CONCORDIA UNIVERSITY

SOEN 6441- ADVANCED PROGRAMMING PRACTICES

PROJECT BUILD 2

Architecture Design Document

TEAM 20

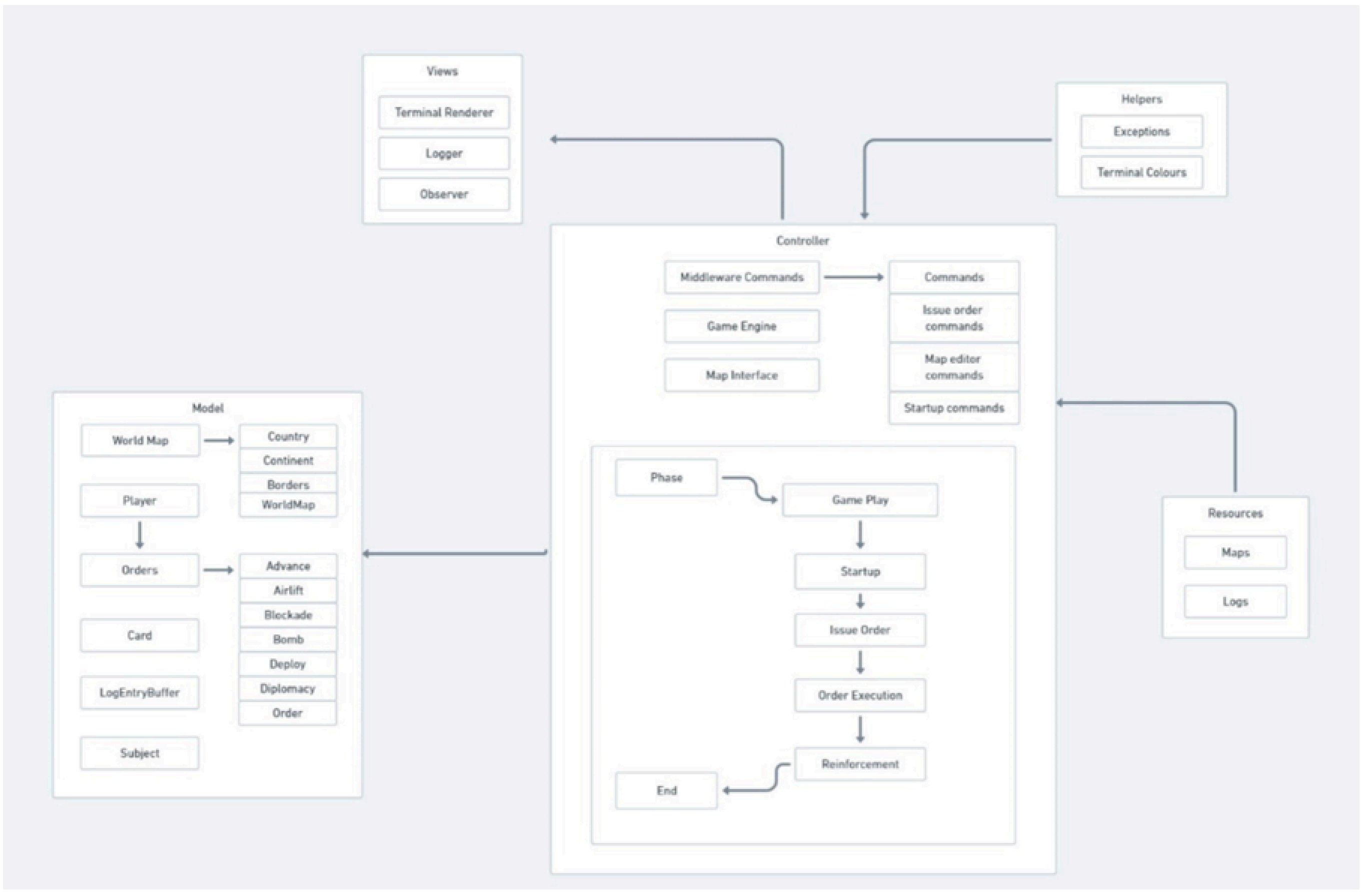
TEAM 20:

1. Devdutt Sharma
2. Priyanshu Adhikari
3. Shashidhar Krovvidi
4. Piyush Satti
5. Eden Almakias
6. Shamita Datta

IMPLEMENTING THE GAME

To accurately implement the game WARZONE, our code is broadly divided into five major components:

- Controller
- Model
- Views
- Helpers
- Resources



1. Controller

Controller contains several methods that are implemented throughout the game.

- **Starting**- the starting phase of the game represents the initial phase when the game begins. It begins with the options to enter the map phase or play phase.
- **Map Interface**- it provides the functionality to load an existing map from the file, save a map and validate the map.
- **Game Engine**- it manages the main logic of the game and handles the game phase, user input and game loop. It is like an interface that coordinates between the map editor and gameplay.
- **Game Play**- it is responsible to implement and execute the command as per the requirement of the player.
- **Middleware Commands**- these include checking the validity of the commands that are used in the startup phase, to edit the map and issue orders for each player.
- **End Phase**- it provides functionality that is specific to the end of the game.

2. Model

Model contains the following functionalities:

- **Worldmap**- it contains the information to validate the maps by checking that the country, continent and worldmap are as per the rules of the game.
- **Orders**- it executes correct orders given by the player such as deploy, bomb, airlift, diplomacy, blockade and advance.
- **Card**- it generates the cards randomly that can be used by a player.
- **Player**- it contains all information pertaining to the players of the game.
- **LogEntryBuffer**- this method is used to implement the generation of the game log file as part of the observer pattern.

3. View

Views contains three classes:

- **Terminal Renderer**- it provides methods for rendering the various components in a terminal interface such as messages to start the game, menu, displaying the map, and ending the game.
- **Logger**- it handles logging the messages to a file as part of the observer pattern.
- **Observer**- it defines the contract for objects that observe the changes in a subject as a part of the implementation of the observer pattern.

4. Helpers

There are two functions of the helper class:

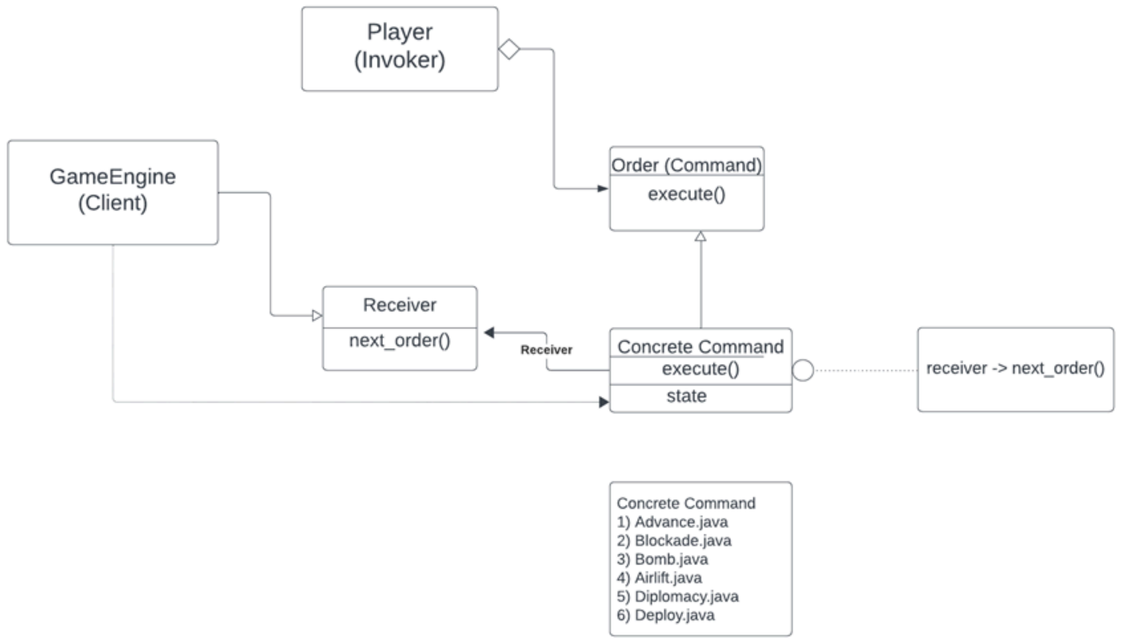
- **Exceptions**- this method is used to handle all the exceptions related to the players and maps and generate the related error command.
- **Terminal Colors**- it provides constants for the ANSI escape codes to represent text colors and background color for the terminal output

5. Resources

This part broadly consists of resources that help the proper functioning of the game.

- **Maps-** it contains several predesigned maps that are already loaded and can be used in the game. When a new map is created, that too is saved in the resources under maps.
- **Logs**- store all the actions that have taken place during the game in each phase of the game.

