# Group U5: REFACTORING

COURSE: SOEN 6441

INSTRUCTOR: AMIN RANJ BAR

## TEAM MEMBERS:

- Abhishek Handa (40231719)
- Rajat Sharma (40196467)
- Harman Singh Jolly (40204947)
- Amanpreet Singh (40221947)
- Anurag Teckchandani (40263724)

## Potential Refactoring Targets:

1. Take Hardcoded Strings in separate classes.
2. Introduce the State Pattern into the Map editor.
3. Apply the State Pattern to the Startup, Issue, and Order phases of gameplay.
4. Incorporate command syntax validation.
5. Utilize the command pattern for order processing.
6. Transfer the logic from the model to the controller in the "IssueOrder" function.
7. Adjust certain naming conventions for improved clarity.
8. Integrate Exception Handling for scenarios such as adding a country without a continent, missing information, and handling typos when adding neighbors.
9. Create additional test cases for the existing logic.
10. Revise the continent check in Map Validation.
11. Add Javadoc comments for private data members.
12. Implement a restriction that prevents the game from proceeding with fewer than two players.
13. Define a constant for the file path.

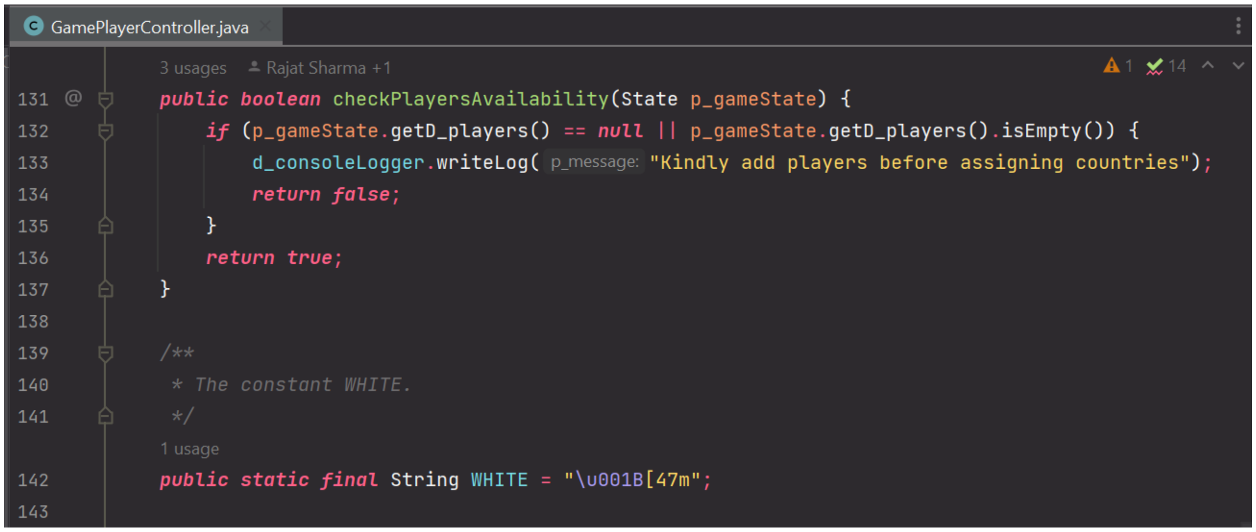
14. Implement the Observer pattern for console log updates.
15. Alter the format for saving the map to match the domination map format.

## Actual Refactoring Targets:

The list of refactoring taken for the above mentioned target list was mainly chosen because of the new requirements established in build 2 and on the greatest pain points and in-consistencies encountered during the development of the first build.

1. Hardcoded strings taken to a separate class called ApplicationConstantsHardcoding. Earlier the hardcodings were in each and every class.

