CPSC 1060: Introduction to Programming in Java
School of Computing, Clemson University
Programming Project # 1
Should be due: Monday, March 15, 10:00pm
Due: Monday, March 22, 10:00pm

This project contains two programs. The first one asks you to implement a series of methods. The second one asks you to implement a larger program, the game Battleship. As a reminder:

Projects are your own work and may not be copied in full or in part from any source.

Program 1: Tricks.java (15 points)

Write a Java program Tricks.java that accepts integers from the user and, depending on the value of each integer, performs the functionality described. The user can continue to try out the functionalities until the program is terminated.

0	Terminate the program.
1	(2 points) Ask the user for a number n; print all **even** numbers that are less than n.
2	(3 points) Ask the user for a letter grade A, B, C, D, or F. Display its corresponding numeric value 4, 3, 2, 1, or 0.
3	(5 points) Pick a random lowercase letter in the alphabet (using the random method) and ask the user to guess the letter. It will tell the user if the random letter is earlier or later in the alphabet. Count the number of trials required by the user to guess the number and print it out (see Figure 1 for example output).

Each functionality needs to be implemented as a method. Invoke these methods in the main method of the program.

```
Please guess which letter of the alphabet I have chosen.
m
Your guess is 'm'.
The correct letter is earlier in the alphabet.
g
Your guess is 'g'.
The correct letter is later in the alphabet.
j
Your guess is 'j'.
The correct letter is earlier in the alphabet.
h
Your guess is 'h'.
The correct letter is later in the alphabet.
i
Well done! The correct letter is 'i'. Finding it took 5 attempts.
```

Figure 1: Example output from the guessing functionality

Program 2: Battleship.java (30 points)

Write a program **Battleship.java** that allows you to play a single player version of the pencil and paper game "Battleship". The rules can be found here:

`http://en.wikipedia.org/wiki/Battleship_(game)`

and here: `http://www.papg.com/show?1TMC`

Create a 10x10 grid (map, board, battle area) with 7 ships of sizes 5, 4, 3, 2, 2, 1, and 1. The ships should not touch each other (be on adjacent grid cells). The user can not see the ships. The user chooses a row and a column. The computer will then determine if the user missed or if a ship was hit. If a ship was hit, the computer will notify the user if the ship has been sunk. The game is finished when all ships have been sunk. See example output for one round below.

```
Choose a row: G
Choose a column: 4

You missed.

   1 2 3 4 5 6 7 8 9 0
----------------------
A| . . . . . . . . . .
B| . . . . ~ . . . . .
C| . ~ . . . . . ~ . .
D| . . . ~ . . . . . .
E| . . . . . . ~ ~ . X
F| . . . . . . . . . X
G| . . . ~ . . . . . X
H| X X . . . . . . . .
I| . . . . . ~ . . . .
J| . . . . . . . . . .


Choose a row:
```

There are of many ways to implement such a game. Follow the steps below for your implementation:

- Save your current state of the game as a 2-dimensional array (matrix). Each element in the matrix can be either a hidden ship (S), an unexplored ocean (.), a hit (X), or a miss (∼).
- The ship's positions can be hard-coded (meaning they are the same for each game).
- Bonus (4 points): Create new valid ship positions randomly.

- Write a method printGrid that prints the game grid as seen in the example output. The method printGrid has two parameters, the first one is the 2-dimensional array, the second one, called displayShips, is a boolean that allows to print your grid with the ships. In the example output shown, displayShips is false. In that case, a hit is represented with 'X', a miss with '∼', and a grid cell that has not been attacked by the user with '.' (period) whether there is a ship or not. Setting displayShips to true should be used for debugging purposes. You might want to use it as long as the game is not fully working. Don't use it for the final version of the game.
- Write a method getUserInput: this method asks the user to choose a row and a column and returns the row and column.
- Write a method attackCell: this method takes the row and column, tests if a ship was hit or just the ocean, and makes the appropriate changes to the grid.
- Implement the functionality of the game using these methods.
- You can of course create additional methods where needed (for example, to determine if a ship has been sunk or the game is finished or to initialize the grid or to convert a String to an array of characters). Do not create objects for this part of the project.

Submission instructions

Submit your solution through canvas. Submit a single zip-file with:

- your Java programs. The programs have to compile correctly on the lab machines. Your name, the date, the course, and the assignment (Project 1) have to be written in the header.
- a readme file (readme.txt, 5 points) in which you describe the functionality of your programs. You might also describe problems you encountered, any improvements or enhancements that you implemented beyond the specification, and any known bugs or features of your program. At the top of your readme file you should include your name, the date, the course, and the assignment (Project 1).

Your programs are not only graded based on functionality but also based on how clean and elegant your code is written or commented. For the Battleship.java program there will be 25 points for functionality, and 5 points for elegance, gameplay, and comments. For the Tricks.java program, the break down is 12 points for functionality and 3 for elegance and comments. Make sure to get them all and enjoy this project.