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 prc38{  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 :**  import java.util.Arrays;  public class prc39 {  public static <T extends Comparable<T>> void sort(T[] array) {  Arrays.sort(array);  }  static class Product implements Comparable<Product> {  private String name;  private double price;  private double rating;  public Product(String name, double price, double rating) {  this.name = name;  this.price = price;  this.rating = rating;  }  public String getName() {  return name;  }  public double getPrice() {  return price;  }  public double getRating() {  return rating;  }  public int compareTo(Product other) {  // Default sorting by name, can be changed as per requirements  return this.name.compareTo(other.name);  }  public String toString() {  return "Product{name='" + name + "', price=" + price + ", rating=" + rating + "}";  }  }  public static void main(String[] args) {  Product[] products = {  new Product("Smartphone", 800.00, 4.7),  new Product("Laptop", 1200.00, 4.5),  new Product("Tablet", 400.00, 4.3)  };  System.out.println("Before sorting:");  System.out.println(Arrays.toString(products));  sort(products);  System.out.println("After sorting by name:");  System.out.println(Arrays.toString(products));  }  } | |

|  |  |  |
| --- | --- | --- |
|  | **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 prc40 {  public static void main(String[] args) {  Map<String, Integer> wordCountMap = new TreeMap<>();  Scanner scanner = new Scanner(System.in);  System.out.println("Enter words one by one. Type 'exit' when you are done:");  while (true) {  String word = scanner.next().toLowerCase(); // Read and convert to lowercase  if (word.equals("exit")) {  break;  }  wordCountMap.put(word, wordCountMap.getOrDefault(word, 0) + 1);  }  System.out.println("Word occurrences in alphabetical order:");  for (Map.Entry<String, Integer> entry : wordCountMap.entrySet()) {  System.out.println(entry.getKey() + ": " + entry.getValue());  }  }  }  **OUTPUT:**    **CONCLUSION:** In this practical, I learned how to use Java's Map and Set classes to count and display the occurrences of word in a given text. |

|  |  |
| --- | --- |
| **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.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  import java.util.HashSet;  import java.util.Set;  public class prc41 {  private static final Set<String> KEYWORDS = new HashSet<>();  static {  KEYWORDS.add("abstract");  KEYWORDS.add("assert");  KEYWORDS.add("boolean");  KEYWORDS.add("break");  KEYWORDS.add("byte");  KEYWORDS.add("case");  KEYWORDS.add("catch");  KEYWORDS.add("char");  KEYWORDS.add("class");  KEYWORDS.add("const");  KEYWORDS.add("continue");  KEYWORDS.add("default");  KEYWORDS.add("do");  KEYWORDS.add("double");  KEYWORDS.add("else");  KEYWORDS.add("enum");  KEYWORDS.add("extends");  KEYWORDS.add("final");  KEYWORDS.add("finally");  KEYWORDS.add("float");  KEYWORDS.add("for");  KEYWORDS.add("goto");  KEYWORDS.add("if");  KEYWORDS.add("implements");  KEYWORDS.add("import");  KEYWORDS.add("instanceof");  KEYWORDS.add("int");  KEYWORDS.add("interface");  KEYWORDS.add("long");  KEYWORDS.add("native");  KEYWORDS.add("new");  KEYWORDS.add("package");  KEYWORDS.add("private");  KEYWORDS.add("protected");  KEYWORDS.add("public");  KEYWORDS.add("return");  KEYWORDS.add("short");  KEYWORDS.add("static");  KEYWORDS.add("strictfp");  KEYWORDS.add("super");  KEYWORDS.add("switch");  KEYWORDS.add("synchronized");  KEYWORDS.add("this");  KEYWORDS.add("throw");  KEYWORDS.add("throws");  KEYWORDS.add("transient");  KEYWORDS.add("try");  KEYWORDS.add("void");  KEYWORDS.add("volatile");  KEYWORDS.add("while");  }  public static void main(String[] args) throws IOException {  String fileName = "D:\\SEMESTER 3\\JAVA\\Practicals\\Part 8\\src\\Practical\_39.java";  int keywordCount = 0;  try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {  String line;  while ((line = reader.readLine()) != null) {  String[] words = line.split("\\s+");  for (String word : words) {  if (KEYWORDS.contains(word)) {  keywordCount++;  }  }  }  }  System.out.println("Number of keywords: " + keywordCount);  }  }  **OUTPUT:**    **CONCLUSION:**  By this practical we learnt how to find the word occurrences of java keywords in a particular file. |