METROPOLITAN COMMUNITY COLLEGE
CSIS 222 – OBJECT-ORIENTED PROGRAMMING IN JAVA

HOMEWORK 03

INSTRUCTIONS

Your assignment is to build a Battleship game program. This assignment will give you practice working in the procedural paradigm and using arrays, and structures. The program also will exercise your ability to decompose a large problem into manageable pieces. A program of this size requires up-front planning, so your first tasks are to carefully design your data structure and map out a sensible decomposition strategy. In the implementation phase, incremental developing and testing are key to putting your plan into action. We highly recommend starting early to ensure that you have enough time to work through the issues, completely debug the functionality, and get help if needed.

In this version of the game of battleship, the computer positions random set of ships of various sizes on a two-dimensional board. Each square of the board is called a cell. Each ship occupies either a horizontal or vertical line of cells. The user attempts to guess where the ships are hidden by choosing a cell on the board. If the user's guess hits a location occupied by a ship, the program indicates that a ship has been hit. Otherwise, the location is marked as a miss. If all of the ship's cells have been hit, the ship is marked as sunk. To make the game more interesting, the user will have a limited number of missiles to sink all of the ships. The game ends when the user sinks all of the ships (the user wins) or runs out of missiles (the computer wins). For variety, each ship is of a particular model (battleship, aircraft carrier, rowboat, etc.).

Ship Data

The ship information is stored in constant variables and that information will be stored in an array. Below is the ship information and cells it occupies.

- Aircraft Carrier – 5 cells
- Battleship – 4 cells
- Destroyer – 3 cells
- Submarine – 3 cells
- Patrol – 2 cells

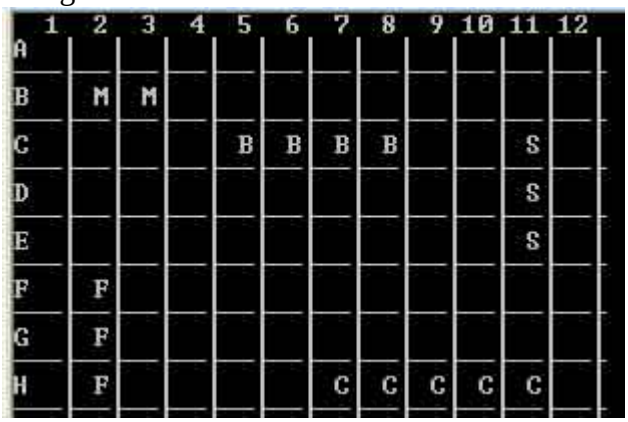
You may give each ship a name to display on the screen, but you must indicate the code and the message the type of ship you hit (example: You hit the destroyer USS Ronald Regan). Each ship will have a letter key associated with it.

Ship Placement

The program randomly places the ships on the board. The ship needs to occupy a horizontal or vertical line of cells, without extending past the board edges. Note that you need to be sure that at most one ship occupies any cell. For this version, ships are allowed to be adjacent to each other.

METROPOLITAN COMMUNITY COLLEGE
CSIS 222 – OBJECT-ORIENTED PROGRAMMING IN JAVA

Game Board
The game board will have numbers on the x-axis and letters in the y-axis. Below is a sample of the board:



You will need to indicate the value of an empty cell, hit, and miss. The board size will be exactly the same size for both horizontal and vertical.

The Player's Attack
The user will enter the cell they wish to attack. The program will check the board to see if that cell is a hit. If so, update the board to show the new hit and congratulate the user. If the cell is empty, ridicule the user and mark the cell to indicate a miss. If that cell had been previously fired upon, tell the user they're a bozo for wasting a missile on a location they've already chosen. If this hit is one that sinks a ship, mark the board to show that ship has been sunk. You also inform the user of the name and model of ship that was just sunk ("You sunk the aircraft carrier USS Mississippi") and tell the user how many more of that model of ship remain to sink ("There are 3 ships still out there"). For each turn, you update the scoreboard to indicate how many missiles are left, how many total ships are yet to sink, and what the user's accuracy is (number of hits divided by the number of total missiles fired – by two decimal places). If the player sinks all the ships, they win. If the player runs out of missiles, the computer wins.