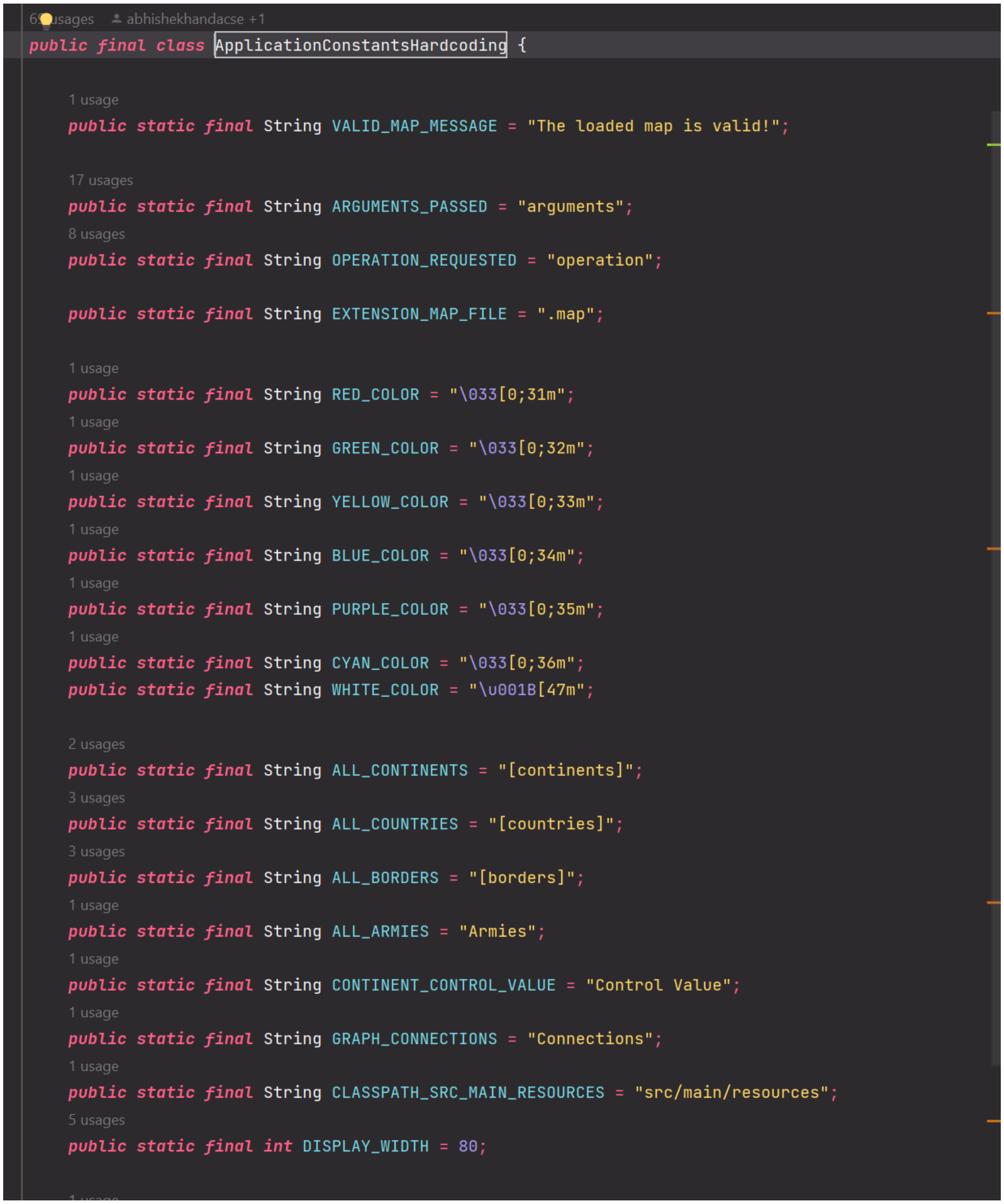
Before Refactoring



GamePlayerController.java

```
3 usages   Rajat Sharma +1
public boolean checkPlayersAvailability(State p_gameState) {
    if (p_gameState.getD_players() == null || p_gameState.getD_players().isEmpty()) {
        d_consoleLogger.writeLog( p_message: "Kindly add players before assigning countries");
        return false;
    }
    return true;
}

/**
 * The constant WHITE.
 */
1 usage
public static final String WHITE = "\u001B[47m";
```

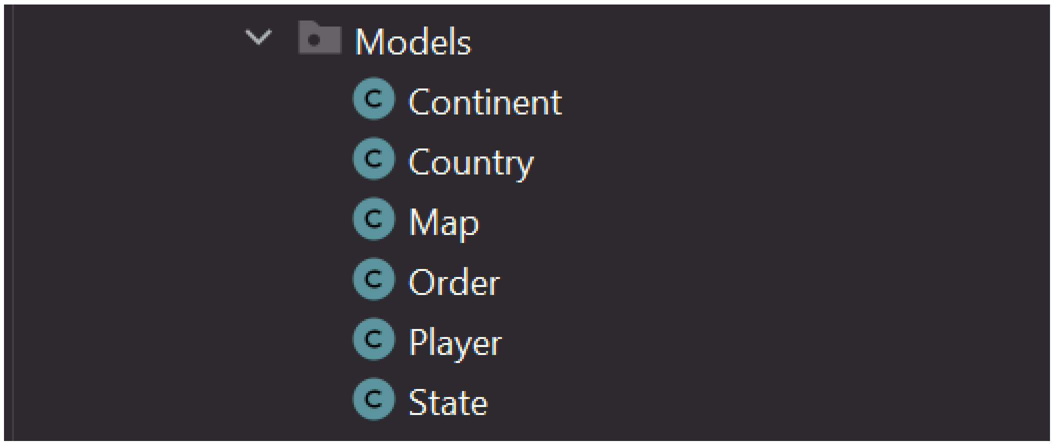
After Refactoring



```
6  usages   abhishekhandacse +1
public final class ApplicationConstantsHardcoding {

    1 usage
    public static final String VALID_MAP_MESSAGE = "The loaded map is valid!";

    17 usages
    public static final String ARGUMENTS_PASSED = "arguments";
    8 usages
    public static final String OPERATION_REQUESTED = "operation";

    public static final String EXTENSION_MAP_FILE = ".map";

    1 usage
    public static final String RED_COLOR = "\033[0;31m";
    1 usage
    public static final String GREEN_COLOR = "\033[0;32m";
    1 usage
    public static final String YELLOW_COLOR = "\033[0;33m";
    1 usage
    public static final String BLUE_COLOR = "\033[0;34m";
    1 usage
    public static final String PURPLE_COLOR = "\033[0;35m";
    1 usage
    public static final String CYAN_COLOR = "\033[0;36m";
    public static final String WHITE_COLOR = "\u001B[47m";

    2 usages
    public static final String ALL_CONTINENTS = "[continents]";
    3 usages
    public static final String ALL_COUNTRIES = "[countries]";
    3 usages
    public static final String ALL_BORDERS = "[borders]";
    1 usage
    public static final String ALL_ARMIES = "Armies";
    1 usage
    public static final String CONTINENT_CONTROL_VALUE = "Control Value";
    1 usage
    public static final String GRAPH_CONNECTIONS = "Connections";
    1 usage
    public static final String CLASSPATH_SRC_MAIN_RESOURCES = "src/main/resources";
    5 usages
    public static final int DISPLAY_WIDTH = 80;
```

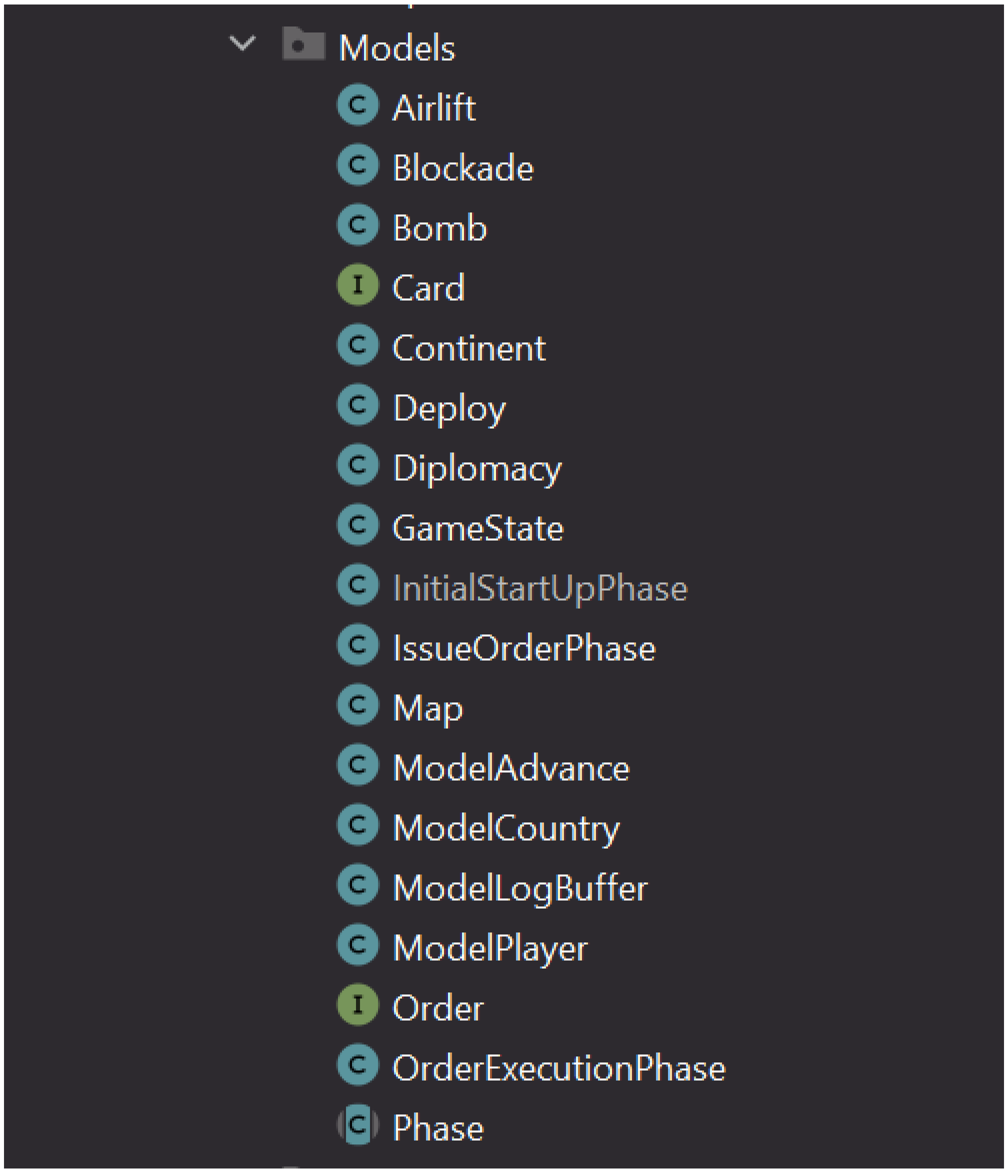
2. Implement State pattern for Phase Change:

Before Refactoring



After Refactoring

State pattern implemented and as asked in the requirements the separate classes created for InitialStartupPhase, IssueOrderPhase and OrderExecutionPhase.


```
Models
  Airlift
  Blockade
  Bomb
  Card
  Continent
  Deploy
  Diplomacy
  GameState
  InitialStartUpPhase
  IssueOrderPhase
  Map
  ModelAdvance
  ModelCountry
  ModelLogBuffer
  ModelPlayer
  Order
  OrderExecutionPhase
  Phase
```

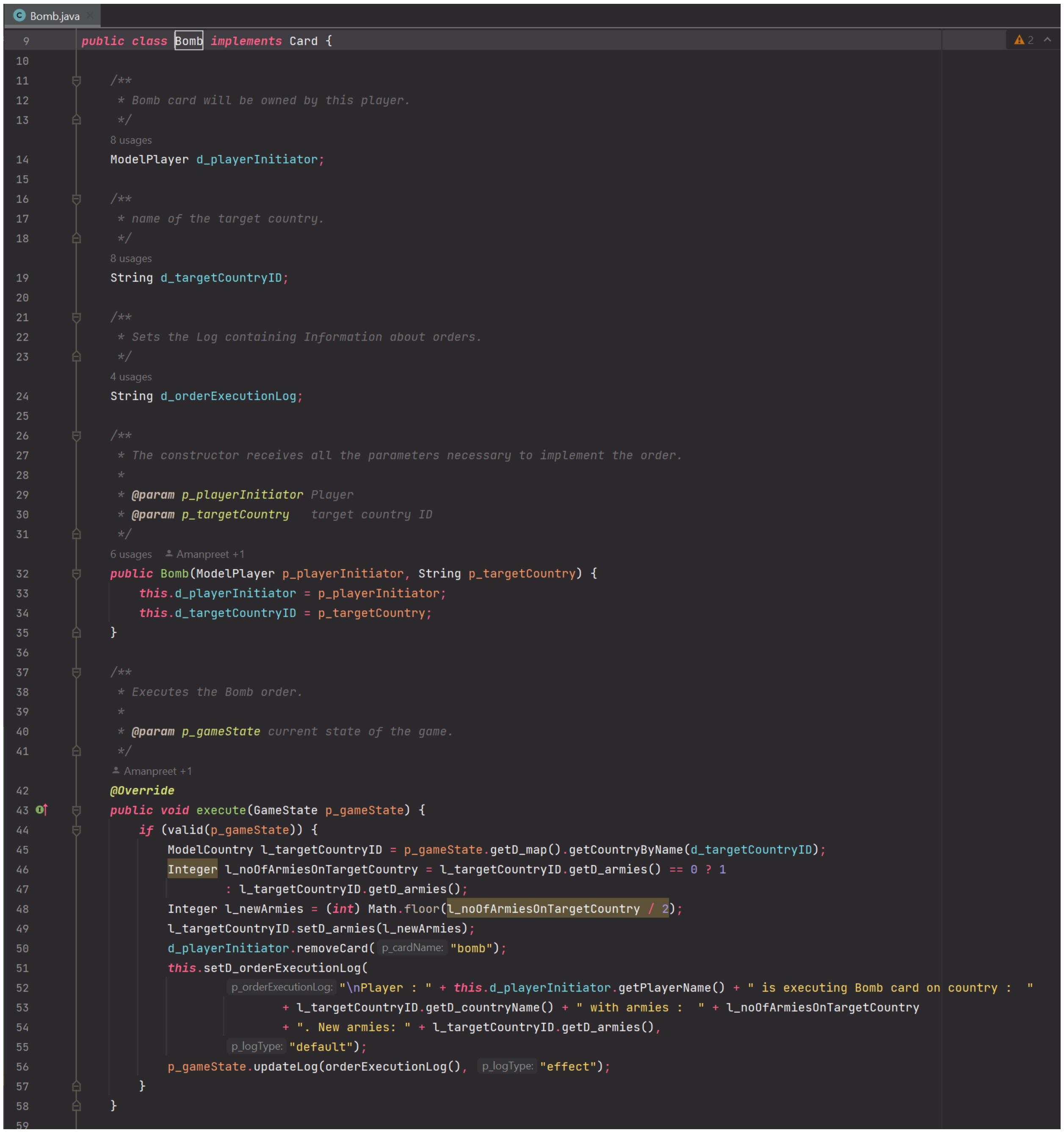
** For detailed class level description, refer to the codebase build2 snippet attached.

