

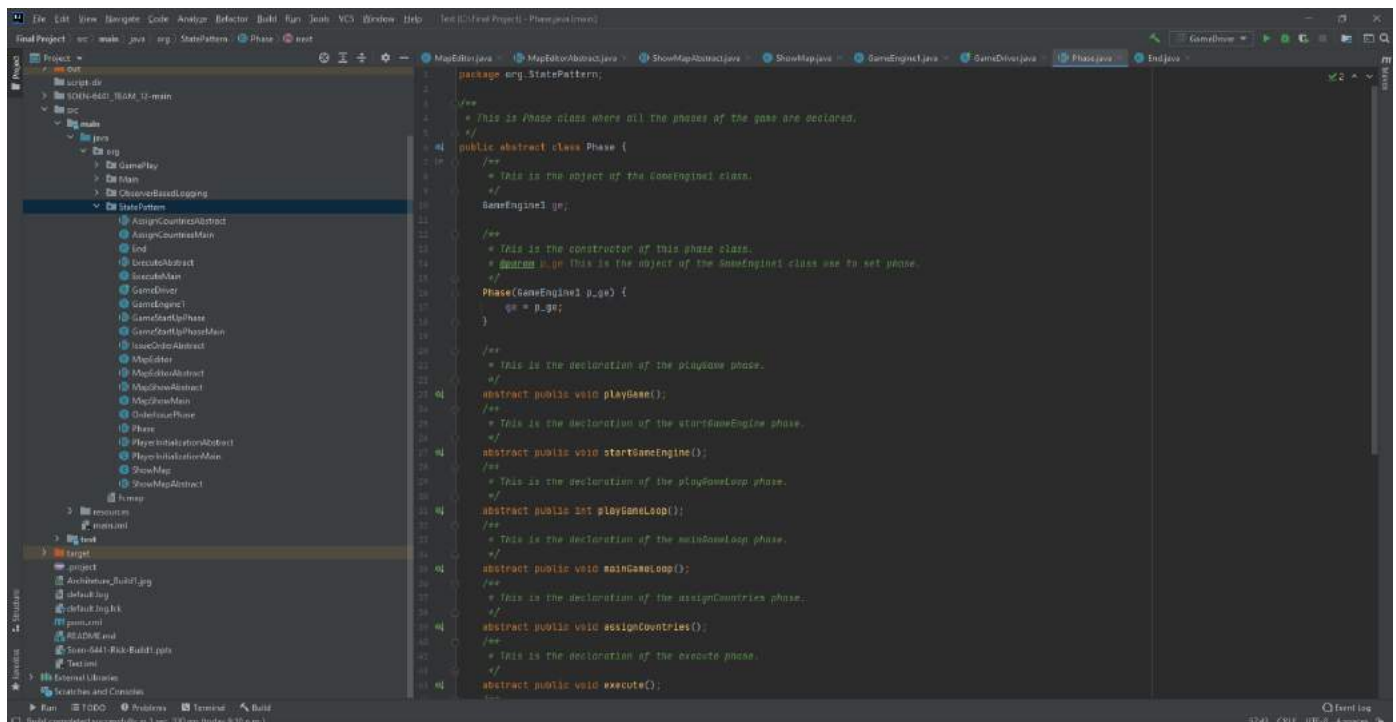
Refactoring Code

1. Introduced State Pattern

- As this was the requirement for the Build-2, so we have introduced it in our code.
- Using state pattern, we can get to know in which phase our game is running.
- We have around 10 different phases of our game.

ASSOCIATED TEST CASES.

1. **public void gameloop()** in playGameTest.java
2. **public void gameloop1()** in playGameTest.java
3. **public void checkStartUpPhase()** in StartUpPhase.java
4. **public void EndGametest()** in EndOfGame.java
5. **public void check()** in checkPhaseChange.java
6. **public void checkNext()** in checkPhaseChange.java

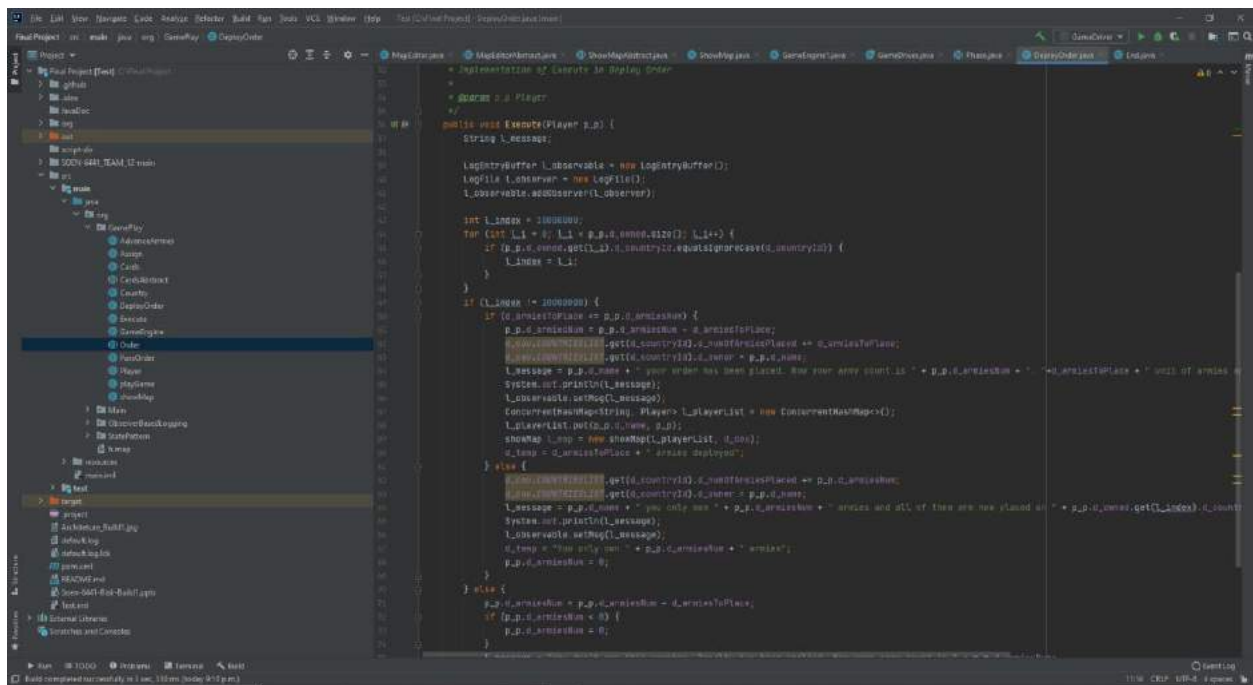


2. Introduced Command Pattern

- This was also the requirement for the Build-2, so we have introduced Command Pattern in our code.
- Using this initially all the orders will be taken from all the players in a go and then will all the orders will be executed in the same manner as they will be taken.

