

## COMP2026 Problem Solving Using Object Oriented Programming

---

### Laboratory 7

#### Part A Discovery Exercises

##### Task 1: Exception Handling

The given **NumberFormat** program reads in a String and converts it into integer type by the `Integer.parseInt()` method.

- a) Run the given **NumberFormat** program with '123' as input and paste the screenshot of the output below.

- b) Run the program again with '123abc' as input and paste the screenshot of the output below.

- c) Run the program again with ' 123 ' as input and paste the screenshot of the output below.

- d) Run the program again with ' 123abc ' as input and paste the screenshot of the output below.

Referring to the Java API Documentation of the `parseInt()` method at <https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html>,

**parseInt**

```
public static int parseInt(String s)
    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all be decimal digits, except that the first character may be an ASCII minus sign '-' ('\u002D') to indicate a negative value or an ASCII plus sign '+' ('\u002B') to indicate a positive value. The resulting integer value is returned, exactly as if the argument and the radix 10 were given as arguments to the `parseInt(java.lang.String, int)` method.

**Parameters:**  
s - a String containing the int representation to be parsed

**Returns:**  
the integer value represented by the argument in decimal.

**Throws:**  
NumberFormatException - if the string does not contain a parsable integer.

the `parseInt()` method throws a `NumberFormatException` if the input string does not contain a parsable integer. A parsable integer is a string with only decimal digits that ends with either a space character or a newline.

An Exception is an unexpected event that interrupts the normal flow of the program. If an exception occurs, which has not been handled by programmer then the program execution gets terminated and a system generated error message is shown to the user as part (b) above.

Try-catch block is one of the ways for handling exception.

e) Modify the **NumberFormat** program to add the try-catch block to handle the exception as below.

```
void runApp() {
    Scanner in = new Scanner(System.in);
    try {
        System.out.print("input> ");
        int num = Integer.parseInt(in.next());
        System.out.println("num: " + num);
    } catch (NumberFormatException e) {
        System.out.println("NumberFormatException encountered!");
        System.out.println("The input value cannot be converted to integer.");
    }
}
```

- f) Run the program again with '123abc' as input and paste the screenshot of the output below.

- g) Modify the **NumberFormat** program to add the `e.printStackTrace()` method in the catch block to print what happened exactly and where the exception occurred in the source code.

```
void runApp() {  
    Scanner in = new Scanner(System.in);  
    try {  
        System.out.print("input> ");  
        int num = Integer.parseInt(in.next());  
        System.out.println("num: " + num);  
    } catch (NumberFormatException e) {  
        System.out.println("NumberFormatException encountered!");  
        System.out.println("The input value cannot be converted to integer.");  
        e.printStackTrace();  
    }  
}
```

- h) Run the program again with '123abc' as input and paste the screenshot of the output below.

Similarly, referring to the Java API Documentation of the Scanner class at <https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html>,

**Scanner**

```
public Scanner(File source)
    throws FileNotFoundException
```

Constructs a new Scanner that produces values scanned from the specified file. Bytes from the file are converted into characters using the underlying platform's default charset.

**Parameters:**  
source - A file to be scanned

**Throws:**  
FileNotFoundException - if source is not found

a `FileNotFoundException` is thrown if the input file is not found while creating the Scanner for file reading file. This exception can also be nicely handled by the try-catch block.

Here is an example:

```
...
File inputFile = new File("input.txt");

try {
    Scanner in = new Scanner(inputFile);
}
catch (FileNotFoundException e){
    e.printStackTrace();
}
...
```

- i) Try to run the given **FileReading** program with the file "input.txt" placed in the project folder. Paste the screenshot of the output below.

- j) Add "throws Exception" to the main() and runApp() method declarations to make the program run without any problem.

