Java Assignment

Report

School:	University of Aberdeen & South China Normal University
Major:	Computer Science
Name:	Ruofan Liao
Student Number:	50079732
Date:	Oct 30th, 2022
Github:	https://github.com/LLLLLrf/JavaAssignment

Contents

1.1 Task status

Task	Status
Task 1 : Create Battle Ship	Completed
Task 2 : Implement the functionality to display the game grid	Completed
Task 3 : Create a mechanism for playing game in rounds and handling the attacks	Completed
Task 4 : Complete your code to implement remaining game controls	Completed
Task 5 : Report and Code Comments	Completed

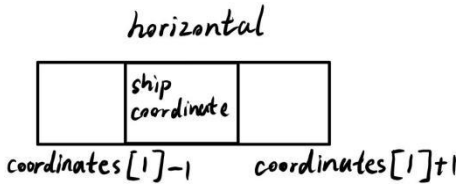
1.2 Task description

1.2.1 Task 1 : Create Battle Ship

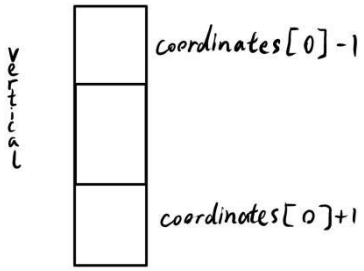
1.1) I created a **BattleShip** class that extends **AbstractBattleShip** class and defined a class constructor that received a string to set the name of ship. I define an array to store two strings “horizontal” and “vertical”, use **nextInt(2)** in **Random** class to randomly set the **shipOrientation**.

1.2) I implemented the getter and setter methods that defined in the **AbstractBattleShip** class to provide access to the class variable.

1.3) The **checkAttck(int row, int column)** method receives two ints as attack coordinates, use **shipCoordinates** and **shipOrientation** to check whether the ship was attacked (Figure 1, Figure 2), if the ship was attacked and the number of hits was less than 3, let hits=hits+1.



(Figure 1)



(Figure 2)

1.2.2 Task 2 : Implement the functionality to display the game grid

2.1) In **GameGrid.java**, **initializeGrid()** use the three input received by class constructor to initialize the game grid, populate the grid with "." characters.

2.2) **generateShips(int numberOfShips)** generate specific number of ships named “Ship 1, Ship 2.....” and store them in a **ships** array.

2.3) In **placeShip(AbstractBattleShip ship)** method, I first use **getShipOrientation()** method to check the orientation of the ship, if the ship was “horizontal” (Figure 3), I narrow the range of width, in order to avoid the ships “overspill” from the grid. Similarly, narrow the range of height when ship is “vertical”. (Figure 4)

```
if (orient.equals(anObject: "horizontal")) {
    row = r.nextInt(height);
    col = r.nextInt(width - 2) + 1;
    coordinates[0][0] = row;
    coordinates[0][1] = col;
    ship.setShipCoordinates(coordinates);
    this.gameGrid[row][col - 1] = "*";
    this.gameGrid[row][col + 1] = "*";
    this.gameGrid[row][col] = "*";
```

(Figure 3)

```
} else if (orient.equals(anObject: "vertical")) {
    row = r.nextInt(height - 2) + 1;
    col = r.nextInt(width);
    coordinates[0][0] = row;
    coordinates[0][1] = col;
    ship.setShipCoordinates(coordinates);
    this.gameGrid[row + 1][col] = "*";
    this.gameGrid[row - 1][col] = "*";
    this.gameGrid[row][col] = "*";
}
```

(Figure 4)