3. Implement command pattern for processing of orders

Before refactoring

In Build1, the processing of deploy order did not exactly follow the command pattern, there was a common single function execute that had the complete logic. In build 2, all of the Commands -Deploy, Advance, Bomb and Blockade, Airlife implements chain of commands and hence command pattern.

After refactoring



Bomb.java

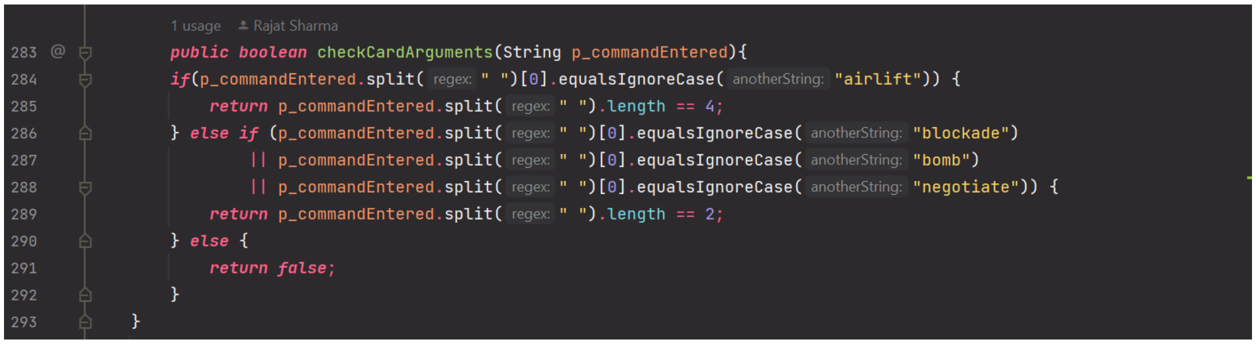
```java
public class Bomb implements Card {

    /**
     * Bomb card will be owned by this player.
     */
    8 usages
    ModelPlayer d_playerInitiator;

    /**
     * name of the target country.
     */
    8 usages
    String d_targetCountryID;

    /**
     * Sets the Log containing Information about orders.
     */
    4 usages
    String d_orderExecutionLog;

    /**
     * The constructor receives all the parameters necessary to implement the order.
     *
     * @param p_playerInitiator Player
     * @param p_targetCountry   target country ID
     */
    6 usages  Amanpreet +1
    public Bomb(ModelPlayer p_playerInitiator, String p_targetCountry) {
        this.d_playerInitiator = p_playerInitiator;
        this.d_targetCountryID = p_targetCountry;
    }

    /**
     * Executes the Bomb order.
     *
     * @param p_gameState current state of the game.
     */
    Amanpreet +1
    @Override
    public void execute(GameState p_gameState) {
        if (valid(p_gameState)) {
            ModelCountry l_targetCountryID = p_gameState.getD_map().getCountryByName(d_targetCountryID);
            Integer l_noOfArmiesOnTargetCountry = l_targetCountryID.getD_armies() == 0 ? 1
                    : l_targetCountryID.getD_armies();
            Integer l_newArmies = (int) Math.floor(l_noOfArmiesOnTargetCountry / 2);
            l_targetCountryID.setD_armies(l_newArmies);
            d_playerInitiator.removeCard( p_cardName: "bomb");
            this.setD_orderExecutionLog(
                    p_orderExecutionLog: "\nPlayer : " + this.d_playerInitiator.getPlayerName() + " is executing Bomb card on country :  "
                            + l_targetCountryID.getD_countryName() + " with armies :  " + l_noOfArmiesOnTargetCountry
                            + ". New armies: " + l_targetCountryID.getD_armies(),
                    p_logType: "default");
            p_gameState.updateLog(orderExecutionLog(), p_logType: "effect");
        }
    }
```