ASSOCIATED TEST CASES.

1. **public void gameloop()** in playGameTest.java
2. **public void gameloop1()** in playGameTest.java
3. **public void checkStartUpPhase()** in StartUpPhase.java
4. **public void EndGametest()** in EndOfGame.java



3. Introduced Switch Case In Cards Class

- In cards class before switch case was not present, it will visit every if condition which matches the card with the and executes the corresponding if loop.
- But after introducing the switch case once the proper card case is found it will execute it and exit out of switch case, now it is not visiting every condition.

ASSOCIATED TEST CASES

1.All test cases in **BombTest.java**

2.All test cases in **AirliftTest.java**

3.All test cases in **BlockadeTest.java**

4.All test cases in **NegotiateTest.java**

Before

```
@Override
void Execute(Player p_p) {

    String decide = "NO";
    if(card.equalsIgnoreCase("bomb")){
        if(p_p.cards.contains("BOMB")) {
            if (!p_p.negotiate.contains(d_country.COUNTRIESLIST.get(CountryID).d_owner)) {
                for (int i = 0; i <= p_p.d_owned.size(); i++) {
                    if (d_country.COUNTRIESLIST.get(p_p.d_owned.get(i).d_countryId).d_neighbours.contains(CountryID)) {
                        decide = "YES";
                        break;
                    }
                }
            }
            else {
                System.out.println("Cannot Bomb this country as you have negotiated with its owner");
            }
        }
    }

    if(decide.equalsIgnoreCase("YES")){
        d_country.COUNTRIESLIST.get(CountryID).d_numOfArmiesPlaced = (int) Math.floor(d_country.COUNTRIESLIST.get(CountryID).d_numOfArmiesPlaced / 2);
    }

}

if (card.equalsIgnoreCase("blockade")) {
    if(p_p.cards.contains("BLOCKADE")) {
        d_country.COUNTRIESLIST.get(CountryID).d_numOfArmiesPlaced *= 3;
        System.out.println("DONE");
        p_p.d_owned.remove(d_country.COUNTRIESLIST.get(CountryID));
        d_country.COUNTRIESLIST.get(CountryID).d_owner = "Neutral";
    }
}
```

AFTER

```
case "blockade":
    if (p.p.d_cards.contains("BLOCKADE")) {
        d_country.COUNTRIESLIST.get(d_CountryID).d_numOfArmiesPlaced -= 3;
        p.p.d_owned.remove(d_country.COUNTRIESLIST.get(d_CountryID));
        d_country.COUNTRIESLIST.get(d_CountryID).d_owner = "Neutral";
        l_message = d_CountryID + " has been neutralized , now nobody owns this country";
        System.out.println(l_message);
        l_observable.setMsg(l_message);
        int l_indexCardName = -1;
        for (String l_cardName : p.p.d_cards) {
            l_indexCardName += 1;
            if (l_cardName.equalsIgnoreCase( anotherString: "BLOCKADE")) {
                p.p.d_cards.remove(l_indexCardName);
                break;
            }
        }
    } else {
        l_message = "You don't have the BLOCKADE card.";
        System.out.println(l_message);
        l_observable.setMsg(p.p + " " + l_message);
    }
    break;
case "airlift":
    if (p.p.d_cards.contains("AIRLIFT")) {
        if (!p.p.d_negotiate.contains(d_country.COUNTRIESLIST.get(d_CountryTo).d_owner)) {
            if (d_country.COUNTRIESLIST.get(d_CountryFrom).d_numOfArmiesPlaced >= d_armiesToAirlift) {
                d_obj = new AdvanceArmies(d_CountryFrom, d_CountryTo, d_armiesToAirlift, d_country, d_card);
                d_obj.Execute(p.p);
                int l_indexCardName = -1;
                for (String l_cardName : p.p.d_cards) {
                    l_indexCardName += 1;
                    if (l_cardName.equalsIgnoreCase( anotherString: "AIRLIFT")) {
                        p.p.d_cards.remove(l_indexCardName);
                    }
                }
            }
        }
    }
}
```

4. Changes in ShowMap class

- We added a new functionality in showMap class that if initially there is no map present in the folder then it will automatically say that there are no map present and let's create a new map. And after that it will directly ask the inputs for the creation of the map.
- Previously, in showMap if there is no map present then it went to EditMap class and from there it went to the creation of map. Now it the direct connection and there is no role of EditMap class.

