main(String[] args) {  HelloWorldThread thread1 = new HelloWorldThread();  thread1.start();  Thread thread2 = new Thread(new HelloWorldRunnable());  thread2.start();  }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates two approaches to creating threads in Java: extending the Thread class and implementing the Runnable interface. Both methods effectively print "Hello World," showcasing the flexibility of Java's concurrency model. |
| 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;  class SumTask implements Runnable {  private int start;  private int end;  private static int totalSum = 0;  public SumTask(int start, int end) {  this.start = start;  this.end = end;  }  public void run() {  int partialSum = 0;  for (int i = start; i <= end; i++) {  partialSum += i;  }  synchronized (SumTask.class) {  totalSum += partialSum;  }  }  public static int getTotalSum() {  return totalSum;  }  }  public class ThreadedSummation {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.print("Enter N: ");  int N = scanner.nextInt();  System.out.print("Enter number of threads: ");  int numThreads = scanner.nextInt();  Thread[] threads = new Thread[numThreads];  int range = N / numThreads;  int remainder = N % 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));  threads[i].start();  start = end + 1;  }  for (Thread thread : threads) {  try {  thread.join();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  System.out.println("Total Sum: " + SumTask.getTotalSum());  }  }  **OUTPUT:**    **CONCLUSION:**  This program effectively demonstrates how to utilize multiple threads in Java to perform a summation task concurrently. By distributing the workload among threads, it showcases improved efficiency in computation, making it a practical example of multithreading in action. |
| 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.Random;  class RandomNumberGenerator extends Thread {  private final Object lock;  public RandomNumberGenerator(Object lock) {  this.lock = lock;  }  public void run() {  Random random = new Random();  while (true) {  int number = random.nextInt(100);  synchronized (lock) {  MultiThreadApplication.lastNumber = number;  lock.notifyAll();  System.out.println("Generated: " + number);  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  }  }  class EvenNumberProcessor extends Thread {  private final Object lock;  public EvenNumberProcessor(Object lock) {  this.lock = lock;  }  public void run() {  while (true) {  synchronized (lock) {  try {  lock.wait();  } catch (InterruptedException e) {  e.printStackTrace();  }  if (MultiThreadApplication.lastNumber % 2 == 0) {  int square = MultiThreadApplication.lastNumber \* MultiThreadApplication.lastNumber;  System.out.println("Square: " + square);  }  }  }  }  }  class OddNumberProcessor extends Thread {  private final Object lock;  public OddNumberProcessor(Object lock) {  this.lock = lock;  }  public void run() {  while (true) {  synchronized (lock) {  try {  lock.wait();  } catch (InterruptedException e) {  e.printStackTrace();  }  if (MultiThreadApplication.lastNumber % 2 != 0) {  int cube = MultiThreadApplication.lastNumber \* MultiThreadApplication.lastNumber \* MultiThreadApplication.lastNumber;  System.out.println("Cube: " + cube);  }  }  }  }  }  public class MultiThreadApplication {  public static int lastNumber;  public static void main(String[] args) {  Object lock = new Object();    RandomNumberGenerator generator = new RandomNumberGenerator(lock);  EvenNumberProcessor evenProcessor = new EvenNumberProcessor(lock);  OddNumberProcessor oddProcessor = new OddNumberProcessor(lock);  generator.start();  evenProcessor.start();  oddProcessor.start();  }  }  **OUTPUT:**    **CONCLUSION:**  This program effectively demonstrates a multi-threaded application where one thread generates random integers, while two other threads process these integers based on their parity. It highlights the use of synchronization in Java to safely share data among threads, showcasing how concurrency can be leveraged for efficient task distribution. |
| 35. | Write a program to increment the value of one variable by one and display it after one second using thread using sleep() method.  **PROGRAM CODE:**  public class IncrementVariable extends Thread {  private int value = 0;  public void run() {  while (true) {  value++;  System.out.println("Value: " + value);  try {  Thread.sleep(1000);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  public static void main(String[] args) {  IncrementVariable incrementer = new IncrementVariable();  incrementer.start();  }  }  **OUTPUT:**    **CONCLUSION:**  This program effectively demonstrates the use of a thread to increment a variable every second. It utilizes the sleep() method to create a delay between increments, showcasing basic thread functionality in Java. |
