Project Coding Standard

The intention of this guide is to provide a set of rules that pertain to how code is to be written.In general, much of our style and conventions mirror the Code Conventions for the Java Programming Language , Google's Java Style Guide, and slides from Dr. Joey Paquet .

Code layout

- Aims at increasing code readability

1. Indentation
   - Code is indented according to its nesting level.

2. Format statements
   - Use statement blocks, minimize code length by appending the open curly brace to the statement that precedes it.

   Example for 1 & 2:

```
public void exchangeCardforArmy() {
    int[] cardTypeNumber = cardTypeNumber();
    for (int counter=0; counter<3; counter++) {
        if (cardTypeNumber[counter]>=3)
        {
            for (int i=0; i<3; i++)
                removeCard(counter);
            return;
        }
    }
    removeCard(0);removeCard(1);removeCard(2);
}
```

3. Blank line
   - For readability purpose, blank lines are added to separate code components/sections

   Example:

```
public int getNumberOfDots() {
    return numberOfDots;
}

public Dice() {
    // to do roll dice
}
```

Naming conventions

- Aims at increasing code understandability.

1. The length of a name depends on its scope.

2. Names that are used pervasively in a program, such as global constants, are long descriptive names.
   Example: showBasicView

3. A name that has a small scope, such as the index variable of a one-line for statement, is short.
   Example: owner;

4. Constants are named with all upper case letters and may include underscores.
   Example:FILE_HEAD_LINE_NUMBER;

5. User-defined type names or class names start with a capital letter.

6. Names that contain multiple words are either separated by a delimiter, such as underscore, or by using an uppercase letter at the beginning of each new word.
   Example for 4 & 5: Fortification View

Commenting conventions

- Aims at increasing code understandability.

1. Comments are used to improve code understandability.

2. Comments do not provide information that can be easily inferred from the code

3. A comment of some kind at used in the following places:
   1) At the beginning of each file there is a comment explaining the purpose of this file in the project.
   2) Each class declaration is preceded by a comment explaining what the class is for.

```
/**
 * The Class GameStartController. It controls the start action of the game.
 * After the button is clicked, A window will show up to ask the player to select a map file and choose the number of players.
 * @author Bingyang Yu
 * @version 1.0
 */
public class GameStartController implements ActionListener {
```

3) Each method or function has comments explaining what it does and how it works, as well as what is the purpose of its parameters.

```
/**
 * This method is triggered by "NEW GAME" button.
 * It opens up a new dialog and allows player to choose map file.
 * The file is passed to PlayerSetupView
 */
@Override
public void actionPerformed(ActionEvent e) {
```

4) All variable declarations, most importantly class data members, are appended with a comment describing its role, unless its name makes it obvious.

```
private List<Player> playerList;
private List<Country> countryList;
private Country clickedCountry;
```

5) In the place where elaborated algorithm is used in a long function, inline comments are used to highlight and explain all the important steps of the algorithm.

```
for(int i = 0; i < countryList.size(); i++)
{
    Country c = countryList.get(i);              // loop for get each country of the map
    Player p = playerList.get(i%num);            // find the corresponding player by the order of the player
    p.getCountryList().add(c);                   // assign country to each player
    c.setOwner(p);
}
```

6) All the preceding can be done using documentation tools such as Javadoc/Doxygen.