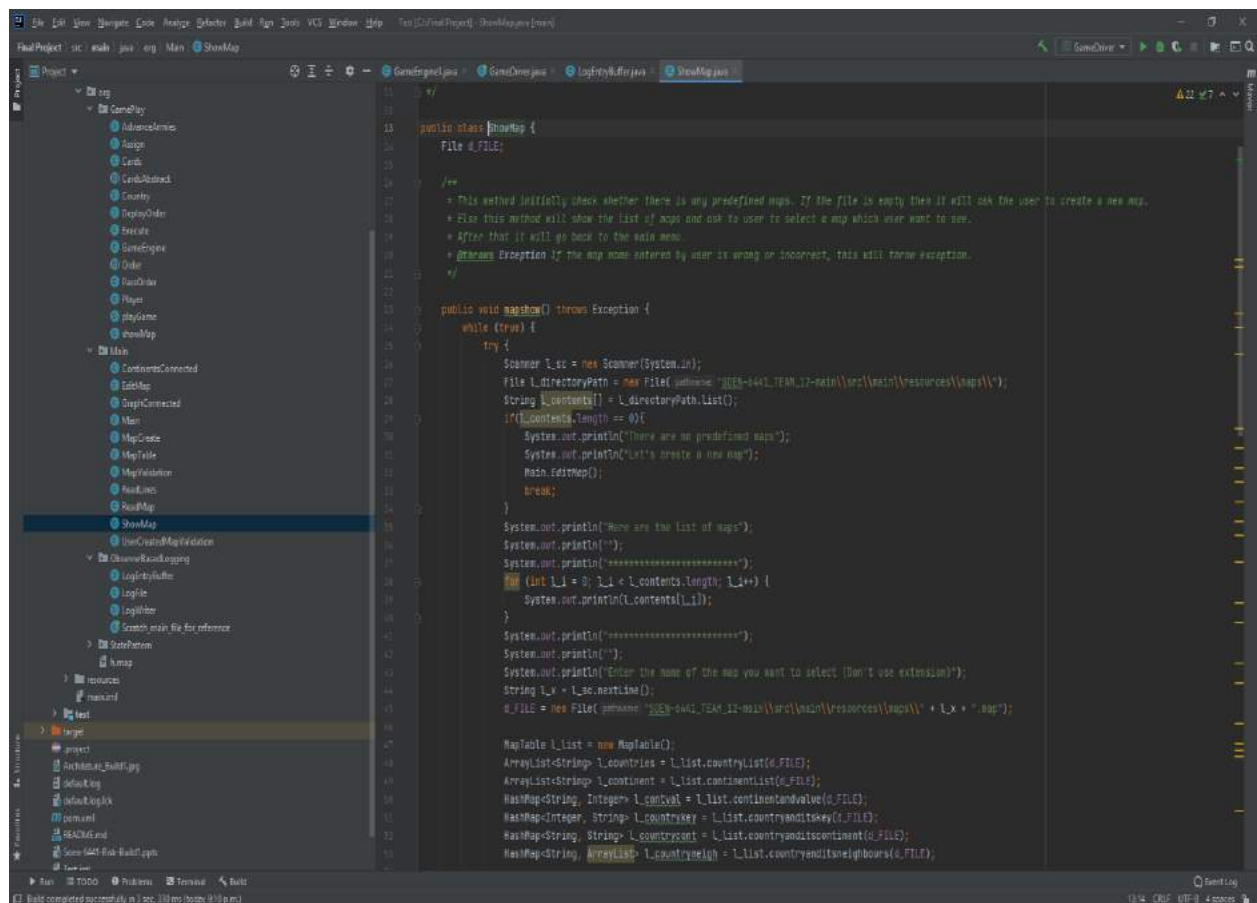
ASSOCIATED TEST CASES.

1.Test cases in MapTableTest.java

2.Test cases in ReadMapTest.java

3.Test cases in ReadlinesTest.java

BEFORE



```
1 //
2
3 public class ShowMap {
4     File d_FILE;
5
6     /**
7      * This method initially check whether there is any predefined maps. If the file is empty then it will ask the user to create a new map.
8      * Plus this method will show the list of maps and ask to user to select a map which user want to see.
9      * After that it will go back to the main menu.
10      * @throws Exception If the map name entered by user is wrong or incorrect, this will throw exception.
11      */
12
13     public void showMap() throws Exception {
14         while (true) {
15             try {
16                 Scanner l_sc = new Scanner(System.in);
17                 File l_directoryPath = new File("D:\\G4G1_TEAM_12\\src\\main\\resources\\maps\\");
18                 String l_contents[] = l_directoryPath.list();
19                 if (l_contents.length == 0) {
20                     System.out.println("There are no predefined maps");
21                     System.out.println("Let's create a new map");
22                     Main.EditMap();
23                     break;
24                 }
25                 System.out.println("Here are the list of maps");
26                 System.out.println("");
27                 System.out.println("=====");
28                 for (int l_i = 0; l_i < l_contents.length; l_i++) {
29                     System.out.println(l_contents[l_i]);
30                 }
31                 System.out.println("=====");
32                 System.out.println("");
33                 System.out.println("Enter the name of the map you want to select (Don't use extension)");
34                 String l_x = l_sc.nextLine();
35                 d_FILE = new File("D:\\G4G1_TEAM_12\\src\\main\\resources\\maps\\" + l_x + ".map");
36
37                 MapTable l_list = new MapTable();
38                 ArrayList<String> l_countries = l_list.countryList(d_FILE);
39                 ArrayList<String> l_continent = l_list.continentList(d_FILE);
40                 HashMap<String, Integer> l_contval = l_list.continentandval(d_FILE);
41                 HashMap<Integer, String> l_countrykey = l_list.countryanditskey(d_FILE);
42                 HashMap<String, String> l_countrycont = l_list.countryanditscontinent(d_FILE);
43                 HashMap<String, ArrayList> l_countryneigh = l_list.countryanditsneighbours(d_FILE);
44             } catch (Exception e) {
45                 e.printStackTrace();
46             }
47         }
48     }
49 }
```

AFTER

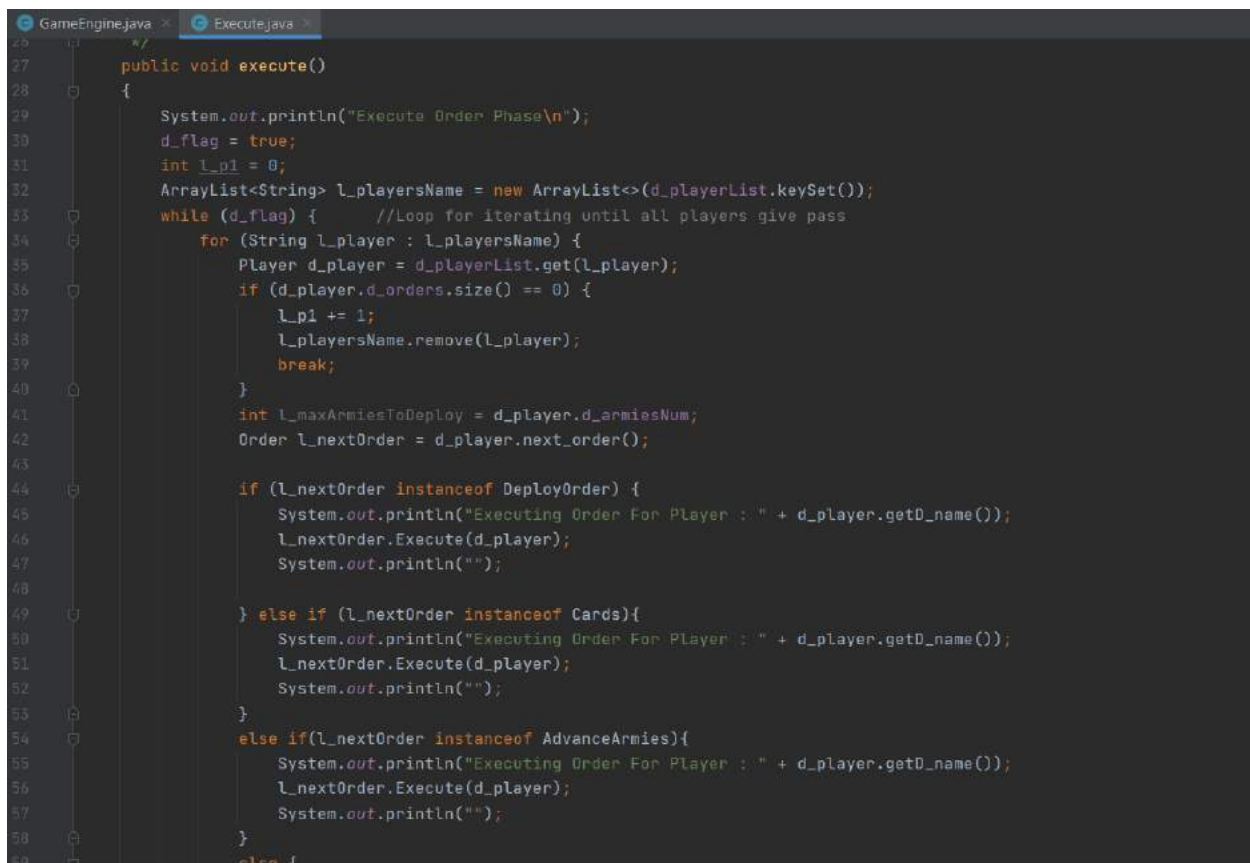
```
public void mapshow() throws Exception {
    while (true) {
        try {
            Scanner l_sc = new Scanner(System.in);
            File l_directoryPath = new File( pathname: "SOEN-6441_TEAM_12-main\\src\\main\\resources\\maps\\");
            String l_contents[] = l_directoryPath.list();
            int flag=0;
            if(l_contents.length == 0){
                System.out.println("There are no predefined maps");
                System.out.println("Let's create a new map");
                while(true) {
                    System.out.println("Enter the name of the map");
                    SC = new Scanner(System.in);
                    String l_filename = SC.nextLine();
                    if(l_filename.endsWith(".map"))
                    {
                        try {
                            File newFile = new File(l_filename);
                            flag=1;
                            Main.createMapByUser(newFile);
                            break;
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                    else {
                        System.out.println("Enter correct command\n");
                    }
                }
            }
            if(flag==1){
                break;
            }
        }
    }
}
```

5. Introduced a new class named Execute

- We have introduced a new class execute. Earlier execution of orders was getting done in the same class in which orders were taken.
- We have created a new class to differentiate the phases of the game i.e., to introduce the execute phase differently.

ASSOCIATED TEST CASES.

All the test cases of the cards and various commands can validate this execute class.

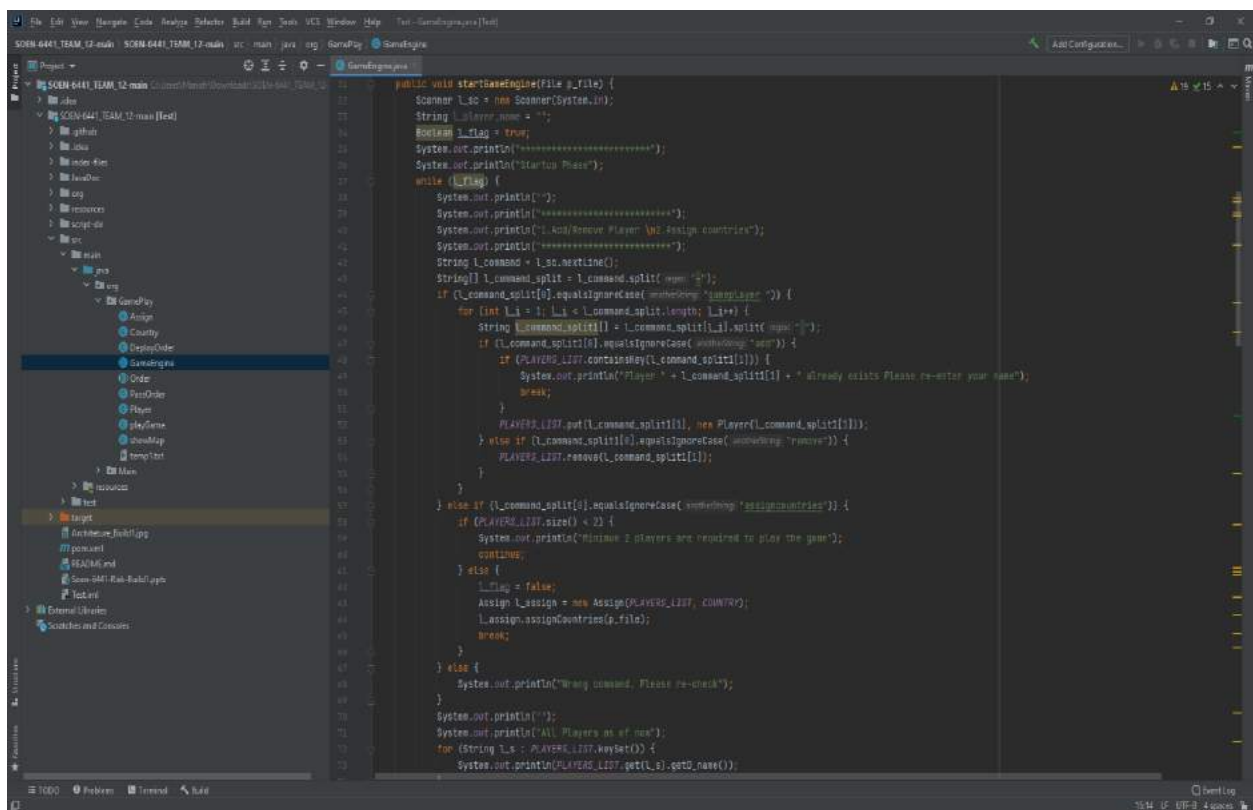


```
27 public void execute()
28 {
29     System.out.println("Execute Order Phase\n");
30     d_flag = true;
31     int l_p1 = 0;
32     ArrayList<String> l_playersName = new ArrayList<>(d_playerList.keySet());
33     while (d_flag) { //Loop for iterating until all players give pass
34         for (String l_player : l_playersName) {
35             Player d_player = d_playerList.get(l_player);
36             if (d_player.d_orders.size() == 0) {
37                 l_p1 += 1;
38                 l_playersName.remove(l_player);
39                 break;
40             }
41             int l_maxArmiesToDeploy = d_player.d_armiesNum;
42             Order l_nextOrder = d_player.next_order();
43
44             if (l_nextOrder instanceof DeployOrder) {
45                 System.out.println("Executing Order For Player : " + d_player.getD_name());
46                 l_nextOrder.Execute(d_player);
47                 System.out.println("");
48             } else if (l_nextOrder instanceof Cards){
49                 System.out.println("Executing Order For Player : " + d_player.getD_name());
50                 l_nextOrder.Execute(d_player);
51                 System.out.println("");
52             }
53             else if(l_nextOrder instanceof AdvanceArmies){
54                 System.out.println("Executing Order For Player : " + d_player.getD_name());
55                 l_nextOrder.Execute(d_player);
56                 System.out.println("");
57             }
58         }
59     }
60 }
```


6. Changes in the GameEngine class

- We have broken the continuity of the game to introduce the state pattern in the game.
- Previously, there was a loop to add player and assign countries together but now we have now broken this continuity so as to create different phases for every part.

BEFORE



```
public void startGameEngine(File p_file) {
    Scanner l_sc = new Scanner(System.in);
    String l_error_name = "";
    boolean l_flag = true;
    System.out.println("*****");
    System.out.println("Startup Phase");
    while (l_flag) {
        System.out.println("");
        System.out.println("*****");
        System.out.println("1.Add/Remove Player and Assign countries");
        System.out.println("*****");
        String l_command = l_sc.nextLine();
        String[] l_command_split = l_command.split(" ");
        if (l_command_split[0].equalsIgnoreCase("addplayer")) {
            for (int l_i = 1; l_i < l_command_split.length; l_i++) {
                String l_command_split1[] = l_command_split[l_i].split(" ");
                if (l_command_split1[0].equalsIgnoreCase("name")) {
                    if (PLAYERS_LIST.contains(l_command_split1[1])) {
                        System.out.println("Player " + l_command_split1[1] + " already exists Please re-enter your name");
                        break;
                    }
                    PLAYERS_LIST.put(l_command_split1[1], new Player(l_command_split1[1]));
                } else if (l_command_split1[0].equalsIgnoreCase("remove")) {
                    PLAYERS_LIST.remove(l_command_split1[1]);
                }
            }
        } else if (l_command_split[0].equalsIgnoreCase("assigncountries")) {
            if (PLAYERS_LIST.size() < 2) {
                System.out.println("Minimum 2 players are required to play the game");
                continue;
            } else {
                l_flag = false;
                Assign l_assign = new Assign(PLAYERS_LIST, COUNTRY);
                l_assign.assignCountries(p_file);
                break;
            }
        } else {
            System.out.println("Wrong command. Please re-check");
        }
        System.out.println("");
        System.out.println("All Players at st. now");
        for (String l_s : PLAYERS_LIST.keySet()) {
            System.out.println(PLAYERS_LIST.get(l_s).getD_name());
        }
    }
}
```