| 36. | Write a program to create three threads ‘FIRST’, ‘SECOND’, ‘THIRD’. Set the priority of the ‘FIRST’ thread to 3, the ‘SECOND’ thread to 5(default) and the ‘THIRD’ thread to 7.  **PROGRAM CODE:**  class MyThread extends Thread {      public MyThread(String name) {          super(name);      }      public void run() {          for (int i = 1; i <= 5; i++) {              System.out.println(getName() + ": " + i);              try {                  Thread.sleep(500);              } catch (InterruptedException e) {                  e.printStackTrace();              }          }      }  }  public class Prac36 {      public static void main(String[] args) {          MyThread firstThread = new MyThread("FIRST");          MyThread secondThread = new MyThread("SECOND");          MyThread thirdThread = new MyThread("THIRD");          firstThread.setPriority(3);          secondThread.setPriority(Thread.NORM\_PRIORITY);          thirdThread.setPriority(7);          firstThread.start();          secondThread.start();          thirdThread.start();      }  }  **OUTPUT:**    **CONCLUSION:**  This program demonstrates thread creation and priority setting in Java. Each thread executes a simple loop, displaying its name and an iteration count, showcasing how thread priority can influence the execution order, although actual execution may vary due to the nature of thread scheduling. |
| 37. | Write a program to solve producer-consumer problem using thread synchronization.  **PROGRAM CODE:**  import java.util.LinkedList;  import java.util.Queue;  class ProducerConsumer {  private final Queue<Integer> queue = new LinkedList<>();  private final int capacity = 5;  public void produce() throws InterruptedException {  int value = 0;  while (true) {  synchronized (this) {  while (queue.size() == capacity) {  wait();  }  queue.add(value);  System.out.println("Produced: " + value);  value++;  notifyAll();  }  Thread.sleep(1000);  }  }  public void consume() throws InterruptedException {  while (true) {  synchronized (this) {  while (queue.isEmpty()) {  wait();  }  int value = queue.poll();  System.out.println("Consumed: " + value);  notifyAll();  }  Thread.sleep(1500);  }  }  }  class Producer extends Thread {  private final ProducerConsumer pc;  public Producer(ProducerConsumer pc) {  this.pc = pc;  }  public void run() {  try {  pc.produce();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  class Consumer extends Thread {  private final ProducerConsumer pc;  public Consumer(ProducerConsumer pc) {  this.pc = pc;  }  public void run() {  try {  pc.consume();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  public class ProducerConsumerExample {  public static void main(String[] args) {  ProducerConsumer pc = new ProducerConsumer();  Producer producer = new Producer(pc);  Consumer consumer = new Consumer(pc);  producer.start();  consumer.start();  }  }  **OUTPUT:**    **CONCLUSION:**  This program effectively demonstrates the Producer-Consumer problem using thread synchronization in Java. The producer generates integers and adds them to a shared queue, while the consumer retrieves and consumes them. Synchronization ensures safe access to the shared resource, preventing data inconsistencies and race conditions. |

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: 2024-25

Part - 8

|  |  |
| --- | --- |
| **No.** | **Aim of the Practical** |
| 38.  39.  40.  41. | Design a Custom Stack using ArrayList class, which mplements following functionalities of stack. My Stack -list ArrayList<Object>: A list to store elements.  +isEmpty: boolean: Returns true if this stack is empty.  +getSize(): int: Returns number of elements in this stack.  +peek(): Object: Returns top element in this stack without  removing it.  +pop(): Object: Returns and Removes the top elements in  this stack.  +push(o: object): Adds new element to the top of this  stack.  **PROGRAM CODE:**  import java.util.ArrayList;  class MyStack {      private ArrayList<Object> list = new ArrayList<>();      public boolean isEmpty() {          return list.isEmpty();      }      public int getSize() {          return list.size();      }      public Object peek() {          if (isEmpty()) {              return "Stack is empty";          }          return list.get(list.size() - 1);      }      public Object pop() {          if (isEmpty()) {              return "Stack is empty";          }          return list.remove(list.size() - 1);      }      public void push(Object o) {          list.add(o);      }  }  public class Prac38 {      public static void main(String[] args) {          MyStack stack = new MyStack();          stack.push(10);          stack.push(20);          stack.push(30);          System.out.println("Top element is: " + stack.peek());          System.out.println("Popped element: " + stack.pop());          System.out.println("Popped element: " + stack.pop());          System.out.println("Is stack empty ? " + stack.isEmpty());          System.out.println("Current stack size: " + stack.getSize());          System.out.println("Top element now: " + stack.peek());      }  }  **OUTPUT:**    **CONCLUSION:**  This Java code defines a stack using an ArrayList to implement basic stack operations such as push, pop, peek, isEmpty, and getSize. The class MyStack manages the stack elements, and Prac38 contains a main method that demonstrates adding elements (push), removing elements (pop), checking the top element (peek), and verifying if the stack is empty.  