SOEN 6441: Advanced Programming Practices

Winter 2019
Project – Risk Game
(Build 2)
Refactoring Document

Submitted By:
Team 30

NAME	ID
1. Gargi Sharma	40042837
2. Jaiganesh Varadharaju	40081862
3. Md Hasibul Huq	40087646
4.Narendran Krishnakumar	40089619
5. Zakiya Jafrin	40021416

Submitted To:
Amin Ranj Bar

Refactoring

Refactoring is the practice for restructuring the existing code or altering the internal structure without changing the external behaviour which improves the code quality.

1. **Refactoring Type**: **Rename method Name**
   Refactoring Target: Understandability
   Refactoring Class: MapController.java
   Description: Change the function name from checkMapFileExists to checkMapFileExistsOrNot.

   Build1

```java
/**
 * This method is used to check if the entered map file name is exists in directory or not.
 */
public void checkMapFileExists() {
    mapModel = new MapModel(); //----refresh----
    String mapPath = mapModel.getMapNameByUserInput();
    File tempFile = new File(mapPath);
    boolean exists = tempFile.exists();
    if (exists) {
        mapModel.readMapFile(mapPath);
        mapModel.printMapValidOrNot();
    } else {
        print.consoleErr("File not found!!!. Please enter the coreect name of map.");
    }
}
```

   Build2

```java
/**
 * This method is used to check if the entered map file name is exists in directory or not.
 */
public void checkMapFileExistsOrNot() {
    mapModel = new MapModel(); //----refresh----
    String mapPath = mapModel.getMapNameByUserInput();
    File tempFile = new File(mapPath);
    boolean exists = tempFile.exists();
    if (exists) {
        mapModel.readMapFile(mapPath);
        mapModel.printMapValidOrNot();
    } else {
        print.consoleErr("File not found!!!. Please enter the coreect name of map.");
    }
}
```

2. **Refactoring Type**: **Change Function**
   Refactoring Target: Readability and understandability
   Refactoring Class: MainController.java
   Description: Change the main function code and create a new function to increase the readability and understandability of the source code.

Build1:

```
     * This function also displays the error message to select valid user input.
     *
     * @param args the arguments
     */
    public static void main(String[] args) {

        MapController mapController = new MapController();
        PrintConsoleAndUserInput print = new PrintConsoleAndUserInput();
        GameController gameController = new GameController();

        int selectMainMenuOption = 0;
        boolean checkMapStatus = false;
        do {
            selectMainMenuOption = displaymainMenu();
            switch (selectMainMenuOption)
            {
            case 1:
                mapController.generateMap();
                break;
            case 2:
                gameController.initializeMap();
                break;
            case 3:
                print.consoleErr("Thanks for playing this Game.");
                System.exit(0);
            default :
                System.err.println("\n\t Error! Select option from the menu list (1 to 5):");
                break;
            }

        }
        while (selectMainMenuOption != 5);
        System.exit(0);
    }
```

Build2

```
public class MainController {

    /**
     * This is a main method to run the game.
     * This function is used to enter the user input and call the functions to create
     * and user can exit if he wants to exit the game.
     * This function also displays the error message to select valid user input.
     *
     * @param args the arguments
     */
    public static void main(String[] args) {
        startMenu();
    }
```

3. **Refactoring Type**: **Consolidate Duplicate Conditional Fragments**
Refactoring Target: Readability
Refactoring Class: MapController.java
Description: Refactor by moving the same code outside of the if else condition.

Build1

```
public void createAndSaveUserMap() {
    mapView.createJframe();
    mapView.saveButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            // TODO Auto-generated method stub
            StringBuffer mapContent = new StringBuffer(mapView.returnMapContent());
            String mapName = mapView.returnMapName();
            boolean checkMapIsCreated;
            checkMapIsCreated = mapModel.createValidateAndSaveMap(mapContent, mapName);

            if (checkMapIsCreated) {
                print.consoleOut(" ****Map has been created successfully in "+print.getMapDir()+ " as " +mapName+".map *
                mapView.closeFrameWindow();
            } else {
                print.consoleErr("**** Error!!!! Map has not been created successfully! ****");
                mapView.closeFrameWindow();
            }
        }
    });
}
```