```
public static void main(String[] args) throws Exception{
    new FileReading().runApp();
}

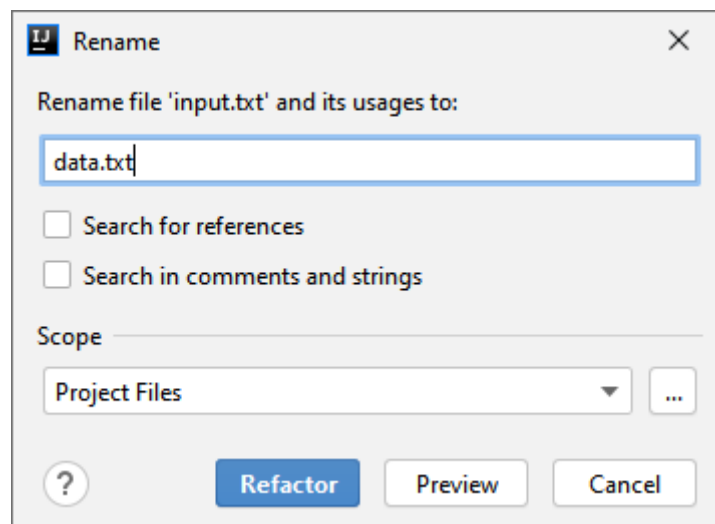
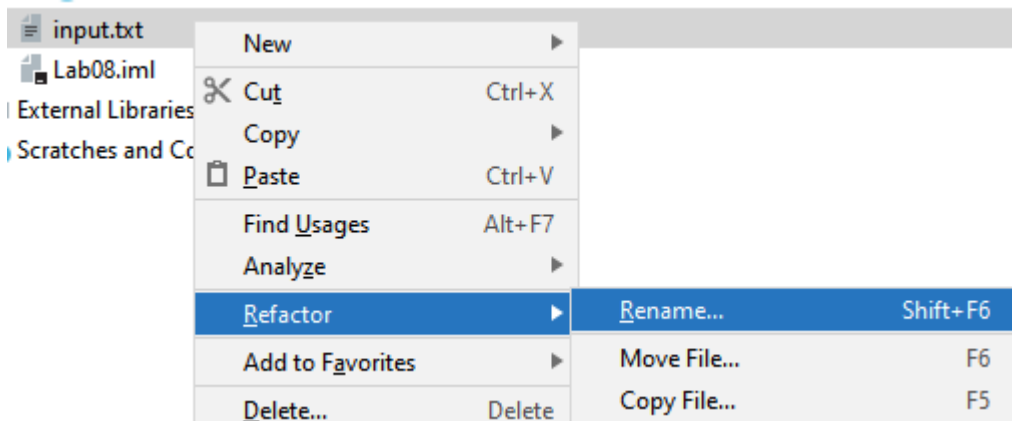
void runApp() throws Exception{

    File inFile = new File( pathname: "input.txt");

