Assignment 1: Let's Play Battleship!

CSCI 2270 - CS2: Data Structures
Kos - Summer 2015

We are emulating a simple version of the game Battleship. On a square board of user-defined size (4x4 for example), a user-defined number of ships are placed at random locations. Each ship occupies just ONE space on the board/grid. The user is then asked repeatedly to guess the location of the ship(s) until all ships are sunk and the game ends.

Here's what a sample output would look like:

```
Welcome to the B A T T L E S H I P !
How vast are your oceans? (Enter an integer for the size of the board):
4

How many ships are in your fleet? (Enter an integer for the number of ships)
3

Let the battle begin! Enter your guess (Enter two integers for the ship's location)
1 5

Out of bounds!

Enter your guess (Enter two integers for the ship's location)
1 3

Sunk! Yay, 2 ships left.

Enter your guess (Enter two integers for the ship's location)
2 2

Miss!

Enter your guess (Enter two integers for the ship's location)
2 1

Sunk! Yay, 1 ships left.

Enter your guess (Enter two integers for the ship's location)
2 4

Sunk! Yay, 0 ships left.

Congratulations! You won the game in 5 moves.
```

Part 1

Create a class ship, including its properties and methods.

Things to consider

Use the printout of the game execution on the previous page to reverse engineer what needs to be included in the ship class.

What are the properties of your ship class and what role does each property contribute in the game? Which properties can change during the game?

Which properties need to be retrieved during the game?

How do you update and retrieve properties of a class?

Implement class methods: a constructor, destructor, get and set methods for the ship class. All code can be included in one .cpp file.

Your program also needs to have a main function. In main, create five ship objects. The location of each ship can either be randomly generated (you will need to do this in part two tomorrow), or set by the user. After the ships have been created, output to the console the location and ship status (sunk or not) of each ship.

This main code is used to test your ship class. It is often good practice to test any class you write with a few test cases to make sure that the class functions as expected. Once you are confident that your ship class is complete, you can move onto part 2. You will not have to include this main code in your final submission; in part 2 you can delete or comment out this code and continue building the battleship game.

Part 2

Modify your main function to implement the functionality of the Battleship game.

1. Initialize the game board
    a. Read input from the user about the board size and the number of ships. You will need to check that the number of ships will fit on the board size.
    b. Create an array of ship objects, each positioned at random locations inside your board. You will need to check that a location is not already occupied by another ship.
2. Create a function called operation() which should be called in a loop until the game ends.
    a. Read input from the user about the next target location to fire upon.
    b. Check if there is a ship at the input location.
    c. If there is no ship send a “Miss!” message to the user.
    d. If a ship is located on the target, send a “Sunk!” message to the user, along with how many ships are left.
    e. If there are no more ships left, congratulate the user and output the number of guesses the user took to beat the game. Exit the game.

Submitting Your Code

Submit your .cpp file through Moodle as Assignment 1. Make sure your code is commented enough to describe what it is doing. Include a comment block at the top of the .cpp file with your name, assignment number, and course instructor