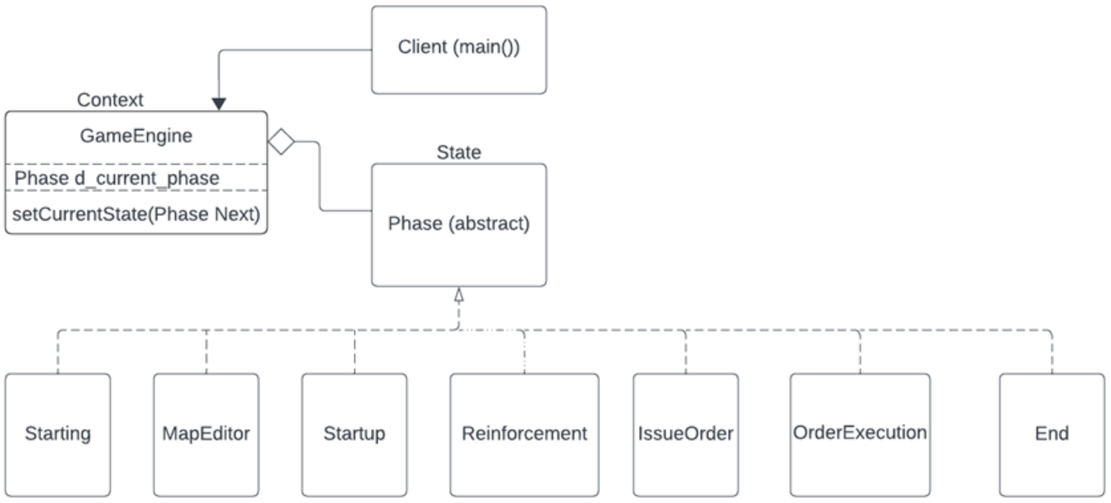
Implementation of Command Pattern



- The above shows how the implementation of the command pattern has been included in the code

- The Command Pattern is a behavioral design pattern that encapsulates a request as an object, thereby allowing parameterization of clients with commands and operations. It enables the separation of the client who sends the command from the execution of the action as per the command.
- The Player class acts as the invoker, Order as the command and GameEngine as the client.
- During the game, as the orders are generated as the player issues orders, these orders are then executed in the game engine as it gets the orders from the players. To execute these orders, the concrete command executes.
- For implementation of the command pattern, different concrete commands are created for the different orders that a player can give in the game:
    - **deploy**: place some armies on one of the current player's territories.
    - **advance**: move some armies from one of the current player's territories (source) to an adjacent territory (target). If the target territory belongs to the current player, the armies are moved to the target territory. If the target territory belongs to another player, an attack happens between the two territories.
    - **bomb**: destroy half of the armies located on an opponent's territory that is adjacent to one of the current player's territories.
    - **blockade**: triple the number of armies on one of the current player's territories and make it a neutral territory.
    - **airlift**: advance some armies from one of the current player's territories to any another territory.
    - **negotiate**: prevent attacks between the current player and another player until the end of the turn.

Implementation of State Pattern



- The above shows how the implementation of the state pattern has been included in the code
- The key idea behind the State Pattern is to encapsulate the behavior associated with each state into separate classes, thus making it easier to add new states or modify existing ones without modifying the context class.
- The Phase method acts as the state class, Game Engine as context and main as client.
- In following the state pattern, the game engine changes its behavior based on the internal state.
- For implementation of the state pattern, different commands are created for the different phases that occur throughout the game:
    - Starting.java
    - MapEditor.java
    - Startup.java
    - Reinforcement.java
    - IssueOrder.java
    - End.java

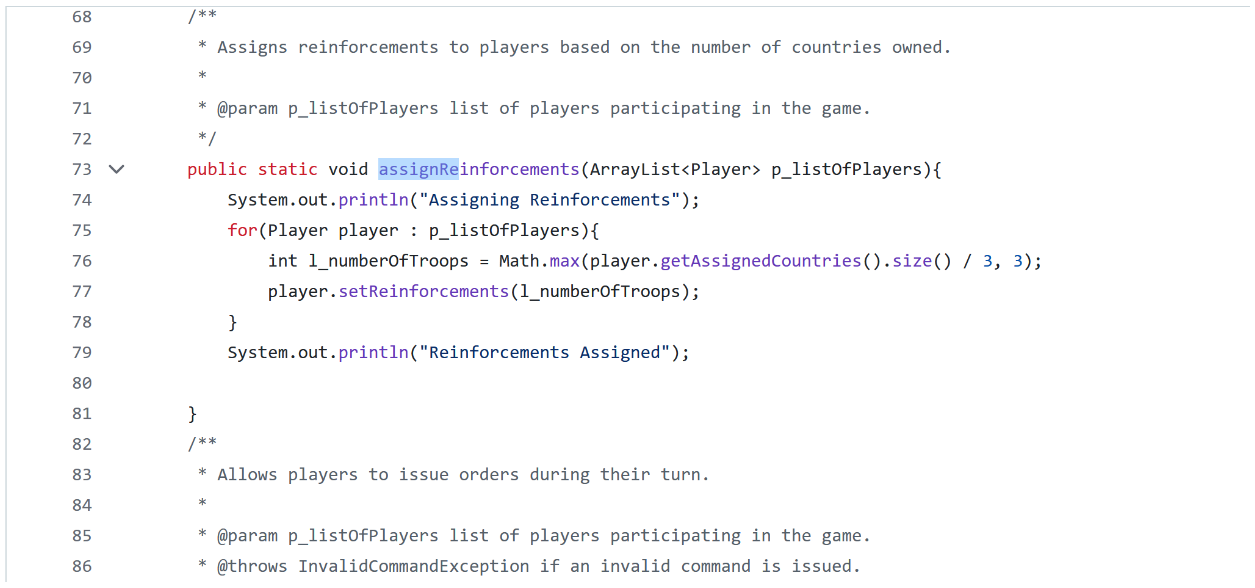
List of 15 Identified Refactoring Targets:

1. Implement state pattern.
2. Implement observer pattern.
3. Refactor assignReinforcements() in Reinforcement.java (Extract method) shifted from PlayGame.java to Reinforcement.java)
4. Refactor CommandValidator.java and split it into separate classes to show the same behavior for validating the user input commands to reduce the number of lines of code,duplicate code and improve code readability.
5. Refactor addCountry() in the WorldMap.java
6. Using common methods to implement editCountryValidator and editContinentValidator (MapEditorCommands.java) to reduce code duplication.
7. Implement the method execute (IssueOrderCommands.java) with the use of functions for the commands to improve code readability.
8. Replace Boolean flags in loadmap (MapInterface.java) with enums to make the code clearer.
9. Implement checkCommandValidity (CommandValidator.java) with switch statements instead of multiple if-else blocks.
10. Create common methods for processValidCommand (CommandValidator.java) to reduce code duplication.
11. Create separate methods to add and remove continents and handle exceptions in editContinents (MapEditorComment.java)
12. Create separate methods to add and remove countries and handle exceptions in editCountry (MapEditorComment.java)
13. Create separate methods to add and remove borders and handle exceptions in editNeighbor (MapEditorComment.java)
14. Create separate methods to load map and handle exceptions in editMap (MapEditorComments.java).
15. Rename method parameter p_GE to p_gameEngine to follow coding conventions.
16. Rename method parameter p_WM to p_worldMap to follow coding conventions.
17. Refactored issue_order() in player.java

Performed Refactorings:

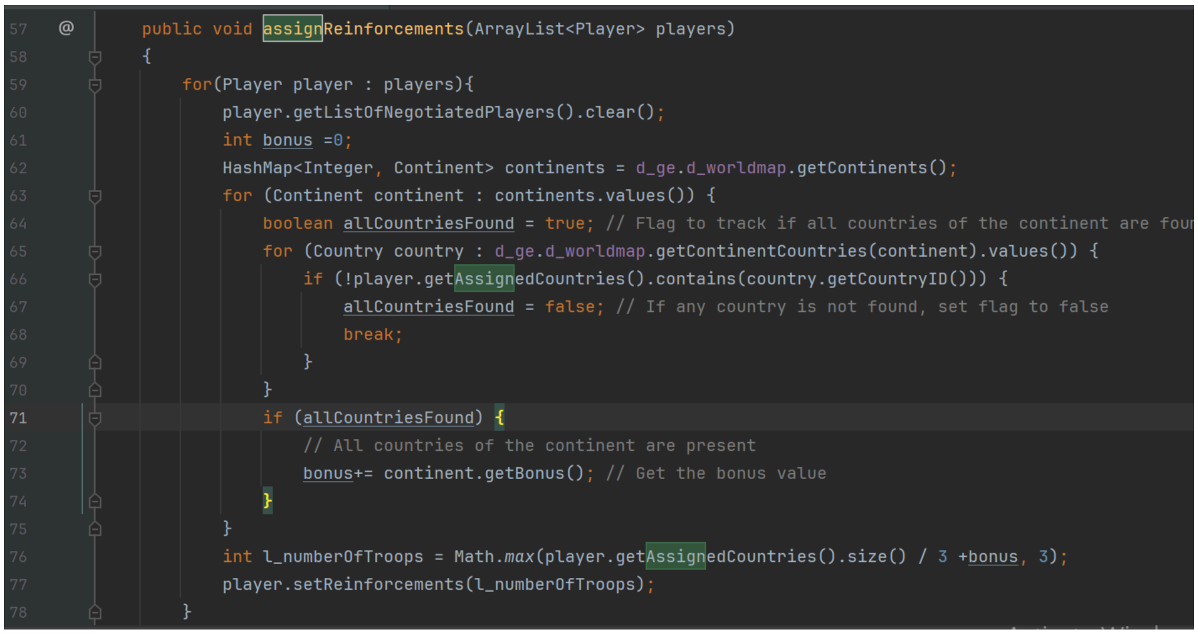
1. Refactor assignReinforcements() in Reinforcement.java (Extract method, could shift from PlayGame.java to Reinforcement.java)

