Refactor List Document

Potential refactoring targets:

1. Refactor the map validate method, so that the program will automatically perform map verification when the user enters the save map command.

2. Refactor the GameDriver class to make user have the option to choice single mode or tournament mode

3. Refactor the method of exception handling.

4. Refactor the Inheritance structure of **AdvanceOder class**

5. Refactor the main menu in mainController class

6. Refactor the ***createOrder()*** method to apply the strategy pattern.

7. Refactor the TextFileReader() method to apply the adapter pattern.

8. Combine the *setup player* method and *assign countries* method in one phase

9. Refactor the inheritance relationship in each state to achieve the purpose of reducing the degree of coupling.

10. Create an interface for the function card, so that the **GameEngine** object will call the method in order creation based on the interface.

11. Display the data changes in the package **Model** by refactoring the method of classes in package **View** to achieve MVC pattern.

12. Create a map list and show up to the user, allowing the user to select a map from the list to play.

13. Automatically display the map after the end of each round so that the user can see the current map status

14. refactor the game phase, ask users if they want to load the saved game progress before they enter *play* command.

15. refactor the game phase, ask users if they want to load the map they just created or edited.

Actual refactoring targets.

Refactor about strategy pattern:

1. To make users able to choose various strategies for AI players, we change the ***createOrder()*** method in player object to ***createOrder()*** method in playerStrategy object. Now the Ai player will based on the strategy that selected by user y to deploy the order.

```
    /**
     * following the command to issue order and add the order to order list.
     *
     * @return true if an order is created
     */
    public boolean issueOrder() {
-       Order l_Order = createOrder();
+       Order l_Order = this.d_Strategy.createOrder();
        if (l_Order != null) {
            addOrderToList(l_Order);
            return true;
@@ -218,37 +231,6 @@ Player extends Observable {
        return false;
```

Refactor about Adapter pattern:

