TP3 on Java Programming

1 Introduction

The objective of the third lab is to practice implementation of classes and objects, the implementation of arrays.

The requirement of your report:

- Submit everything in a .zip file named: `JAVA_TP3_prenom_nom.zip`;
- Include a .pdf file to answer open questions;
- Name your project folder `TP3_Prenom_Nom`, and inside this project folder, include the following:
  - Write comments in your code specifying which question you are answering;
  - Inside the `src` folder, include:
    1. A package called `arrayMuni`, containing the following classes [45%]:
       1) `ArrayMuni.java`
    2. A package called `contact` [45%], you can try your own structure design in order to finish it.
    3. A package called `CM3`, containing the following classes [10%] [a]:
       1) `ArraySum.java` –Exercise 2;
       2) `ArrayMaxMin` – Exercise 3;
       3) `RangeScore.java` – Exercise 4, and 5;

Deadline: The deadline of group1 is: 18:00:00, 04/11/2024; the deadline of group2 is: 18:00:00, 18/11/2024.

[a] These exercises are included in CM3 at `https://www.qiongliu.info/assets/teaching/java/Java_cm3.pdf`

2 Arrays

Create a class `ArrayMuni`, it has the constructor that creates 1-dimension array with length n.

1. It should have methods:
   - `geti()`, which returns i-th element in the array.
   - `seti()`, which sets i-th element in the array.
   - With error handling, by using `ArrayIndexOutOfBoundsException`.
   - `toString()`, which returns all value of elements in the array.
   - `subArray()`: it returns a continuous subarray[1] with the maximum sum (the subarray contains at least one element).[2]
2. In the `TestArray` file, create an instance from class `ArrayMuni`, and check the output of each method.
   - `arr1 = {-3,5,-3,6,-2,4,11,-5,4}`.

[1] Examples of a continuous subarray A_1 from $A = \{a_1, a_2, a_3, a_4\}$, can be $A_1 = \{a_1\}$ or $A_1 = \{a_2, a_3\}$, or $A_1 = \{a_1, a_2, a_3, a_4\}$

[2] This method is crucial in scenarios like financial analyses where you might want to find the best buying and selling points in a stock price array.

- Display the message by calling method Geti(i), where i is the position of element in `MyArr1`, $i = \{0, 2, 10\}$.
- `arr2 =` {`1,1,1,1,1,1,1,1,1,1`}
- `arr3 =` {`5,4,-1,7,8`}
- `arr4` has length n=20, each element is an integer, which is randomly generated with value between -50 and 50.

Questions:

1. Explain what how you design the algorithm to find optimal subarray with maximum sum.
2. How many combinations of subarrays did you find for a 1-dimension array of length n?

Several Tips:

- **Tip 1:** You can create constructors in various ways to make your array list more flexible:

  1) Fix the length of your array, i.e.

```java
public ArrayMuniTest(int size) {
    this.arr = new int[size];
}
```

  2) Do initialization with a given array, i.e.,

```java
public ArrayMuniTest(int[] initialArray) {
    this.arr = initialArray;
}
```

- **Tip 2:** Here are two methods to find the subarray with the maximum sum, you can chose one of them:

  1. *Using a Double Loop*: You can iterate through all possible subarrays and calculate their sums. By comparing these sums, you can identify the subarray with the highest sum. This approach checks every possible subarray, ensuring that you find the one with the maximum sum.
  2. *Using a Sliding Window Technique*: This method involves five local variables:
     - `startPoint` and `endPoint` to keep track of the current subarray bounds,
     - `currentSum` to store the sum of the current subarray,
     - `maxSum` to keep track of the maximum sum found so far,
     - and `tempStart` to mark the beginning of a new subarray when the current sum becomes negative.
  3. – Initializes the variable `maxSum` to the smallest possible integer value.

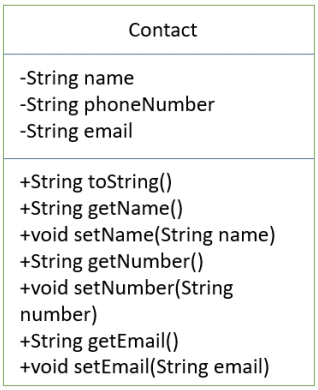
```java
import java.util.Arrays;
import java.util.Random;

int maxSum=Integer.MIN_VALUE;
```

3 Contact

We would like to create a simple Phonebook system. First it contains the ability of creating new contact, with name, phone number and email. Then, the newly created contact is added to the Phonebook. Phonebook should have a maximum capacity of m = 100 contacts.

Tips: I'll provide a design outline, of course you can use your own design structure.

- Contact.java: This class defines the structure of a contact, including fields for name, phone number, and email, along with appropriate getters, setters, and a method to format the contact as a string.
- Phonebook.java: This class manages the Phonebook operations such as adding new contacts and ensuring that the Phonebook does not exceed its capacity.
- PhonebookTest.java: This class is used to test the functionality of the Phonebook by adding contacts and printing them to verify correct behavior.



Questions:

1. Write down the description you read from the class diagram, as well as the relationships between classes.
2. Create new `Contact` using different ways, either defining it directly in the test file, or by `Scanner`.
3. Use `(if {}else {throw})` for Exceptions when the new contact already exists.
4. Create the test file, and add following new contacts.
   - Jean, 0612121212, jean@gmail.com
   - Paul, 0618181818, paul@yahoo.com
   - Luc, 0646985221, luc@msn.com
   - Jean, 0612121212, jean@gmail.com