Before:
This refactoring operation was deemed necessary



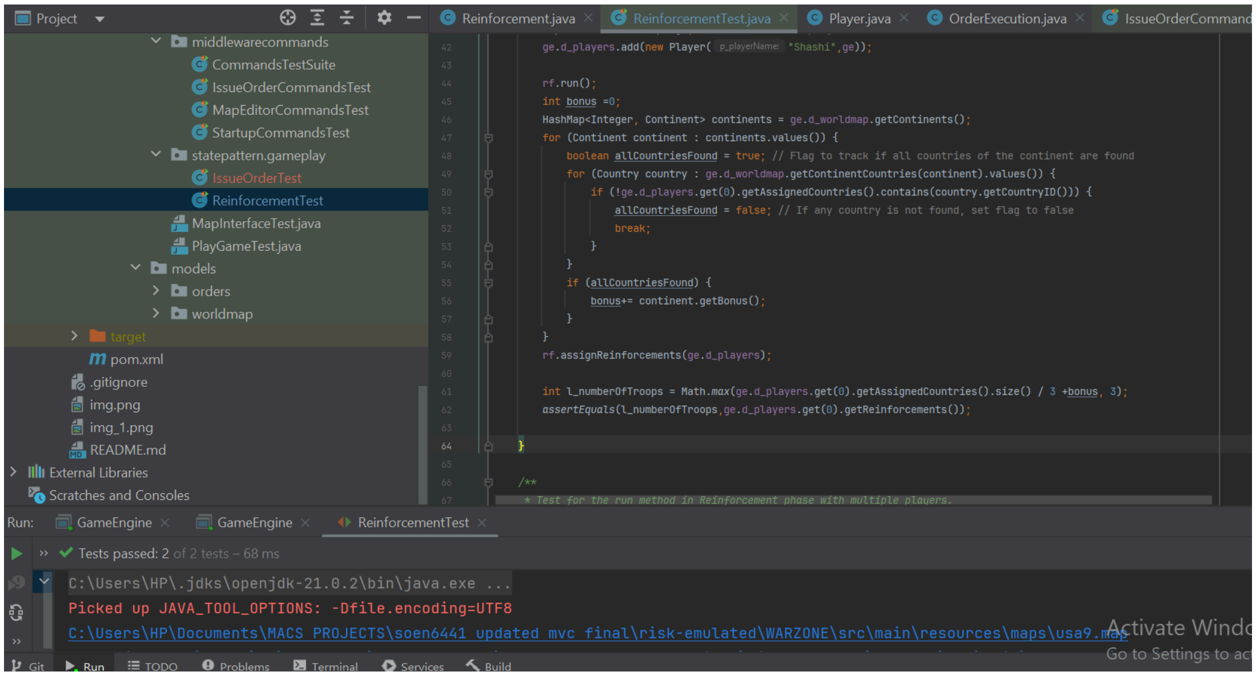
```java
/**
 * Assigns reinforcements to players based on the number of countries owned.
 *
 * @param p_listOfPlayers list of players participating in the game.
 */
public static void assignReinforcements(ArrayList<Player> p_listOfPlayers){
    System.out.println("Assigning Reinforcements");
    for(Player player : p_listOfPlayers){
        int l_numberOfTroops = Math.max(player.getAssignedCountries().size() / 3, 3);
        player.setReinforcements(l_numberOfTroops);
    }
    System.out.println("Reinforcements Assigned");

}
/**
 * Allows players to issue orders during their turn.
 *
 * @param p_listOfPlayers list of players participating in the game.
 * @throws InvalidCommandException if an invalid command is issued.
```

After:



```java
public void assignReinforcements(ArrayList<Player> players)
{
    for(Player player : players){
        player.getListOfNegotiatedPlayers().clear();
        int bonus =0;
        HashMap<Integer, Continent> continents = d_ge.d_worldmap.getContinents();
        for (Continent continent : continents.values()) {
            boolean allCountriesFound = true; // Flag to track if all countries of the continent are fou
            for (Country country : d_ge.d_worldmap.getContinentCountries(continent).values()) {
                if (!player.getAssignedCountries().contains(country.getCountryID())) {
                    allCountriesFound = false; // If any country is not found, set flag to false
                    break;
                }
            }
            if (allCountriesFound) {
                // All countries of the continent are present
                bonus+= continent.getBonus(); // Get the bonus value
            }
        }
        int l_numberOfTroops = Math.max(player.getAssignedCountries().size() / 3 +bonus, 3);
        player.setReinforcements(l_numberOfTroops);
    }
```

Junit Test Results:



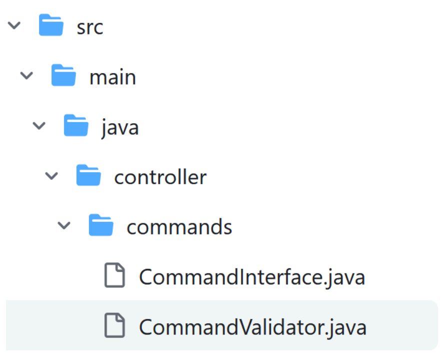
```java
        ge.d_players.add(new Player( p_playerName: "Shashi",ge));

        rf.run();
        int bonus =0;
        HashMap<Integer, Continent> continents = ge.d_worldmap.getContinents();
        for (Continent continent : continents.values()) {
            boolean allCountriesFound = true; // Flag to track if all countries of the continent are found
            for (Country country : ge.d_worldmap.getContinentCountries(continent).values()) {
                if (!ge.d_players.get(0).getAssignedCountries().contains(country.getCountryID())) {
                    allCountriesFound = false; // If any country is not found, set flag to false
                    break;
                }
            }
            if (allCountriesFound) {
                bonus+= continent.getBonus();
            }
        }
        rf.assignReinforcements(ge.d_players);

        int l_numberOfTroops = Math.max(ge.d_players.get(0).getAssignedCountries().size() / 3 +bonus, 3);
        assertEquals(l_numberOfTroops,ge.d_players.get(0).getReinforcements());

    }

    /**
     * Test for the run method in Reinforcement phase with multiple players.
```

```
Tests passed: 2 of 2 tests – 68 ms
C:\Users\HP\.jdks\openjdk-21.0.2\bin\java.exe ...
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF8
C:\Users\HP\Documents\MACS PROJECTS\soen6441 updated mvc final\risk-emulated\WARZONE\src\main\resources\maps\usa9.map
```

