OBJECT-ORIENTED PROGRAMMING
LAB 9: EXCEPTIONS AND FILE HANDLING

I. Objective

After completing this lab tutorial, you can:

- Understand and implement exception handling in Java programs.
- Understand and implement file handling in Java programs.

II. Motivation

There are three types of errors:

- **Syntax errors** occur when the rule of the language is violated and detected by the compiler.
- **Run-time errors** occur when the computer detects an operation that cannot be carried out (e.g., division by zero, x/y is syntactically correct, but if y is zero at run-time a run-time error will occur).
- **Logic errors** occur when a program does not perform the intended task.

Instead of deciding how to deal with an error, Java provides the exception mechanism:

- Indicate an error (exception event) has occurred.
- Let the user decide how to handle the problem in a separate section of code specific to that purpose.
- Crash the program if the error is not handled.

III. Exception Indication

1. Use built-in exception class

There are many useful predefined exception classes, for example:

- `ArithmeticException`
- `NullPointerException`
- `IndexOutOfBoundsException`
- `IllegalArgumentException`
- `InputMismatchException`

The following Java programs will illustrate the use of *Exception Indication*.

```
import java.util.Scanner;
import java.util.InputMismatchException;

public class SampleException {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        boolean isError = false;

        do {
            System.out.println("Enter an integer number");
            try {
                int num = sc.nextInt();
                System.out.println("num = " + num);
                isError = false;
            } catch (InputMismatchException e) {
                System.err.println("Incorrect input");
                sc.nextLine(); // skip newline
                isError = true;
            }
        } while (isError);
    }
}
```

```
public class SampleException {
  public static double factorial(int n) throws IllegalArgumentException {
      if (n < 0) {
          IllegalArgumentException obj = new IllegalArgumentException(n + " is
invalid.");
          throw obj;
      } else {
          double output = 1;
          for (int i = 2; i <= n; i++) {
              output *= i;
          }
          return output;
      }
  }

  public static void main(String[] args) {
      System.out.println("n = 5 --> " + factorial(5));
      System.out.println("n = -1 --> " + factorial(-1));
      /*
      try {
          System.out.println("n = -1 --> " + factorial(-1));
      } catch (IllegalArgumentException err) {
          System.out.println(err);
      }
      */
      System.out.println("n = 6 --> " + factorial(6));
  }
}
```

2. Define new Exception class

New exception classes can be defined by deriving from class Exception. Then it can be used in `throw` statements and `catch` blocks.

```
public class MathException extends Exception {
    public MathException() {
        super();
    }

    public MathException(String s) {
        super(s);
    }
}
```

```
public class SampleException {
  public static double factorial(int n) throws MathException {
      if (n < 0) {
          throw new MathException(n + " is invalid.");
      } else {
          double output = 1;
          for (int i = 2; i <= n; i++) {
              output *= i;
          }
          return output;
      }
  }

  public static void main(String[] args) throws MathException {
      System.out.println("n = 5 --> " + factorial(5));
      System.out.println("n = -1 --> " + factorial(-1));
      /*
      try {
          System.out.println("n = -1 --> " + factorial(-1));
      } catch (MathException err) {
          System.out.println(err);
      }
      */
      System.out.println("n = 6 --> " + factorial(6));
  }
}
```

IV. Java File Handling

File handling is an important part of any application.

Java has several methods for *creating*, *reading*, *updating*, and *deleting* files.

The *File* class from the java.io package, allows us to work with files.

To use the *File* class, create an object of the class, and specify the filename or directory name

```
import java.io.File;  // Import the File class

File myObj = new File("filename.txt"); // Specify the filename
```

The File class has many useful methods for creating and getting information about files.

Method	Type	Description
canRead()	boolean	Tests whether the file is readable or not
canWrite()	boolean	Tests whether the file is writable or not
createNewFile()	boolean	Create an empty file

delete()	boolean	Deletes a file
exists()	boolean	Tests whether the file exists
getName()	String	Returns the name of the file
getAbsolutePath()	String	Returns the absolute pathname of the file
length()	long	Returns the size of the file in bytes
list()	String[]	Returns an array of the files in the directory
mkdir()	boolean	Creates a directory

