# CS F213 Object Oriented Programming

# Minor Assignment 2

**General instructions for all questions:**

* ***Your output should not have any leading or trailing whitespace(s) or newline character(s).***
* **More offline test cases will be used for the final grading.**
* **On the VPL interface on Moodle:**
  + **Go to the question**
  + **You can either go to the Submission tab or the Edit tab. The Submission tab lets you upload a program that you have written offline. The Edit tab lets you use the in-built IDE to write your program**
  + **In both cases, you can click the run / evaluate button to run your program and test it against the online test cases**
  + **The interface will tell you how many test cases have passed**
  + **The main method should be in a public class. The name of the public class should be the same as the Java filename**
* **The plagiarism penalty remains the same as Minor Assignment 1.**

## 

**Question 1:**

Write a Java program that reads input from a file (the filename is passed as a commandline input) and writes the final output to the console. The input provided will contain 3 nXn matrices (A, B and C) in a format described below. The program should compute the product (multiplication) of the 3 input matrices ((AxB)xC), and print the output to the console according to the format described below. Your program should employ multiple Java threads for the computation, as execution time will be an evaluation parameter.

In the program, you need to wait for all the threads to complete execution and only then exit the program. You will also need to add all the relevant exception-handling logic based on your implementation (read below on guidelines for exception handling).

Please note that how many threads to create and how to use them within the program is completely up to your implementation.

**Constraints:**

* 1 <= n <= 4000
* 1 <= Matrix[i][j] <= 1000000

**NOTE**

* Use relevant packages as required.
* You are not allowed to use any package or library which provides matrix arithmetic classes and methods.

**Input format**

First line in input file will tell the value of n

Next three lines will contain nXn elements in row-major format, with each element separated by a single space.

2

1 2 3 4

5 6 7 8

12 34 45 67

Here, each of the nXn matrix elements will be considered in the following manner:

1 2 3 4 -> [ [ 1, 2 ], [ 3, 4 ] ]

**Output format (on the console)**

A single line containing nXn single-space-separated elements which is the product of the three input matrices in row-major format.

1218 2120 2766 4812

The result matrix should be converted to a single line in the following manner:

[ [ 1, 2 ], [ 3, 4 ] ] -> 1 2 3 4

**Exception handling**

The exception messages, if any, should be clear and relevant and from the glossary of exceptions provided at the end of this document.

**Grading guidelines**

A few of the factors that will be considered for grading:

* The number of threads employed to solve the program
* Exception handling
* The execution speed
* Number of test cases (on and off the platform) passed

# 

**Question 2:**

Write a Java program that creates a banking system with concurrent deposits and withdrawals using threads. The deposits and withdrawals may happen in any order and concurrently and must preserve the semantic meaning of a bank balance.

You will have to initialize the Bank class to a balance of 0, and after every transaction (deposit/withdrawal) you will have to print the new balance to the console. All transactions will involve only integer values.

In the program, you have to wait for all the threads to complete execution and then only exit the program. You will also need to add all the relevant exception-handling logic based on your implementation.

Please note that how many threads and how to use them within the program is completely up to your implementation, but faster execution time gets more credit.

**Constraints:**

* 1 <= deposit\_amount, withdrawal\_amount <= 1000000000

**Code Constraints:**

Your code is required to use the following structure for the main method:

// main() starts here

public static void main(String[] args) {

BankAccount account = new BankAccount();

// Read filename from console, l number of lines in the file

for (int i = 0; i < l; i++) {

//Read a line from the input file

//check if deposit

// start a new thread for deposit

//check if withdrawal

// start a new thread for withdrawal

}

for (int i = 0; i < l; i++) {

threads[i].start();

}

for (Thread thread: threads) {

try {

thread.join();

} catch (InterruptedException e) {

//e.printStackTrace();

}

}

}

// main() ends here

**Input format:**

A text file (filename is passed as a commandline input) with one transaction per line. Each transaction starts with the character d or w representing deposit or withdrawal respectively:

d1000

w500

d100

w600

**Output format:**

One integer value representing the balance after each transaction (deposit/withdrawal) on a new line.

1000

500

600

0

Explanation: Take the output to be the case when

1. Deposit 1000
2. Withdraw 500
3. Deposit 100
4. Withdraw 600

**Exception handling**

The exception messages, if any, should be clear and relevant and from the glossary of exceptions provided at the end of this document.

In case of any transaction that will lead the balance to go below 0, throw an **ArithmeticException** with the message mentioned in the glossary. And that transaction should be ignored.

**Grading guidelines**

A few of the factors that will be considered for grading:

* The number of threads employed to solve the program
* The exception handling
* The execution speed
* Number of test cases (on and off the platform) passed

**Question 3:**

Write a Java program that reads one integer, n, from the console. The program should give the output of the sum of all the prime numbers between 1 and n (both inclusive); and the sum of all the multiples of 3 or 5 or 7 between 1 and n (both inclusive). Your program should employ multiple threads for the computation. The program should print the output to the console according to the format below.

In the program, you have to wait for all the threads to complete execution and only then exit the program. You will also need to add all the relevant exception-handling logic based on your implementation, choosing from the glossary of exceptions at the end of this document.

Please note that how many threads and how to use them within the program is completely up to your implementation.

**Constraints:**

* 10 <= n <= 500000000

**NOTE**

* Use relevant packages as required

**Input format (on the console):**

100

**Output format**

A single line containing two space-separated integers, which is the sum of the prime numbers until 100 and the sum of the multiples of 3 or 5 or 7.

1060 2838

**Exception handling**

The exception message should be clear and relevant, choosing from the glossary of exceptions at the end of this document

**Grading guidelines**

A few of the factors that will be considered for grading:

* The number of threads employed to solve the program
* The exception handling
* The execution speed
* Number of test cases (on and off the platform) passed

**Question 4:**

Write a Java program that sorts an array of integers using multiple threads.

The program divides the sorting task among multiple threads and performs parallel sorting to improve the performance of sorting large arrays. You need to use merge sort as the sorting algorithm, and you will have to implement it from scratch. The input will be read from a file, the filename is supplied as a commandline argument. The file contains space separated integers.

In the program, you have to wait for all the threads to complete execution and then only exit the program. You will also need to add all the relevant exception-handling logic based on your implementation.

Please note that how many threads and how to use them within the program is completely up to your implementation.

**Constraints:**

* 1 <= number of integers in the file <= 1000
* 0 <= Array[i] <= 1000000

**NOTE**

* Use relevant packages as required

**Input format**

Space separated integers are the contents of the array

1 2 3 4 5 6 7 8 9 0

**Output format**

A single line containing n space-separated integers in ascending order.

0 1 2 3 4 5 6 7 8 9

**Exception handling**

The exception message should be clear and relevant, choosing from the glossary of exceptions at the end of this document

**Grading guidelines**

A few of the factors that will be considered for grading:

* The number of threads employed to solve the program
* The exception handling
* The execution speed
* Number of test cases (on and off the platform) passed

## Exception Glossary

The output format of exception handling should be as follows

*(Example with IOException):*

java.io.IOException: Error in IO

*(Example with IllegalArgumentException):*

java.lang.IllegalArgumentException: Wrong argument provided

* ArithmeticException: Illegal mathematical operation
* ArrayIndexOutOfBoundsException: Invalid array index
* ClassNotFoundException: No class with that name
* FileNotFoundException: Error opening the specified file
* IOException: Error in IO
* InterruptedException: Thread interrupted
* NoSuchFieldException: Unknown variable
* NoSuchMethodException: Unknown method
* NullPointerException: Null object
* NumberFormatException: Error in string to number conversion
* RuntimeException: Runtime error
* StringIndexOutOfBoundsException: Invalid string index
* IllegalArgumentException : Wrong argument provided