For my final project, I set up the game BattleShip. It works similarly to the real-world BattleShip board game. Two players take different kinds of ships and place them on their boards. Each ship takes up a different amount of squares. Once both players have placed their ships, they begin “bombing” the other team’s board, trying to hit each square that a ship is placed on, by calling out a coordinate on their turn. They keep track of the coordinates they have tried to bomb on a blank board adjacent to their own board. The game ends when one players’ ships have all been hit/sunk.

Some concessions I made from the real life game is that the second player is a computer and chooses the placement and bombing of ships randomly. The board is smaller (6x6) then it would be in real life which is 10x10. Additionally, both players can only use one of each kind of Ship by default.

To start the game, run “java BattleShipGame. This will start the game, displaying two boards (your *playerBoard* and what will be your *trackingBoard*) with a “Play Game” button at the bottom. All dialogue will show up at the bottom to prompt the interaction with the game. Once the “Play Game” button is clicked, you will see a “Carrier” and “Clear” button. The “Clear” button will be present anytime you are placing ships so you can undo the tiles you have clicked and try again. To start placing ships, click the “Carrier” button. Each ship(Carrier,Cruiser,BattleShip, Submarine, Destroyer) you will need to place will follow this pattern of having a button to click to get started with placement. Once you click the “Carrier” button, you will see a “Done” button and the click handlers of the *playerBoard* (the left board) will be enabled. To place the ship, click the squares you want to set it on. You can can click any squares you want but have a valid placement, the squares need to be in order horizontally or vertically and the correct number must have been clicked. Once the player has clicked all of the required squares, they will click “Done” button which runs the validation. If it passes, the player moves to placing the next ship. If the validation fails, nothing will happen and player should click the “Clear” button to try again. Once the player has placed all of their ships, they will see a button saying “Go!!!” and this will kick off the game. At all times, you will see the placement of your ships on the *PlayerBoard.*

The UI will switch up slightly. The *playerBoard* (left board) will become unclickable and the *TrackingBoard* (right board) will become clickable. The computer will be trying to bomb the *playerboard* so on each turn, the player will see X if the bomb was a miss and a H if a ship was hit on their *playerBoard* and conversely when they bomb their opponent which is tracked on the *trackingBoard*. The player has first turn and will click the square on the right they think holds a ship. Once the player clicks, the computer will make its move and this will continue until all ships are sunk for a player.

I have enclosed a class diagram and a flow diagram to assist with understanding my program. In the spirit of disclosure, I want to disclose things that could be recognized as bugs but are intentional designs based upon time and/or not worth the risk.

1) makeComputerMove() within TrackingBoard.java -> the method looks for a random coordinate to place the Ship and checks against the ArrayList of integers that have been

used. As the game progresses, the available squares that haven't been chosen gets much less and the method will struggle to find one. I have added a check for it to try at least 20 times then give up if it can't find one. This will mean that it picks to bomb a square that is already bombed. This will change nothing in the flow of the game except the computer didn't actually complete a turn. I think this could be optimized by adding a check on the size of the used number ArrayList and if above a certain number, try to find the numbers missing from that list. Since the numbers would be 0-55, if we sorted the arraylist(or put it in a structure that allowed it), we could find it fairly easily.

2) setRandomSelection() within Ship.java -> This method selects a random coordinate and from there, tries to place the ship. It looks horizontally and vertically for enough squares to fit the ship. If it can not find one, I have the method bailing out and it does not place the ship. It does check to see if the squares are used. To see if it bailed out, I have logging on the command line to show the coordinates the computer chose for the ships and you would see "Can't find placement" for that ship. I have put in code to not show the Ship within the ShipCounter and it is not taken into calculation for hits left in deciding if a user won the game.
3) I set up Status enums on the Ship and Board class to track their status. I did not really use them but I think if the program was to be expanded, it would be useful so i did not remove them.
4) checkLocation() within PlayerBoard.java. I borrowed code from StackOverflow. I credited it.
5) addEvents() within TrackingBoard.java. I had to remove the action listeners before adding them back. I was unable to find the extra ones that were being added.
6) checkShipLocation within TrackingBoard.java. I do not think a HashMap was the best data structure to use for this but I needed a way to pass back the Ship object with a result (true or false) and HashMap was the only data structure i knew within Java to pass two variables.
7) I handled the coordinates of the board multiple ways throughout the code. If I had more time to refactor, I would have chose one and stuck with it since it makes it difficult to add methods since the JButton, the boards, and the usedNumber array use the numbers differently.
8) The UI is inspired by the actual BattleShip board game but i would go back and remove the second board until we need to use it so the window is more capable of being resized instead of the right board folding underneath the left board.
9) With more time, I would implement a reset button so you would not need to close the game to restart it. I looked into doing this but felt it wasn't worth the additional testing time.
10) In my initial design, I had a HumanPlayer and ComputerPlayer class, both which inherited from a Player class. I scraped them because I didn't use them but I do think it would have been a better design to have done so.