Difficulty Levels
Prompt the user for the level of difficulty they want to play the game. Below are the specifications:

LEVEL	BOARD SIZE	MISSLES
Beginner	6 x 6	30
Standard	9 x 9	50
Advance	12 x 12	75

All levels will have the same amount of ships. You may output a message for each level after user choice (ex: Advance: So looking for a challenge, then you got one. Let's go!).

Designing a Data Structure
One of your most important tasks for this program comes before you do any coding, when designing your data structures. Think carefully through the information you need to store: the ships, their models, their locations on the board, and so on. A well-designed approach for storing and accessing the data is essential. As a small example, just consider the problem of storing the ship location information. At times you need to go quickly from a board location to the ship that occupies it. In other situations, you have a ship and need to quickly access its placement on the board. Although you could always determine one given the other, it is worthwhile to build a bit of redundancy into your data structures in order to facilitate both types of access. A two-dimensional array is a good choice for representing the board and provides direct access to each cell. Each cell could keep

information to the ship that occupies that cell, or a symbol to indicate the cell is empty. This gives quick access to the ship at a given location. Every cell on the array will have information and not display "NULL" information. There is other information that you need in various data structures (whether each cell has been fired upon, number of missiles remaining, what model a ship is, and so on) and it will be up to you to work out the appropriate organization for this data. There are many interesting issues to consider: choices between static (fixed-size) and dynamic allocation, where to use structures, and so on. All of these are valuable opportunities for becoming familiar with the programming code, syntax, and structure for managing data.

Data should not be haphazardly thrown together. If you can't think of a good name for a particular structure, this might be a sign it isn't a logical grouping. Keep wasted space to a minimum and avoid unnecessary redundancy. Sometimes these goals come in conflict with one another. You are trying to strike the right overall balance. A decision that's right for this program might go the other way in a different context. Taking care when designing your structure makes the entire program much easier to develop and debug.

Getting the program to work is only half the challenge. I am looking for a well-crafted program with good style— sensible decomposition, clear readability, and intelligent comments. Your program should be easy to read and stylistically elegant. Be conscious of the style you are developing and aim for consistency. Comment thoughtfully. Take care to avoid any unnecessary code repetition—if you need something more than once, write one function and call it from both places. Functions should represent small, meaningful subtasks and be named and parameterized appropriately.

Data Structure framework
There will be three java files:

- battleship.java: This will be the main file that will run the game. This will call the actiongame.java file to run.
- actiongame.java: This file will be running the game.
- gameboard.java: This file will contain code to build game board, placement of pieces, tracking score, and other code for the game.

You will be submitting all three files. The name of the Java Class package will be "battleship".

DELIVERABLES

- Proper information declared as constants, variables, class, objects, and functions.
- Proper use of comments, calculations, and code space
- Creating proper objects
- Make sure to fully test your program, including tests with ships extending out-of-bounds, overlapping ships, firing out of bounds and firing twice at the same spot.

BONUS (15 points)
Create a fourth file called "computermove.java" to play against the computer. This file is where the computer moves will be accomplished. The computer will randomly choose pieces on your board. If he gets a hit, then the computer starts working around and find the next hit, figures the pattern, and executes it until your ship has been sunk. This will become a turn based game. The winner of the game is the following (in order):

- Winner sinks all ships before running out of missiles
- If all missiles run out, the person with the most hits wins,
- If both players have same hits, then the average hits break the tie.

The only addition to the three original files are the following:

- battleship.java: add choice if user wants to play the computer or not

- actiongame.java: build board piece for the computer.

In order to receive full bonus points, all aspects of the bonus portion must be in working order as well as the user able to play in normal game (standard game as instructed above).