Due before: **11:59 PM on Wed Sept 14 2022**

CSE-271: Object-Oriented Programming

Homework #2: Battleship

Deadline: Wed Sept 14 2022 before 11:00 PM

Delayed (by no more than 24-hours) submissions earn only 80% credit

Maximum Points: 23

Key objectives of this project are:

- Gain experience of working with Exceptions
- Continue to build strength in fundamentals of problem solving
- Recap Java programming using an IDE like Eclipse
- Review and adhere to CSE department's Style guide
- Use Javadoc to document methods and their return values

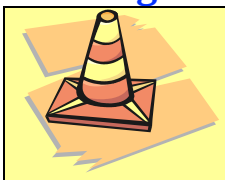
Submission Instructions

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure your program compiles successfully, without any warnings or style errors. Ensure you have documented the methods. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload just the PointProcessor.java source file onto Canvas via the CODE plug-in.

1. The 1 Java source file developed for this part of the homework.

General Note: Upload each file associated with homework (or lab exercises) individually to Canvas. Do not upload archive file formats such as zip/tar/gz/7zip/rar etc.

Grading Rubric:



The program submitted for this homework **must pass necessary base case test(s) in order to qualify for earning any score at all**. Programs that do not meet basic requirements or just skeleton code will be assigned zero score! Programs that do not compile, **have even 1 method longer than 25 lines**, or just some skeleton code will be assigned zero score.

- **NOTE:** Violating CSE programming style guidelines is an error! Your program should not have any style errors.
- **Delayed submission: Only 80% points:** Submission delayed **by no more than 24-hours** will be accepted for a partial credit of maximum 80% of the points.
- **Formatting & Documentation:** 4 **points** – Reserved for good Javadoc style documentation for each method you implement. If your Javadoc is low quality, then you lose points in this category.

Project Overview

In this project you will be completing a simplified version of the Battleship game – *i.e.*, the player has to guess the row and column where a battleship is hidden. The basic logic of the lame is given to you as a Java class called `BattleGrid`. This class generates a square grid of random size (between 8 and 20) and places the battleship at a random location in the grid. The location of the battleship is guessed by repeated calls via the `BattleGrid.guess()` method. The interesting part is that this method does not return values and instead, it throws exceptions to signify different outcomes. So, are expected to suitably handle exceptions to help a user play the word guessing game.

The `BattleGrid` class does not provide any user interactions. Accordingly, in this project you will be implementing the `BattleGame` class to provide necessary user interactions as described below. For this project, you have been provided with the following files:

Starter code

1. **`BattleGrid.java`**: This file contains the basic logic for the game. **You should not modify or submit this file**. Review the documentation in the source code, particularly for the `guess` methods and the different exceptions it throws.
2. **`BattleGame.java`**: This file contains very basic starter code. You will be implementing the necessary features in this class, starting with the `play` method. Of course, you may add as many helper methods and additional static instance variables, as you see fit. It is up to you how you choose to implement the required features.

Requirements for the game:

It would be best to first see the sample outputs below to understand some of the operations of the game. Here are **requirements** for implementing the `BattleGame.java` class:

Required features for this project (Startup operations & Input loop)

1. **Startup operations [4 points]**: Determine and the dimension of the battle grid by repeatedly calling the `guess` method with increasing column position, until the `guess` method throws the custom `BattleGrid.IllegalLocation` exception. Inform the user about the grid dimensions using the `BattleGame.GRID_SIZE_MSG` (see its Javadoc for details).

    i. **Default row & column values**: For improved user experience, the `BattleGame` method maintains a default guess by the player for the row and column of the battleship. This value is initialized to zeros and updated each time the user guesses a row or column value.

2. **Input loop [3 points]**: Next, repeatedly perform the following operations (of course, using helper methods as you see fit as no method should be longer than 25-lines of code) to let the player guess the location:
    - Prompt the user for input via `System.out.printf` using `INPUT_PROMPT`. (See constant defined in the starter code)

- **Inputs**: Read a line from the user (yes, using a `Scanner` and its `nextLine` method) and perform the following operations (using suitable helper methods that you add as you see fit) based on the input:
    i. If input is "`quit`" then the loop ends and `play` method must return.
    ii. If input is "`grid`"., then print the current play grid as discussed in the optional feature below. If this feature is not implemented, then simply ignore this input.
    iii. Otherwise assume the input is the player's guess and process it as discussed in the *Processing row or column inputs* feature below.

Optional feature: Processing row or column inputs [7 points]

The player may enter just row, just column, or both as input. The row and column values are specified in the form **`r`**`<num>` or `c<num>` respectively, where `<num>` is an integer. Examples: `r5` (row 5), `c7` (column 7), `r5 c7` (row 5 and column 7), or `c7 r5` (row 5 and column 7), etc.