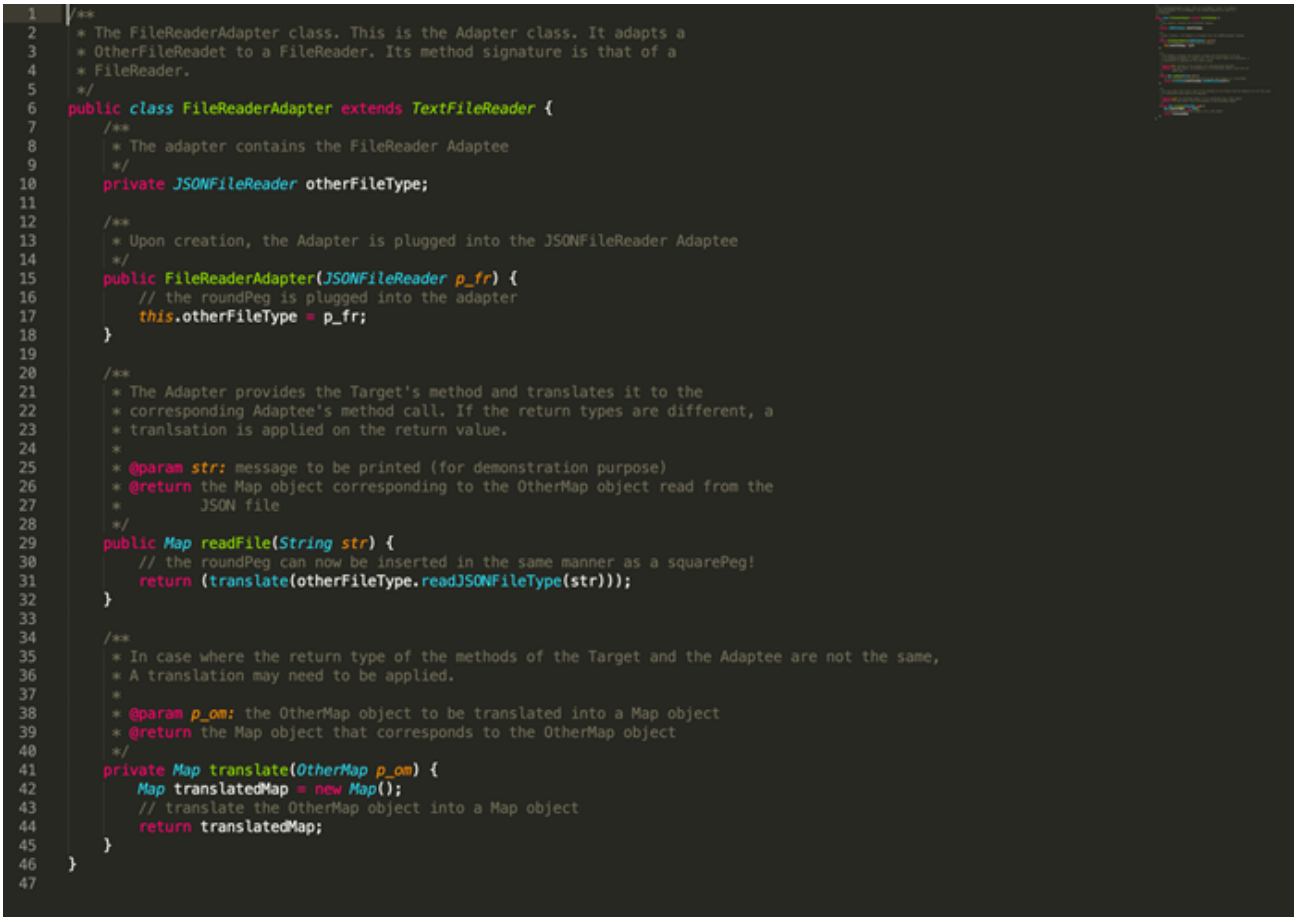
2. In the previous version, users can only load or edit the maps with the “domination” type. Now we have changed the previous structure of the map class to apply the adapter pattern. So That the adapter can convert the map in *conquest* type into the *domination* type for loading and editing.

```
1   map.map
2
3   [continents]
4   Asia 3
5   Europe 4
6
7   [countries]
8   1 China 1
9   2 Korean 1
10  3 Russia 2
11  4 Finland 2
12
13  [borders]
14  1 2 3
15  2 1 3
16  3 1 2 4
17  4 3
```

domination map file

```
1   conquest.map
2
3   [Continents]
4   Asia=3
5   Europe=4
6
7   [Territories]
8   China,Asia,Korean,Russia
9   Korean,Asia,China,Russia
10
11  Russia,Europe,China,Korean,Finland
12  Finland,Europe,RUssia
13
```

conquest map file



```java
/**
 * The FileReaderAdapter class. This is the Adapter class. It adapts a
 * OtherFileReadet to a FileReader. Its method signature is that of a
 * FileReader.
 */
public class FileReaderAdapter extends TextFileReader {
    /**
     * The adapter contains the FileReader Adaptee
     */
    private JSONFileReader otherFileType;

    /**
     * Upon creation, the Adapter is plugged into the JSONFileReader Adaptee
     */
    public FileReaderAdapter(JSONFileReader p_fr) {
        // the roundPeg is plugged into the adapter
        this.otherFileType = p_fr;
    }

    /**
     * The Adapter provides the Target's method and translates it to the
     * corresponding Adaptee's method call. If the return types are different, a
     * tranlsation is applied on the return value.
     *
     * @param str: message to be printed (for demonstration purpose)
     * @return the Map object corresponding to the OtherMap object read from the
     *         JSON file
     */
    public Map readFile(String str) {
        // the roundPeg can now be inserted in the same manner as a squarePeg!
        return (translate(otherFileType.readJSONFileType(str)));
    }

    /**
     * In case where the return type of the methods of the Target and the Adaptee are not the same,
     * A translation may need to be applied.
     *
     * @param p_om: the OtherMap object to be translated into a Map object
     * @return the Map object that corresponds to the OtherMap object
     */
    private Map translate(OtherMap p_om) {
        Map translatedMap = new Map();
        // translate the OtherMap object into a Map object
        return translatedMap;
    }
}
```

common refactor:

3. We **super** the parent class **Order** and transfer the value of each methods to the defined variables in **AdvanceOder** class, instead of using **this()** to call another constructor in the same class

```
-60,7 +62,12 @@ public class AdvanceOrder extends Order {
   * @param p_OrderInfo the order info
   */
  public AdvanceOrder(OrderInfo p_OrderInfo) {
      this(p_OrderInfo.getInitiator(), p_OrderInfo.getDeparture(), p_OrderInfo.getDestination(), p_OrderInfo.getNumberOfArmy());
      super();
      setType("Advance");
      d_Player = p_OrderInfo.getInitiator();
      d_AttackCountry = p_OrderInfo.getDeparture();
      d_DefendCountry = p_OrderInfo.getDestination();
      d_NumberOfArmies = p_OrderInfo.getNumberOfArmy();
      this.setOrderInfo(p_OrderInfo);
  }
```

4. We made some rectifications to the main menu of the previous version of **mainController** class. In single game mode, the user needs to enter the **start** command to start the game after the play setup phase, and the user will only need to enter command for his/her own turns durning the issue and execute phase.

```
@@ -54,9 +54,10 @@ public class SingleGameController extends MainPlayController{
54                  System.out.println("| 4.  Play:Startup:LoadMap       : loadmap           |");
55                  System.out.println("| 5.  Play:Startup:AddPlayer     : addPlayer         |");
56                  System.out.println("| 6.  Play:Startup:AssignCountry : assign            |");
-                   System.out.println("| 7.  Play:MainPlay:IssueOrder   : issue            |");
-                   System.out.println("| 8.  Play:MainPlay:ExecuteOrder : execute          |");
-                   System.out.println("| 9.  Any                        : end               |");
57 +                System.out.println("| 7.  Play:MainPlay:start to play: start            |");
58 +                System.out.println("| 8.  Play:MainPlay:IssueOrder   : issue (user)     |");
59 +                System.out.println("| 9.  Play:MainPlay:ExecuteOrder : execute  (user)  |");
60 +                System.out.println("| 10. Any                        : end              |");
```

5. We adopted a try-catch method to handle the exception, if the target map files are not found, then we track the map file from the top of stack.

```
28      public void loadMap() {
29
-           if (!d_MapInputDoneFlag) {
-               stringMapPathInputProcess();
30 +           try {
31 +               d_Ml.d_MapFile = new File(d_Ml.d_GameEngineController.getMapFilePath());
32 +           } catch (IOException e) {
33 +               e.printStackTrace();
34          }
```

6. refactor the structure of GameDriver class to make users have the option to choose single mode or tournament mode before they start to play.

```
4  -        // new game
5  -        GameDriver l_GameDriver = new GameDriver();
37 +        Scanner l_Scanner = new Scanner(System.in);
38 +
39 +        System.out.println("Welcome to WAR-ZONE game! ");
40 +
41 +        GameDriver l_GameDriver;
42 +
43 +        while (true) {
44 +            System.out.println("===== game mode selection(single,tournament)=====");
45 +            String l_Mode=l_Scanner.nextLine();
46 +            switch (l_Mode){
47 +                case "single":
48 +                    // new single game
49 +                    System.out.println(" *****single game mode*****");
50 +                    l_GameDriver = new GameDriver(new SingleGameController());
51 +                    l_GameDriver.d_MainPlayController.Start();
52 +                    break;
53 +                case "tournament":
54 +                    // new tournament game
55 +                    System.out.println("*****tournament game mode*****");
56 +                    System.out.println("not implement yet");
57 +                    continue;
58 +                default:
59 +                    System.out.println("Error: mode not found");
60 +                    continue;
61 +            }
62 +            break;
63 +        }
64 +
65 +
```

Unit Test

All related union tests have passed: