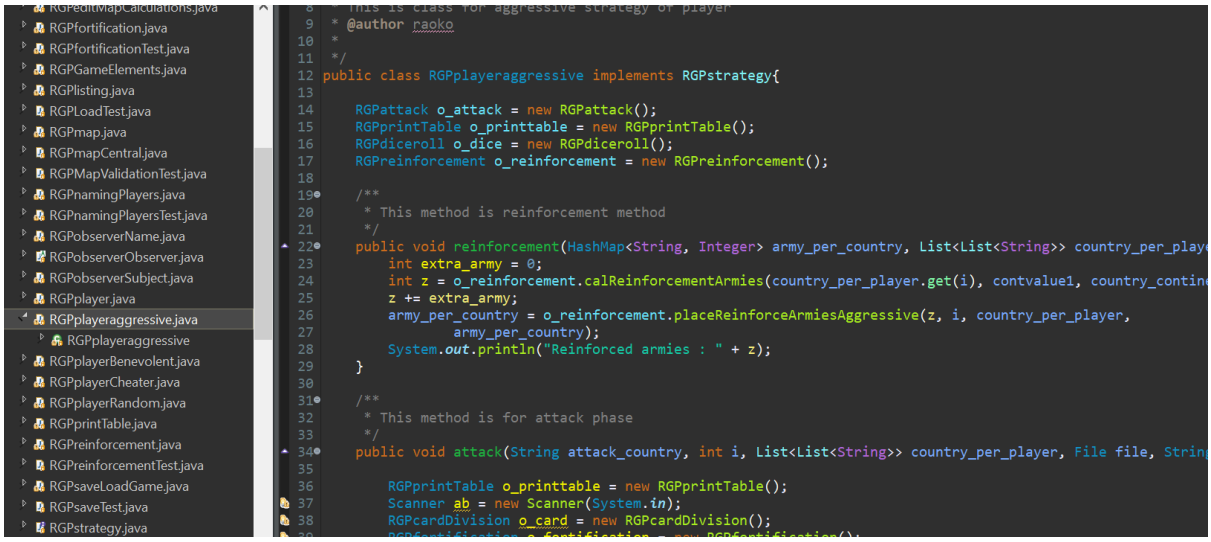
Refactoring Code

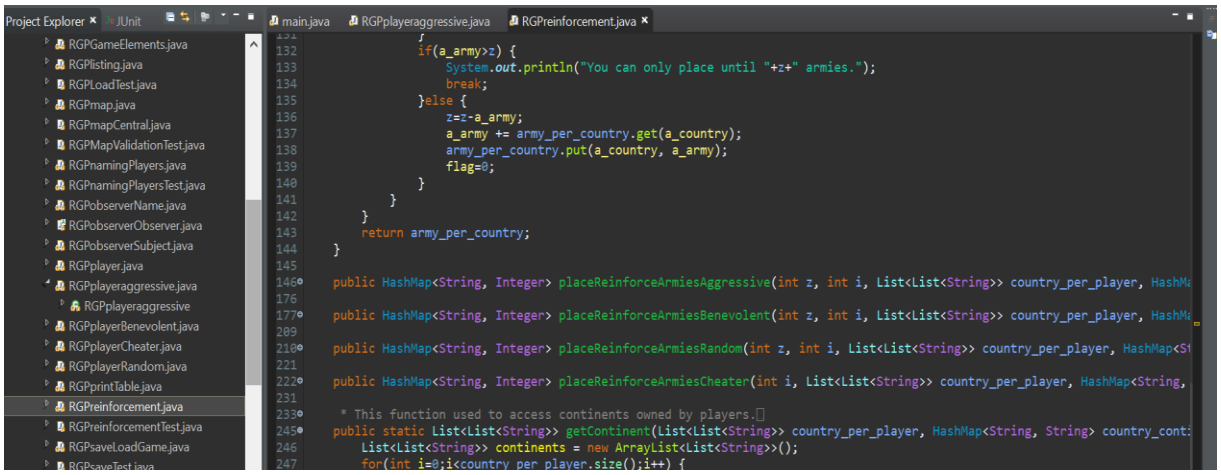
- **Change in Player Class:**
  - We added new functionality in player class to ease our strategy as we created new classes such as RGPplayerAggressive, RGPplayerBenevolent, RGPplayerRandom, RGPplayerCheater.
  - This strategy player classes implementes player class and are changed as required by the mode they are playing.

```
import java.io.File;

public class RGPplayer {
    public int reinforcement(HashMap<String, Integer> army_per_country, List<List<String>> country_per_player, int j,int total


    public void attack(String attack_country, int i, List<List<String>> country_per_player, File file, String defend_country,A

    public void fortify(File file, int i,List<List<String>> country_per_player,HashMap<String, Integer> army_per_country) thro

}
```

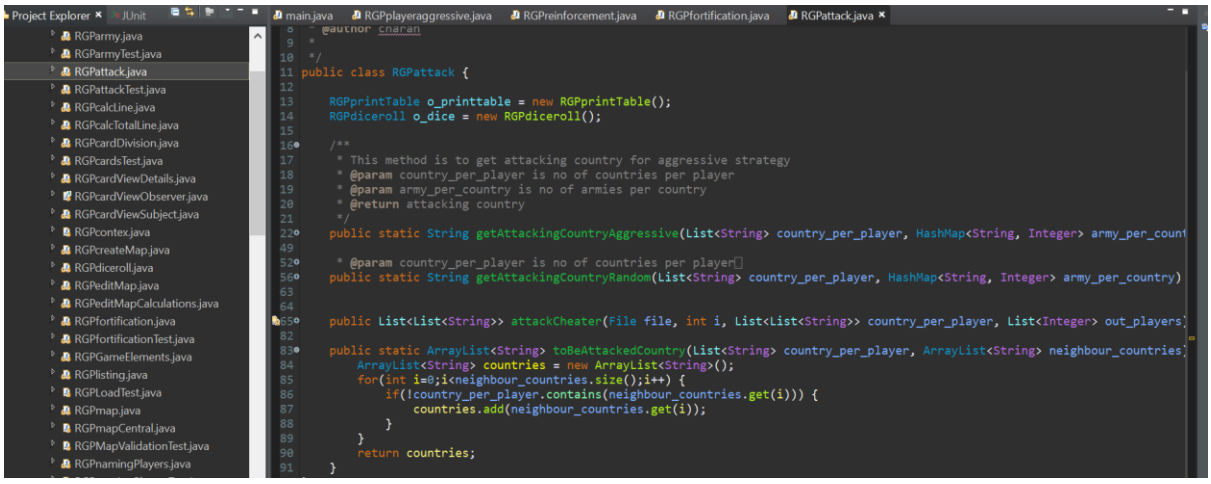


- **Changes in Phases:**
  - Changes were made in Reinforcement Phase, Attack Phase, Fortification Phase as to play using new strategies.

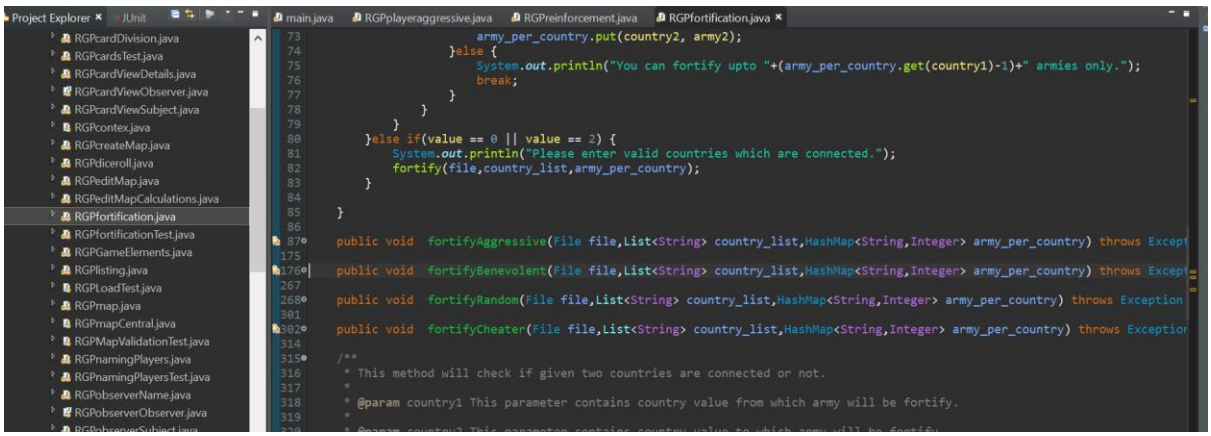
Reinforcement:



Attack:



Fortify:



- **Add brackets to single line loops.**

- **Creating new methods.**
  - In some files certain method contained too much of code so a bunch of code were extracted from the complex method to create a new method for better understanding of code as well as to reduce confusion.