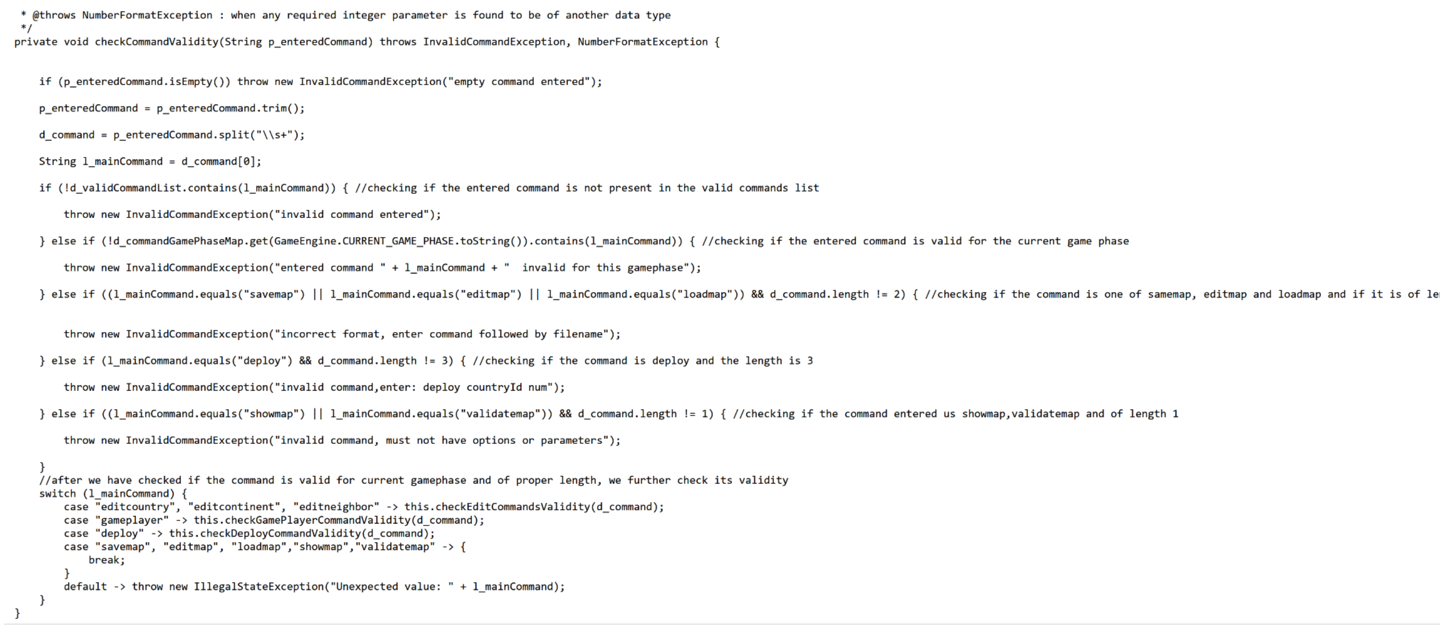
2. **Refactor CommandValidator.java and split it into separate classes to show the same behavior for validating the user input commands to reduce the number of lines of code,duplicate code and improve code readability. Also, integrate CommandInterface.java class's functionality into these new splitted classes.**

This refactoring operation was deemed necessary due to the complexity of the code in CommandValidator.java and CommandInterface.java and large number of lines of code and existence of duplicate code.

Before:

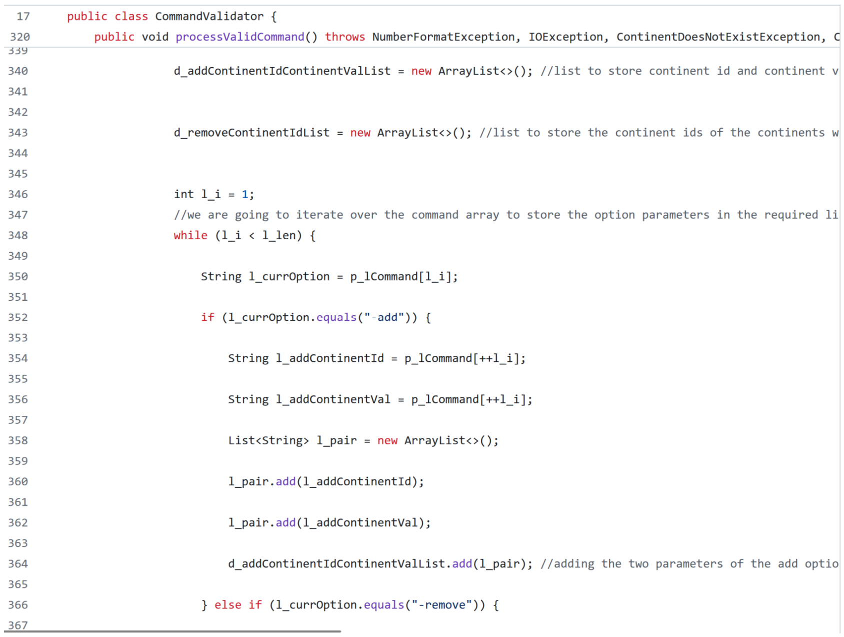


```
src
  main
    java
      controller
        commands
          CommandInterface.java
          CommandValidator.java
```



```
 * @throws NumberFormatException : when any required integer parameter is found to be of another data type
 */
private void checkCommandValidity(String p_enteredCommand) throws InvalidCommandException, NumberFormatException {


    if (p_enteredCommand.isEmpty()) throw new InvalidCommandException("empty command entered");

    p_enteredCommand = p_enteredCommand.trim();

    d_command = p_enteredCommand.split("\\s+");

    String l_mainCommand = d_command[0];

    if (!d_validCommandList.contains(l_mainCommand)) { //checking if the entered command is not present in the valid commands list

        throw new InvalidCommandException("invalid command entered");

    } else if (!d_commandGamePhaseMap.get(GameEngine.CURRENT_GAME_PHASE.toString()).contains(l_mainCommand)) { //checking if the entered command is valid for the current game phase

        throw new InvalidCommandException("entered command " + l_mainCommand + "  invalid for this gamephase");

    } else if ((l_mainCommand.equals("savemap") || l_mainCommand.equals("editmap") || l_mainCommand.equals("loadmap")) && d_command.length != 2) { //checking if the command is one of samemap, editmap and loadmap and if it is of le


        throw new InvalidCommandException("incorrect format, enter command followed by filename");

    } else if (l_mainCommand.equals("deploy") && d_command.length != 3) { //checking if the command is deploy and the length is 3

        throw new InvalidCommandException("invalid command,enter: deploy countryId num");

    } else if ((l_mainCommand.equals("showmap") || l_mainCommand.equals("validatemap")) && d_command.length != 1) { //checking if the command entered us showmap,validatemap and of length 1

        throw new InvalidCommandException("invalid command, must not have options or parameters");

    }
    //after we have checked if the command is valid for current gamephase and of proper length, we further check its validity
    switch (l_mainCommand) {
        case "editcountry", "editcontinent", "editneighbor" -> this.checkEditCommandsValidity(d_command);
        case "gameplayer" -> this.checkGamePlayerCommandValidity(d_command);
        case "deploy" -> this.checkDeployCommandValidity(d_command);
        case "savemap", "editmap", "loadmap","showmap","validatemap" -> {
            break;
        }
        default -> throw new IllegalStateException("Unexpected value: " + l_mainCommand);
    }
}
```

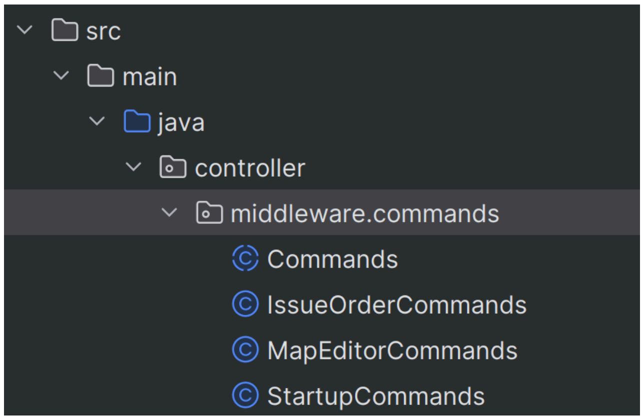
Above snippet is an example of code which is long and has multiple if/else conditions



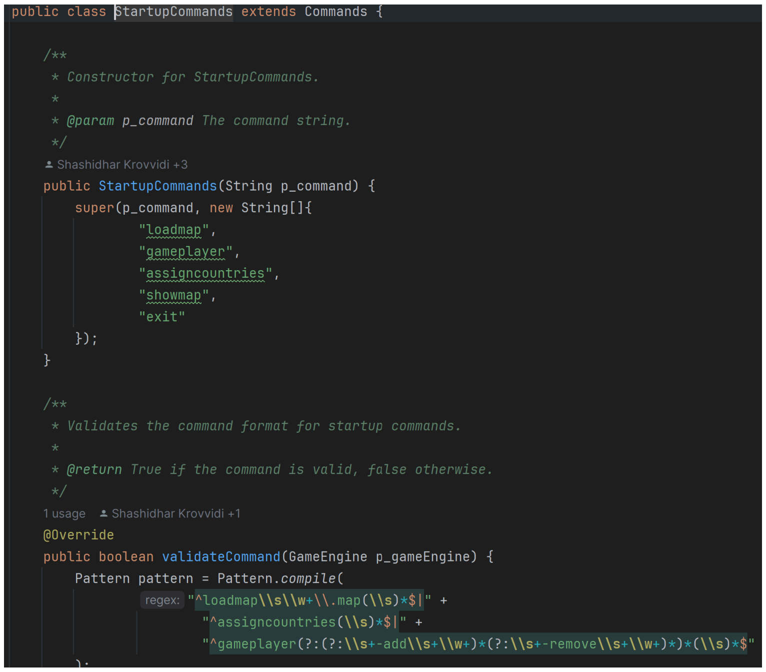
```
public class CommandValidator {
    public void processValidCommand() throws NumberFormatException, IOException, ContinentDoesNotExistException, C

                d_addContinentIdContinentValList = new ArrayList<>(); //list to store continent id and continent v


                d_removeContinentIdList = new ArrayList<>(); //list to store the continent ids of the continents w


                int l_i = 1;
                //we are going to iterate over the command array to store the option parameters in the required li
                while (l_i < l_len) {

                    String l_currOption = p_lCommand[l_i];

                    if (l_currOption.equals("-add")) {

                        String l_addContinentId = p_lCommand[++l_i];

                        String l_addContinentVal = p_lCommand[++l_i];

                        List<String> l_pair = new ArrayList<>();

                        l_pair.add(l_addContinentId);

                        l_pair.add(l_addContinentVal);

                        d_addContinentIdContinentValList.add(l_pair); //adding the two parameters of the add optio

                    } else if (l_currOption.equals("-remove")) {
```

Above piece of code is an example of code smell since it has been used at different places inside processValidCommand() for creating lists to store parameters of add and remove options of the editcountry,editcontinent and editneighbor commands.

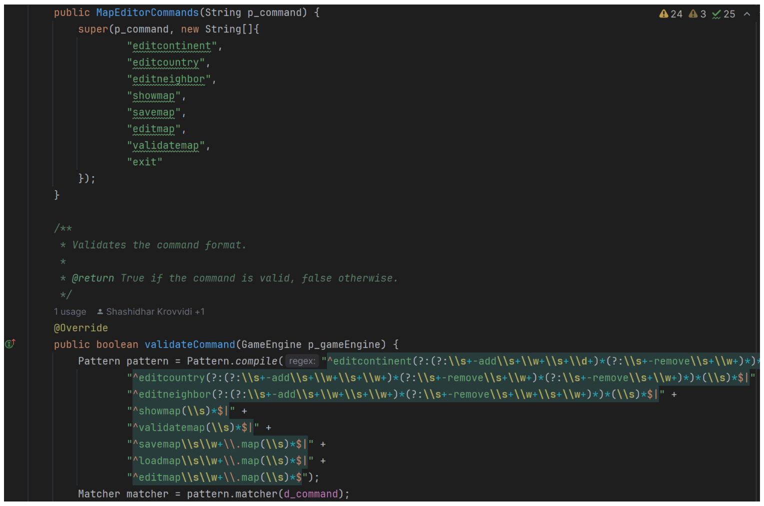
After:



```
src
  main
    java
      controller
        middleware.commands
          Commands
          IssueOrderCommands
          MapEditorCommands
          StartupCommands
```

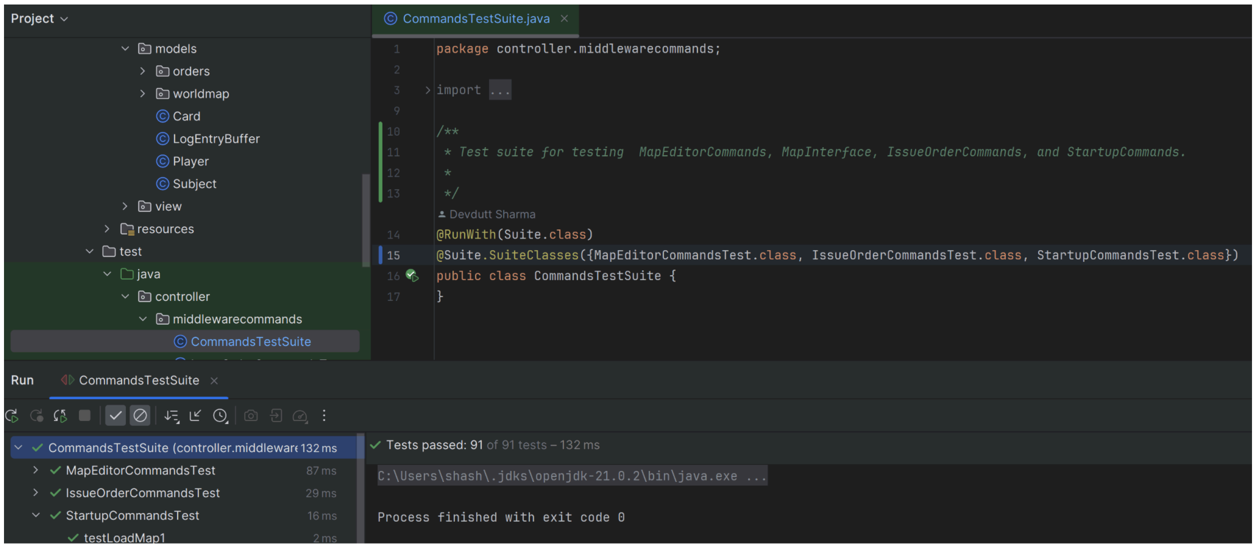


```java
public class StartupCommands extends Commands {

    /**
     * Constructor for StartupCommands.
     *
     * @param p_command The command string.
     */
    Shashidhar Krovvidi +3
    public StartupCommands(String p_command) {
        super(p_command, new String[]{
                "loadmap",
                "gameplayer",
                "assigncountries",
                "showmap",
                "exit"
        });
    }

    /**
     * Validates the command format for startup commands.
     *
     * @return True if the command is valid, false otherwise.
     */
    1 usage   Shashidhar Krovvidi +1
    @Override
    public boolean validateCommand(GameEngine p_gameEngine) {
        Pattern pattern = Pattern.compile(
                regex: "^loadmap\\s\\w+\\.map(\\s)*$|" +
                        "^assigncountries(\\s)*$|" +
                        "^gameplayer(?:(?:\\s+-add\\s+\\w+)*(?:\\s+-remove\\s+\\w+)*)*(\\s)*$"
        );
```



```java
    public MapEditorCommands(String p_command) {                              ▲24 ▲3 ✓25
        super(p_command, new String[]{
                "editcontinent",
                "editcountry",
                "editneighbor",
                "showmap",
                "savemap",
                "editmap",
                "validatemap",
                "exit"
        });
    }

    /**
     * Validates the command format.
     *
     * @return True if the command is valid, false otherwise.
     */
    1 usage   Shashidhar Krovvidi +1
    @Override
    public boolean validateCommand(GameEngine p_gameEngine) {
        Pattern pattern = Pattern.compile( regex: "^editcontinent(?:(?:\\s+-add\\s+\\w+\\s+\\d+)*(?:\\s+-remove\\s+\\w+)*)*
                "^editcountry(?:(?:\\s+-add\\s+\\w+\\s+\\w+)*(?:\\s+-remove\\s+\\w+)*(?:\\s+-remove\\s+\\w+)*)*(\\s)*$|"
                "^editneighbor(?:(?:\\s+-add\\s+\\w+\\s+\\w+)*(?:\\s+-remove\\s+\\w+\\s+\\w+)*)*(\\s)*$|" +
                "^showmap(\\s)*$|" +
                "^validatemap(\\s)*$|" +
                "^savemap\\s\\w+\\.map(\\s)*$|" +
                "^loadmap\\s\\w+\\.map(\\s)*$|" +
                "^editmap\\s\\w+\\.map(\\s)*$");
        Matcher matcher = pattern.matcher(d_command);
```

Junit Test Results:



```java
package controller.middlewarecommands;

import ...

/**
 * Test suite for testing  MapEditorCommands, MapInterface, IssueOrderCommands, and StartupCommands.
 *
 */
Devdutt Sharma
@RunWith(Suite.class)
@Suite.SuiteClasses({MapEditorCommandsTest.class, IssueOrderCommandsTest.class, StartupCommandsTest.class})
public class CommandsTestSuite {
}
```

```
✓ CommandsTestSuite (controller.middleware 132 ms
  ✓ MapEditorCommandsTest   87 ms
  ✓ IssueOrderCommandsTest  29 ms
  ✓ StartupCommandsTest     16 ms
    ✓ testLoadMap1           2 ms
```