Note the following aspects regarding these inputs and their format:

- Inputs will always be valid and in lower case letters.
- If row or column is not specified then the current default value for it is used (see Default row & column values).
- If both row and column are specified then they will be separated by 1 blank space in the input.
- It is easiest to read them as strings and then parse out the numeric value using suitable substring operation and the `Integer.parseInt` method (to convert string to `int`).
- The default row and column are suitably updated to reflect the current input by the user.

Calling the `BattleGrid.guess` method & handling exceptions:

Once the row or column inputs have been processed, the values should be used to call the `guess` method in the `BattleGrid` class and exceptions (if any) must be processed as follows:

- If no exceptions are thrown, that means the row and column values guessed by the player are correct. Print the `SUCCESS_MSG`. After this, the game is complete and the input-loop must end (and the program stops normally)
- If `BattleGrid.InvalidLocationException` is thrown, then just print the message associated with the exception.
- If `BattleGrid.InvalidRowException` is thrown, then print message that column is valid using the `COL_CORRECT_MSG` string followed by the message in the exception. See sample output
- If `BattleGrid.InvalidColException` is thrown, then print message that row is valid using the `ROW_CORRECT_MSG` string followed by the message in the exception. See sample output.
- If `BattleGrid.IllegalLocationException` is thrown, then just print the message associated with the exception.

Optional feature: Printing the battle grid [5 points]

If the user input is the string "grid", then the play grid must be printed as shown in sample outputs further below. Locations (*i.e.*, row and col) previously guessed by the user are marked with an '`X`' character while other locations are shown with a '`.`' (period). Note that to show previously guessed locations, it would be easiest to maintain an `ArrayList<String>` that contains the list of previously guessed locations stored as "`row col`" format. Of course, you need to add to this list after each guess by the player.

Testing and sample outputs:

Note that user inputs are show in green color:

Basic startup functionality (Test #1) [Required]

```
Welcome to exceptional Battleship game.
The battle grid size is 12 rows and columns.
Guess the row and column of the ship.
Enter input [r<num>, c<num>, grid, quit]: quit
```

Process inputs (Test #2):

```
Welcome to exceptional Battleship game.
The battle grid size is 12 rows and columns.
Guess the row and column of the ship.
Enter input [r<num>, c<num>, grid, quit]: r8
Invalid row. It must be lower
Invalid column. It must be higher
Enter input [r<num>, c<num>, grid, quit]: c10
Invalid row. It must be lower
Invalid column. It must be lower
Enter input [r<num>, c<num>, grid, quit]: quit
```

Process inputs (Test #3):

```
Welcome to exceptional Battleship game.
The battle grid size is 12 rows and columns.
Guess the row and column of the ship.
Enter input [r<num>, c<num>, grid, quit]: c10 r8
Invalid row. It must be lower
Invalid column. It must be lower
Enter input [r<num>, c<num>, grid, quit]: r7 c10
The row value of 7 is correct
Invalid column. It must be lower
Enter input [r<num>, c<num>, grid, quit]: c9
The row value of 7 is correct
Invalid column. It must be lower
Enter input [r<num>, c<num>, grid, quit]: c8
The row value of 7 is correct
Invalid column. It must be lower
Enter input [r<num>, c<num>, grid, quit]: c3
The row value of 7 is correct
Invalid column. It must be higher
Enter input [r<num>, c<num>, grid, quit]: r7 c5
You guessed the correct location of the battleship!
```

Grid inputs (Test #4):

```
Welcome to exceptional Battleship game.
The battle grid size is 12 rows and columns.
Guess the row and column of the ship.
Enter input [r<num>, c<num>, grid, quit]: c10 c1
Invalid row. It must be higher
Invalid column. It must be higher
Enter input [r<num>, c<num>, grid, quit]: r5 c4
Invalid row. It must be higher
```

```
Invalid column. It must be higher
Enter input [r<num>, c<num>, grid, quit]: c0 r0
Invalid row. It must be higher
Invalid column. It must be higher
Enter input [r<num>, c<num>, grid, quit]: r11 c11
Invalid row. It must be lower
Invalid column. It must be lower
Enter input [r<num>, c<num>, grid, quit]: r12 c12
The row and column must be in the range 0 to 12
Enter input [r<num>, c<num>, grid, quit]: r6 c6
Invalid row. It must be higher
Invalid column. It must be lower
Enter input [r<num>, c<num>, grid, quit]: grid
Battle grid (previous guess marked with X)
XX.........
...........
...........
...........
...........
....X......
......X.....
...........
...........
...........
...........
..........X
Enter input [r<num>, c<num>, grid, quit]: quit
```

Submission:

This homework assignment must be turned-in electronically via Canvas CODE plug-in. Ensure your program compiles successfully, without any warnings or style errors (you should not have methods longer than 25-lines of code). Ensure you have documented the methods. Ensure you have tested operations of your program as indicated. Once you have tested your implementation, upload just the `BattleGame.java` source file onto Canvas via the CODE plug-in.