    Scanner inputFile = new Scanner(inFile);
    while (inputFile.hasNextLine()) {
        String s = inputFile.nextLine();
        System.out.println(s);
    }
    inputFile.close();
}
```

Paste the screenshot of the output below.

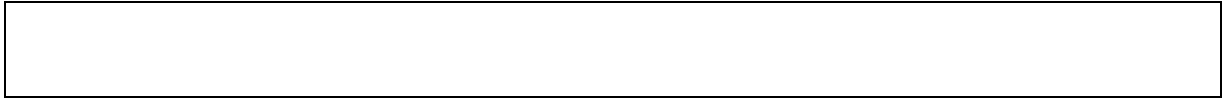
- k) In the project folder, rename the file “input.txt” to “data.txt”. Do NOT modify the program.



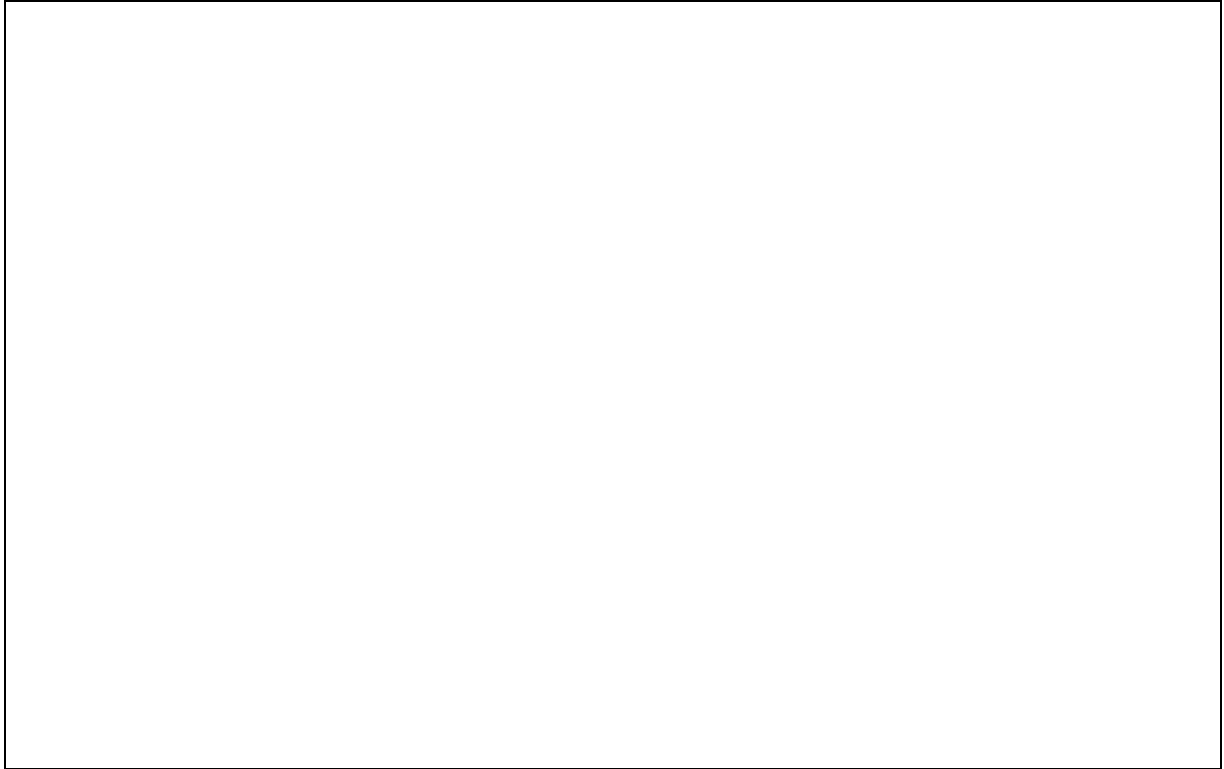
Run the program and paste the screenshot of the output below.

- l) Modify the program to catch the `FileNotFoundException` by the try-catch block and print “Input file does not exist!” as the error message. Run the program and paste the screenshot of the output below.

- m) Remove the “throws Exception” in the main() and runApp() method declarations. Run the program and paste the screenshot of the output below.



- n) Copy and paste the source code of the program done in part m) below.



## Task 2: Try-catch-finally

### Handling Errors

Many errors can occur during the runtime of a program, including hardware errors, communication errors, and memory access errors. Java uses “exceptions” to provide error-handling capabilities.

### Exceptions

When an error occurs within a method, the method creates an exception object and hands it off to the runtime system. This is called throwing an exception. The exception object contains information about the error, which is useful for debugging and error handling.

### The “Catch or Throws” Requirement

In Java, if a method may throw an exception, the exception must be either:

- Caught within the method (by a try-catch block)
- Not caught within the method, but the method declaration must specify the exception that may be thrown by the throws keyword, e.g.:

```
public void someMethod() throws Exception { ... }
```

This leaves the exception handling to the caller of `someMethod()`.

### Try-catch block

A try-catch block is used to run codes that may raise exceptions and catch it if it is thrown.

```
try {  
    //Program logic that may throw exceptions  
} catch (Exception e) {  
    //Do something;  
}
```



**TryCatchExample.java**

```
import java.io.*;
import java.util.*;

public class TryCatchExample {
    public static void main(String[] args) {
        new TryCatchExample().runApp();
    }

    void runApp() {
        Scanner in = new Scanner(System.in);
        String filename = "output.txt";
        PrintWriter out = null;
        try {

            out = new PrintWriter(filename);
            int num;
            do {
                System.out.print("Input>");
                String s = in.nextLine();
                num = Integer.parseInt(s);
                if (num != -1) {
                    out.println(num * num);
                }
            } while (num != -1);
            out.close();
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

a) Run the given TryCatchExample.java program with the following input.

```
Input>1
Input>2
Input>3
Input>4
Input>5
Input>-1
```

b) Paste the content of the `output.txt` below.

c) Run the given `TryCatchExample.java` program again with the following input.

```
Input>1
Input>2
Input>a
```

d) What's the content of the `output.txt`? Why?