4. Implement Command syntax validation:

Before Refactoring:

In build 1, the validation for commands was not implemented strictly. In Build 2, we explicitly added the function ValidateCommand to check the validation for all the commands.

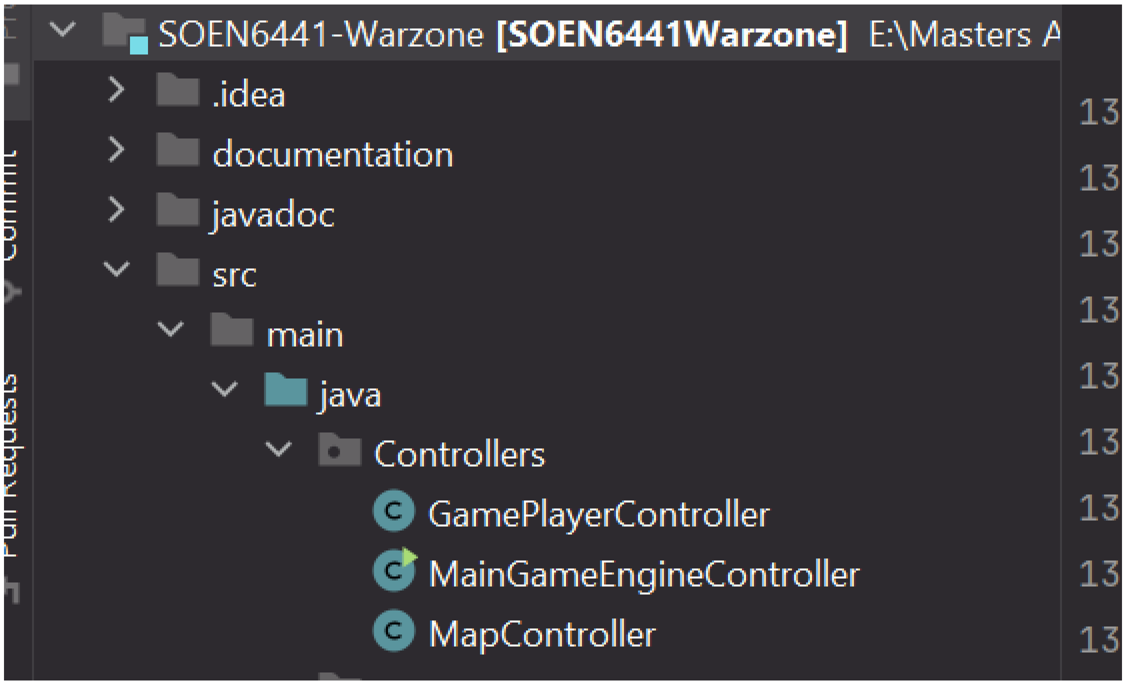
After Refactoring:



```
1 usage   Rajat Sharma
public boolean checkCardArguments(String p_commandEntered){
if(p_commandEntered.split( regex: " ")[0].equalsIgnoreCase( anotherString: "airlift")) {
    return p_commandEntered.split( regex: " ").length == 4;
} else if (p_commandEntered.split( regex: " ")[0].equalsIgnoreCase( anotherString: "blockade")
        || p_commandEntered.split( regex: " ")[0].equalsIgnoreCase( anotherString: "bomb")
        || p_commandEntered.split( regex: " ")[0].equalsIgnoreCase( anotherString: "negotiate")) {
    return p_commandEntered.split( regex: " ").length == 2;
} else {
    return false;
}
}
```

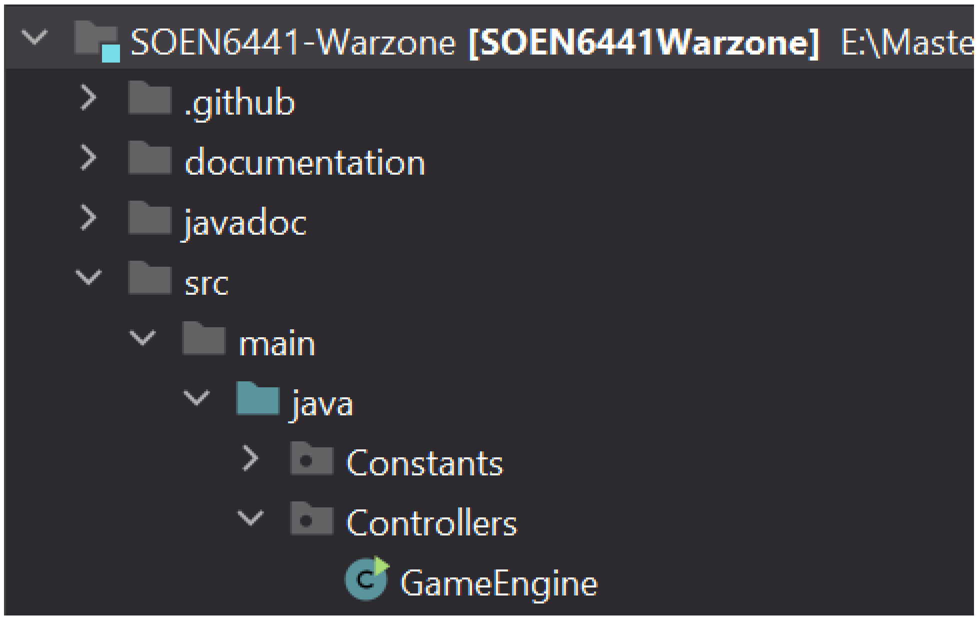
5. Expose only a single Controller rather than having redundant controllers in build 1

Before Refactoring:



```
SOEN6441-Warzone [SOEN6441Warzone]  E:\Masters A
  .idea
  documentation
  javadoc
  src
    main
      java
        Controllers
          GamePlayerController
          MainGameEngineController
          MapController
```

After Refactoring:

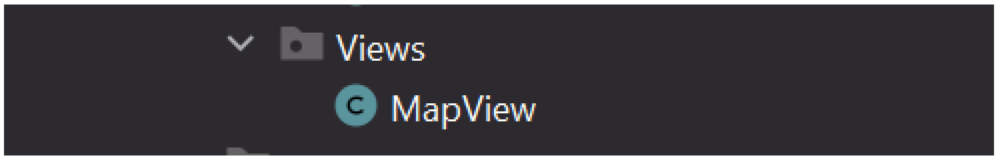


```
SOEN6441-Warzone [SOEN6441Warzone]  E:\Maste
  .github
  documentation
  javadoc
  src
    main
      java
        Constants
        Controllers
          GameEngine
```

** For detailed class level changes, refer to the code attached.

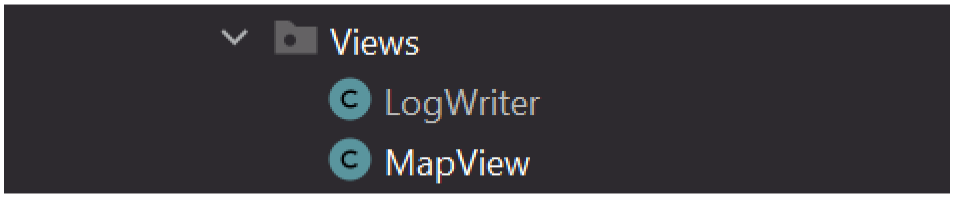
6. Make separate view for Log Writing to Console

Before refactoring:



Views
MapView

After refactoring:

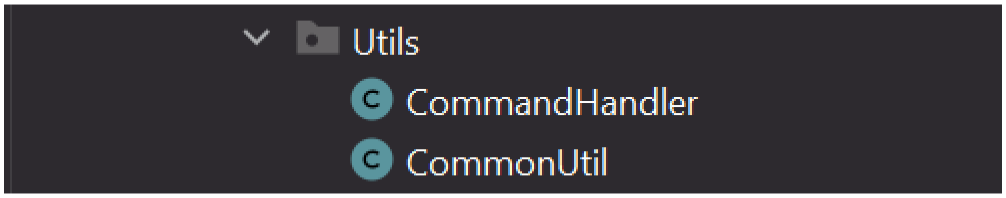


Views
LogWriter
MapView

** For detailed class level changes, refer to the code attached.

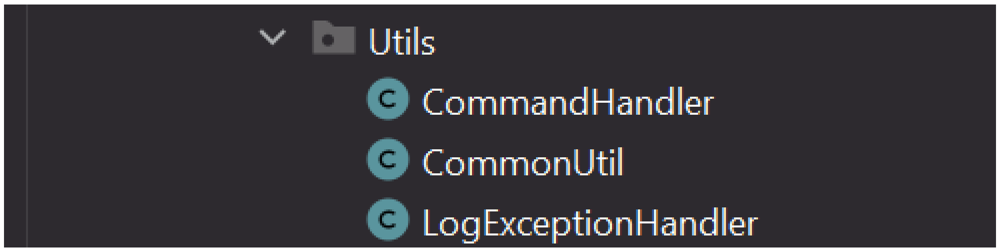
7. Introduced LogExceptionHandler as Utility class for logging correctly.

Before refactoring:



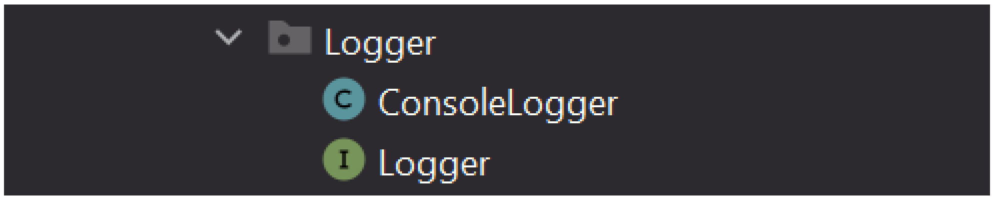
Utils
CommandHandler
CommonUtil

After refactoring:

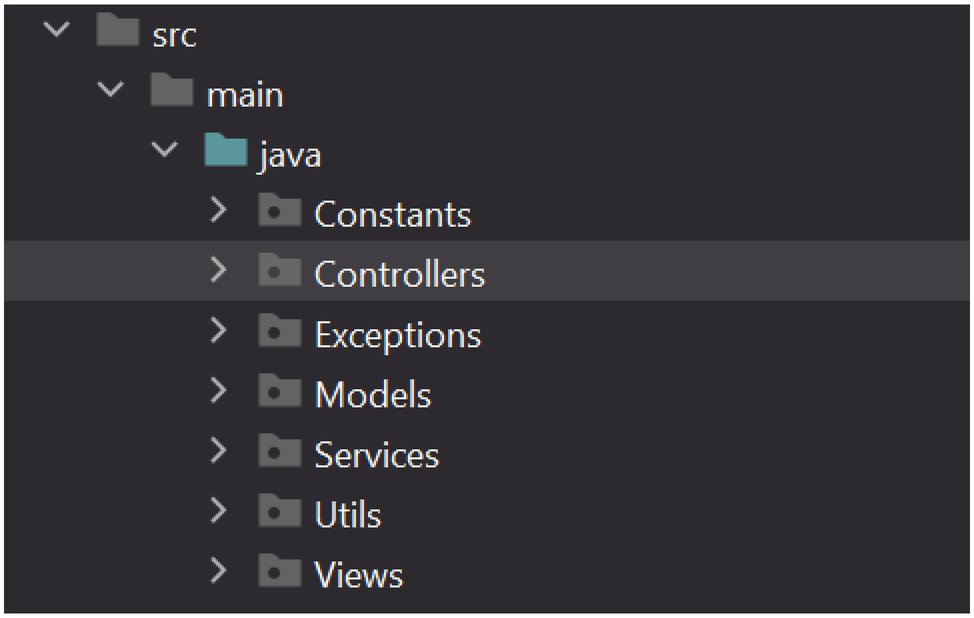


Utils
CommandHandler
CommonUtil
LogExceptionHandler

8. Removed Logger package
   Before refactoring:



   After refactoring:
   No Logger package


src
main
java
Constants
Controllers
Exceptions
Models
Services
Utils
Views