SOEN 6441- Advanced Programming Practices

Winter 2019

Coding Convention Document

Recipient: Prof Amin Ranj Bar

Team 11

Gursimran Singh – 40080981
Karandeep Karandeep – 40104845
Namita Faujdar – 40105179
Karanbir Singh Pannu - 40047216
Shriyans Athalye - 40037637

This document will cover the coding conventions followed in the project. These conventions were followed according to the Coding convention for Java Programming Language by Oracle Corporation.

File Basics

1. File name

The source file name must consist of case-sensitive name of top-level class. The format should consist of .java extension in addition.

2. Java Source File

Each java file contains single public class or interface. The public class should be the first class or interface in the file. Java source file should have the following ordering:

- Beginning comments
- Package and Import statements
- Class and interface declarations

2.1 Beginning Comments

All source files should be with a c-style comment that lists the class name, version information and date. Following is the commenting structure which was followed in this project.

```
/**
* "GamePlayModel" model basically consists of an object that consist of the current map that the   game is being played upon
* and the players that are playing the game.
*/
```

2.2 Package and Import statements

The first non-commented line of most programs is a package statement. After package statement line, starts the import statement line. For example, from the project

```
package com.risk.gameplayrequirements;
import com.risk.model.ContinentModel;
import com.risk.model.CountryModel;
import javafx.scene.control.Alert;
```

2.3 Class and Interface Declarations

Classes declared should being with Javadoc. This allows to know which author has worked for which classes and related class functionality.

```
 /**
 * This class is for Human player type
 *
 * @author Namita
 */
```

3. Indentation

3.1 Line length

Four space should be used as unit of indentation. Avoid lines longer that 80 characters, since they are not handled well by many terminals and tools.

3.2 Wrapping Lines

When an expression will not fit on a single line, break is used

- After a comma
- Before an operator
- Align new line with beginning of expression at same level on previous line

```
else if ("Aggressive".equals(PlayerType)) {
        gamePlayObj.getMapRiskModelModObj().getPlayerTurn()
            .setStrategy(new PlayerAggresiveController(gamePlayObj));
        gamePlayObj.getMapRiskModelModObj().getPlayerTurn().executeReinforcement();
        gamePlayObj.getMapRiskModelModObj().getPlayerTurn().executeAttack();
```

4. Comments

Java programs can have two kinds of comments: implementation comments and documentation comments. Implementation comments are for different programming language whereas documentation comments also known as doc comments are for Java-only.

4.1 Block Comment

Block comments are used to provide descriptions of files, methods, data structures and algorithms which are defined at beginning of each file and each method. These can also be used in other places like within methods.

```
 /*
 * This class is for Human player type
 */
```

4.2 Single Line comment

Short comments can appear on a single line indented to the level of code that follows. In case a comment not being written in single line, it should follow the block comment.

```
/** current players playing the game*/
```

4.3 Trailing Comments

Very short comments can appear on same line as code it describes. These comments should be separated from the statements.

4.4 End of Line comments

The // comment delimiter can comment out a complete line or only a partial line. It shouldn't be used on consecutive multiple lines for text comments. Although this can be used in consecutive multiple lines for commenting out sections of code.

5. Declarations

5.1 Number per line

One declaration per line is recommended since it encourages commenting. No different types can be declared on same line.

```
File readFile = getReadFile();
BufferedReader bfr;
ArrayList<CountryModel> countryModelList;
```

5.2 Initialization

Initializing the local variable where it is declared is a good practice. If its not declared, no harm in doing so but the declared variables initial value depends on some computation occurring first.

5.3 Placement

Declarations should only be added at the beginning of blocks. It is not necessary to wait until the variables first use which confuses programmer and hampering code portability.

5.4 Class and Interface Declarations

There are some rules required to keep in mind when coding class and interfaces. These rules are:

- No space is allowed between method name and parenthesis.
- Open brackets appear at end of same line as declaration statement
- Closed brackets start a line by itself indented to match its corresponding opening statement until it is a null statement that appears immediately after the opened bracket.

```
if (e.getSource().equals(d_mapCreateCountryView.addButton))
    {
      if (d_mapCreateCountryView.countryValue.getText() != null
           && !d_mapCreateCountryView.countryValue.getText().equals(""))
      {
```

6. Statements

6.1 Simple Statements

Each line contains at most one statement. For example

```
if (d_mapCreateCountryView.countryValue.getText() != null
        && !d_mapCreateCountryView.countryValue.getText().equals(""))
```

6.2 Compound Statements

Compound statements contains list of statements enclosed in curly brackets. These enclosed statements should be indented one more level that the compound statement.

The opening brackets should be at end of line that begins the compound statement. Closing brace should begin a line and be indented to beginning of compound statement.

6.3 Return Statement

A return statement with a value should not use parentheses unless they make return value more obvious.

```
JOptionPane.showOptionDialog(null, "Please enter a valid input", "Invalid", JOptionPane.DEFAULT_OPTION,
JOptionPane.INFORMATION_MESSAGE, null, new Object[] {}, null);

        return;
```

6.4 If, if-else, if else-if else statements

If statements always use braces. The if-else class of statement should have the following form:

```
if (condition) {
    statements;
}
```

For example

```
        if (gamePlayObj.getPlayersList().size() > index) {
            gamePlayObj.getMapRiskModelModObj().setIndexOfPlayer(index);
            gamePlayObj.getPlayersList().get(index).callObservers();
        } else {
            index = 0;
            gamePlayObj.getMapRiskModelModObj().setIndexOfPlayer(index);
            gamePlayObj.getPlayersList().get(index).callObservers();
        }
```

6.5 While Statements

While statement should have following form:

```
while ( condition ) {
statements;
}
```

For example

```
while (bfr.readLine() != null) {
String bfr1 = bfr.readLine(); }
```

6.6 try-catch statements

A try catch should have following format:

```
try {
statements;
} catch ( ExceptionClass e ) {
statements;
}
```

For example

```
try {
    bfr = new BufferedReader(new FileReader(readFile));
}catch (IOException e) {
    e.printStackTrace();
}
```

7. White spaces

7.1 Blank Lines

Blank lines improve readability by setting off sections of code that are logically related.

At least two blank lines should always be used in following circumstances

- Between sections of source file
- Between class and interface definitions

There should be one blank line in the following circumstances

- Between methods
- Between local variables in a method and its first statement
- Before block or single line comment
- Between logical sections inside a method

7.2 Blank Spaces

Blank spaces are necessary for following circumstances:

- A keyboard followed by parenthesis should have space in between
- Blank lines should appear after commas in argument lists
- For statement having expressions should be blank space separated

```
for (int i = 0; i < listOfNeighbouringCountries.size(); i++)
```

8. Naming Conventions

Naming conventions make program more understandable. This makes it easier to read. These names also give information about function of the identifier whether it is constant, package or class which can be helpful for code understanding.

Package name: Prefix of package name is always written in lower case ASCII letters and should be one of the top-level domain names.

```
com.risk.controller
```

Classes Name: Class names should be nouns in mixed case with the first letter of each internal word capitalized.

```
Class MainGame
```

Interface Name: Interface name should be capitalized like class names.

```
StrategyInterface
```

Methods Name: Method names should have verb, be in mixed case with first letter lowercase of each internal word capitalized.

```
exitGame();
```

Variable Name: Variable names should be short but meaningful. The name should indicate casual observer its intent of use.

```
int i;
```

Constant: Name of variables in declared class constants and ANSI constants should be all uppercase with words separated by underscore.

```
static final int MAX_WIDTH = 400;
```