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 Prac39 {      public static <T extends Comparable<T>> void sort(T[] array) {          for (int i = 0; i < array.length - 1; i++) {              for (int j = 0; j < array.length - i - 1; j++) {                  if (array[j].compareTo(array[j + 1]) > 0) {                      T temp = array[j];                      array[j] = array[j + 1];                      array[j + 1] = temp;                  }              }          }      }      public static void main(String[] args) {          Product[] products = {              new Product("Laptop", 1200),              new Product("Phone", 800),              new Product("Tablet", 600)          };          sort(products);          for (Product product : products) {              System.out.println(product.getName() + " - $" + product.getPrice());          }      }  }  class Product implements Comparable<Product> {      private String name;      private int price;      public Product(String name, int price) {          this.name = name;          this.price = price;      }      public String getName() {          return name;      }      public int getPrice() {          return price;      }      @Override      public int compareTo(Product other) {          return Integer.compare(this.price, other.price);      }  }  **OUTPUT:**    **CONCLUSION:**  This Java code defines a generic sorting method using the bubble sort algorithm to sort arrays of any type that implements the Comparable interface. The Product class represents products with a name and price and implements Comparable<Product> to compare products based on price. In the main method of Prac39, an array of Product objects is created, sorted by price using the sort method, and the sorted products are then printed with their names and prices.  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) {          String text = "Apple Banana Apple Orange Banana Orange Apple Mango Grape Banana";          Map<String, Integer> wordCountMap = new TreeMap<>();          String[] words = text.split("\\s+");          for (String word : words) {              wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);          }          Set<Map.Entry<String, Integer>> entrySet = wordCountMap.entrySet();          for (Map.Entry<String, Integer> entry : entrySet) {              System.out.println(entry.getKey() + ": " + entry.getValue());          }      }  }  **OUTPUT:**    **CONCLUSION:**    This Java code counts the occurrences of each word in a given text using a TreeMap. The string text is split into individual words, and the TreeMap stores each word as a key and its count as the value. The program iterates through the words, updating the count for each word, and then prints the word along with its frequency in sorted (alphabetical) order.  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 Prac41 {      private static final HashSet<String> keywords = new HashSet<>();      static {          String[] keywordArray = {              "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class",              "const", "continue", "default", "do", "double", "else", "enum", "extends", "final",              "finally", "float", "for", "goto", "if", "implements", "import", "instanceof", "int",              "interface", "long", "native", "new", "package", "private", "protected", "public",              "return", "short", "static", "strictfp", "super", "switch", "synchronized", "this",              "throw", "throws", "transient", "try", "void", "volatile", "while"          };          for (String keyword : keywordArray) {              keywords.add(keyword);          }      }      public static void main(String[] args) {          Scanner scanner = new Scanner(System.in);          System.out.print("Enter the path of the Java source file: ");          String filePath = scanner.nextLine();          try {              File file = new File(filePath);              Scanner fileScanner = new Scanner(file);              int keywordCount = 0;              while (fileScanner.hasNext()) {                  String word = fileScanner.next();                  if (keywords.contains(word)) {                      keywordCount++;                  }              }              System.out.println("Number of Java keywords in the file: " + keywordCount);              fileScanner.close();          } catch (FileNotFoundException e) {              System.out.println("File not found: " + filePath);          }      }  }  **OUTPUT:**    **CONCLUSION:**  This Java program counts the number of Java keywords in a given source file. It reads the file path from the user, scans the file for words, and checks if they match any predefined Java keywords. If found, it increments a counter and finally displays the total keyword count. |