S12BLH31 PROGRAMMING IN JAVA

Unit – III

3 c. Write a Java program to handle multiple exceptions (ArithmeticException, ArrayIndexOutOfBoundsException, and NullPointerException) using separate try-catch blocks.

Algorithm:

Step 1: **ArithmeticException**:

- The program attempts to divide a number by zero, which throws an ArithmeticException.
- This exception is caught, and an appropriate message is displayed.

Step 2: **ArrayIndexOutOfBoundsException**:

- The program tries to access an element at an index that is out of bounds for the array, which throws an ArrayIndexOutOfBoundsException.
- This exception is caught, and an appropriate message is displayed.

Step 3: **NullPointerException**:

- The program attempts to call a method on a null object reference, which throws a NullPointerException.
- This exception is caught, and an appropriate message is displayed.

Program:

```
public class MultipleExceptionsExample {
   public static void main(String[] args) {
      // 1. Handling ArithmeticException
      try {
         int numerator = 10;
         int denominator = 0;
         int result = numerator / denominator;  // This will throw ArithmeticException
         System.out.println("Result of division: " + result);
      } catch (ArithmeticException e) {
         System.out.println("Caught ArithmeticException: " + e.getMessage());
      }
```

```
        // 2. Handling ArrayIndexOutOfBoundsException
        try {
            int[] numbers = {1, 2, 3};
            System.out.println("Accessing fourth element: " + numbers[3]);  // This will throw ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Caught ArrayIndexOutOfBoundsException: " + e.getMessage());
        }

        // 3. Handling NullPointerException
        try {
            String text = null;
            System.out.println("Text length: " + text.length());  // This will throw NullPointerException
        } catch (NullPointerException e) {
            System.out.println("Caught NullPointerException: " + e.getMessage());
        }

        // Program continues after handling all exceptions
        System.out.println("Program continues after handling the exceptions.");
    }
}
```

Output:

Caught ArithmeticException: / by zero

Caught ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3

Caught NullPointerException: Cannot invoke "String.length()" because "text" is null

Program continues after handling the exceptions.

Unit – IV FILE STREAMS AND COLLECTION FRAMEWORK

4a. Write a Java program to read and display the content of the file sample.txt

Algorithm:

Step 1: **Open** a file named sample.txt for writing.
Step 2: Use FileWriter to create a file named sample.txt.
Step 3: **Create** a PrintWriter object to write to the file.
Step 4: **Write** the text "Welcome to File handling" to the file using the PrintWriter object.

Step 5: **Close** the PrintWriter object to finalize writing.
Step 6: **Close** the FileWriter object to release the file resources.

Program:

```
import java.io.*;
    public class Writeonfile
    {
        public static void main(String arg[]) throws IOException
        {
                //opening the file for writing
                FileWriter f=new FileWriter("sample.txt");
                //creation of the object for writing
                PrintWriter out=new PrintWriter(f);
                //writing text on the file
                out.println("Welcome to File handling");
                //closing the output channel and the file
                out.close();
                f.close();
        }
}
```

Output

Sample.txt

Welcome to File handling

4b. Program for Copying a File Containing Sentences Using FileInputStream and FileOutputStream

Algorithm:

Step1: Assume source.txt contains the following three sentences:

This is the first sentence.
This is the second sentence.
This is the third sentence.

Step 2: **FileInputStream** class reads the file source.txt byte by byte.

Step 3: **FileOutputStream** class writes the content read from source.txt to destination.txt.

Step 4:

- The program reads each byte from source.txt and writes it to destination.txt until the entire content is copied.
- The process continues until the end of the file is reached (indicated by fis.read() returning -1).

Step 5: **Exception Handling is used** to close the file streams automatically and handles potential IOException.

Program:

```
import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.io.IOException;


public class FileCopyAndDisplayExample {
   public static void main(String[] args) {
      String sourceFile = "source.txt";
      String destinationFile = "destination.txt";


      // Copy the content from source.txt to destination.txt
      try (FileInputStream fis = new FileInputStream(sourceFile);
          FileOutputStream fos = new FileOutputStream(destinationFile)) {


         int byteData;
         while ((byteData = fis.read()) != -1) {
            fos.write(byteData);
         }
         System.out.println("File copied successfully!");


      } catch (IOException e) {
         e.printStackTrace();  // Handle any IO exceptions
      }


      // Display the copied data from destination.txt
      try (FileInputStream fis = new FileInputStream(destinationFile)) {
```

```
            System.out.println("Copied data from " + destinationFile + ":");

            int byteData;
            while ((byteData = fis.read()) != -1) {
                System.out.print((char) byteData);
            }

        } catch (IOException e) {
            e.printStackTrace();  // Handle any IO exceptions
        }
    }
}
```

Output:

File copied successfully!

Copied data from destination.txt:

This is the first sentence.

This is the second sentence.

This is the third sentence.

4c. Program to implement Collection Framework using ArrayList, HashSet, and HashMap

Algorithm:

Step 1:

Create an ArrayList, a resizable array implementation of the List interface. There is the option to allow duplicate elements and maintain insertion order. Add some fruit names to the ArrayList, including a duplicate "Apple", then iterate over the list and print each fruit.

Step 2:

Create a HashSet, an implementation of a Set interface, in which duplicate elements are not allowed, and no particular order is guaranteed. Using the ArrayList, create a HashSet that automatically removes the duplicate "Apple" and then iterate over the set and print each unique fruit.

Step 3:

Create a HashMap, which implements the Map interface and maps keys to values. A key can be mapped to a maximum of one value. It counts the fruit occurrences in the ArrayList and stores them in a HashMap. Then, iterate over the map and print each fruit along with its count.

Program:

```
import java.util.ArrayList;

import java.util.HashSet;

import java.util.HashMap;


public class CollectionsExample {
   public static void main(String[] args) {
      // ArrayList Example
      ArrayList<String> fruits = new ArrayList<>();
      fruits.add("Apple");
      fruits.add("Banana");
      fruits.add("Orange");
      fruits.add("Apple");  // Duplicate element
      System.out.println("ArrayList:");
      for (String fruit : fruits) {
         System.out.println(fruit);
      }

      // HashSet Example
      HashSet<String> uniqueFruits = new HashSet<>(fruits);  // Remove duplicates
      System.out.println("\nHashSet:");
      for (String fruit : uniqueFruits) {
         System.out.println(fruit);
      }

      // HashMap Example
      HashMap<String, Integer> fruitCount = new HashMap<>();
      for (String fruit : fruits) {
```

```
        fruitCount.put(fruit, fruitCount.getOrDefault(fruit, 0) + 1);  // Count occurrences
    }

    System.out.println("\nHashMap:");
    for (String fruit : fruitCount.keySet()) {
        System.out.println(fruit + ": " + fruitCount.get(fruit));
    }
  }
}
```

Output:

ArrayList:

Apple

Banana

Orange

Apple

HashSet:

Banana

Orange

Apple

HashMap:

Apple: 2

Banana: 1

Orange: 1