1. Create a File

You can use the `createNewFile()` method, to create a file in Java. This method returns a boolean value: `true` if the file was successfully created, and `false` if the file already exists. Note that the method is enclosed in a `try…catch` block. This is necessary because it throws an `IOException` if an error occurs (if the file cannot be created for some reason):

```
import java.io.File;  // Import the File class
import java.io.IOException;  // Import the IOException class to handle errors

public class CreateFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      if (myObj.createNewFile()) {
        System.out.println("File created: " + myObj.getName());
      } else {
        System.out.println("File already exists.");
      }
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Note: To create a file in a specific directory (requires permission), specify the path of the file and use double backslashes to escape the "\" character (for Windows). You can just write the path on Mac and Linux, like */Users/name/filename.txt*.

2. Write to a File

In the following example, we use the `FileWriter` class together with its `write()` method to write some text to the file we created in the example above. Note that when you are done writing to the file, you should close it with the `close()` method:

```
import java.io.FileWriter;   // Import the FileWriter class
import java.io.IOException;  // Import the IOException class to handle errors

public class WriteToFile {
  public static void main(String[] args) {
    try {
      FileWriter myWriter = new FileWriter("filename.txt");
      myWriter.write("Hello world! Hello TDTU!");
      myWriter.close();
      System.out.println("Successfully wrote to the file.");
    } catch (IOException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Note: Many available classes in the Java API can be used to write files in Java: `FileWriter`, `BufferedWriter`, `FileOutputStream`, etc.

3. Read a File

In the following example, we use the Scanner class to read the contents of the text file we created in the example above.

```
import java.io.File;  // Import the File class
import java.io.FileNotFoundException;  // Import this class to handle errors
import java.util.Scanner; // Import the Scanner class to read text files

public class ReadFile {
  public static void main(String[] args) {
    try {
      File myObj = new File("filename.txt");
      Scanner myReader = new Scanner(myObj);
      while (myReader.hasNextLine()) {
        String data = myReader.nextLine();
        System.out.println(data);
      }
      myReader.close();
    } catch (FileNotFoundException e) {
      System.out.println("An error occurred.");
      e.printStackTrace();
    }
  }
}
```

Note: Many available classes in the Java API can be used to read in Java: `FileReader`, `BufferedReader`, `File`, `Scanner`, `FileInputStream`, etc. Which one to use depends on the Java version you're working with and whether you need to read bytes or characters, the size of the file/lines, etc.

4. Get File Information

To get more information about a file, use any of the File methods:

```
import java.io.File;  // Import the File class

public class GetFileInfo {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.exists()) {
      System.out.println("File name: " + myObj.getName());
      System.out.println("Absolute path: " + myObj.getAbsolutePath());
      System.out.println("Writeable: " + myObj.canWrite());
      System.out.println("Readable " + myObj.canRead());
      System.out.println("File size in bytes " + myObj.length());
    } else {
      System.out.println("The file does not exist.");
    }
  }
}
```

5. Delete a File

To delete a file in Java, use the delete() method:

```
import java.io.File;  // Import the File class

public class DeleteFile {
  public static void main(String[] args) {
    File myObj = new File("filename.txt");
    if (myObj.delete()) {
      System.out.println("Deleted the file: " + myObj.getName());
    } else {
      System.out.println("Failed to delete the file.");
    }
  }
}
```

V. Exercises

1. Create class Calculator has 2 methods below:

   - **public double divide(int a, int b)**
   - **public int multiply(int a, int b)**

   Implement exceptions:

- If parameter b = 0, the method throws an exception `ArithmeticException` with the message "divide by zero".
- If the values of a and b are outside the range [-1000, 1000], the method throws an exception `NumberOutOfRangeException` with the message "Number is outside the computation". `NumberOutOfRangeException` is a student-defined exception.

2. Write a Java program:
   - Read all contents from an **input.txt** file.
   - Then uppercase all contents and write the results to the output.txt file.

   For example:

input.txt	**output.txt**
information technology object oriented programming	INFORMATION TECHNOLOGY OBJECT ORIENTED PROGRAMMING

3. Write a method **public <E> boolean writeFile(String path, ArrayList<E> lst)** to write the generic **ArrayList** and apply it to write the result of exercise 3 in the Classwork section of Lab 8 to a file with format **sName - sID - gpa** with each line is one **Student** object, instead of print on the command prompt.
4. Write a Java program:
   - Get specific files by extensions from a given folder.
   - Check if a file or directory specified by pathname exists or not.
   - Check if the given pathname is a directory or a file.
   - Append text to an existing file.
   - Find the longest word in a text file.
5. Write a Java program:
   - Read all integers from an *input.txt* file.
   - Then calculate the sum of them and write the result to the *output.txt* file.

input.txt	**output.txt**
0 12 8 4 6 100 1 9	140