Programming Hand in 1

Brooks, Mortensen & Odderskov

For this and the remaining three handins you must work in your semester project group.

For each of the below exercises there is an associated template. You can find all templates in the zip file 'dmaProg1.zip'. Create a new project in IntelliJ for all your DMA programming handins. It is up to you how you want to structure your hand ins but it may be a good idea to have a module for each of the four hand ins. Unpack the zip file to the newly created project folder.

Do not change any of the existing code, e.g. do not rename methods, etc., although you may need to change the reference to the package (first line in templates). You will only need to add the body of your methods in the relevant template. For each of the programs, replace the comment "// TODO implement method" with your code. Note, you do not need to create methods for user inputs. All you really need to do is to write the logic of the methods, i.e. the algorithms.

Each exercise is accompanied by a test class. You can use this to test whether your method works correctly. E.g. in order to test findMax, right-click on testFindMax and press 'Run 'testFindMax.main()".

In the last three exercises, it is possible to obtain an extra point if you find a fast solution. What defines a fast solution is stated in each exercise. The test class will let you know whether you found a correct solution - and if you did whether this solution is sufficiently fast for the extra point.

You upload your hand in as a zip-file where you simply zip the handin1 module with all your code/files.

If you get stuck doing any of the exercises, it is recommended seeking out the student instructor at the study café (on Discord).

Program 1: `findMax`

In this problem, the input to your program are two integers, and your program must output the largest of the two integers. For instance, if the input is (97, 66) your program should output 97.

Concretely, you must implement a public method named `findMax` that takes two integers as argument and returns an `int`. Use the template **max.java**.

Scoring:

- 1 point for correct algorithm

Program 2: `findMax2`

In this problem, the input to your program is an array of N positive integers, and your program must output the largest integer in the input. For instance, if the input array contains these ten elements

```
<< 31, 41, 57, 26, 53, 59, 97, 93, 23, 84 >>
```

then your program should return 97.

Concretely, you must implement a public method named findMax2 that takes an `int[]` as argument and returns an `int`. Use the template **max2.java**.

Scoring:

- 1 point for correct algorithm

Program 3: `decToBinary`

In this problem, the input to your program is a positive integer, and your program should output the binary representation of the integer as a `string`. For instance, if the input is 17, then your program should return `10001`.

Concretely, you must implement a public method named `convertDec2Bin` that takes an int as argument and returns a `string`. Use the template `dec2Bin.java`.

Please note that the template already contains code that prompts for user input. Thus, if you have programmed the method correctly, you can enter an integer and its binary representation will be displayed.

Scoring:

- 1 point for correct algorithm

You are NOT allowed to simply use `Integer.toBinaryString()`.

Program 4: `findClosest`

In this problem, the input to your program is an arraylist of N integers, and your program must output the smallest difference between two numbers in the input. For instance, if the input array contains these ten elements

```
<< 31, -41, 57, 26, -53, 59, 97, -93, -23, 84 >>
```

then your program should return the difference 2, i.e. $59 - 57 = 2$.

Concretely, you must implement a public method named `findClosest` that takes an `ArrayList<Integer> input` as argument and returns an `int`. Use the template **closest.java**.

Scoring:

- 1 point for correct algorithm
- 1 extra point for correct and fast algorithm

An $\mathcal{O}(N \log N)$ solution is fast enough to get the extra point.

Hint:
You are allowed to use `Collections.sort();` to sort the input list (or any list you may create as part of your solution). That is, after running `Collections.sort(input);` in `findClosest()`, the input list is sorted into ascending order. This might be useful to you when solving the exercise.

Remember to have `import java.util.*;` in the beginning of your solution in order to be able to use `Collections.sort()` (which is defined in `java.util`). It is imported as default in the template.

Program 5: `closestBall`

This problem is based on a similar problem from Aarhus University.

Dodgeball Madness is a game in which two teams of N players play against each other using M balls. It is your team's turn to play, and you must quickly determine which of the N players on your team is closest to one of the M balls you can throw, since the faster your team gets the first throw in, the more likely it is that you will hit one of the players on the other team!

Since all players and all balls are currently on a straight line, the distance between a player at position x and a ball at position y is simply $|x - y|$, that is, the absolute value of the difference between the two numbers x and y.

Your task is to write a program that takes the positions of the N players and the M balls and outputs the shortest distance between a player and a ball.

For instance, if the players are located at positions

```
<< 95, 66, 82, 63, 17 >>
```

and the balls at positions

```
<< 75, 38, 25, 77 >>
```

then your program should output 5, i.e. $|82 - 77| = 5$.

Concretely, you must implement a public method named `computeClosest` that takes an `ArrayList<Integer> players` and an `ArrayList<Integer> balls` as arguments and returns an `int`. Use the template **closestBall.java**.

Scoring:

- 1 point for correct algorithm
- 1 extra point for correct and fast algorithm

An $\mathcal{O}(N \log N + M \log M)$ solution is fast enough to get the extra point.

Hint: For the fast solution, you're allowed to use `Collections.sort()` just like in the `findClosest` program.

Program 6: `maxSubSum`

This problem is based on Bentley Chp. 8.

In this problem, the input to your program is an array of N integers, and your program must output the maximum sum found in any *contiguous* subarray of the input. For instance, if the input array contains these ten elements

```
<<31, -41, 57, 26, -53, 59, 97, -93, -23, 84>>
```

then your program should return the sum $186 = 57 + 26 + (-53) + 59 + 97$. (For convenience, the maximum sum is 0 if all the integers are negative.)

Concretely, you must implement a public method named `findMaxSubSum` that takes an `ArrayList<Integer> input` as argument and returns an `int`. Use the template **maxSubSum.java**.

Scoring:

- 1 point for correct algorithm
- 1 extra point for correct and fast algorithm

A linear solution is fast enough to get the extra point.