This is a GROUP Project. You may work in groups of upto 3 people.

Battleship is a game that you probably played as kids. It's a guessing game between 2 players. You have to implement this on the computer.

The game itself involves two grids positioned over locations in the ocean. Each player may only see the ships that are in their own grid and their own attacks on the opponent's screen. Players take turns launching missiles into individual grid locations with the goal of sinking the opponent's ships. When a missile is launched, the player is told whether the missile "hit" or "missed". The game also says on a "hit" if the ship was "sunk", meaning that all of the locations the ship occupies have been hit. The game is won when all locations that the enemy's ships cover have been "hit".

The requirements of the game are as follows.

- 2 Players
- Each player has a 10X10 board to place the ships.
- Players will be given 5 ships that are 3 units long and 1 unit wide
- Ships can be oriented vertically or horizontally.
- Each player sees 2 grids - their own with the ships placed and their opponent's grid with a Record of where they have attacked.
- Each player takes turn attacking the opponents ships.
- The result of an attack can be one of:
    - A 'Hit' if the opponent has a ship covering the coordinate
    - A 'Miss' if there is no ship covering the coordinate
    - 'Sunk' if all coordinates a ship covers have been hit
    - 'Win' if all ships on the opponent's board have been sunk

The game ends when all of a player's ships have been sunk.

Game Progress:

1. A starter code in Swing has been provided to you in BattleShip.zip under Files. It consists of the following files
    a. BattleShip: this is the main class. It creates the PlayerScreens.
    b. PlayerScreen: Represents the screen of a player. Feel free to let your creative juices flow and make it easy for the user. Currently it has two grids for each player 1 for self and 1 for attack
    c. BattleGrid: Implements the gridlayout used for self and attack grids
    d. SelfGrid: Shows player's own position of ships (Needs implementation)
    e. AttackGrid: Shows attack positions and results. (Needs implementation)
2. Setting up the game: Initially the game starts with just Player1's grid visible. The player can click on any of the grid. It will create a ship 3 units long. Show the ship in a different color than the grid. Start with horizontal orientation (left to right). Require the player to choose 5 grids (for 5 ships). You must make sure that there are no overlaps. For ex if there is a ship at (0,0) covering

(0,0),(0,1),(0,2), the user cannot choose (0,1) or (0,2) for the next ship. Choosing (0,0) does nothing. Once the player1 has setup the game, he/she can click on the next button to give control to player2 for setting up the game.

3. Playing: when it's a player's turn he uses the attack grid to aim a missile. See above for the results of attack. Indicate the position of the attack and the result. (For simplicity, it is ok to only show hit (green) or miss(red) on the grid and use the status section to show the number of ships remaining)
4. The game finishes when all ships of a player are sunk. Display the result and congratulate the winner

Things to implement

1. The grids need to handle the clicks. The attack grid should check the opponent's ship positions (Beware: breaking encapsulation is not permissible. Think about how you can enhance the attackgrid/playerscreen class to do this).
2. There are a lot of state transitions in this game. Use the State Design Pattern discussed in class to implement this. Where should this reside? New class? BattleShip? This state pattern should handle the toggling of the player screens
3. The data for displaying on the grid needs to be maintained somewhere (where?). For simplicity, you may choose to do this in a 2D array in PlayerScreen. (Remember you will need 2 2D arrays - 1 for own and 1 for attack) Ideally create a separate class for PlayerData and use that to check/update the cells. (Model-View approach discussed in class). See #5 for additional data that this class can track.
4. Modify the cell's paintComponent to set the color based on the underlying data. This would automatically maintain the visual state as you resize or toggle the visibility.
5. Add a status region to the SOUTH of PlayerScreen to show the following. Use the hint in #3 to drive this display.
    a. Number of own ships:
    b. Number of own ships sunk:
    c. Number of enemy ships sunk:
    d. Current state (Can be one of player1 setup, player2 setup, player1 attack, player2 attack, Game over (result))
6. When a player attacks, you will need to determine the result. Decide which is the most appropriate place for doing so.

Submission:

1. Submit a runnable jar and all source files in a single zip.
2. Include a README with names of the team members and student ID.
3. Only one submission per team.

Extra Credit:

1. During setup, clicking on the first cell of a ship toggles the orientation of the ship.
2. During setup, clicking on the any cell of a ship toggles the orientation of the ship.