2.4) In **PlayerGameGrid.java** and **OpponentGameGrid.java**, not only will it print out the game grid as task required, it will also print out the coordinates for each row and each column, which is more convenient for player to choose the coordinates they want to attack. (Figure 5)

```
HIT Ship 4!!!

Opponent's GameGrid
  0 1 2 3 4 5
0 % % X . . .
1 . . . . . .
2 . . . . . .
3 . . . X . .
4 . . . . . .
5 . . . . . .
6 . . . . . .
7 . . . . . .

Opponent is attacking...
MISS!!!
My GameGrid
  0 1 2 3 4 5
0 . % . . . .
1 % . . . . .
2 . . % . . .
3 . . * * * .
4 . . . * . .
5 . . * * . .
6 . . * * . .
7 . . * . . %
Please enter the position you wish to attack
enter 'exit' to quit the game, enter 'check' to see the current record
```

(Figure 5)

1.2.3 Task 3 : Create a mechanism for playing game in rounds and handling the attacks

3.1) The constructor in **Game.java** will receive three parameters and use them to instantiate both player’s and opponent’s grid. I also implemented the two getter methods to retrieve player’s and opponent’s grids

3.2) Each round of the game, I check the input by using regular expressions (Figure 6), if the input involve “exit”, it will invoke the exit method and print out “Exiting game-thank you for playing.....”. Additionally, if the input involve “check” it will invoke **checkShip()**, which is an extra method I wrote. (Figure 7)

```
String pt1 = "(\\d{1,})[\\s,/.](\\d{1,})[\\s,/.](\\d{1,})";
String pt2 = "(-?\\d{1,})[\\s,/.](-?\\d{1,})";
String pt3 = "exit|check";
Pattern p1 = Pattern.compile(pt1);
Pattern p2 = Pattern.compile(pt2);
Pattern p3 = Pattern.compile(pt3);
```

(Figure 6)

```
__________myships__________
Ship 1 dead
Ship 2 alive hits:2
Ship 3 alive hits:1
Ship 4 alive hits:2
Ship 5 alive hits:1
Ship 6 alive hits:2
_________opponentship_________
Ship 1 dead
Ship 2 alive hits:2
Ship 3 dead
Ship 4 dead
Ship 5 alive hits:2
Ship 6 dead
press 'enter' to return to the game
```

(Figure 7)

3.3) Also we need use **checkVictory()** method to check whether game is over and who won the game. I iterate through all the ships and if all of one’s ships have been destroyed, he or she lose the game, but if all ships in both sides were destroyed, the program will print out “End in a tie!”.

```
boolean Iwin = false;
boolean Ilose = false;
if (my_deadships == myGameGrid.ships.length) {
    Ilose = true;
}
if (opp_deadships == oppGameGrid.ships.length) {
    Iwin = true;
}
if (Iwin) {
    if (Ilose) {
        RunGame.clear();
        System.out.println("End in a tie!");
        return true;
    }
    RunGame.clear();
    System.out.println("You have won!");
    return true;
}
if (Ilose) {
    RunGame.clear();
    System.out.println("You have lost!");
    return true;
```

(Figure8)

3.4) In **PlayRound()** method, I separated the input coordinates and turn them into integer and check whether the coordinates are reasonable, if not, print out the error. We then iterate each opponent’s ship, use **checkAttack()** method to see whether the ship was attacked. If any ship was attacked, print “HIT <ship name>!!!”,

print “MISS!!!” if not. Then use **printGrid()** method to show the game grid. After that is opponent’s turn, I use **Thread.sleep()** and **nextInt()** in **Random** to simulate the opponent’s thinking time. Randomly generate two integer to be the coordinates, if the coordinates have already been attacked, the program will regenerate new coordinates until it haven’t been attacked before. And similarly print whether any ships were hit.

1.2.4 Task 4 : Complete code to implement remaining game control

4.1) In **RunGame** class, I use the arguments of the main method to create an instance of **Game** class. (I also provide a version that don’t need arguments from main method, receive from user console input instead)

4.2) & 4.3) The code for each round of the game is put into a while loop, only when “exit” is involved in user’s console input or the return value of **checkVictory()** method is true, which means the game is over.

4.4) pass the attack coordinates input by player to the **playRound()** method.

