UNIVERSITY OF ABERDEEN

University of Aberdeen

School of Natural and Computing Sciences

Department of Computing Science

2022 – 2023

Assessment Item 1 of 1 Briefing Document – Individually Assessed (no teamwork)

Title: CS2020 – Software Programming	Note: This assessment accounts for 25% of your total mark of the course.

Learning Outcomes

On successful completion of this component a student will have demonstrated competence in the following aspects:

- Ability to write and run basic Java applications
- Ability to judge how Java applications should be structured
- Understanding and ability to apply, techniques to support correct code in a Java application
- The ability to analyse simple problems and design a program solution
- The understanding of software libraries and their application in order to classify them for usage
- The ability to apply programming concepts for solving problems in order to create a new program
- Knowledge and understanding of basic programming concepts and their application,

Information for Plagiarism: The source code and your report may be submitted for plagiarism check (e.g., Turnitin or other tools). Please refer to the slides available at MyAberdeen for more information about avoiding plagiarism before you start working on the assessment. Please also read the following information provided by the university: https://www.abdn.ac.uk/sls/online-resources/avoiding-plagiarism/

Report Guidance & Requirements

Your assessment code and report must conform to the below structure and include the required content as outlined in each section. Each subtask has its own marks allocated. You **must** supply a written report, along with the corresponding code, containing all distinct sections/subtasks that provide a full critical and reflective account of the processes undertaken.

This assessment includes coding tasks and a final written task outlining the steps taken as aforementioned. You will be provided with a template code for some of the classes to help you get started. The goal of this assessment is to build a simple Battleships game with a text-based interface. The goal of the game is to guess the location of all your opponent's ships positioned on a 2D grid before your opponent discovers your ships.

You'll also find submission details, which you need to follow, at the end of this document.

Use Git to Save Your Work

While we're not marking your use of Git in this assignment, you are strongly encouraged to do this work inside a Git repository. By doing that you gain the benefit of rolling back your edits, or doing work in branches, and then merging it to the main branch as you accomplish tasks along the way.

Perhaps most importantly, by using Git and pushing your work to a remote repository on GitHub, you provide a safe backup of your work. You'd be surprised by how regularly fellow students discover failed hard disk drives, or that the laptop they're using needs to be returned to its owner, or be repaired due to some liquid being spilt on the keyboard.

If you push your work regularly to GitHub, then you can easily (a) pull it into Codio and carry on if you lose your laptop, and (b) move it to a different device if you need to.

By the way, 'regularly' in this case means making a commit every 5 or 10 minutes locally, and then pushing it to GitHub every hour. Small local commits mean you can roll back small changes without much trouble, and pushing remotely less frequently stores your work safely offsite.

To test your progress the project template includes 11 tests. They are initially commented out so they don't throw compilation errors because they expect certain classes that you will have to create.

With a clean template after running **mvn clean test** you should see the following:

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.562 s
[INFO] Finished at: 2022-08-29T12:32:16Z
[INFO] ------------------------------------------------------------------------
codio@dragon-scorpio:~/workspace/Assignments/2022/assignment1_template$
```

Once you have completed a subtask, uncomment relevant test(s) and corresponding imports. For example, after completing task 1.1. your test file should look like this:

```
import CS2020.assignment1.game.BattleShip;
/*
import CS2020.assignment1.game.Game;
import CS2020.assignment1.game.GameControls;
import CS2020.assignment1.game.GameGrid;
import CS2020.assignment1.game.OpponentGameGrid;
import CS2020.assignment1.game.PlayerGameGrid;
*/
```

Note, we need to import the BattleShip class which is expected by the tests. Then uncomment t1_1a and t1_1 b after completing the task 1.1.

```
@Test
public void t1_1_a() {
    BattleShip ship = new BattleShip ("Ship");
    try {
        assertEquals ("Ship", ship.name);
    } catch (Exception | Error e) {
        fail(ANSI_WHITE_BACKGROUND +ANSI_BLACK+"T1.1_a: Make sure the constructor of BattleShip class is modified and
    }

}

@Test
public void t1_1_b() {
    int horizontal = 0;
    int vertical = 0;

    for (int i=0;i<100;i++) {

        BattleShip ship = new BattleShip ("Ships" +i);

    if (ship.shipOrientation.equals("vertical")) {
        vertical++;
    }
    if (ship.shipOrientation.equals("horizontal")) {
        horizontal++;
    }
    }

    if (horizontal<20||vertical<20) {
        fail(ANSI_WHITE_BACKGROUND +ANSI_BLACK+"T1.1_b The ship orientation should be decided at random. Out of 100 sh