### Try-catch-finally block

The **finally** block will be run no matter if an exception is caught or not. If an exception is caught, the **finally** block will be run after running the catch block.

```
try {
    //Program logic that may throw exceptions
} catch (Exception e) {
    //Do something;
} finally {
    //Do something;
}
```

e) Add the finally block to the TryCatchExample.java as follow.

```
...
void runApp() {
    Scanner in = new Scanner(System.in);
    String filename = "output.txt";
    PrintWriter out = null;
    try {
        out = new PrintWriter(filename);
        int num;
        do {
            System.out.print("Input>");
            String s = in.nextLine();
            num = Integer.parseInt(s);
            if (num != -1) {
                out.println(num * num);
            }
        } while (num != -1);
        //out.close(); removed
    } catch (Exception e) {
        System.out.println(e.getMessage());
    } finally {
        out.close();
    }
}
...
```

f) Run the given TryCatchExample.java program again with the following input.

```
Input>1
Input>2
Input>a
```

g) What's the content of the output.txt? Why?

### Task 3: Creating Exceptions

#### Age.java

```
import java.util.Scanner;

public class Age {
    public static void main(String[] args) {
        new Age().runApp();
    }

    void runApp() {
        while (true) {
            age();
        }
    }

    void age() {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your age>");
        int age = in.nextInt();
        System.out.println("You are " + age + " years old.");
    }
}
```

- a) Run the given Age.java program and enter "18" as input. Paste the output below.

- b) Run the given Age.java program again and enter "-10" as input. Paste the output below.

In part b, the input does not make sense. Apart from handling the problem by some if-else statements, we could handle it by throwing an exception.

c) Modify the Age.java as follow.

```
import java.util.Scanner;

public class Age {
    public static void main(String[] args) {
        new Age().runApp();
    }

    void runApp() {
        while (true) {
            try {
                age();
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }

    void age() throws Exception{
        Scanner in = new Scanner(System.in);
        System.out.print("Enter your age>");
        int age = in.nextInt();
        if (age < 0) {
            throw new Exception("Invalid Age!");
        }
        System.out.println("You are " + age + " years old.");
    }
}
```

d) Run the Age.java program again and enter "-10" as input. Paste the output below.

## Task 4: Debugger

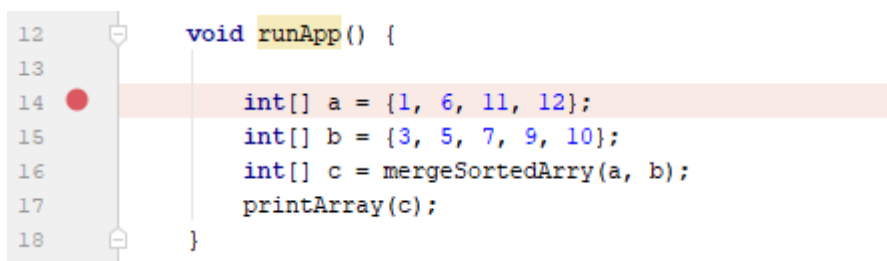
Computer programs rarely run perfectly the first time. It can be quite frustrating to find the bugs. We can insert print commands, run the program, and try to analyse the printout. If the printout does not clearly point to the problem, we may need to add and remove print commands and run the program again. That can be a time-consuming process.

Debugger is a tool that help us locate bugs by letting us follow the execution of a program. We can stop and restart the program and see the contents of variables whenever the program is temporarily stopped.

Common functions in a debugger:

### Breakpoints

Breakpoint allows stopping program execution at certain point. Breakpoints can be set by hovering the mouse over the Editor's gutter area and clicking on it. In IntelliJ, breakpoints are denoted using red circle symbols. To remove breakpoint just click on the same symbol.



```
12 void runApp() {  
13  
14     int[] a = {1, 6, 11, 12};  
15     int[] b = {3, 5, 7, 9, 10};  
16     int[] c = mergeSortedArray(a, b);  
17     printArray(c);  
18 }
```

The screenshot shows a code editor with a method named `runApp()`. A red circle breakpoint is set on line 14, which contains the declaration of array `a`. The line is highlighted in orange.

### Step over

Execute the current line of code and move to the next line of code.

### Step into

If a method is encountered while debugging, move into the method instead of the next line of code.

### Step out

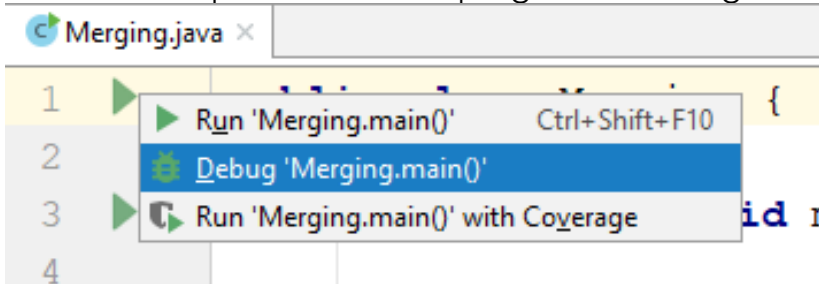
Finish the execution of the current method and jump out of the method.

- a) Create a project in IntelliJ and add the given **Merging.java** into it. This program merges the content of two sorted arrays into a new sorted array. However, there are some bugs in this program. Let's find the bugs by using the debugger.
- b) Run the program and paste the output below.

- c) Add breakpoints to line 9 and line 11.


```
7 void runApp() {  
8  
9 int[] a = {1, 6, 11, 12};  
10 int[] b = {3, 5, 7, 9, 10};  
11 int[] c = mergeSortedArray(a, b);  
12 printArray(c);  
13 }  
14
```

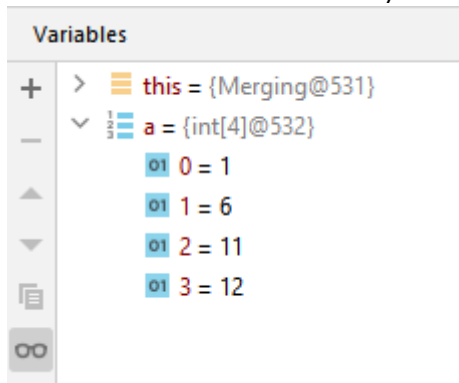
- d) Click the green arrow icon at the top left corner of the editor window and select the second option to run the program in debug mode.



- e) The problem execution pauses at line 9 now.


```
7 void runApp() {  
8  
9 int[] a = {1, 6, 11, 12};  
10 int[] b = {3, 5, 7, 9, 10};  
11 int[] c = mergeSortedArray(a, b);  
12 printArray(c);  
13 }
```

- f) Click the step over button  once to execute line 9. In the Variables window, view the content of array **a**.



- g) Click the step over button again to execute line 10. In the Variables windows, view the content of array **b** and past the screenshot of the Variables window below.



- h) Click the step into button  to move into the **mergeSortedArray** method. Step through the method line by line. Observe the content of array **c**, and the values of variables **i**, **j** and **k**. Paste the screenshot of the content of array **c** when the value of **k** reaches 4.





- i) Keep stepping through the **mergeSortedArray** method, find the errors and fix the errors. Paste the corrected method below. Make minimal changes to the method while you are fixing it.

## Part B Programming Exercises

### Task 1: Sale Amount

A hotel salesperson enters sales in a text file. Each line contains the following, separated by commas: The name of the client, the service sold (Dinner, Conference, and Lodging), and the amount of the sale. Write a program named **SaleAmount** that reads such a file and displays the total amount for each service category. Display an error if the file does not exist or the format is incorrect.

Note:

- Use the `split()` method in the `String` library to extract the values from a line
- Use the `equalsIgnoreCase()` method to compare `String` values of the service sold
- Use `Double.parseDouble()` to parse a `String` into a `double`

