Introduction

Hello folks! In this assignment, you will tackle 3 main challenges: an object and class problem, a single-dimensional array problem, and a multidimensional array problem. You are also required to come up with your own tests to make sure your code has the correct behavior as described in this writeup. You should not change/delete any given starter code (except some return -1; and comments). You should complete this assignment individually.

Part 1: Get started

1. Make sure there is no problem with your Java coding environment. If there is any, review assignment 1 or come to the office hour before you start this assignment.
2. Review lecture 5 - 8 and/or read chapter 7 - 9.4 of Introduction to Java Programming and Data Structures.
3. Download the starter code.
   If you work with your local machine, download it from Piazza -> Resources -> Homework -> Assignment3.java and MyIntArray.java.
   Start terminal/command line and navigate to the directory that contains Assignment3.java and MyIntArray.java.
   If you decide to use Linux Cloud, use the following command to copy the starter code to a new directory called HW3:

   ```bash
   $ cd ~
   $ mkdir HW3
   $ cp -a ../public/assignments/PA3 ./HW3
   ```

4. You should compile MyIntArray.java first and then compile Assignment3.java. You will run Assignment3 for this assignment.
Part 2: MyIntArray.java

Note: In this assignment we will use member variables, fields, and attributes interchangeably, as they all refer to the same thing.

First, you need to implement the object class called MyIntArray storing integer values. Your class object should have an array under the hood to store all the values. Before you write any code, make sure you understand how java class works, and how getter/setter works.

The MyIntArray object should contain the following member variables/fields/attributes:

1. A int[] field storing integer elements called intArray.
2. An int field called length

The MyIntArray object should contain the following member methods:

1. public MyIntArray(int[] newArray)
   This is the constructor that initializes your intArray with the given parameter. It will also sort the array. It also initializes length to be the length of your array.

2. public int[] getIntArray()
   This is the getter method that returns the intArray field.

3. public void setIntArray(int[] newArray)
   This is the setter method that sets the intArray field. You should not sort the array in the setter.

4. public int getLength()
   This is the getter method that returns the length field.

You’re expected to use this object in the following parts. Since it contains several attributes, everytime you want to update the attributes/retrieve the attributes values, you are required to use the setter/getter. You will lose points if you access your MyIntArray object’s attributes without setters/getters for the whole assignment. Please note that the intArray attribute requires you to sort the array. Don’t panic. We are allowing you to use the sort method from java.util.Arrays package. The usage of that sort method is:
Arrays.sort(intArray)

In this assignment, you should only use the above sort method **once in the constructor** of this class. You will lose points if there are other usages of this method in your code.

**Part 3: Implement three methods**

In class Assignment3 of the starter code, five methods are already declared: `findInsertIndex`, `insertInMyIntArray`, `escapeFromMaze`, `unitTests` and `main`. Your task in this part is to implement the first four methods. But before you implement your methods, make sure to take a look at the constants on the top. All error messages are declared as constant String in Assignment3 class for you. They all start with the keywords “private final static”. **Use them to ensure that your code always gives correct output, and you don’t need to worry about misspelling. Hardcoding will not receive any credit.**

1. **findInsertIndex:**
   This method takes a **MyIntArray** object you created in part 2, and an integer **insertingInteger** that you want to insert into your **myIntArray**. The method should return an integer as the insert position. Remember in part 2, **myIntArray** contains a sorted integer array. We want to keep **myIntArray** always sorted, so the returned insert position integer should be the right position of insertion so that **myIntArray** is still sorted if we insert the **insertingInteger** to the returned position. Also, you need to find the right insert position.
   Sample:
   ```java
   myIntArray.getIntArray() returns-> [1,2,3,4,10]
   findInsertIndex(myIntArray, 4) returns-> 3 or 4 (both are correct)
   findInsertIndex(myIntArray, -1) returns-> 0
   findInsertIndex(myIntArray, 8) returns-> 4
   findInsertIndex(myIntArray, 100000) returns-> 5
   ```

2. **insertInMyIntArray:**
   This method takes three inputs: an **MyIntArray** object, an integer **insertingPosition**, and an integer **insertingInteger**. In this method, you should insert **insertingInteger** into the **insertingPosition** index of the integer array of the **MyIntArray** object. Then you need to update your **MyIntArray** object with your **setter**.
Sample:

```java
myIntArray.getIntArray() returns-> [1,2,3,4,10]
insertInMyIntArray(myIntArray, 4, 4)
myIntArray.getIntArray() returns-> [1,2,3,4,4,10]
```

3. escapeFromMaze:

You’re given a **maze** in the form of a 2d-array which is a square of size 6x6. Imagine you are at the **top left corner** of this maze. You want to find a path to go to the **bottom right corner** of the maze, which is the exit. You can **only** go right OR go down. All the 1’s in this 2d-array mean blockers, and all 0’s means empty spaces you can move to.

The given i and j are indexes you can use (i denotes the row number, j denotes the column number).

The **endIndex** represents the maximum value your i and j can take in this maze (which is 5 of course).