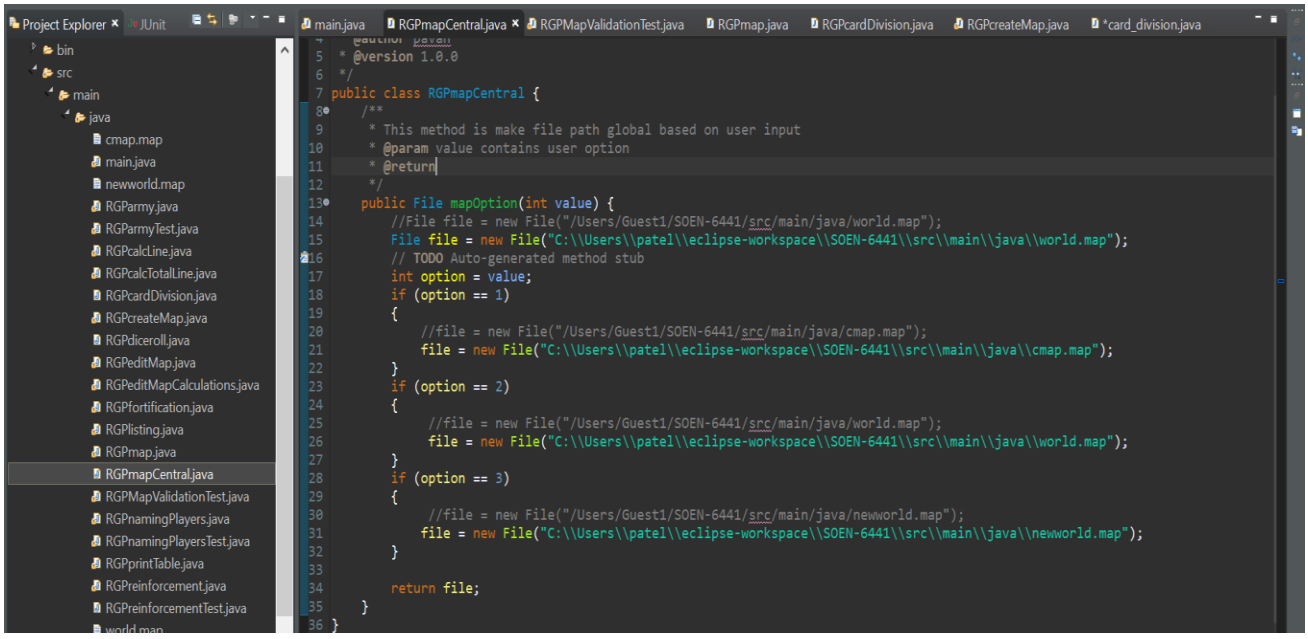
Before:

```
    ArrayList<String> new_country = new ArrayList<String>();
    for(int i=0;i<noc;i++)
        {
        country_names.add("country "+i);
        hm = hm1(country_names);
        }
    System.out.println("Elements in 1st ArrayList "+country_names);
    new_country.add(country_names.remove(2));
    System.out.println("Elements in 2nd Arraylist "+new_country);
    System.out.println("Elements in 1st ArrayList after deletion "+country_names);
    System.out.println("Elements in hashmap "+hm);

    HashMap<String,Integer> a1 =new HashMap<String,Integer>();
    HashMap<Object,Integer> b1 =new HashMap<Object,Integer>();
    HashMap<String,Integer> c1 =new HashMap<String,Integer>();

 if(a1.containsValue(3) && b1.containsValue(3) && c1.containsValue(3))
 {
     System.out.println("Reinforcement +8");
 }
 else if(a1.containsValue(2) && b1.containsValue(2) && c1.containsValue(2))
 {
     System.out.println("Reinforcement +6");
 }
 else if(a1.containsValue(1) && b1.containsValue(1) && c1.containsValue(1))
 {
     System.out.println("Reinforcement +4");
 }
 else if(a1.containsValue(1) && b1.containsValue(2) && c1.containsValue(3)||
         a1.containsValue(1) && b1.containsValue(3) && c1.containsValue(2)||
         a1.containsValue(2) && b1.containsValue(1) && c1.containsValue(3)||
```

After:

```
public static void check_cards_deal(HashMap<String,Integer> a, HashMap<Object,Integer> b, HashMap<String,Integer> c) {

public static void add_sub(ArrayList<String> a, ArrayList<String> b){

public static void hash_add_sub(HashMap<String,Integer> a, HashMap<Object,Integer> b){

public static Object getting_key(HashMap<String,Integer> a, Object value){
```

- **Map modules centralized.**
  - Previously several class were to be changed if there was any updation in map file.
  - Now all the map modules are centralized in a Map Central Method which will occupy less space and time, as no other file would be updated rather than Map Central.



- **Consolidate Duplicate Conditional Fragments:**
  - Previously, same fragment of code was in all branches of a conditional expression, so by refactoring them by moving it outside of the expression and adding the fragment after all the conditions were completed.

- **Adding brackets.**
  - In many methods the control statement which have one line of command were not having brackets.
  - This created confusion so brackets were added to reduce confusion.

Before:

```
public File mapOption(int value) {
    //File file = new File("/Users/Guest1/SOEN-6441/src/main/java/world.map");
    File file = new File("C:\\Users\\patel\\eclipse-workspace\\SOEN-6441\\src\\main\\java\\world.map");
    // TODO Auto-generated method stub
    int option = value;
    if (option == 1)
        file = new File("C:\\Users\\patel\\eclipse-workspace\\SOEN-6441\\src\\main\\java\\cmap.map");
    if (option == 2)
        file = new File("C:\\Users\\patel\\eclipse-workspace\\SOEN-6441\\src\\main\\java\\world.map");
    if (option == 3)
        file = new File("C:\\Users\\patel\\eclipse-workspace\\SOEN-6441\\src\\main\\java\\newworld.map");
```

After:

```
if (option == 1)
{
    //file = new File("/Users/Guest1/SOEN-6441/src/main/java/cmap.map");
    file = new File("C:\\Users\\patel\\eclipse-workspace\\SOEN-6441\\src\\main\\java\\cmap.map");
}
if (option == 2)
{
    //file = new File("/Users/Guest1/SOEN-6441/src/main/java/world.map");
    file = new File("C:\\Users\\patel\\eclipse-workspace\\SOEN-6441\\src\\main\\java\\world.map");
}
if (option == 3)
{
    //file = new File("/Users/Guest1/SOEN-6441/src/main/java/newworld.map");
    file = new File("C:\\Users\\patel\\eclipse-workspace\\SOEN-6441\\src\\main\\java\\newworld.map");
}
```

- **Restructuring of code.**
  - There were many “if..else” statements, they created ambiguity in code. So some were replaced by other methods and some were replaced by “switch()..” statement.
  - Refactoring by combining them into a single conditional expression and extracting it.

- **Remove unnecessary comments.**
    - Comments were written to describe different methods or variables. Now variables are changed to appropriate “self-explanatory” name so unnecessary comments are now deleted.

Before:

```
public static void main(String args[]){
    int noc=10;                                                    //no of countries;
    HashMap<String,Integer> hm = new HashMap<String,Integer>();
    ArrayList<String> c1 = new ArrayList<String>();                // country 1
    ArrayList<String> c2 = new ArrayList<String>();                // country 2
    for(int i=0;i<noc;i++){                                        // if condition for adding country to Arraylist
        c1.add("country "+i);
        hm = hm1(c1);
        }
```

After:

```
public static void main(String args[]){
    int no_countries = 42;
    HashMap<String,Integer> hm = new HashMap<String,Integer>();
    ArrayList<String> country_1 = new ArrayList<String>();
    ArrayList<String> country_2 = new ArrayList<String>();
    for(int i=0;i<no_countries;i++){
        country_1.add("country "+i);
        hm = hm1(country_1);
        }
```

- **Renaming of Necessary Variables and Methods.**
    - Adding new functionalities to the project tends to add new methods and class, because of that some methods and classes were renamed, so as to remove unambiguity and create a clear picture of each and every method.

    Before: We had a single method doing map validation, creating and editing.

    After: Separate classes for each method were created, so as to avoid confusion and were easy to maintain.