SOEN 6441- Advanced Programming Practices

Summer 2019

Coding Standards for Battleship Game Project

Submitted to: Prof. Nagi Basha

Team 2

Gurleen Sethi	40079565
Sandeep Singh	40043110
Vaibhav Malhotra	40079373
Karandeep Karandeep	40104845
Navjyot Kaur Minhas	40073551

1. Project Structure

- View/GUI files are kept in **view** folder. Depending on if its a GUI file or Console file, it is kept in **gui** or **console** folder respectively.
- ViewModels are kept in **viewmodels** folder. Subfolders can be created within this folder, if required.
- Custom exceptions are kept at a top level **exceptions** folder.
- Enums are kept in a top level **enums** folder.
- Files that are common throughout the application, such as constants are kept in a top level **common** folder.
- Tests are structured in exactly the same manner as application. It makes it easier to find where the test of a certain class is.

2. File Structure

- Package statement should be the first line in every java file.
- Import statements comes after the package statements. Import statements are grouped as follows:
    1. Project file imports and 3rd party library imports.
    2. Java's inbuilt classes.
    3. Static imports.

3. Formatting

- Indentation is 4 white spaces.
- Line breaking is added when the code line gets too long or to separate the code logically.
- Multiple conditions in an if statement need to be on separate lines or are grouped logically. No more that 4 conditions are allowed on the same line.

```
if (ship.getStartX() < 0
        || ship.getStartY() < 0
        || ship.getEndX() >= grid.getGridSize()
        || ship.getEndY() >= grid.getGridSize()) {
            ...
}
```

4. Naming

- All variables (except of static constants) and methods should strictly follow camel case.
- Short variables names such as **a, b, i, j** should not be used and are only reserved for loop variables wherever required. Variable names should be well thought of and should represent the data that they contain.
- Using **i**, **j** for loop variables should be prevented wherever possible. Instead more meaningful names should be selected, such as using **x** and **y** when iterating over a grid, **x** representing the x-coordinate and **y** representing the y-coordinate.

- All private static variables start with an **s**. (Exception: instances of logger).

  ```
  private static GameController sGameController;
  ```

- Method to get an instance of a singleton class will should be named **getInstance**.

  ```
  public static GameController getInstance() {
          ...
  }
  ```

- Static variables representing a constant should be upper case and delimited by underscore.

  ```
  public static final String INITIAL_USER_INPUT = "/initialUserInput";
  public static final String SHIP_PLACEMENT = "/shipPlacement";
  public static final String GAME_PLAY = "/gamePlay";
  ```

- Whenever creating a new Scene in **gui** folder, always append **Scene** at the end of class name. Eg: ShipPlacementScene, GamePlayScene.

5. Comments

- Every variable and functions should have a JavaDoc comment explaining its purpose, arguments, return type (if any) and exceptions thrown (if any).

```
/**
* Places a ship on the board with the provided Ship object.
*
* @param ship The ship to be placed
* @throws DirectionCoordinatesMismatchException If the starting and ending coordinates in ship
*                                               don't match with the provided direction.
* @throws CoordinatesOutOfBoundsException       If any coordinate of the ship is not on the plane.
* @throws InvalidShipPlacementException         If there is another ship already on one of the coordinates that
*                                               the ship is being placed on.
*/
public void placeShip(Ship ship) throws Exception {
```

- Methods declared in interfaces also require JavaDoc as this makes the user of the interface to understand what the methods do without having to look into implementation of it.
- Some variables don't require JavaDoc, such as an instance of a Logger.
- Regular comments should be used thoughtfully and not unnecessarily. Comments should answer the question of *Why you did that?* and not *What is happening*?, as that can be deduced from the code itself.
- Commented out code should be deleted before submitting a pull request to master.

6. Workflow Practices

- **master** branch is locked and you are not allowed to add commits directly to it. Always create a new branch when adding/removing anything.
- Every pull request needs at least one review from a peer before getting merged into master.

7. Best Practices

- Always use **checkNotNull** function from Google's guava library to check if an object is null or not.

- For any class providing some functionality to other classes, try to code it to an interface and provide that interface rather than providing a concrete implementation directly.
- When using RxJava, always export a Subject as an Observable to the client, unless absolutely required.
- When using RxJava, use PublishSubject over BehaviourSubject when not required to cache previous value.