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

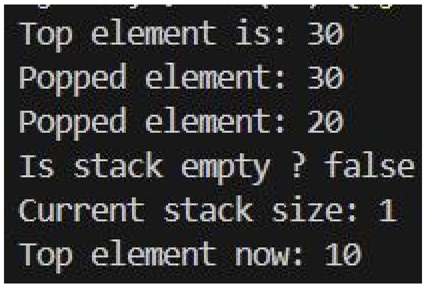
SET- 6

No	Aim of the Practical

38.	Design a Custom Stack using ArrayList class, whichimplements 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 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:



```
Top element is: 30
Popped element: 30
Popped element: 20
Is stack empty ? false
Current stack size: 1
Top element now: 10
```

CONCLUSION:

This program demonstrates the implementation of a custom stack using the ArrayList class in Java. It provides functionalities to push, pop, peek, check if the stack is empty, and get the current size of the stack. The program effectively showcases how to manage a dynamic collection of elements while adhering to stack principles.

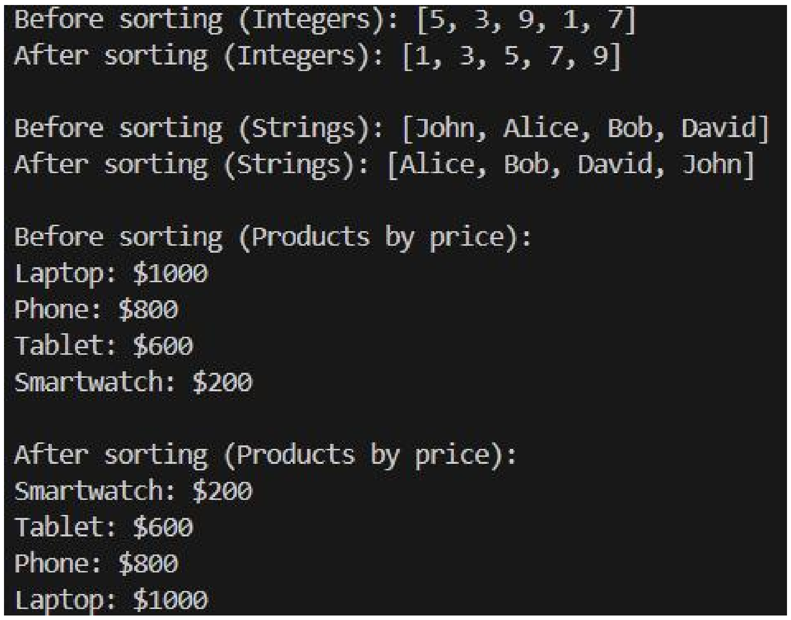
39 Imagine you are developing an e-commerce application. The platform needs to sort lists of products based on different criteria, such as price, rating, or name. Each product object implements the Comparable interface to define the natural ordering. To ensure flexibility and reusability, you need a 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 P39 {

public static <T extends Comparable<T>> void sortArray(T[] array) {

Arrays.sort(array);

}

public static void main(String[] args) {

Integer[] numbers = {5, 3, 9, 1, 7};

System.out.println("Before sorting (Integers): " + Arrays.toString(numbers));

sortArray(numbers);

System.out.println("After sorting (Integers): " + Arrays.toString(numbers));

String[] names = {"John", "Alice", "Bob", "David"};

System.out.println("\nBefore sorting (Strings): " + Arrays.toString(names));

sortArray(names);

System.out.println("After sorting (Strings): " + Arrays.toString(names));
Product[] products = {
new Product("Laptop", 1000),
new Product("Phone", 800),
new Product("Tablet", 600),
new Product("Smartwatch", 200)
};
System.out.println("\nBefore sorting (Products by price): ");
for (Product p : products) {
System.out.println(p);
}
sortArray(products);
System.out.println("\nAfter sorting (Products by price): ");
for (Product p : products) {
System.out.println(p);
} } }
class Product implements Comparable<Product> {
private String name;
private int price;
public Product(String name, int price) {
this.name = name;
this.price = price;
}
@Override
```

```
public int compareTo(Product other) {
return this.price - other.price;
}
@Override
public String toString() {
return name + ": $" + price;
} }
```

OUTPUT:



```
Before sorting (Integers): [5, 3, 9, 1, 7]
After sorting (Integers): [1, 3, 5, 7, 9]

Before sorting (Strings): [John, Alice, Bob, David]
After sorting (Strings): [Alice, Bob, David, John]

Before sorting (Products by price):
Laptop: $1000
Phone: $800
Tablet: $600
Smartwatch: $200

After sorting (Products by price):
Smartwatch: $200
Tablet: $600
Phone: $800
Laptop: $1000
```

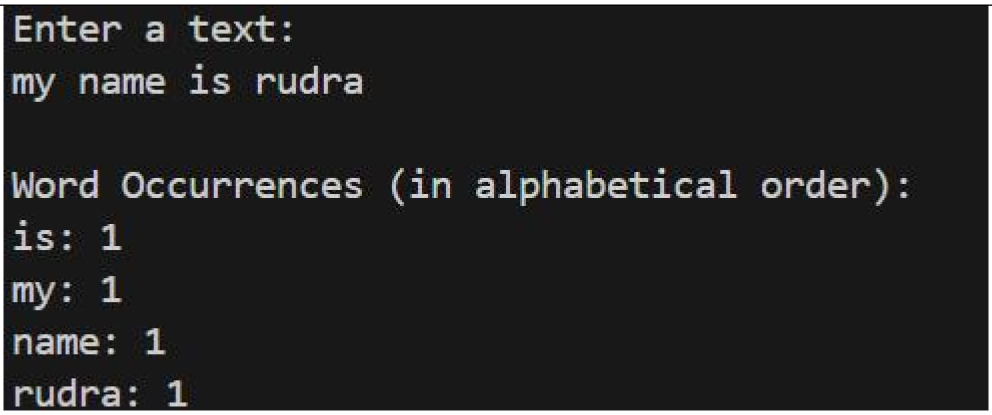
CONCLUSION:
This program demonstrates the use of generics in Java to create a versatile sorting method for arrays of different types. By implementing the Comparable interface in the Product class, it enables sorting of custom objects based on specific criteria, such as price. The output shows the effective sorting of integers, strings, and products, highlighting the flexibility and reusability of the generic sorting method.

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 P40 {
public static void main(String[] args) {
Map<String, Integer> wordMap = new TreeMap<>();
Scanner scanner = new Scanner(System.in);
System.out.println("Enter a text:");
String text = scanner.nextLine();
String[] words = text.toLowerCase().split("\\W+");
for (String word : words) {
if (!word.isEmpty()) {
wordMap.put(word, wordMap.getOrDefault(word, 0) + 1);
} }
System.out.println("\nWord Occurrences (in alphabetical order):");
Set<Map.Entry<String, Integer>> entrySet = wordMap.entrySet();
for (Map.Entry<String, Integer> entry : entrySet) {
System.out.println(entry.getKey() + ": " + entry.getValue());
} } }
```

OUTPUT:



```
Enter a text:
my name is rudra

Word Occurrences (in alphabetical order):
is: 1
my: 1
name: 1
rudra: 1
```

CONCLUSION:

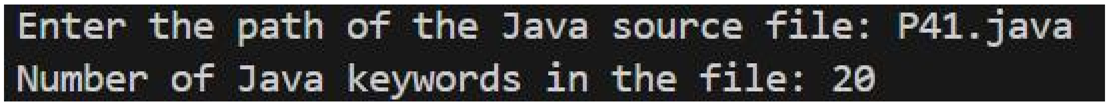
This program demonstrates how to count and display the occurrences of words in a given text using Java's Map and Set classes. The words are stored in a TreeMap, ensuring that they are presented in alphabetical order. The use of getOrDefault() simplifies the counting process, showcasing efficient word frequency analysis.

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",
};
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:



```
Enter the path of the Java source file: P41.java
Number of Java keywords in the file: 20
```

CONCLUSION:

This program demonstrates the use of a HashSet to efficiently count Java keywords in a source file. By reading each word from the file and checking for its presence in the set of keywords, it showcases how to utilize collections for rapid lookups. The result is the total number of keywords, providing a simple yet effective tool for analyzing Java code.