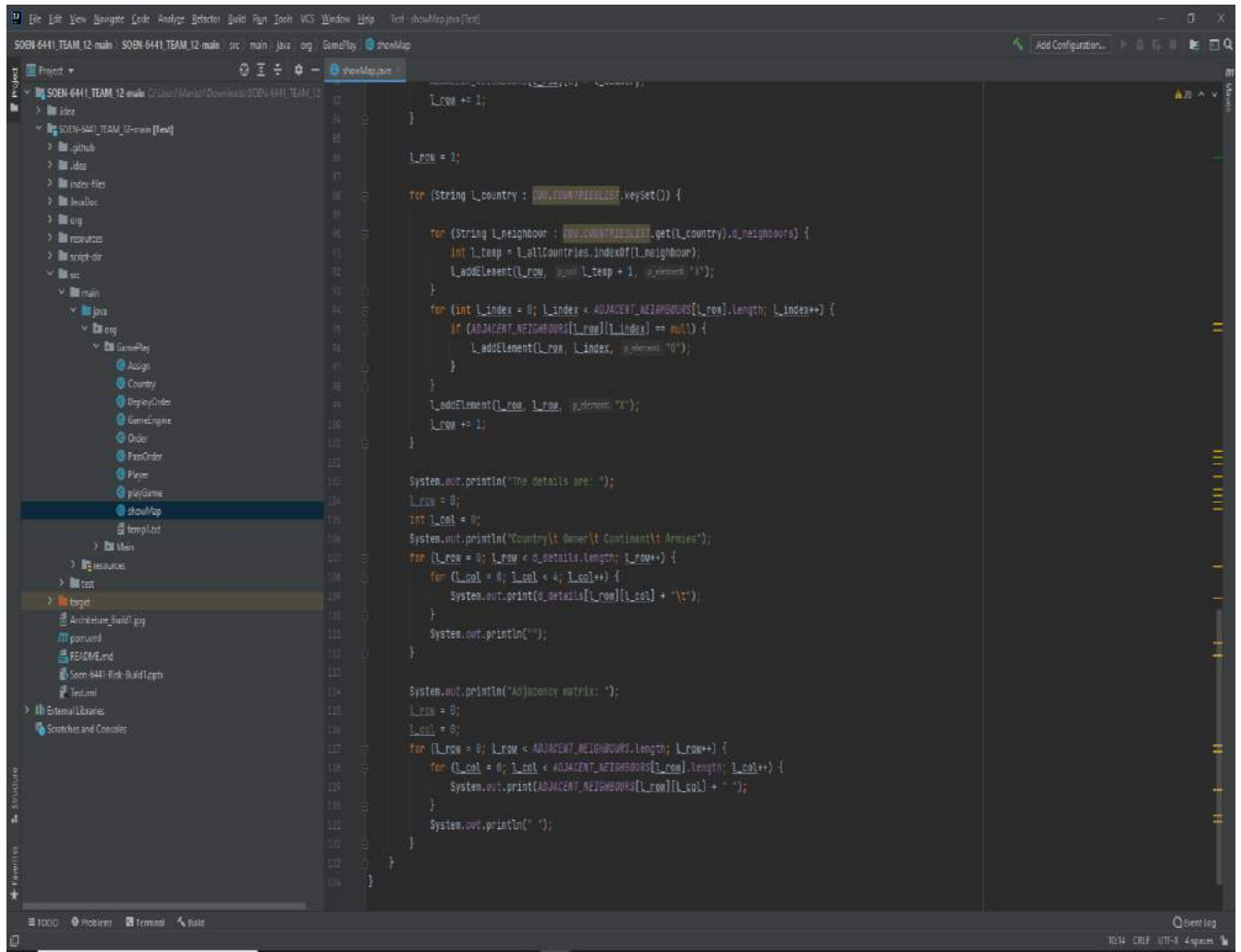

AFTER

```
GameEngine.java
38
39 public void startGameEngine(File p_file) {
40     Scanner l_sc = new Scanner(System.in);
41     String l_player_name = "";
42     Boolean l_flag = true;
43     System.out.println("*****");
44     System.out.println("Startup Phase");
45     while (l_flag) {
46         System.out.println("");
47         System.out.println("*****");
48         System.out.println("1.Add/Remove Player");
49         System.out.println("*****");
50         String l_command = l_sc.nextLine();
51         String[] l_command_split = l_command.split(" ");
52         if (l_command_split[0].equalsIgnoreCase("gameplayer ") && l_command_split.length > 1) {
53             for (int l_i = 1; l_i < l_command_split.length; l_i++) {
54                 String l_command_split1[] = l_command_split[l_i].split(" ");
55                 if (l_command_split1[0].equalsIgnoreCase("add")) {
56                     if (PLAYERS_LIST.containsKey(l_command_split1[1])) {
57                         System.out.println("Player " + l_command_split1[1] + " already exists Please re-enter your name");
58                         break;
59                     }
60                     PLAYERS_LIST.put(l_command_split1[1], new Player(l_command_split1[1]));
61                     l_flag = false;
62                 } else if (l_command_split1[0].equalsIgnoreCase("remove")) {
63                     PLAYERS_LIST.remove(l_command_split1[1]);
64                     l_flag = false;
65                 }
66             }
67         } else {
68             System.out.println("Wrong command. Please re-check");
69         }
70         System.out.println("");
    }
```

7. Introduced changes in ShowMap in Gameplay

- Previously no cards were shown to any player in the game.
- Now, every player can see cards i.e., which player which cards.

BEFORE



```
114 1_row += 1;
115  }
116  1_row = 1;
117
118  for (String l_country : ROW_COUNTRY_LIST.keySet()) {
119
120      for (String l_neighbour : ROW_COUNTRY_LIST.get(l_country).get_neighbours()) {
121          int l_tesp = l_allCountries.indexOf(l_neighbour);
122          l_addElement(l_row, l_col = l_tesp + 1, l_element: "X");
123      }
124      for (int l_index = 0; l_index < ADJACENT_NEIGHBOURS[l_row].length; l_index++) {
125          if (ADJACENT_NEIGHBOURS[l_row][l_index] == null) {
126              l_addElement(l_row, l_index, l_element: "0");
127          }
128      }
129      l_addElement(l_row, l_col, l_element: "X");
130      l_row += 1;
131  }
132
133  System.out.println("The details are:");
134  l_row = 0;
135  int l_col = 0;
136  System.out.println("Country\tOwner\tContinent\tArmies");
137  for (l_row = 0; l_row < o_details.length; l_row++) {
138      for (l_col = 0; l_col < 4; l_col++) {
139          System.out.print(o_details[l_row][l_col] + "\t");
140      }
141      System.out.println("");
142  }
143
144  System.out.println("Adjacency matrix:");
145  l_row = 0;
146  l_col = 0;
147  for (l_row = 0; l_row < ADJACENT_NEIGHBOURS.length; l_row++) {
148      for (l_col = 0; l_col < ADJACENT_NEIGHBOURS[l_row].length; l_col++) {
149          System.out.print(ADJACENT_NEIGHBOURS[l_row][l_col] + " ");
150      }
151      System.out.println("");
152  }
153  }
```

AFTER

```
Cards.java x ShowMap.java x showMap.java x
113
114     System.out.println("Adjacency matrix: ");
115     l_row = 0;
116     l_col = 0;
117     for (l_row = 0; l_row < ADJACENT_NEIGHBOURS.length; l_row++) {
118         for (l_col = 0; l_col < ADJACENT_NEIGHBOURS[l_row].length; l_col++) {
119             System.out.print(ADJACENT_NEIGHBOURS[l_row][l_col] + " ");
120         }
121         System.out.println(" ");
122     }
123     System.out.println("");
124
125     System.out.println("Cards of the Players: ");
126
127     for(String l_player:PLAYERS_LIST.keySet())
128     {
129         if(PLAYERS_LIST.get(l_player).d_cards.size()!=0) {
130             System.out.print(l_player + "= ");
131             for (String l_cards : PLAYERS_LIST.get(l_player).d_cards) {
132                 System.out.print(l_cards + " ");
133             }
134             System.out.println("");
135         }
136         else
137         {
138             System.out.println("No cards for player: "+l_player);
139         }
140     }
141 }
142 }
```

8. Code repetition in continent connectivity check

- In Graph Connected class for countries connectivity there is dfs() function and finally checking the connectivity ifGraphConnected() function is there.
- In ContinentCheck both above functions can be used but instead whole code is written from scratch.
- This can be avoided by making dfs() and ifGraphConnected() function generalized.

```
GraphConnected.java
120 public boolean ContinentsCheck() throws Exception {
121     ContinentsConnected d_obj = new ContinentsConnected(FILE);
122     HashMap<Integer, ArrayList> l_list = d_obj.ifContinentCountriesConnected();
123     for(Map.Entry<Integer, ArrayList> l_entry: l_list.entrySet()) {
124         HashMap<Integer, ArrayList> l_check = d_obj.CountryNeighbours(FILE, l_entry.getValue());
125         List<Integer> l_nodes = l_entry.getValue();
126         ArrayList<Integer> l_empty = new ArrayList<>();
127         if(l_nodes.size() == 1){
128             l_check.put(l_nodes.get(0), l_empty);
129         }
130         int l_max = Collections.max(l_nodes);
131         d_scan = new boolean[l_max + 1];
132         boolean [] l_traversed = new boolean[l_max + 1];
133         int l_key_1 = l_check.keySet().stream().findFirst().get();
134         int p_start = l_key_1;
135         d_stack.push(p_start);
136         l_traversed[p_start] = true;
137         d_country_index = l_nodes;
138         d_scan[0] = true;
139         for (int l_i = 0; l_i < d_country_index.size(); l_i++){
140             d_scan[d_country_index.get(l_i)] = true;
141         }
142         for (int l_j = 0; l_j < l_nodes.size(); l_j++){
143             if(l_check.containsKey(l_nodes.get(l_j)) == false){
144                 int l_note = l_nodes.get(l_j);
145                 l_check.put(l_note, l_empty);
146             }
147         }
148         int l_count = 0;
149         while (true) {
150             if (d_stack.isEmpty() == true) {
151                 break;
152             }
153         }
154     }
155 }
```

9. Code repetition in EditMap Class

- There is validation call part in the EditMap class which is called thrice in the same class.
- We could have refactored this one too, but this is not an important part which we need to do so.
- Also, this is making the understandability of code much easier.

```
File l_file3 = new File(l_address);
l_flag_error=0;
while(true) {
    try {
        System.out.println("Enter validatemap command");
        String l_reply = d_SC.nextLine();
        if (l_reply.equalsIgnoreCase( anotherString: "validatemap")) {
            MapValidation l_validate = new MapValidation();
            l_validate.mapValidate(l_file3, l_countries, l_continent, l_cont_val, l_country_key, l_country_conti
            break;
        }
        else{
            System.out.println("Enter valid command");
        }
    } catch (Exception p_e) {
        p_e.printStackTrace();
    }
}
```

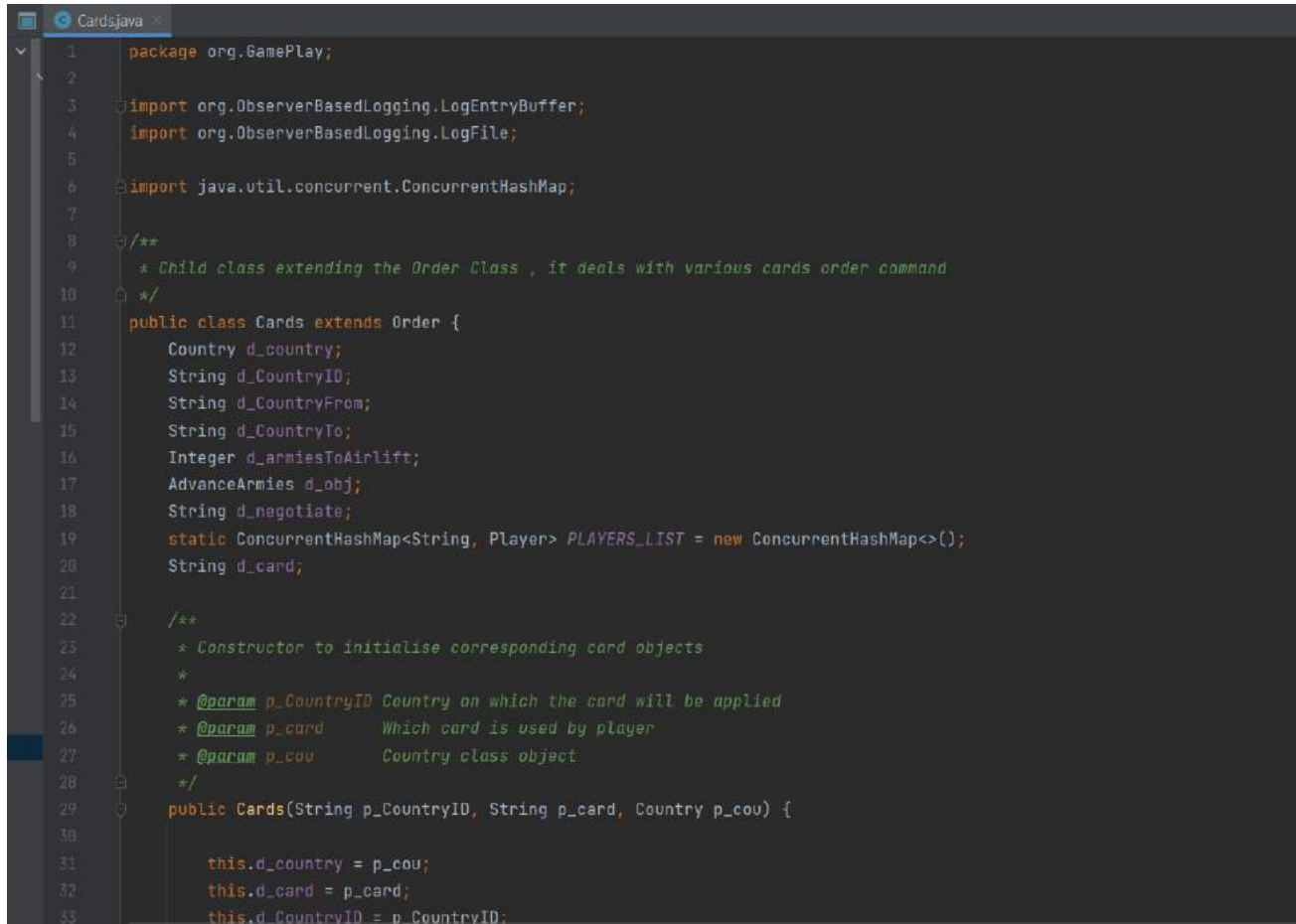
```
File l_file2 = new File(l_address);
l_flag_error=0;
while(true) {
    try {
        System.out.println("Enter validatemap command");
        String l_reply = d_SC.nextLine();
        if (l_reply.equalsIgnoreCase( anotherString: "validatemap")) {
            MapValidation l_validate = new MapValidation();
            l_validate.mapValidate(l_file2, l_countries, l_continent, l_cont_val, l_country_key, l_country_cont
            break;
        }
        else{
            System.out.println("Enter valid command");
        }
    } catch (Exception p_e) {
        p_e.printStackTrace();
    }
}
```

```
File l_file3 = new File(l_address);
l_flag_error = 0;

while(true) {
    try {
        System.out.println("Enter validatemap command");
        String l_reply = d_SC.nextLine();
        if (l_reply.equalsIgnoreCase( anotherString: "validatemap")) {
            MapValidation l_validate = new MapValidation();
            l_validate.mapValidate(l_file3, l_countries, l_continent, l_cont_val, l_country_key, l_country_cont
            break;
        }
        else{
            System.out.println("Enter valid command");
        }
    } catch (Exception p_e) {
        p_e.printStackTrace();
    }
}
```