Build2

```
/**
 * This method is used to create the user map and save it in directory.
 *
 */
public void createAndSaveUserMap() {
    mapModel = new MapModel();//----refresh----
    mapView.createJframe();
    mapView.saveButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            // TODO Auto-generated method stub
            StringBuffer mapContent = new StringBuffer(mapView.returnMapContent());
            String mapName = mapView.returnMapName();
            boolean checkMapIsCreated;
            checkMapIsCreated = mapModel.createValidateAndSaveMap(mapContent, mapName);

            if (checkMapIsCreated) {
                print.consoleOut(" ****Map has been created successfully in "+print.getMapDir()+ " as " +mapName+".map *
            } else {
                print.consoleErr("**** Error!!!! Map has not been created successfully! ****");
            }
            mapView.closeFrameWindow();
        }
    });
}
```

4. **Refactoring Type**: **Move method**
Refactoring Target: Understandability
Refactoring Class: Player.java, Game.java
Description: Move the calculationForNumberOfArmiesInReinforcement method from Game.java to Player.java

Build1

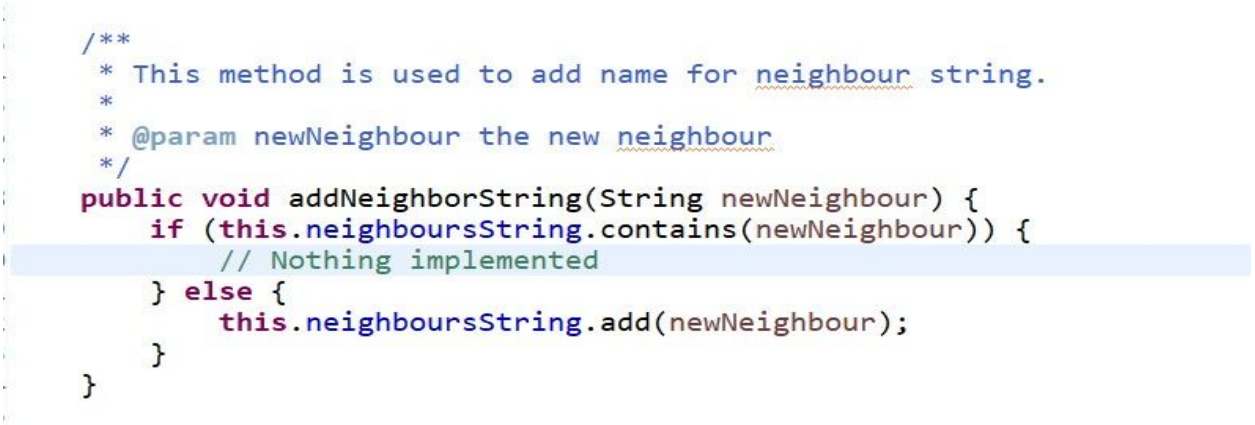
```
/**
 * This method calculates the corresponding reinforcement armies from a particular player from the number of countri
 * @param player Player
 * @return total number of armies in reinforcement
 */
public int calculationForNumberOfArmiesInReinforcement(Player player) {
    return (int) Math.floor(playerCountry.get(player).stream().count() / 3);
}
```

Build2

```
/**
 * This method calculates the corresponding reinforcement armies from a particular player from the number of countries owned by the layer.
 * @param playerCountry Player
 * @param continents Continents
 * @return total number of armies in reinforcement
 */
public int calculationForNumberOfArmiesInReinforcement(HashMap<Player, ArrayList<Country>> playerCountry,ArrayList<Continent> continents) {
    int countries_count = (int) Math.floor(playerCountry.get(this).stream().count() / 3);
    if (playerCountry.containsKey(this)) {
        ArrayList<Country> assignedCountries = playerCountry.get(this);

        List<Integer> assignedCountryIds = assignedCountries.stream().map(c -> c.getCountryId()).collect(Collectors.toList());

        for (Continent continent : continents) {
            List<Integer> continentCountryIds = continent.getCountryList().stream().map(c -> c.getCountryId()).collect(Collectors.toList());

            boolean hasPlayerAllCountries = assignedCountryIds.containsAll(continentCountryIds);

            if (hasPlayerAllCountries){
                countries_count += continent.getControlValue();
            }
        }
    }
    countries_count = countries_count+ initialArmiesafterExchange;
    return countries_count;
}
```