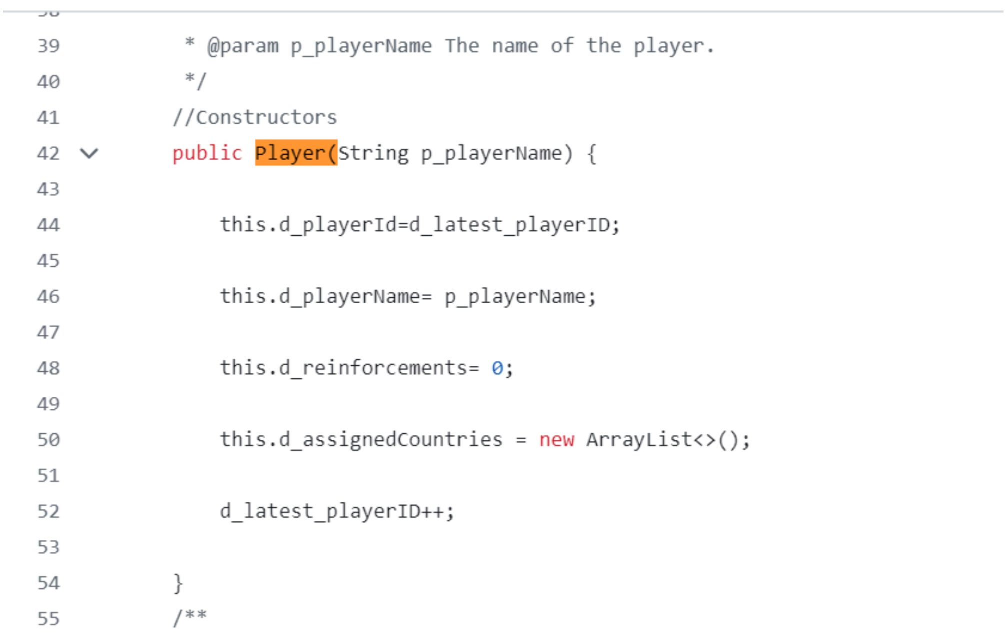
```
✓ Tests passed: 91 of 91 tests – 132 ms

C:\Users\shash\.jdks\openjdk-21.0.2\bin\java.exe ...

Process finished with exit code 0
```

—---

3. Refactored Player class constructor.

This refactoring was deemed necessary to implement functionalities of cards such as airlift, negotiate,etc.

Before:

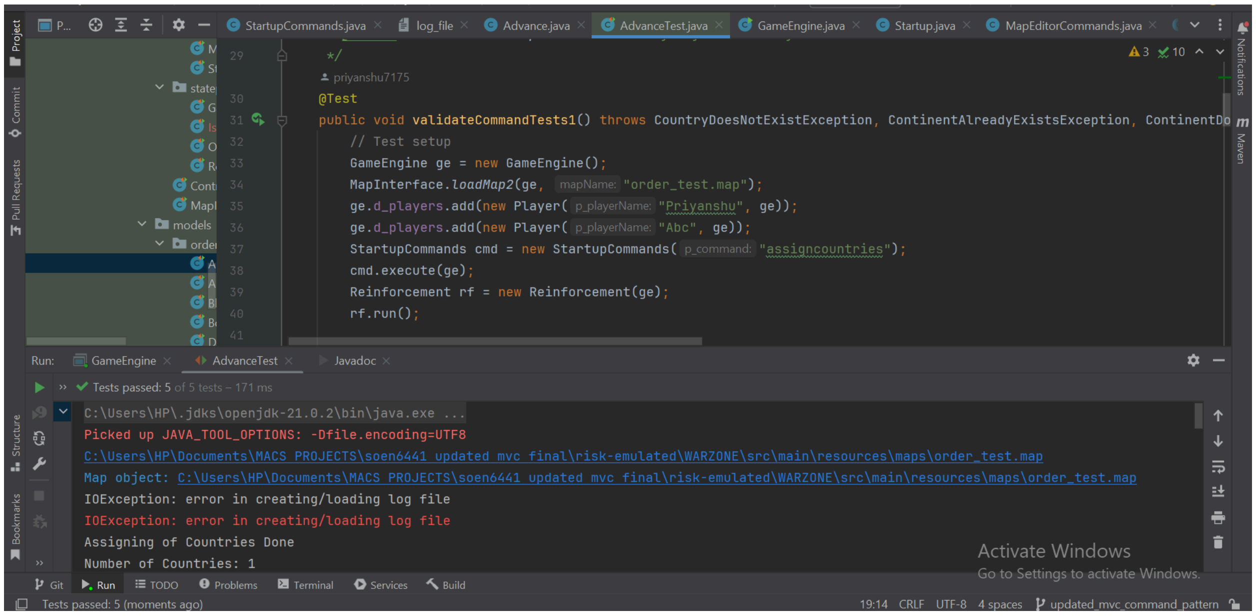


```java
     * @param p_playerName The name of the player.
     */
    //Constructors
    public Player(String p_playerName) {

        this.d_playerId=d_latest_playerID;

        this.d_playerName= p_playerName;

        this.d_reinforcements= 0;

        this.d_assignedCountries = new ArrayList<>();

        d_latest_playerID++;

    }
    /**
```

After:

```
    ,
//Constructors
public Player(String p_playerName, GameEngine p_gameEngine) {

    this.d_playerId = d_latest_playerID;
    this.d_playerName = p_playerName;
    this.d_reinforcements = 0;
    this.d_assignedCountries = new ArrayList<>();
    this.d_listOfCards = new ArrayList<>();
    this.d_listOfNegotiatedPlayers = new ArrayList<>();
    this.d_gameEngine = p_gameEngine;
    this.d_terminalRenderer = new TerminalRenderer(this.d_gameEngine);
    this.d_orderSuccess = false;
    this.d_finishedIssueOrder = false;
    d_latest_playerID++;
}
```

Junit Test Results:



```
 */
priyanshu7175
@Test
public void validateCommandTests1() throws CountryDoesNotExistException, ContinentAlreadyExistsException, ContinentDo
    // Test setup
    GameEngine ge = new GameEngine();
    MapInterface.loadMap2(ge, mapName: "order_test.map");
    ge.d_players.add(new Player( p_playerName: "Priyanshu", ge));
    ge.d_players.add(new Player( p_playerName: "Abc", ge));
    StartupCommands cmd = new StartupCommands( p_command: "assigncountries");
    cmd.execute(ge);
    Reinforcement rf = new Reinforcement(ge);
    rf.run();
```

Tests passed: 5 of 5 tests – 171 ms

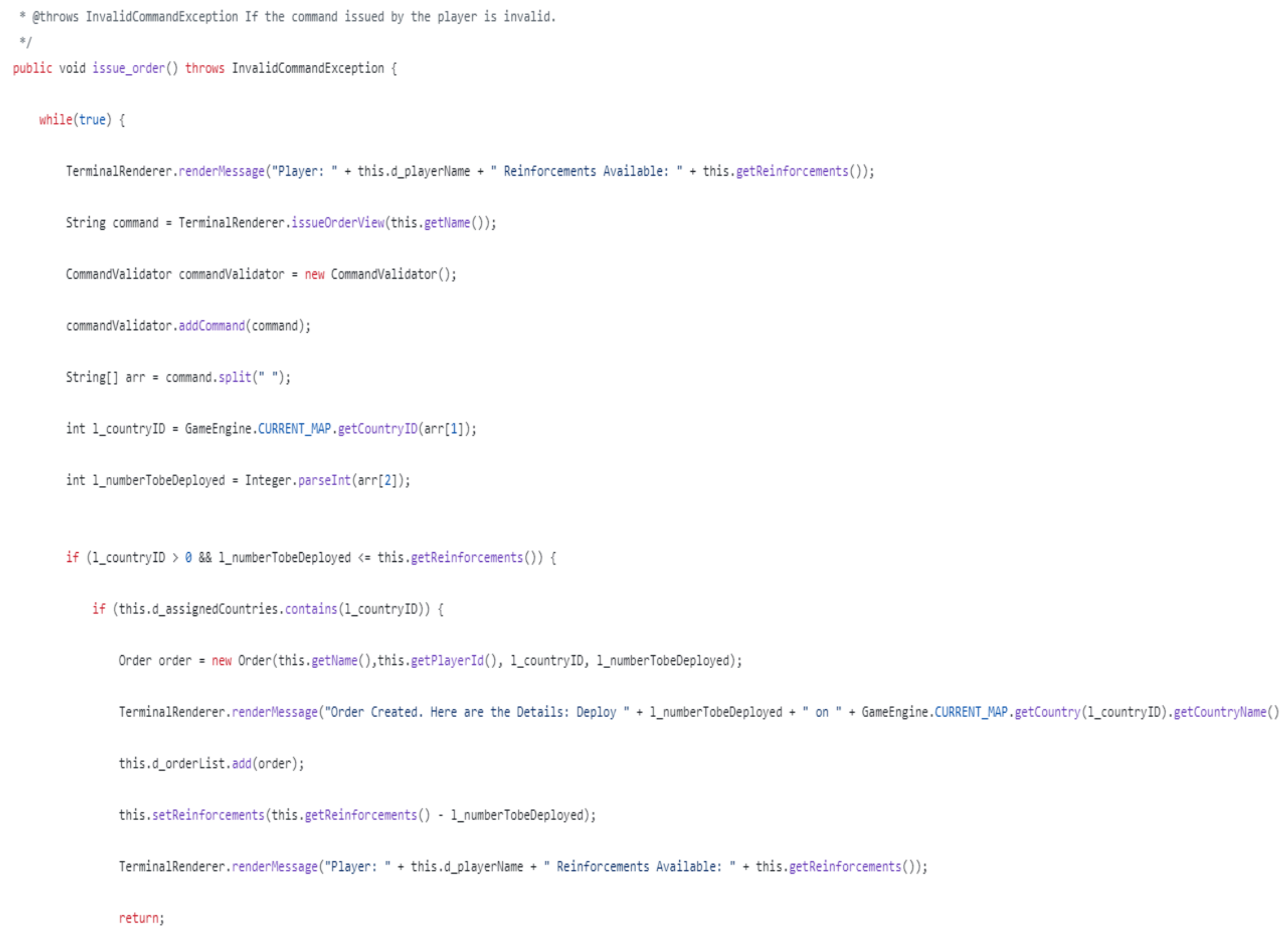
```
C:\Users\HP\.jdks\openjdk-21.0.2\bin\java.exe ...
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF8
C:\Users\HP\Documents\MACS_PROJECTS\soen6441_updated_mvc_final\risk-emulated\WARZONE\src\main\resources\maps\order_test.map
Map object: C:\Users\HP\Documents\MACS_PROJECTS\soen6441_updated_mvc_final\risk-emulated\WARZONE\src\main\resources\maps\order_test.map
IOException: error in creating/loading log file
IOException: error in creating/loading log file
Assigning of Countries Done
Number of Countries: 1
```

