PART-VIII Collection Framework and Generic

No.	Aim of the Practical
38.	**AIM** : 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 :

```java
import 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;
    }
}
```

```java
    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:

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part8\"
Current top = 4
Pushing 7,8,9 in the stack
Elements in the stack are:
8
7
4
3
2
1
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part8> █
```

OUTPUT: PRACTICAL-38

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.

2

39.

AIM : 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 :

```java
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);
        }
    }
```

```java
}

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:

```
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part8> cd "
c_39 }
Headphones - $150.0 - Rating: 4.7
Monitor - $300.0 - Rating: 4.4
Phone - $800.0 - Rating: 4.3
Laptop - $1200.0 - Rating: 4.5
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part8> 
```

OUTPUT: PRACTICAL-39

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. **AIM :** 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 :

```java
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();
    }
```
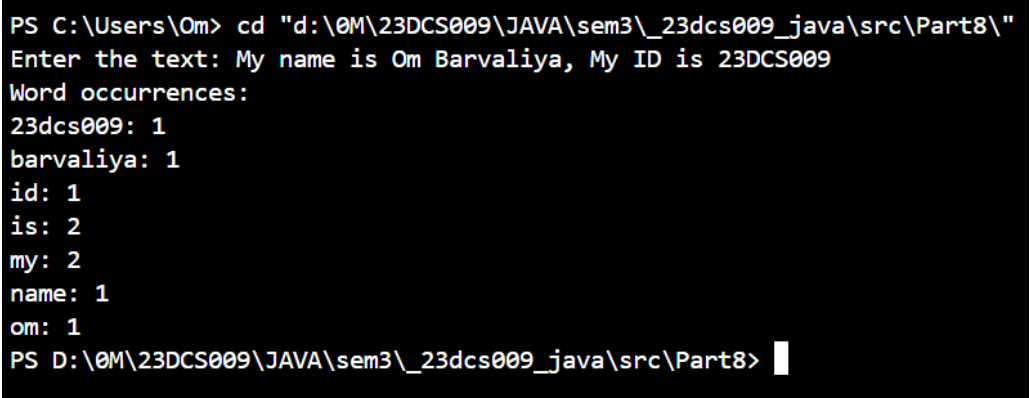
}

OUTPUT:

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part8\"
Enter the text: My name is Om Barvaliya, My ID is 23DCS009
Word occurrences:
23dcs009: 1
barvaliya: 1
id: 1
is: 2
my: 2
name: 1
om: 1
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part8>
```

OUTPUT: PRACTICAL-40

CONCLUSION:

In this practical, We learned how to use Java's Map and Set classes to count and display the occurrences of words in a given text. We 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. **AIM :** 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:

```
PS C:\Users\Om> cd "d:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part8\" ;
Enter the file name you want to scan : Prac_40.java
Total keywords are : 18
PS D:\0M\23DCS009\JAVA\sem3\_23dcs009_java\src\Part8>
```

OUTPUT: PRACTICAL-41

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.