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 - VIII

|  |  |
| --- | --- |
| **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 inthis 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 P38 {      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 code defines a custom stack implementation using an `ArrayList` and performs basic stack operations like push, pop, and peek. It also checks if the stack is empty and retrieves its size. The program demonstrates stack functionality with a simple usage example. |
| 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 SortUtil {      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 program demonstrates a generic bubble sort method that sorts an array of `Comparable` objects, specifically `Product` objects, based on their price. The `Product` class implements the `Comparable` interface, allowing the sorting to be based on the price attribute. After sorting, the products are displayed in ascending order of price. |
| 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 WordCounter {      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 program counts the occurrences of each word in a given text and displays them in alphabetical order using a `TreeMap`. It demonstrates basic string manipulation, word counting, and sorting capabilities. |
| 41. | Write a code which counts the number of the keywords in a Java source file. Store all the keywords in a HashSet and use the contains () method to test if a word is in the keyword set.  **PROGRAM CODE:**  import java.io.\*;  import java.util.\*;  public class P41 {  private static final HashSet<String> keywords = new HashSet<>();  static {  String[] keywordArray = {  "abstract", "assert", "boolean", "break", "byte", "case", "catch", "char", "class",  "const", "continue", "default", "do", "double", "else", "enum", "extends", "final",  "finally", "float", "for", "goto", "if", "implements", "import", "instanceof", "int",  "interface", "long", "native", "new", "package", "private", "protected", "public",  "return", "short", "static", "strictfp", "super", "switch", "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 program counts the number of Java keywords in a source file by reading the file and checking each word against a predefined set of keywords stored in a `HashSet`. It demonstrates keyword detection using file handling and basic string comparison. |