

Group 15: REFACTORING BUILD 3  
COURSE: SOEN 6441 APP  
INSTRUCTOR: Prof. JOEY PAQUET

TEAM MEMBERS:

- Mehakveer Singh Bal (40293616)
- Jaiwant Singh Mahun (40270569)
- Chappalwala Amir Asif (40290282)
- Punamkumar Patel (40290462)
- Heeba Shaikh (40278184)
- Pankaj Sharma (40269802)

## **Potential Refactoring Targets:**

The following list of refactoring targets has been compiled based on the requirements identified from inconsistencies and pain points observed in build 2, as well as insights gained throughout the entire development process of build 3.

- 1) Adding logs for newly added functionality and remaining functions from build2.
- 2) Restructuring the Adapter pattern for loading and saving Domination and Conquest map types.
- 3) Adding the Strategy pattern for player behavioral strategies.
- 4) Enhancing the presentation of information on the console.
- 5) Reworking error handling through exceptions.
- 6) Modularizing and fully segregating the observer into the view directory.
- 7) Overhauling the game to include both single and tournament modes.
- 8) Validating the Command pattern.
- 9) Refactoring and rectifying in accordance with coding conventions.
- 10) Refactoring functions to include tournament mode command.
- 11) LoadMap function changed to incorporate Conquest Map format.
- 12) Game Services added to enable savegame and loadgame functionality.
- 13) Javadoc addition to private data members.

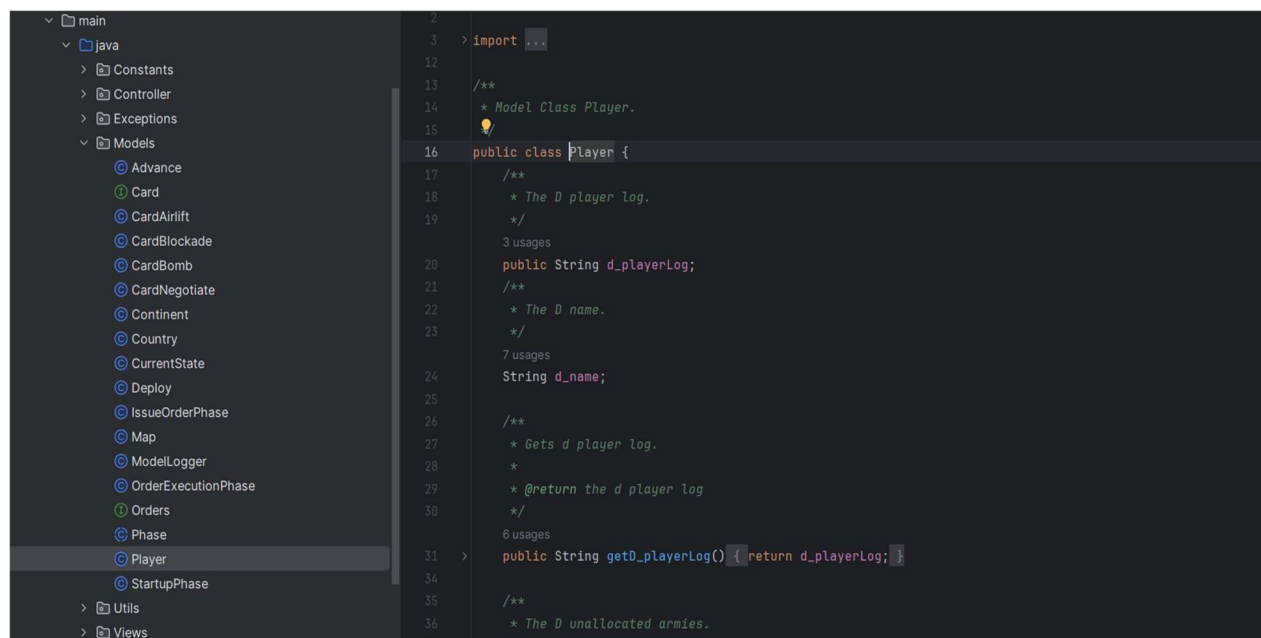
Additional test cases implementation.

- 1) Save Game and Load Game addition.
- 2) Refactoring with Save Game and Load Game.
- 3) Tournament Mode test Case Addition.

# Actual Refactoring Targets:

- 1) Restructure the `issueOrder()` method in the `Player` class to implement the Strategy pattern: In alignment with the specifications of `build3`, we've redesigned the `issueOrder()` method within the `Player` class, incorporating distinct strategies for each player.

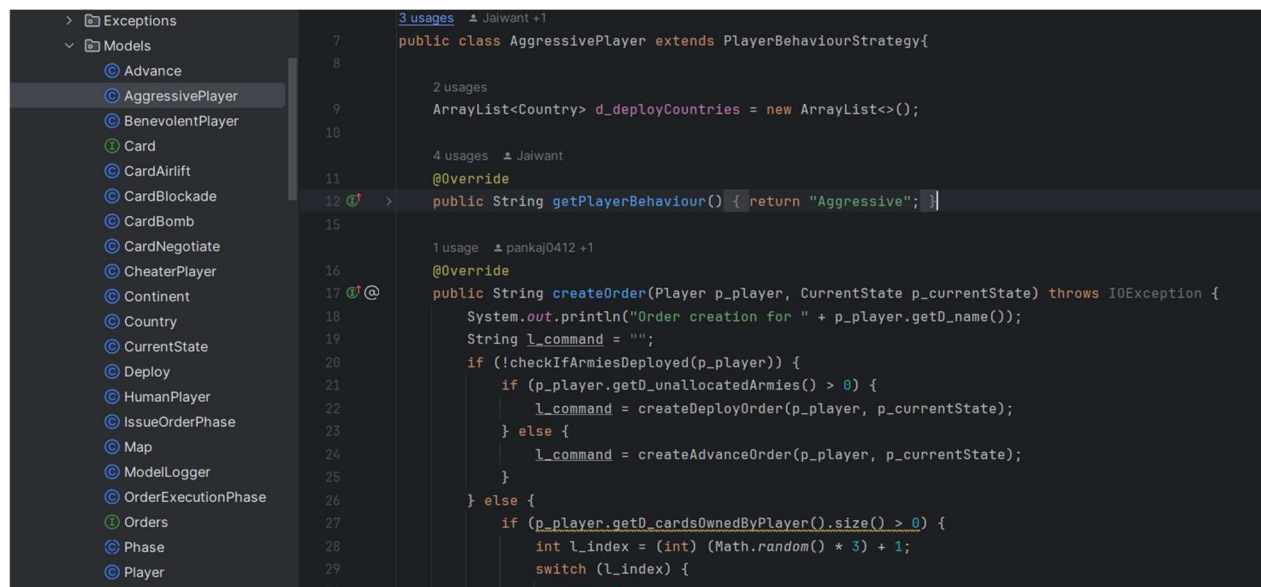
## Before Refactoring



The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a package structure with 'main' and 'java' folders. Under 'java', there are sub-packages like 'Constants', 'Controller', 'Exceptions', and 'Models'. The 'Models' package contains several classes, with 'Player' selected. The code editor displays the `Player` class with the following code:

```
2
3 > import ...
12
13 /**
14  * Model Class Player.
15  */
16 public class Player {
17     /**
18      * The d player log.
19      */
20     public String d_playerLog;
21     /**
22      * The d name.
23      */
24     String d_name;
25
26     /**
27      * Gets d player log.
28      *
29      * @return the d player log
30      */
31     public String getD_playerLog() { return d_playerLog; }
32
33     /**
34      * The d unallocated armies.
35      */
36 }
```

## After Refactoring



The screenshot shows the same IDE after refactoring. The project explorer now shows the 'Models' package with several new classes, including 'AggressivePlayer', 'BenevolentPlayer', 'Card', 'CardAirLift', 'CardBlockade', 'CardBomb', 'CardNegotiate', 'CheaterPlayer', 'Continent', 'Country', 'CurrentState', 'Deploy', 'HumanPlayer', 'IssueOrderPhase', 'Map', 'ModelLogger', 'OrderExecutionPhase', 'Orders', 'Phase', and 'Player'. The 'Player' class is still selected. The code editor displays the `AggressivePlayer` class, which extends `PlayerBehaviourStrategy`. The code is as follows:

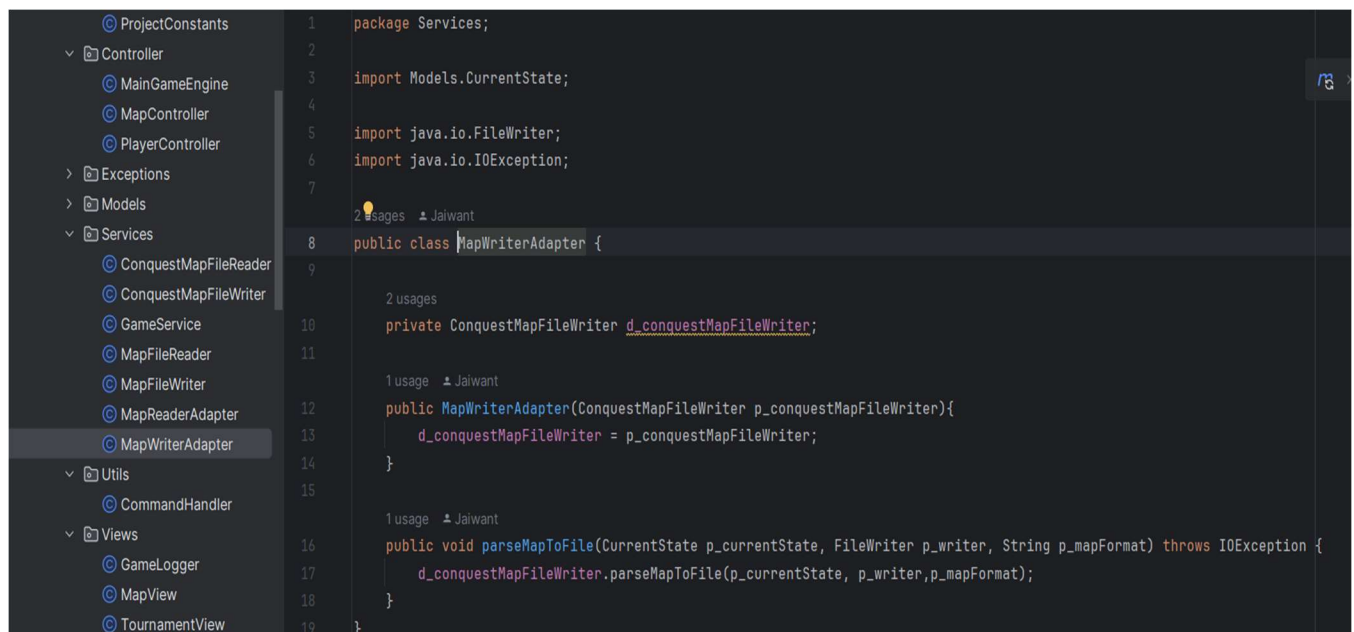
```
7
8 public class AggressivePlayer extends PlayerBehaviourStrategy{
9
10     ArrayList<Country> d_deployCountries = new ArrayList<>();
11
12     @Override
13     public String getPlayerBehaviour() { return "Aggressive"; }
14
15
16     @Override
17     public String createOrder(Player p_player, CurrentState p_currentState) throws IOException {
18         System.out.println("Order creation for " + p_player.getD_name());
19         String l_command = "";
20         if (!checkIfArmiesDeployed(p_player)) {
21             if (p_player.getD_unallocatedArmies() > 0) {
22                 l_command = createDeployOrder(p_player, p_currentState);
23             } else {
24                 l_command = createAdvanceOrder(p_player, p_currentState);
25             }
26         } else {
27             if (p_player.getD_cardsOwnedByPlayer().size() > 0) {
28                 int l_index = (int) (Math.random() * 3) + 1;
29                 switch (l_index) {
30                     case 1:
31 
```

2) Utilizing the Adapter pattern to handle the loading and saving of Domination files in build3.

## Before Refactoring

Adapter pattern was not present in build2

## After Refactoring

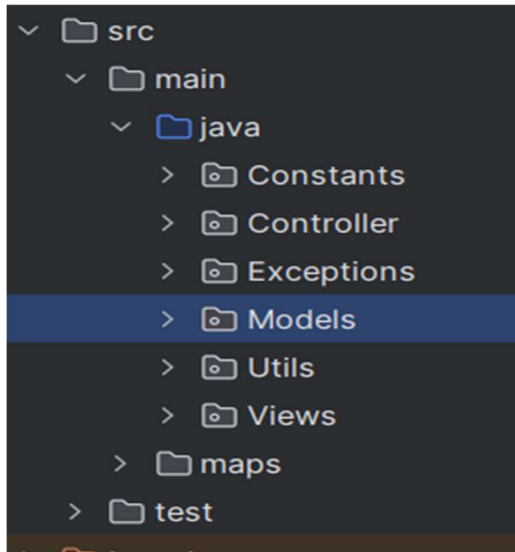


The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a package structure with 'Services' containing 'MapWriterAdapter'. The code editor displays the implementation of 'MapWriterAdapter' in the 'Services' package. The code includes imports for 'Models.CurrentState', 'java.io.FileWriter', and 'java.io.IOException'. It defines a private field 'd\_conquestMapFileWriter' of type 'ConquestMapFileWriter', a constructor that initializes this field, and a 'parseMapToFile' method that delegates the parsing task to 'd\_conquestMapFileWriter'. Usage counts are shown for the field and the method.

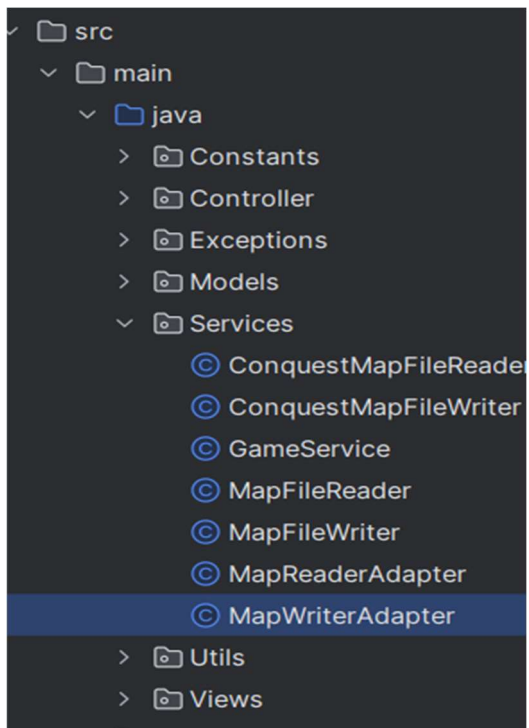
```
1 package Services;
2
3 import Models.CurrentState;
4
5 import java.io.FileWriter;
6 import java.io.IOException;
7
8 2 usages  ▲ Jalwant
9 public class MapWriterAdapter {
10
11     2 usages
12     private ConquestMapFileWriter d_conquestMapFileWriter;
13
14     1 usage  ▲ Jalwant
15     public MapWriterAdapter(ConquestMapFileWriter p_conquestMapFileWriter){
16         d_conquestMapFileWriter = p_conquestMapFileWriter;
17     }
18
19     1 usage  ▲ Jalwant
20     public void parseMapToFile(CurrentState p_currentState, FileWriter p_writer, String p_mapFormat) throws IOException {
21         d_conquestMapFileWriter.parseMapToFile(p_currentState, p_writer, p_mapFormat);
22     }
23 }
```

- 3) New Services Added for implementing Map Adapter Pattern and adding other functionalities such as loadgame and savegame.

Before Refactor

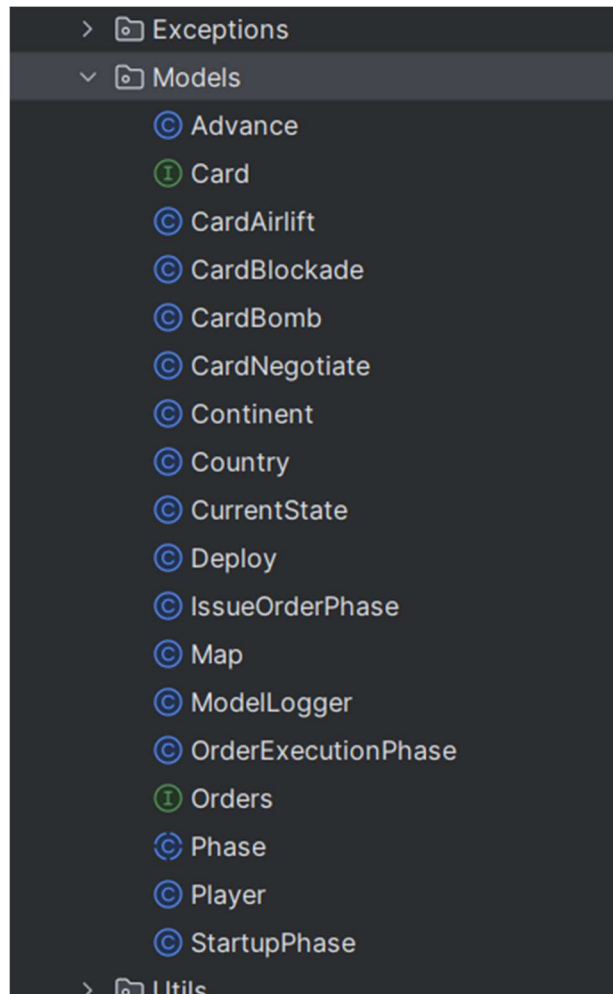


After Refactor

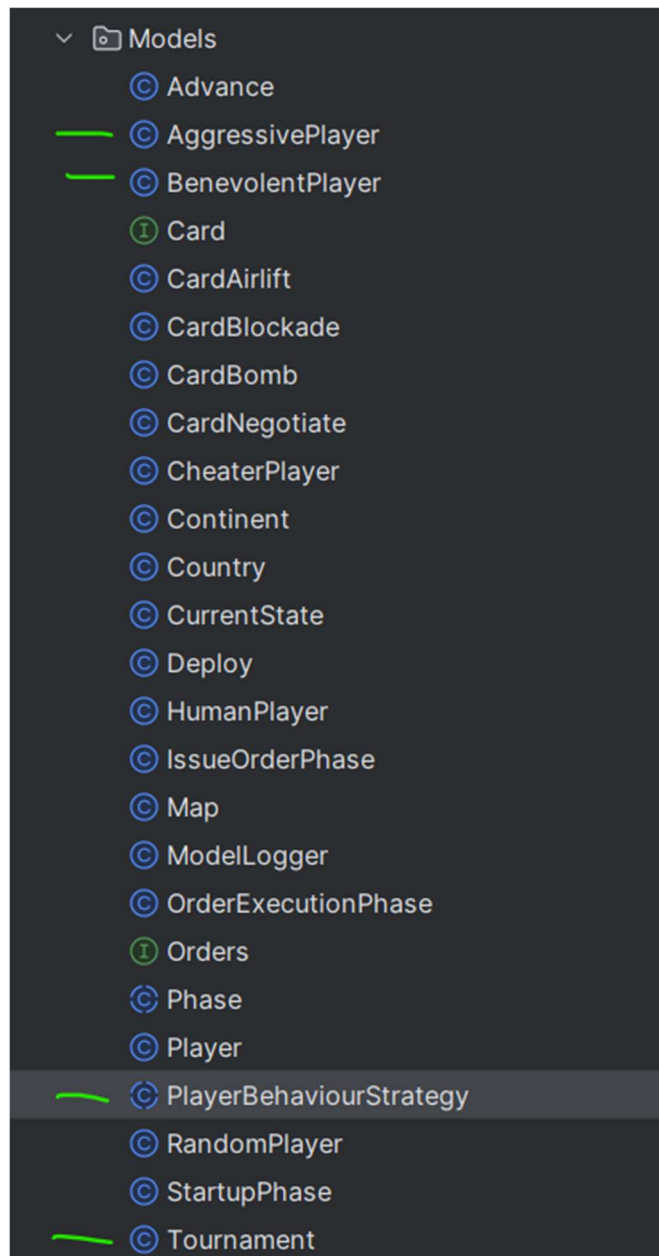


- 4) New classes and Models introduced depending upon the need and requirements of Build 3.

**Before Refactor:**



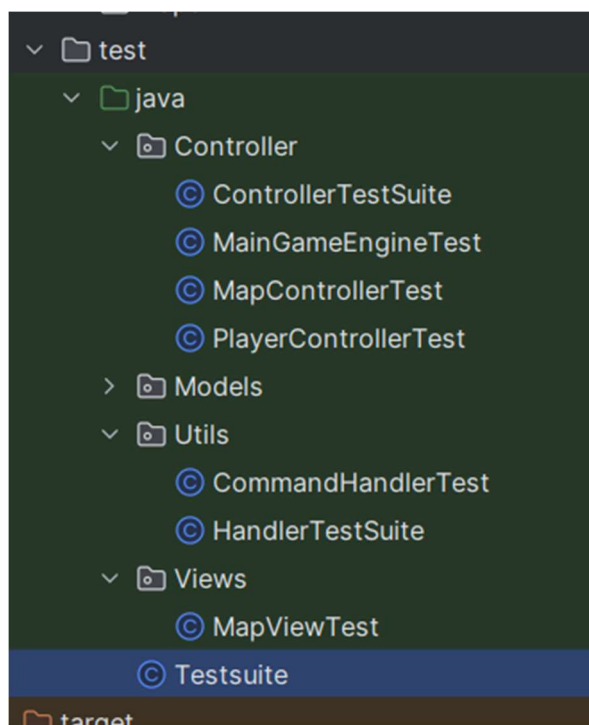
## After Refactor:



## 5) More test cases added

More test-cases were implemented in addition to the previous ones and these test cases were mapped to the name of the actual model class.

### Before refactor





## After refactor

