BATTLESHIP GAME REPORT

TASK.1 - BattleShipClass - COMPLETED

Tasks 1.1 and 1.2 implementing the getter and setter methods, and extending the **AbstractBattleShip** class are fairly straightforward. On the other hand task, 1.3 is confusing for me. Each BattleShip class had a 2-d variable for storing the coordinates of ship size N. The test should have initialized the ship coordinates size as three rows and two columns.
That:

```
int [][] shipCoordinates = new int [2][1];
```

Instead of :

```
int [][] shipCoordinates = new int [1][2];
```

Also, I also struggled with the **IndexOutOfBoundsException.** Instead of using nested loop statements and nested if, I used a simplified for loop and used an and (&&) symbol.

```
for (int [] coors: this.shipCoordinates) {
    int x = coors[0];
    Int y = coors[1];
 }
```

Instead of :

```
for(int i=0; i<this.shipCoordinates.length) {
   for(int j=0; j<this.shipCoordinates[0].length) {
    }
}
```

Since ship coordinates for the game only have x and y values, a nested loop is not necessary.
As far as task 1 was concerned, it was straightforward.

TASK.2 - GameGridClass - COMPLETED

As far as I can tell, I found creating the gameGrid class, initializing it, printing it, and generating ships to be fairly straightforward.
For the **placeship** method, it has the main while loop that checks the random x and y points generated to see if the space is empty before placing the ship using a class I added called **isEmptySpace.** These random generated x and y points are generated between values of 0, width - 2 and 0, height - 2 respectively to reduce the chances of those points falling out of the grid.

```
   private boolean isEmptySpace(int [][] coordinates) {
       boolean isEmpty = true;
       for (int [] cor: coordinates){
```

```
            int x = cor[0];
            int y = cor[1];
            isEmpty = gameGrid[x][y].equals("*") ? false: true;
        }
        return isEmpty;
    }
```

This function does not work well with a game grid of width less than 10 and of height less than 10

TASK.3 - GameClass - COMPLETED

The most complex task was task 3, while tasks 3.1, 3.2, and 3.3 were fine, but task 3.4 with the playground method proved to be the most challenging.
The **checkVictory** method checks whether the player has won or lost; if the player has won, it returns true to indicate that the game is over; if not, it returns false to indicate that the game has not ended. The boolean value returned by it was used in task 4's main loop.
The **PlayRound** Function takes in input for x and y coordinates on the game grid,
checks for a ship attack in all ships in the opponent instance,
create Random values, and also checks for a ship attack in all ships in the player instance.
Checking for hits, printing a Hit or Miss message, and registering were the hit or miss occurred are the main purposes of this function.
This code is encapsulated in a try-catch block that checks for incorrect input values and
the **IndexOutOfBoundsException.**

TASK.4 - RunGameClass - COMPLETED

The **RunGame** class in my code has a main method and a main loop that checks for errors in the input passed by the person running when the **RunGame** class.
The loop checks if the game is running, whether the input word is “exit”, checks for player and opponent victory or loss, and checks if the input is the wrong value type.

It takes input from the main method and the user through the "System in" method, creates an instance of the game, plays it, checks for the exit keyword, checks for victory, and determines whether to end the game.

HOW TO RUN THE CODE

Open the **README.md** file in the root directory and please follow the instructions to run **RunGame.java**.