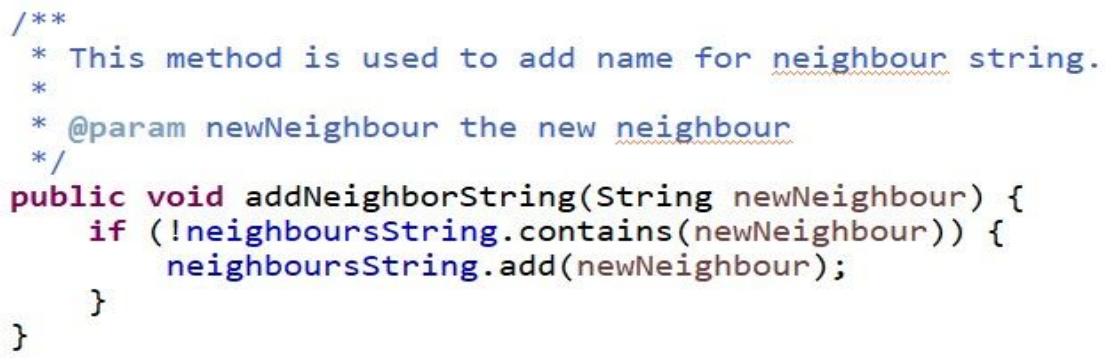
5. **Refactoring Type**: **Restructure Conditional Statement**
Refactoring Target: Readability
Refactoring Class: Country..java
Description: Change the Conditional statement and remove if condition.

Build1



```
/**
 * This method is used to add name for neighbour string.
 *
 * @param newNeighbour the new neighbour
 */
public void addNeighborString(String newNeighbour) {
    if (this.neighboursString.contains(newNeighbour)) {
        // Nothing implemented
    } else {
        this.neighboursString.add(newNeighbour);
    }
}
```

Build2



```
/**
 * This method is used to add name for neighbour string.
 *
 * @param newNeighbour the new neighbour
 */
public void addNeighborString(String newNeighbour) {
    if (!neighboursString.contains(newNeighbour)) {
        neighboursString.add(newNeighbour);
    }
}
```

6. **Refactoring Type**: **Change method Implementation**
Refactoring Target: Understandability
Refactoring Class: Game.java, Player.java
Description: Change the implementation of method and move it to player class as per the Build2 requirements.

Build1

```
//Functions called by addMoveArmyButtonListener() from GameController.
/**
 * This method checks whether the source and destination countries belongs to the player and moves the armies from sourc
 * @param source  source as string
 * @param destination destination countries as string
 * @param armies count of armies as int
 * @return true
 */
public boolean fortificationPhase(String source, String destination, int armies){
    Player player = getCurrentPlayer();

    Country sourceCountry = playerCountry.get(player).stream()
            .filter(c -> c.getCountryName().equalsIgnoreCase(source)).findAny().orElse(null);

    Country destinationCountry = playerCountry.get(player).stream()
            .filter(c -> c.getCountryName().equalsIgnoreCase(destination)).findAny().orElse(null);


    if (sourceCountry == null || destinationCountry == null) {
        print.consoleOut("Source or destination country is invalid!");
        return false;
    }

    if (armies == 0) {
        print.consoleOut("No armies to move");
        return true;
    }
    sourceCountry.decreaseArmyCount(armies);
    destinationCountry.increaseArmyCount(armies);
    this.setupNextPlayerTurn();
    setGamePhase(gamePhase.Reinforcement);
    reinforcementPhaseSetup();
    notifyObserverslocal(this);
    return true;
}
```

Build2

```
350     /**
351      * This method checks whether the source and destination countries belongs to the player and moves the armies from source to destination.
352      * @param sourceCountry  source as string
353      * @param destinationCountry destination countries as string
354      * @param armies count of armies as int
355      * @return true
356      */
357     public boolean fortificationPhase(Country sourceCountry, Country destinationCountry, int armies){
358
359         if(!checkFortificationCondition(sourceCountry, destinationCountry,armies)) {
360             return false;
361         }
362
363         sourceCountry.decreaseArmyCount(armies);
364         destinationCountry.increaseArmyCount(armies);
365
366         return true;
367     }
```

References:

From the professor slides(Chapter - Testing, MVC and refactoring)
https://moodle.concordia.ca/moodle/pluginfile.php/3524902/mod_resource/content/2/Lecture%205.pdf