10. Introduced a new class named cards

- We have introduced a new class named cards in which all the cards are defined and their functionalities as defined.
- Every card has it's own functionality, when player use any of the card they will work according to the functionalities wrote in the code.



```
1  package org.GamePlay;
2
3  import org.ObserverBasedLogging.LogEntryBuffer;
4  import org.ObserverBasedLogging.LogFile;
5
6  import java.util.concurrent.ConcurrentHashMap;
7
8  /**
9   * Child class extending the Order Class , it deals with various cards order command
10  */
11  public class Cards extends Order {
12      Country d_country;
13      String d_CountryID;
14      String d_CountryFrom;
15      String d_CountryTo;
16      Integer d_armiesToAirlift;
17      AdvanceArmies d_obj;
18      String d_negotiate;
19      static ConcurrentHashMap<String, Player> PLAYERS_LIST = new ConcurrentHashMap<>();
20      String d_card;
21
22      /**
23       * Constructor to initialise corresponding card objects
24       *
25       * @param p_CountryID Country on which the card will be applied
26       * @param p_card       Which card is used by player
27       * @param p_cou        Country class object
28       */
29      public Cards(String p_CountryID, String p_card, Country p_cou) {
30
31          this.d_country = p_cou;
32          this.d_card = p_card;
33          this.d_CountryID = p_CountryID;
```

11. Add Card Function is never used in Player Class

- add_card() function inside player class is never used.
- Instead ArrayList d_cards is directly accessed in various parts of our program.
- Either we can remove this function or we can use this function in order to access d_cards arraylist where it is accessed in our code.

```
/**
 * Method to add the cards to the list.
 * @param cards is the Name of the card that player has won.
 */
public void add_card(String cards) { this.d_cards.add(cards); }
/**
```

12. Type of Order Variable is never used in Deploy Order Class

- Initially d_typeOfOrder variable was created to differentiate between different orders.
- Now after introducing different design patterns this variable is of no use.
- We can simply remove it.

```
public class DeployOrder extends Order {
    String d_typeOfOrder = "DeployOrder";
    String d_countryId;
    Integer d_armiesToPlace;
    Country d_cou;
    String d_temp = "";
```


13. Invalid Command Handling can be encapsulated inside a function in playGame class.

- No when the user types command ,to validate this command all the handling is done using if-else statements.
- We can do this handling inside a function and simply call it at the appropriate place.

```
} else if (l_commandSplit[0].equalsIgnoreCase( anotherString: "bomb")) {
    if (l_commandSplit.length == 2) {
        String l_countryId = l_commandSplit[1];
        if (d_country.COUNTRIESLIST.containsKey(l_countryId)) {
            if (!l_p.d_owned.contains(d_country.COUNTRIESLIST.get(l_countryId))) {
                l_flag_1 = false;
                Cards l_card = new Cards(l_countryId, l_commandSplit[0], d_country);
                l_p.issue_order(l_card);
            } else {
                System.out.println("Cannot Bomb its own country");
                continue;
            }
        } else {
            System.out.println("Not a Valid Country");
            continue;
        }
    } else {
        System.out.println("Not a Valid Command");
        continue;
    }
}
} else if (l_commandSplit[0].equalsIgnoreCase( anotherString: "advance")) {
    if (l_commandSplit.length == 4) {
        String l_country_from = l_commandSplit[1];
        String l_country_to = l_commandSplit[2];
        String l_armies = l_commandSplit[3];
        String l_card = "empty";
        int l_armiesToAdvance = Integer.parseInt(l_armies);
        if (d_country.COUNTRIESLIST.containsKey(l_country_from) && d_country.COUNTRIESLIST.containsKey(l_country_to)) {
            if (l_p.d_owned.contains(d_country.COUNTRIESLIST.get(l_country_from))) {
                if (d_country.COUNTRIESLIST.get(l_country_from).d_neighbours.contains(l_country_to)) {
                    if (l_p.d_armiesNum >= l_armiesToAdvance) {
```

14. In Cards we can encapsulate each card functionality into a function and simply call it.

- We have simply just implemented the functionality of each card directly in the respective switch cases.
- We can create functions for each card and then simply call it under the switch case.

```
case "blockade":
    if(p_p.d_cards.contains("BLOCKADE") && p_p.d_owned.contains(d_country.COUNTRIESLIST.get(d_CountryID))) {
        d_country.COUNTRIESLIST.get(d_CountryID).d_numOfArmiesPlaced -= 3;
        p_p.d_owned.remove(d_country.COUNTRIESLIST.get(d_CountryID));
        d_country.COUNTRIESLIST.get(d_CountryID).d_owner = "Neutral";
        l_message = d_CountryID + " has been neutralized , now nobody owns this country";
        System.out.println(l_message);
        l_observable.setMsg(l_message);
        int l_indexCardName=-1;
        for(String l_cardName: p_p.d_cards)
        {
            l_indexCardName++;
            if(l_cardName.equalsIgnoreCase( anotherString: "BLOCKADE"))
            {
                p_p.d_cards.remove(l_indexCardName);
                break;
            }
        }
    }
    else{
        l_message = "You don't have the BLOCKADE card.";
        System.out.println(l_message);
        l_observable.setMsg(p_p+" "+l_message);
    }
    break;
case "airlift":
    if(p_p.d_cards.contains("AIRLIFT")) {
        if (!p_p.d_negotiate.contains(d_country.COUNTRIESLIST.get(d_CountryTo).d_owner)) {
            if (d_country.COUNTRIESLIST.get(d_CountryFrom).d_numOfArmiesPlaced >= d_armiesToAirlift) {
                d_obj = new AdvanceArmies(d_CountryFrom, d_CountryTo, d_armiesToAirlift, d_country, d_card);
                d_obj.Execute(p_p);
            }
        }
    }
}
```

15. Introduced Observer Pattern

- This was again the requirement of the Build-2, so we have introduced Observer Pattern as well in our code.
- Using this we have created a LogEntryBuffer which act as a observable.
- This buffer keeps the track of the game.