    }
}
/*
@Test
public void t1_3() {
```

After running **mvn clean test** you should see the following if you have completed the task correctly:

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running CS2020.assignment1.game.GameTests
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.032 s - in CS2020.assignment1.game.GameTests
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.733 s
[INFO] Finished at: 2022-08-29T12:54:03Z
[INFO] ------------------------------------------------------------------------
```

Note that not all tasks have tests defined. **Make sure you mention in the report which tasks you have completed.**

In later stages, you can use the command **mvn package** to generate a jar file which then can then run with **java -jar assignment1-1.0-SNAPSHOT.jar <your parameters>** from the target folder. Alternatively, you can run the main method directly with **mvn compile exec:java -Dexec.mainClass="CS2020.assignment1.game.RunGame" -Dexec.args="arg1 arg2 arg 3".**

When the game starts, the console output should be similar to this (depending on the grid size and number of ships specified):

```
Player's grid

....

***.

....

....

Opponent's grid

....

....

....

....

Please enter the position you wish to attack
```

The following provides a detailed description of each task. To complete these tasks, follow the instructions and ensure you use the provided template. All processes taken to run the final project must be clearly described. For submission instructions refer to the later sections.

****Please read all the information below carefully****

Task 1: Create Battleship objects (10 points)

1.1) Create **BattleShip.java** that extends **AbstractBattleShip** class. Define a class constructor that sets the name of the ship based on the passed method parameter, and randomly decides the ship orientation by assigning “horizontal” or “vertical” values in the **shipOrientation** variable defined in

the abstract class. You can utilise the **Random** class and **nextInt** () to give you random integers within certain range.

1.2) Implement the getter and setter methods defined in the **AbstractBattleShip** class to provide access to the class variables (i.e., **getName** (), **getHits**(), **getShipOrientation**(), **sethits**(), **getShipCoordinates**(), and **setShipCoordinates**())

1.3) Implement the functionality of the **checkAttack** method. The method should check if the ship has been already destroyed (i.e., received three hits) and return **false** if this is the case even if the attack coordinates match the position. If any of ship's coordinates match the attack coordinates represented by the row/column parameters passed to the method and the number of hits is less than 3 then the method should return **true**. In case of a successful attack, the method should also increase the amount of hits registered for this ship.

Task 2: Implement the functionality to display the game grid (35 points)

2.1) Create **GameGrid**.java that extends the **AbstractGameGrid**.java. Create a class constructor that takes three inputs width, height, and number of ships that should be created. The constructor should create a 2D String array representing the grid and initialize it using the **initializeGrid** method defined in the abstract class with every element having value set to ".".

2.2) Implement the functionality of the **generateShips** method to generate required number of ships and store it in the ships array. Each ship should have assigned name "Ship 1", "Ship 2", etc.

2.3) Implement the functionality of **placeShip** method. This method should place ships randomly on the grid. You can utilise the **Random** () object and **nextInt** () to give you random integers within certain range. Make sure that you respect the ship orientation (either vertical or horizontal) when placing the ship on the grid. Overlapping ships are fine. However, make sure that you don't try to place ships that would "overspill" from the grid (e.g., first element of the horizontal ship to the last element on the right). Update the grid so the ships are denoted by the "*" symbol. Each ship object should have it's coordinates stored in the **shipCoordinates** array (defined in the **AbstractBattleShip**) and all ship objects should be stored in the ships array (defined in the **AbstrcatGameGrid**).

Example of two ships positioned on the grid:

.***..*...

......*...

......*...

2.4) Create **PlayerGameGrid**.java and **OpponentGameGrid**.java that extend **GameGrid** class. Within both classes define method **printGrid** which when called will print either "Player's grid" or "Opponent's Grid" message followed by the contents of the grid. Opponent's grid should not reveal the position of the ships (i.e., "*" symbols) but should show the hits (i.e., using the "X" symbol), misses (i.e., using the "%" symbol), and everything else as ".".

Task 3: Create a mechanism for playing game in rounds and handling the attacks (35 points)

3.1) Create **Game.java** which should implement the **GameControls** interface. Define a class constructor that will create and initialise player's and opponent's grids with the appropriate size of the grid and the number of ships using the classes defined in 2.4. The constructor should take three parameters number of rows, number of columns, and number of ships to be placed on the grid. Implement getter methods to retrieve player's and opponent's grids.

3.2) Implement the **exitGame** method that ends the program if the input contains value "exit". The following message should be printed into the console before exiting: "Exiting game – thank you for playing". Note: There are no automated tests for this sub-task

3.3) Implement **checkVictory** method that checks whether either all opponent's or player's ships have been destroyed (i.e. all received three hits). If all ships have been destroyed message should printed into the console informing the player who won by printing:

"You have won!"

Or

"You have lost!"

3.4) Implement the **playRound** method. The method should expect input in a form of a string where there are two numbers separated by ",". First number represents the x coordinate and the second number the y coordinate on our grid (hint: you will need to split the string and parse characters into numbers). Complete the method so the user input is used to check the opponent's grid for ship positions. If the coordinate matches part of the ship print "HIT <ship name>!!!" into the console, update the number of hits recorded in the corresponding ship object, and update the grid with symbol "X" denoting the hit.

```
0,1
HIT Ship 1!!!
.X.......

..........

..........
```

If there is no ship in that position, print "MISS!!!" into the console and update the grid with symbol "%" denoting the miss.

```
0,2
MISS!!!
.X%.....

..........

..........
```

After the user input is processed, the method should generate a random set of coordinates to simulate an attack on player's grid. No need to be very sophisticated here at this time (e.g., utilising the positions of partially hit ships) but if you have time you can make your "bot" smarter.

NOTE: you can break up your code in as many helper methods as you want to avoid a code repetition.

Task 4: Complete your code to implement remaining game controls (10 points)

Note: There are no automated tests for this task

Create a **RunGame.java** that starts your game. The following functionality should be implemented:

4.1) **RunGame** class contains the main method that starts the game with user defined inputs for game grid width, height, and number of ships (hint: use the arguments of the main method). The main method should create a single instance of **Game** class**.**

4.2) The code continuously plays next rounds of the game (i.e., checks for user console input) until termination conditions are met.

4.3) There are two termination conditions:

- User types a word "exit". Utilise the method from 3.2.

- Either player's or opponent's ships are destroyed. Utilise the method from 3.3

4.4) Input from the player defining attack coordinates should be passed to the **playRound** method defined in the **GameControls** interface and implemented in the **Game** class

4.5) Cathc an exception thrown with an unexpected input (i.e, if users enters anything else apart from "exit" or "x,y" coordinates. When the error occurs print "Incorrect input" into the console.

Task 5: Report and Code Comments (10 points)

Write a report (5 page max) listing all the tasks and their status (i.e., completed, partially completed, not attempted). For each tasks also include a brief paragraph outlining your approach. At the end of your report include a section with instructions how to run your code. In your .java files, please include comments explaining the purpose of individual functions.

Example Game output:

```
Please enter the position you wish to attack

3,3
Player is attacking
MISS!!!
Opponent is attacking
MISS!!!
Player's grid

%...

***.

....

%...

Opponent's grid

.%..

....

...%

...%

Please enter the position you wish to attack

2,1
Player is attacking
HIT Ship 1!!!
Opponent is attacking
MISS!!!
Player's grid

%...

***.

...%

%...

Opponent's grid

.%..

....

.X.%

...%

Please enter the position you wish to attack

wrong input
Player is attacking
Incorrect input
Please enter the position you wish to attack

Player is attacking
```

Useful Information

- Please describe and justify each step that is needed to reproduce / run your work by using clear descriptive writing, supporting code-snippets and screenshots. When using screenshots please make sure they are clearly readable.

- If you use open source code, you must point out where it was obtained from (even if the sources are online tutorials or blogs) and detail any modifications you have made to it in your tasks. You should mention this in both your code and report. *Failure to do so will result in zero marks being awarded on related (sub)tasks*

Marking Criteria

- Quality of the source code, including the commenting of the code
- Technical correctness of the code, and structure.
- Reproducibility. How easy is it for another student to repeat your work based on your report and code?
- Quality of the report, including structure, clarity, and brevity

Submission Instructions

You should submit a PDF version of your report along with your code. The name of the PDF file should have the form "CS2020_Assessment1_< your Surname>_<your first name>_<Your Student ID>". For instance, "CS2020_Assessment1_Smith_John_4568985.pdf", where 4568985 is your student ID.

It is your responsibility to ensure that your code runs in Codio. If you do it on another device, and then move it to Codio, you need to ensure it runs there, and that it runs as expected.

You should submit your **code and the report using Codio**.

Your folder structure should look like follows:

- CA4_CS2020_yourName/

  - Report/

    - CS2020_Assessment1_Smith_John_4568985.pdf

  - assignment1/

    - <your code here> ….

Any questions pertaining to any aspects of this assessment, please address them to the course coordinator Milan Markovic (milan.markovic@abdn.ac.uk)