4.5) In this part, I match the player's input with regular expressions in case of some careless mistyping. The following two methods **findCoordinate()** and **findCommand()** are used to match the input. If both methods can not match anything in the input, it will print “Incorrect input”. (Figure 9)

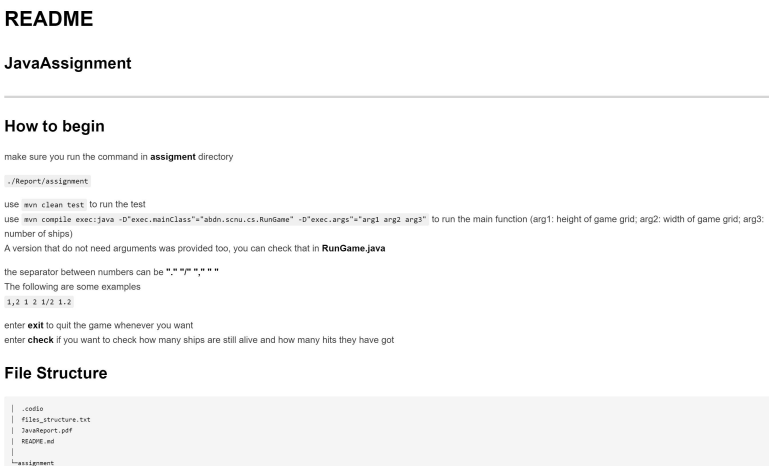
```
// match the coordinate to avoid mistyping
public static String[] findCoordinate(Pattern p, String input) {
    Matcher m = p.matcher(input);
    String[] coord = { "none", "none", "none" };
    while (m.find()) {
        coord[0] = m.group(group: 1);
        coord[1] = m.group(group: 2);
        try {
            coord[2] = m.group(group: 3);
        } catch (IndexOutOfBoundsException NumberFormatException) {
        }
        return coord;
    }
    return coord;
}
// match the commands to avoid mistyping
public static String findCommand(Pattern p, String input) {
    Matcher m = p.matcher(input);
    String coord="none";
    while (m.find()) {
        coord = m.group(group: 0);
        return coord;
    }
    return coord;
```

(Figure 9)

1.2.5 Task 5 : Report and Code Comments

To make sure everything is clear to the users, not only did I wrote a large amount

of comments to explain the code, I also wrote a **README.md** to show how to get started. (Figure 10)



README

JavaAssignment

How to begin

make sure you run the command in **assigment** directory

```
./Report/assignment
```

use `mvn clean test` to run the test

use `mvn compile exec:java -D"exec.mainClass"="abdn.scnu.cs.RunGame" -D"exec.args"="arg1 arg2 arg3"` to run the main function (arg1: height of game grid; arg2: width of game grid; arg3: number of ships)

A version that do not need arguments was provided too, you can check that in **RunGame.java**

the separator between numbers can be **"." "/" "," " "**

The following are some examples

`1,2` `1 2` `1/2` `1.2`

enter **exit** to quit the game whenever you want

enter **check** if you want to check how many ships are still alive and how many hits they have got

File Structure

```
| .codio
| files_structure.txt
| JavaReport.pdf
| README.md
|
└─assignment
```

(Figure 10)

2. How to begin

````
make sure you run the command in assigment directory
```./Report/assignment```
use mvn clean test to run the test
use mvn compile exec:java -D"exec.mainClass"="abdn.scnu.cs.RunGame" D"exec.args"="arg1
arg2 arg3" to run the main function (arg1: height of game grid; arg2: width of game grid;
arg3: number of ships)
A version that do not need arguments was provided too, you can check that in RunGame.java
the separator between numbers can be "." "/" "," " "
The following are some examples
```1,2     1 2     1/2     1.2```
Enter exit to quit the game whenever you want
Enter check if you want to check how many ships are still alive and how many hits they
have got
````

3. Reference list

1. https://blog.csdn.net/weixin_39683021/article/details/114456413

In this article I learned how to clear the console output, so that I can clear the console when each game round start.

2. All the development processes are documented in my github repository:

Github: https://github.com/LLLLLrf/JavaAssignment