Northeastern Illinois University

CS-207: Programming II
Research Lab: Battleship
Due: Sunday October 7 at 11:59 PM

Goal:
The goal of this research lab is to further understand the Java API while utilizing an object-oriented approach by recreating the classic game Battleship. As this is a "research" lab, you will also need to investigate and analyze the code that has been provided for you to be able to correctly use this code to complete the lab.

The Problem:
The game Battleship has its origin in history dating back to World War I, where the French created a game called *L'Attaque* that was played during World War I that resembles the game. Battleship would have its first commercial release in the United States with the game *Salvo,* published by the Starex company in 1931, which was an early version of Battleship that came on pre-printed pads of paper. However, the one version that is iconic with the game now involves the 1967 Milton Bradley version of Battleship, which was created by Ed Hutchins. This version did away with the pads of paper and instead came with plastic boards for the grids, where players could place miniature plastic ships and use pegs to indicate hits or misses.

Instructions:

- You should work in groups of 2-3 individuals. Groups of more than 3 are **not** permitted.
- Each group should submit ONE lab write-up. It is the responsibility of each group member to ensure that their name is on the write-up.
- The lab write-up should be typed! Type each question (and the question number) followed by your group's answer. **Convert your lab write-up to a .pdf.**
- You should use complete sentences and proper grammar in your write-up. Use spell-check! This counts as part of your grade.
- Do **not** copy/paste directly from your sources for your answers (this is called plagiarism). Instead, you should re-word the information in your own words.
- Submit the pdf and `.java` files (`BattleshipMain.java` *and* `Battleship.java`) to D2L by the specified due date.
- Each member of the group must turn in a **readable** digital copy of the peer evaluation to **PLTL Lab 1 Peer Evaluation** on D2L by the assigned due date and time. The peer assessment counts as a significant part of your grade and you will receive a **zero** for that portion of the research lab grade if you do not turn it in.

Part A: Getting Started

Download the provided file from D2L. Research is often a very ambiguous process and it can be difficult to figure out how to get started on a problem. However, this first lab is designed to help you ask and answer the questions that are important when trying to solve a challenging research problem.

Question #A.1
Since we are coding the version of Milton Bradley's Battleship, we want to look up how many ships there are in the game as well as the size of these ships. What are the names of the ships as well as the size of each ship?

Question #A.2
Open the `BattleshipMain.java` file and look at the import statements at the top of the file. In your own words, what is JavaFX?

Question #A.3
What are the instance variables in this class? The instance variables are all object types – specify whether each instance variable is part of the Java API or a custom-built class.

Note that there is a lot of code in this file, however we will attempt to understand some of this code as we begin to create the game. However, in order to make this code work, you will need to spell and capitalize the code you create **exactly as described** otherwise this file will not work properly for you!

Part B: Creating the Game Itself

Though the graphical parts of the game have already been created for you, the game does not function as the `Battleship` class does not exist yet. To complete this research lab, you will need to create the `Battleship` class.

Question #B.1
Before we begin coding, we must consider what values we are using to represent a ship, a ship being hit and a shot missing a ship. Looking at the `startingGame()` method in the `BattleshipMain` file, we can see a nested loop that is used to update either the player's or the computer's board. Inside the nested loop, we see a Rectangle object being created and a color fill being set that is dependent on the number located at the row and column of the 2D array used. What values and color fills are used to represent a ship, a hit ship and a hit missing a ship? **Hint:** The light gray color fill represents the empty spaces of the board, corresponding with the value `0`.

Coding Rules

Create a new `java` file named Battleship that will contain the following code:

1. Properly encapsulated instance variables that represent three rectangular grids: one for the player's board, one for the computer's board (which will contain the location of the computer's ships), and another for the computer's board (which will be displayed on the screen). These three instance variables will also have getter methods. You will need to determine what the instance variable type and names of the getter methods should be by looking at the BattleshipMain class.

2. Two static instance variable integers to keep track of how many times the player has hit a ship and how many times the computer has hit a ship. You can chose the names of these variables.

3. A constructor that initializes the three rectangular grids. Look at the BattleshipMain class to see what the parameter types should be.

4. A method named `randomizingBoard()` that takes a rectangular grid as a parameter and does not return anything. This method will randomly set the ships on both the player's board and computer's hidden board by using the size of each ship. For the method to randomize these boards, the method must first decide randomly between a vertical or horizontal orientation for the ship, then randomly choose coordinates to place that ship on the board, taking into consideration the size of the ship and whether the ship is to be placed vertically or horizontally (i.e. will it fit?). The method should then check if the coordinates are already occupied by another ship. If the coordinates are occupied by another ship, the code will continue to run until that ship is able to be placed on the board. Call this method in the constructor created earlier, using the player board and computer hidden board instance variables as parameters. **Hint:** You can test to see if this method does its job by creating a test java file, calling the getter method for either board and printing out the grid using a nested for-loop.

5. A method named `updateComputerBoards()` that will update both of the computer boards after the user has input valid x and y coordinates (user input is done for you in the `BattleshipMain` class). This method should check whether the user input coordinates have hit or missed a ship on the computer's hidden board, change the value at that coordinate on both boards according to the result of this check, and increase the player counter by one if they have hit a computer ship. Look at the `BattleshipMain` class to see what parameters this method takes and whether the method returns anything.

6. A method named `computerTurn()` that does not return anything and does not take any parameters. This method should randomly select coordinates for the computer and check to see if player has a ship at those coordinates and change the value of the player's grid to correspond to whether a ship has been hit or missed. Increase the computer counter by one if the computer has hit a user's ship. If the random

coordinates have been previously produced (i.e. there is already a hit or miss on the player's board for that location), new random coordinates should be generated again until it targets an open location on the player's board.

7. Two methods that check whether the player has won, or the computer has won. Look at the `BattleshipMain.java` class to see what to name these methods, if they take any parameters, and if they return anything.

8. **Note:** You can create additional private helper methods to organize the code in the above methods. However, the above methods **must** exist.

Part C: Explaining Some of the Intricate Code

If you have correctly created the Battleship class, when you run the `BattleshipMain` class, you will be presented with a separate window that will display your grid on top of the screen and the computer grid on the bottom, with two text boxes on the right side of the screen with a button to send the coordinates you input.

Let us do some research as to why the `BattleshipMain` code is able to display our code in a new window!

Question #C.1
The whole graphical aspect of Battleship is completely dependent on the `GridPane` instance variable in the `BattleshipMain` class. In your own words, what is a `GridPane` object and where does it exist in the JavaFX package?

Question #C.2
The `startingGame()` method in the `BattleshipMain` class handles adding the information onto the `GridPane` object. There are two `GridPane` method calls chained together that enable the boards to be displayed when the class is run. What are these two `GridPane` methods and how do they work (Hint: Java docs!)?

Question #C.3
If you have continued playing the game, you will notice that when you input coordinates that have already been used or you have entered an incorrect x or y coordinate, a message for each specific error will appear along with a message to pick again that disappears after a while. There is a method in the `BattleshipMain` class that handles the appearance and disappearance of the text. What is the name of this method and the name of the object that is returned? In what JavaFX package does this object reside in?

Part D: Final Summary

Whenever you complete a project, it is important to assess what you think went well and what you need to improve on.

Question #D.1
What was the most challenging part of this research lab for your group?

Question #D.2
What did your group learn/find the most useful by doing this research lab?

Question #D.3
What was the most fun aspect of doing this research lab?