You are also given an int[][] **path** that records each position you visit. For example, at the beginning you are at (i, j) = (0, 0), so your **path[0]** should be an array containing [0,0]. If you go right, then the new position should be (i, j) = (0, 1), so your **path[1]** should be an array containing [0, 1].

Your job is to fill out the **path** containing 11 index pairs (think about why it has 11 index pairs). You’re required to record the position of each step you take on your way out and you need to return that **path**. The **path** is also a 2d-array that contains all index pairs of the right path and the exit pair is already filled out for you. You might find the **pathCounter** helpful on your way out.

**Important**: You need to write a program to find a way out for you. Keep in mind that the example **private final static int[][] MAZE** is only one example!!! Your program should also work if the maze is slightly changed. You can assume that the maze is guaranteed to have at least one path. If it has two or more paths, any solution is accepted. You will receive 0 credit if you hard code your answer by just giving the fixed path from the maze showing below.
Part 4: Test the correctness of three methods

Testing is a very important part in programming. In this course, we will get you familiar with unit test. For this and all future assignments, you will be asked to create your own tests to check whether your code works as expected. In this part, you need to implement your own test cases in the method called `unitTests`.

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. To get full credit, for each method, create at least two test cases that cover different situations. (Hint: make sure your methods return correct messages when the input is invalid). We suggest making some print messages in each of your test cases so that you will know which test case is failing. The `unitTests` method should return true only when all the test cases are passed, otherwise return false.

Part 5: Complete main

After completing part 4, compile and run Assignment3. You should see the message “All unit tests passed”. If not, it’s very likely that you have bugs in your code. Read previous instructions carefully while inspecting your code to fix bugs (we call process “debug”).

The main method is the method that will be called when running program Assignment 3. In the starter code, the main method looks as below:
The code to run `unitTests` and print prompt according to the testing result is already given to you. Don’t change any code above the comment “Start the user-machine interaction below”. Below that comment, your last task is to implement a ask-answer interaction functionality via command line.

First, your program should print the following prompt:

```
Which problem do you want to solve today? Please Enter:
1 to Insert number into a sorted array
2 to Escape from maze
end to Stop running
```

and wait for the user to enter feedback. After the number is entered, it will read the input via Scanner. Then,

1. If the input is “1”, the program should print another prompt “Please enter an integer”. It will then read the user input number as an `insertingInteger`. It will create a `MyIntArray` object with `ARRAY_TO_INSERT` as its array. It will call the method `findInsertIndex` with `myIntArray` and `insertingInteger` to get the `insertPosition`. Then it will use the generated `myIntArray`, `insertPosition`, and `insertingInteger` to make the insertion. Finally the program will print out the inserted array attribute from your `MyIntArray` object. For 1d-array, the following method will help you convert the array to a string. This method is from `java.util.Arrays` package.

```
Arrays.toString(someArray)
```

2. If the input is “2”, the program should call the `escapeFromMaze` method with `MAZE` as the input 2d-array. Finally the program will print out the result. For 2d-array, the following
method will help you convert the array to a string. This method is from java.util.Arrays package.

```
Arrays.deepToString(someArray)
```

3. If the input is “end”, return from the main method, which will terminate the execution of the program.

4. If the input is invalid (e.g. other than 1, 2, or end), print error message “Invalid method. Available options are: 1, 2, and end”.

After performing one of the above, your program should print the prompt “Which method do you want to call?” again and repeat the whole process. The repetition should never stop until “end” is entered or Ctrl + C is pressed to forcibly stop the program.

Example: you should be able to reproduce this output with your program

```
(base) andyj@MacBook-Pro Assignment 3 % java Assignment3_solution
All unit tests passed.

Which problem do you want to solve today? Please Enter:
1 to Insert number into a sorted array
2 to Escape from maze
end to Stop running
1
Please enter an integer
4
[-1, 1, 2, 4, 6, 7, 32, 65, 1000]

Which problem do you want to solve today? Please Enter:
1 to Insert number into a sorted array
2 to Escape from maze
end to Stop running
2
[[0, 0], [0, 1], [1, 1], [1, 2], [1, 3], [2, 3], [3, 3], [3, 4], [4, 4], [5, 4], [5, 5]]

Which problem do you want to solve today? Please Enter:
1 to Insert number into a sorted array
2 to Escape from maze
end to Stop running
end
```
Part 5: Coding style

When coding in Java, there are several style rules that people usually follow to make the code clean and readable. In this course, you are asked to follow rules specified in link below:

https://cseweb.ucsd.edu/classes/fa20/cse8B-a/styleguide.html

Read the coding style guide carefully and refine your code for this and all future assignments.

Submission

Very important! Please follow the instructions below carefully and make the exact submission format. This is important since we will use scripts to grade so if you don't follow the same submission format you probably will receive a zero.

1. Go to Gradescope and click on PA3.
2. Click the DRAG & DROP section and directly select the required file (Assignment3.java and MyIntArray.java). Drag & drop is fine. Please make sure you don't submit a zip. Just the file solely. Make sure the name of the file is correct.
3. You can resubmit unlimited times before the due date. Your score will depend on your final submission, even if your former submissions have a higher score.
4. The autograder is for the use of the instructional team. You won't see the result of the autograder. As long as you uploaded your file you're good to go.