—--------------------------

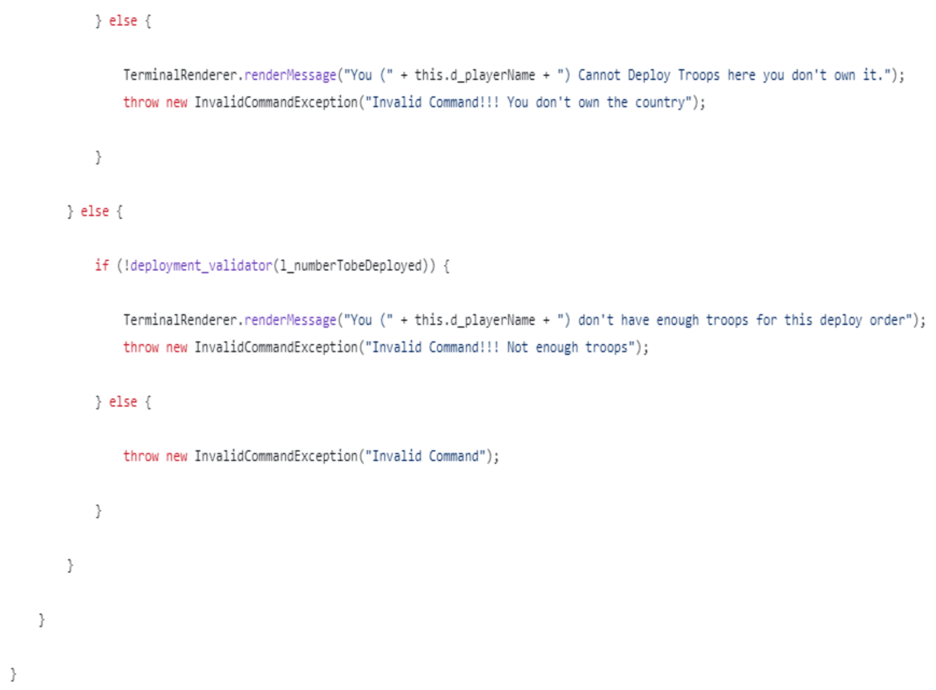
4. Refactored issue_order() in player.java

This refactoring done in order to implement command pattern in a better way by moving some logic from inside of issue_order to outside. Enhancing readability of the issue_order method.

Before:

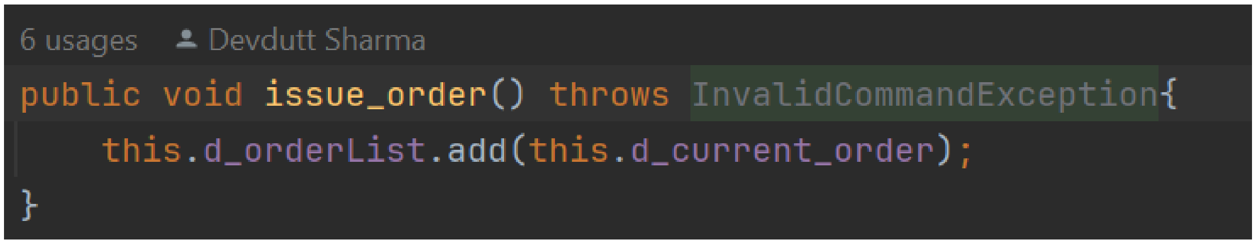


```
 * @throws InvalidCommandException If the command issued by the player is invalid.
 */
public void issue_order() throws InvalidCommandException {

    while(true) {

        TerminalRenderer.renderMessage("Player: " + this.d_playerName + " Reinforcements Available: " + this.getReinforcements());

        String command = TerminalRenderer.issueOrderView(this.getName());

        CommandValidator commandValidator = new CommandValidator();

        commandValidator.addCommand(command);

        String[] arr = command.split(" ");

        int l_countryID = GameEngine.CURRENT_MAP.getCountryID(arr[1]);

        int l_numberTobeDeployed = Integer.parseInt(arr[2]);


        if (l_countryID > 0 && l_numberTobeDeployed <= this.getReinforcements()) {

            if (this.d_assignedCountries.contains(l_countryID)) {

                Order order = new Order(this.getName(),this.getPlayerId(), l_countryID, l_numberTobeDeployed);

                TerminalRenderer.renderMessage("Order Created. Here are the Details: Deploy " + l_numberTobeDeployed + " on " + GameEngine.CURRENT_MAP.getCountry(l_countryID).getCountryName()

                this.d_orderList.add(order);

                this.setReinforcements(this.getReinforcements() - l_numberTobeDeployed);

                TerminalRenderer.renderMessage("Player: " + this.d_playerName + " Reinforcements Available: " + this.getReinforcements());

                return;
```



```
            } else {

                TerminalRenderer.renderMessage("You (" + this.d_playerName + ") Cannot Deploy Troops here you don't own it.");
                throw new InvalidCommandException("Invalid Command!!! You don't own the country");

            }

        } else {

            if (!deployment_validator(l_numberTobeDeployed)) {

                TerminalRenderer.renderMessage("You (" + this.d_playerName + ") don't have enough troops for this deploy order");
                throw new InvalidCommandException("Invalid Command!!! Not enough troops");

            } else {

                throw new InvalidCommandException("Invalid Command");

            }

        }

    }

}
```

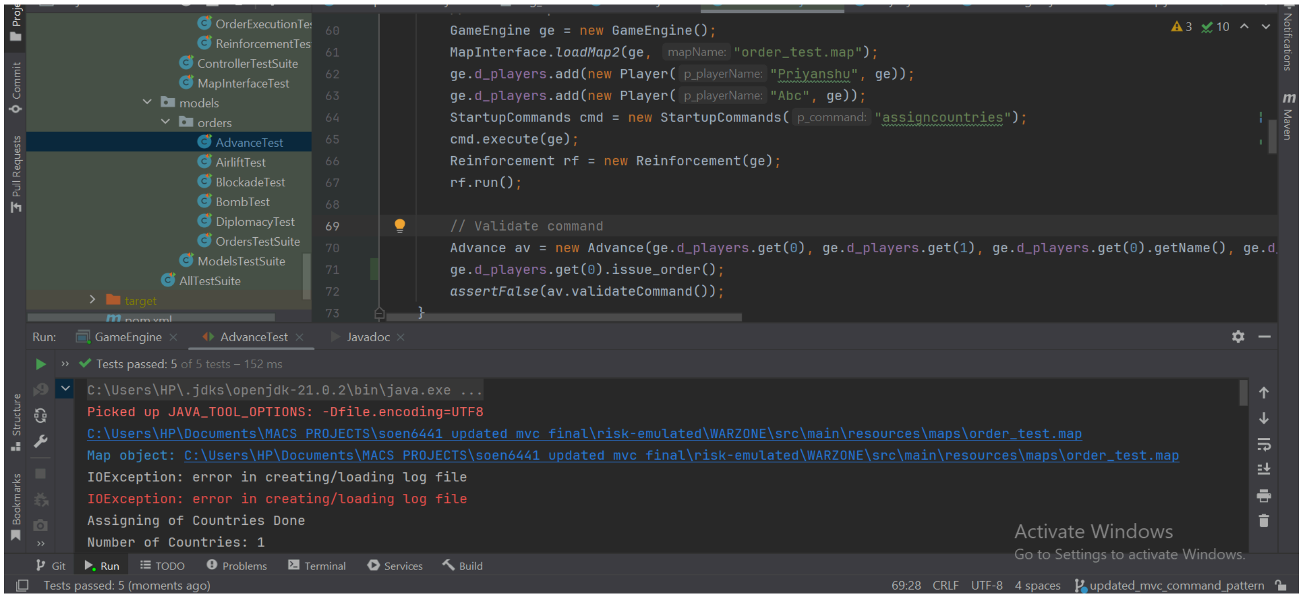
After



6 usages   Devdutt Sharma

```
public void issue_order() throws InvalidCommandException{
    this.d_orderList.add(this.d_current_order);
}
```

Junit Test Results:



```
        GameEngine ge = new GameEngine();
        MapInterface.loadMap2(ge, "order_test.map");
        ge.d_players.add(new Player( "Priyanshu", ge));
        ge.d_players.add(new Player( "Abc", ge));
        StartupCommands cmd = new StartupCommands( "assigncountries");
        cmd.execute(ge);
        Reinforcement rf = new Reinforcement(ge);
        rf.run();

        // Validate command
        Advance av = new Advance(ge.d_players.get(0), ge.d_players.get(1), ge.d_players.get(0).getName(), ge.d
        ge.d_players.get(0).issue_order();
        assertFalse(av.validateCommand());
    }
```

```
Tests passed: 5 of 5 tests – 152 ms
C:\Users\HP\.jdks\openjdk-21.0.2\bin\java.exe ...
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF8
C:\Users\HP\Documents\MACS_PROJECTS\soen6441_updated_mvc_final\risk-emulated\WARZONE\src\main\resources\maps\order_test.map
Map object: C:\Users\HP\Documents\MACS_PROJECTS\soen6441_updated_mvc_final\risk-emulated\WARZONE\src\main\resources\maps\order_test.map
IOException: error in creating/loading log file
IOException: error in creating/loading log file
Assigning of Countries Done
Number of Countries: 1
```

```
Tests passed: 5 (moments ago)          69:28  CRLF  UTF-8  4 spaces  updated_mvc_command_pattern
```

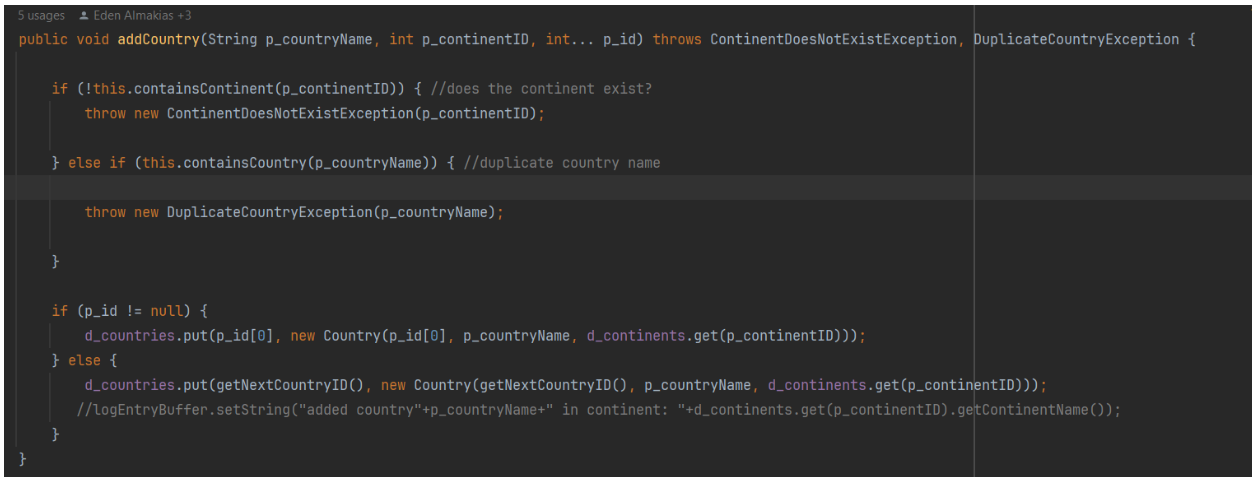
5. Refactored addCountry() in WorldMap.java

This refactoring is done in order to provide more flexibility by allowing for country ID to be either provided as a parameter or generated internally. This allows callers to specify a custom ID if needed or let the method generate one automatically if not provided.

Before

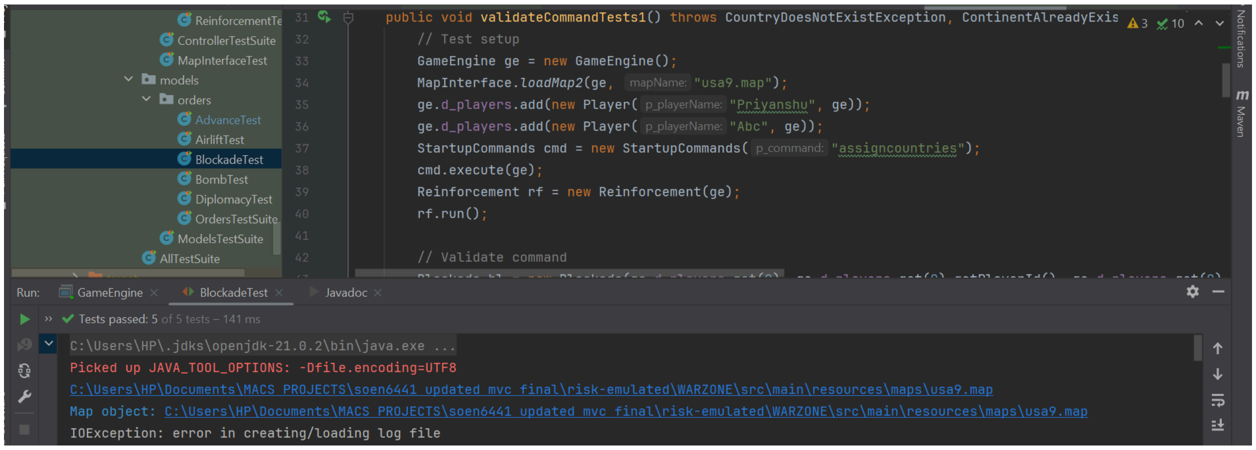
```
 * @param p_countryID ID of country to be added
 * @param p_continentID ID of continent to which country will be added
 * @param p_countryName name of new country
 * @throws ContinentDoesNotExistException if continent does not exist
 * @throws DuplicateCountryException if country already exists
 */
public void addCountry(int p_countryID, int p_continentID, String p_countryName) throws ContinentDoesNotExistException, DuplicateCountryException {

    if (!this.containsContinent(p_continentID)) { //does the continent exist?

        throw new ContinentDoesNotExistException(p_continentID);

    } else if (this.containsCountry(p_countryID)) { //duplicate country id

        throw new DuplicateCountryException(p_countryID);

    } else if (this.containsCountry(p_countryName)) { //duplicate country name

        throw new DuplicateCountryException(p_countryName);

    }

    d_countries.put(p_countryID, new Country(p_countryID, p_countryName, d_continents.get(p_continentID)));

}
```

After:



```java
public void addCountry(String p_countryName, int p_continentID, int... p_id) throws ContinentDoesNotExistException, DuplicateCountryException {

    if (!this.containsContinent(p_continentID)) { //does the continent exist?
        throw new ContinentDoesNotExistException(p_continentID);

    } else if (this.containsCountry(p_countryName)) { //duplicate country name

        throw new DuplicateCountryException(p_countryName);

    }

    if (p_id != null) {
        d_countries.put(p_id[0], new Country(p_id[0], p_countryName, d_continents.get(p_continentID)));
    } else {
        d_countries.put(getNextCountryID(), new Country(getNextCountryID(), p_countryName, d_continents.get(p_continentID)));
       //logEntryBuffer.setString("added country"+p_countryName+" in continent: "+d_continents.get(p_continentID).getContinentName());
    }
}
```

Junit Test:



```java
public void validateCommandTests1() throws CountryDoesNotExistException, ContinentAlreadyExis
    // Test setup
    GameEngine ge = new GameEngine();
    MapInterface.loadMap2(ge, mapName: "usa9.map");
    ge.d_players.add(new Player( p_playerName: "Priyanshu", ge));
    ge.d_players.add(new Player( p_playerName: "Abc", ge));
    StartupCommands cmd = new StartupCommands( p_command: "assigncountries");
    cmd.execute(ge);
    Reinforcement rf = new Reinforcement(ge);
    rf.run();

    // Validate command
```

Tests passed: 5 of 5 tests – 141 ms

```
C:\Users\HP\.jdks\openjdk-21.0.2\bin\java.exe ...
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF8
C:\Users\HP\Documents\MACS_PROJECTS\soen6441_updated_mvc_final\risk-emulated\WARZONE\src\main\resources\maps\usa9.map
Map object: C:\Users\HP\Documents\MACS_PROJECTS\soen6441_updated_mvc_final\risk-emulated\WARZONE\src\main\resources\maps\usa9.map
IOException: error in creating/loading log file
```