DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Concordia University

Montreal, Quebec, Canada

Advanced Programming Practices
SOEN 6441

(Risk Game)
Project Build Version -3

Professor - Amin Ranj Bar

CODING CONVENTIONS

Team-35

Name	**Student ID**
Gursharan Deep	40039988
Harmanpreet Singh	40059358
Raghav Sharda	40053703
Yaswanth Choudary Narra	40087595
Saman Soltani	40093581

Introduction

This document reflects the Java Language Coding Conventions of the second build of the Risk Game project.

1. **FileNames :**

- The file names used reflect the functionality of that particular file.
- For **example**, GameDriver class handles the drivers of the game.
- Map files used have '.bmp' extension and data of a map is stored in a '.map' file.

File Type	**Suffix**
Java Source	.java
Map	.bmp
Map Data	.map
View	.html

2. File Organization :

- Different sections in a file are separated by blank lines and an optional comment lines for identifying each section.
- File with more than 2000 lines are difficult to work on and should be avoided.
- Basic idea of File Organization used is as follows

```
package java.blah;

import java.somelib.*;

/**
 *
   Class description goes here.
 * @author Name
 * @version some version
*/
public class ClassName extends SomeClass {


    /** classVar1 documentation comment
    */ public static int classVar1;


    private static Object classVar2;
```

3.Naming Conventions:

- Classes have been named as per their functionality and the architecture adopted under the package.
- All class names start with upper case letters.
- Classes in ‘controllers’ package are named with CamelCase for separation of words .(ex: TheMainController)
- Classes in other packages (Model, View ,game.messages) are also named by using case change for word separation.(ex: ReadMap, ControlsConsole etc,)
- Variables are named using Camel case format.

Identifier	Naming rules	Examples
Classes	Change case.	GameDriver,GameTurnManager
Methods	start lower case letters	getCountries()
Variables	short and meaningful Common alphabet for temporary variables are i, j, k, m and n.	countryList, countryName

4.Comments :

- Commenting is done as per conventions of JavaDoc.
- Description is mentioned at the beginning of every class or method.
- @return for return value of a method.
- @author for the name of the programmer.
- @version for the version of the build.
- @param for parameters used in methods.
- @see for linking an API documentation.

5.Indentation:

- One tab (4 spaces) is used as a unit of Indentation.
- Each line should start with a unit tab space before for indentation.
- sample :

```
 public NodeOfCountry (String name , ArrayList<NodeOfCountry>neighbour , int[] coordi            -
nate)
 {
             this.CountryName = name;
             this.Neighbours = neighbour;
             this.Coordinate = coordinate;
             this.PlayerCountry = null;
             this.Armies = 0;
            }
```

6.Declarations:

- One declaration per line is recommended as follows.

```
int sample;   //sample comment
int sample1; //sample1 comment
```

- More than one declaration per line is not ideal.

```
int sample 1; int sample 2;     //this is not ideal
```

- Declarations must be put at the beginning of a block.
- Do not put different types on same line.
- Declaring variables at their first use in not recommended as it can confuse the programmer.
- Try to initialise variables where they are declared.
- Indexes for loops can be declared on the same line for example we can consider a for loop.

```
for (int i = 0; i < maxLoops; i++) { ...
} </blockquote>
```

```
void myMethod() {
    int int1 = 0;         // beginning of method block

    if (condition) {
        int int2 = 0;     // beginning of "if" block
        ...
    }
}
```

Class Declarations:

- Open brace appears at the end of the same line as the declaration statement.

```
class Sample extends Object {
    int ivar1;
    int ivar2;

    Sample(int i, int j) {
        ivar1 = i;
        ivar2 = j;
    }

    int emptyMethod() {}

    ...
}
```

- Closed brace appears on its separate own line and is intended to match the corresponding opening statement.
- Methods are separated by a blank line or an optional comment.

7.Statements :

- Each line should contain at most one statement.

```
argv++;             // Correct
argc--;             // Correct
 argv++; argc--;    // NOT IDEAL
```

- Return statements:
  - Return statements should be immediately followed by a return value.
  - A return statement with a value should not use parenthesis unless they return value more obvious in some way .
  - Example:

```
return;

return myDisk.size();
```

```
if (condition) {
    statements;
}

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

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

- If, If-else, if-else-if-else Statements:
  - These statements use the following format.
- for Statements:
  - Avoid the complexity of using more than 3 variables in the initialisation of a for loop.
  - Nesting of for loops should be according to the organisation format.
  - for statement should be declared as follows.

```
for (initialization; condition; update) {
    statements;
}
```

- while statements:
  - A while statement should be written in the following format.

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

- try-catch statements:
  - try-catch statements are as follows.

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

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

- try-catch statements can also be followed by finally which executes regardless of whether or not try block has completed successfully.

- 8. Programming Practices:

  - Do not make any *instance* or class variable *public* without a good reason.
  - Avoid assigning same values for different variables in a same line.
  - Try to use classname instead of using an object to access a class variable or a method.

```
classMethod();              //OK
AClass.classMethod();       //OK
anObject.classMethod();     //AVOID!
```

- Numerical constants should not be coded directly, except for -1,0 and 1 which can appear in a *for* loop as counter values.
- Do not use assignment operator a place where it can be easily confused with equality operator.

```
if (c++ = d++) {        // AVOID! (Java disallows)
    ...
}
</blockquote>
```

```
if ((c++ = d++) != 0) {  //OK
    ...
}
```

- Do not use embedded assignments in an attempt to improve run-time performance.

```
d = (a = b + c) + r;        // AVOID!
</blockquote>

}
```

```
a = b + c;     //OK
d = a + r;     //OK
```

9.Miscellaneous practices :

- Returning values :
  - Try to make the structure of the program match the intent.

```
if (
             booleanExpression) {
    return true;
} else {
    return false;
}
```

should instead be written as

```
return
             booleanExpression;
```

- Operator precedence :
  - Even if operator precedence seems clear to you, it might not be for the others , you shouldn't assume other programmers know precedence clearly.
  - Try to use operators as follows.

```
if (a == b && c == d)      // AVOID!
if ((a == b) && (c == d))  // RIGHT
```

These are the coding conventions used for the third project build of the Risk Game.