NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practice Paper 1 for Practical Assessment
CS1010J — Programming Methodology

Time Allowed: 1.5 hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper consists of 2 exercises on 5 pages.
2. This is an **OPEN BOOK** assessment. You may bring in any printed material and a calculator, but **not** other electronic devices such as laptops, iPads or thumb-drives.
3. In line with the university rules, any form of communication with other students, or the use of unauthorised materials is considered cheating and you are liable to the disciplinary action.
4. The first 10 minutes of the assessment is reserved for you to read questions and design algorithms. You are not allowed to type your programs in this period of time.
5. You are to use **DrJava** or **VS Code** to write your programs. No other IDEs are allowed.
6. You may assume that all input data are valid; hence no input validation is needed.
7. Once finish, you may submit your programs to **CodeCrunch** for automatic grading.
8. You are **NOT** allowed to use any Java language syntax not covered in weeks 1-9 (e.g., string functions or recursion).

ALL THE BEST!

Exercise 1: Set Operations [50 marks]

Mathematically, a set is a collection of distinct objects. Given two sets A and B,

- Their intersection, denoted by $A \cap B$, is a set of objects that exist in both A and B. For example, if A = {4, 5, 14}, B = {1, 2, 3, 5, 9}, $A \cap B$ = {5}.
- The complement of B in A, denoted by $A - B$, is a set of objects that exist in A but not in B. For instance, if A = {4, 5, 14}, B = {1, 2, 3, 5, 9}, $A - B$ = {4, 14}.

Write a program **SetOp.java** that reads two sets of integers (**`setA`** and **`setB`**) and complete the following two methods.

1) **`int countIntersection(int[] setA, int[] setB)`** that returns the number of integers in the intersection of **`setA`** and **`setB`**.
2) **`int[] returnComplements(int[] setA, int[] setB)`** that returns a new set that is the complement of **`setB`** in **`setA`**. (i.e. **`setA-setB`**). You may assume that **`setA`** and **`setB`** each contains at least two distinct integers already sorted in ascending order. The new set returned by the method should also be sorted in ascending order.

Write in the skeleton program that has been loaded into your working directory. You must **NOT** change the method headers given in the skeleton program. You are **NOT** allowed to change the **`main`** method as well. You may write additional methods as necessary.

Two sample runs of the program are shown below with the user's input shown in **bold**.

Sample run #1

```
Enter the number of integers in set A: 3
Enter set A: 4 5 14
Enter the number of integers in set B: 5
Enter set B: 1 2 3 5 9
A ^ B contains 1 integer(s)
A - B = [4, 14]
```

Sample run #2

```
Enter the number of integers in set A: 3
Enter set A: 5 6 7
Enter the number of integers in set B: 3
Enter set B: 5 6 7
A ^ B contains 3 integer(s)
A - B = []
```

Exercise 2: City Skyline [50 marks]

[Source: Wikipedia] A skyline is best described as the overall or partial view of a silhouette of a city's tall buildings and structures consisting of many skyscrapers in front of the sky in the background. It can also be described as the artificial horizon that a city's overall structure creates.

Your task in this exercise is to print out the skyline of a city area, given the heights of buildings. The city area can be modeled as a *m x n* map, where the top of the map is north and the bottom of the map is south. Each slot of the map is a positive integer stating the height of the building in that slot (in meters). If there is no building at a place, the value of that slot will be 0.

The following is an example map and its corresponding skyline when you look from south to north. Each hash ('#') in the skyline represents a height of 10 meters.

north

10	10	10	10	10	10
10	40	40	10	20	20
70	10	10	80	20	20
0	0	0	0	0	0
0	0	0	10	10	0

south

```
     #
#    #
#    #
#    #
####
####
######
######
```

skyline

Note that when you look from south to north at sea level (0 meter), taller buildings would "override" low buildings. So the skyline essentially consists of the tallest building in each column.

Write a program **Skyline.java** that reads the map of a city area and prints out its skyline. In the skeleton program, complete the following four methods:

1) **`int[][] readMap()`** that reads a map into a two-dimensional array and returns it. You may assume the following: a given map contains at least 2 rows and 2 columns, there is at least one building in the given map, and the height of every building is multiple of 10 meters.

2) **`int[] checkHeights(int[][] map)`** that returns a one-dimensional array that stores the maximum height of each column of the given **`map`**. For example, such an array for the above example map will be {70, 40, 40, 80, 20, 20}.

3) **`int findMaximumHeight(int[] heights)`** that takes in an array that stores the maximum height of each column in a map, and returns the maximum height among all the columns. For example, given the **`heights`** array {70, 40, 40, 80, 20, 20}, maximum height will be 80.

4) **`void printSkyline(int[] heights)`** prints the skyline of a map. The parameter **`heights`** array stores the maximum height of each column of a map.

You must **NOT** change the method headers given in the skeleton program. You are **NOT** allowed to change the **`main`** method as well. You may write additional methods as necessary.

Two sample runs of the program are given below with the user's input shown in **bold**. For your convenience of testing, test inputs are also appended to the back of the skeleton program.

Sample run #1

```
Enter the size of map: 5 6
Enter 5*6 map:
10 10 10 10 10 10
10 40 40 10 20 20
70 10 10 80 20 20
 0  0  0  0  0  0
 0  0  0  0  0  0
   #
#  #
#  #
#  #
####
####
######
######
```

Tip: a skyline actually consists of hashes ('#') and spaces (' '). If you replace spaces (' ') with dashes ('-') for easy identification, the above skyline would be:

```
---#--
#--#--
#--#--
#--#--
####--
####--
######
######
```

Sample run #2

```
Enter the size of map: 10 10
Enter 10*10 map:
 0   0   0   0   0   0   0   0   0   0
90  90  70  70   0  40  40   0  80  80
90  90  70  70   0  40  40   0  80  80
90  90  70  70   0  40  40   0 130 130
90  90  70  70   0  40  40   0 130 130
 0   0   0   0   0   0   0   0   0   0
20  20   0 120 120  20  20  20  80  80
20  20   0 120 120  20  20  20  80  80
20  20   0 120 120   0   0   0   0   0
20  20   0 120 120   0   0   0   0   0
         ##
   ##    ##
   ##    ##
   ##    ##
## ##    ##
## ##    ##
#####    ##
#####    ##
#####    ##
####### ##
####### ##
##########
##########
```

Example Rough Marking Scheme for an Exercise of 50 Marks

1. Style: 10 marks
   - Write program description and student number in the program header.
   - Write a short and meaningful description for every method (except main).
   - Apply consistent indentation and good naming of variables
2. Correctness and design: 40 marks
   - (**In actual PA**) Graders will manually go through your program and award method by method.
3. Additional penalties:
   - Program is not compilable: deduct 5 marks.
   - Break coding restrictions: deduct 5 - 20 marks, depending on severity.

=== END OF PAPER ===