Sample outputs:

```
Enter the filename: sale.txt
Error: The file sale.txt is not found.

Process finished with exit code 0
```

```
Enter the filename: sale1.txt
Dinner Sale Amount: 505000.0
Conference Sale Amount: 645000.0
Lodging Sale Amount: 1159000.0

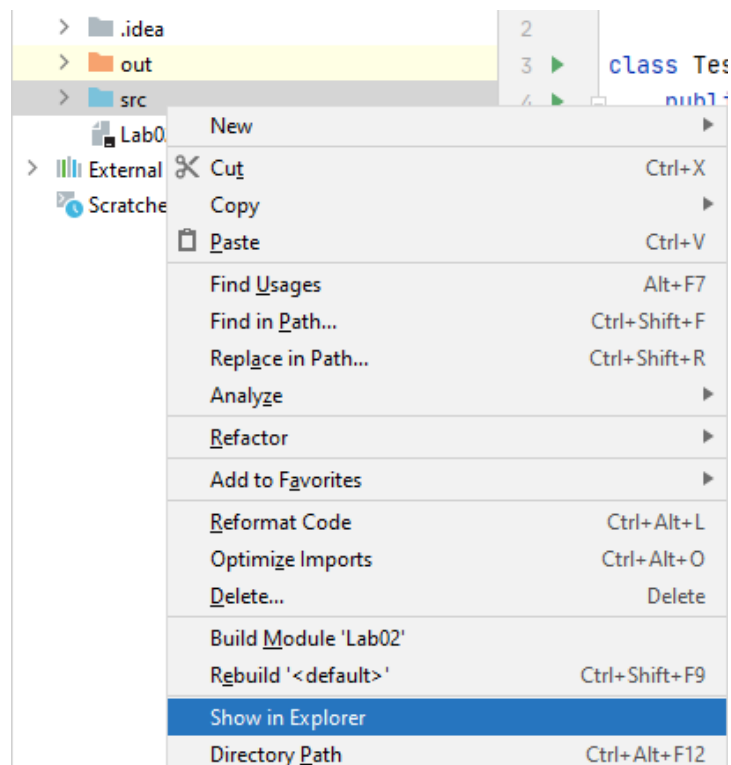
Process finished with exit code 0
```

```
Enter the filename: sale2.txt
Error: Line 2: Incorrect line format!
Error: Line 4: Incorrect line format!
Error: For input string: "I54000.0"
Error: Line 7: Incorrect service category!
Error: Line 9: Incorrect service category!
Dinner Sale Amount: 505000.0
Conference Sale Amount: 345000.0
Lodging Sale Amount: 1543000.0

Process finished with exit code 0
```

## Part C Submitting Exercises

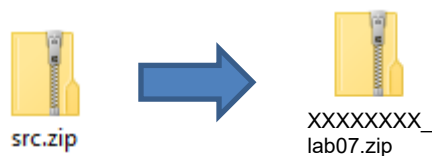
Step 1: Right-click the **src** folder and select **Show in Explorer**



Step 2: Zip the **src** folder into **src.zip**



Step 3: Rename the **src.zip** file to **XXXXXXXX\_lab07.zip** where **XXXXXXXX** is your **student id**



Step 4: Submit **XXXXXXXX\_lab07.zip** and **XXXXXXXX\_lab07.docx** to Moodle.



## References

- [1] Bravaco, R., & Simonson, C. (2009). *Java programming: From the ground up*. Dubuque, IA: McGraw-Hill.
- [2] Dean, J., & Dean, R. (2008). *Introduction to programming with Java: A problem solving approach*. Boston: McGraw-Hill.
- [3] Farrell, J. (2012). *Java programming*. Boston, MA: Course Technology Cengage Learning
- [4] Forouzan, B. A., & Gilberg, R. F. (2007). *Computer science: A structured programming approach using C (3rd ed.)*. Boston, MA: Thomson Course Technology.
- [5] Gaddis, T. (2016). *Starting out with Java (6th ed.)*. Pearson.
- [6] Liang, Y. D. (2013). *Introduction to Java programming: Comprehensive version*. (8<sup>th</sup> ed.). Pearson.
- [7] Schildt, H. ( 2006). *Java a beginner's guide*. New York: McGraw Hill.
- [8] Schildt, H., & Skrien, D. J. (2013). *Java programming: A comprehensive introduction*. New York: McGraw-Hill.
- [9] Wu, C. T. (2010). *An introduction to object-oriented programming with Java*. Boston: McGraw Hill Higher Education
- [10] Xavier, C. (2011). *Java programming: A practical approach*. New Delhi: Tata McGraw Hill.
- [11] yet another insignificant Programming Notes. (n.d.). Retrieved from <https://www3.ntu.edu.sg/home/ehchua/programming>
- [12] Zakhour, S., Kannan, S., & Gallardo, R. (2013). *The Java tutorial: A short course on the basics (5th ed.)*.