Group 15: REFACTORING DOCUMENT COURSE: SOEN 6441

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:

After the completion of build 1, some refactoring steps were needed according to the description and requirements of the Build 2. Major refactoring steps were performed throughout the development of this Build that include: -

1. Implementation of a Logger file to store major events of the game.
2. Implementation of Logger using Observer pattern to update all the views.
3. Replace hard-coded strings with constants all grouped together in separate class to enhance reusability.
4. Applying State Pattern to different phases Startup, Issue Order and Order Execution phase of the game.
5. Remove unnecessary and unused imports.
6. Adding Javadoc for private data members and ensuring that Javadoc is completed using Xdoclint.
7. Adding syntax validation for various commands.
8. Ensuring that at least 2 players are present in a game.
9. Adjusting naming conventions for better understanding.
10. Using Command patterns for handling various order commands.

11. Modifying the existing test cases according to new project structure.
12. Add more test cases for the existing functions.
13. Display error messages to the user whenever an invalid operation is carried out.
14. Validate existing command operations and informing the user whenever an invalid operation is carried out.
15. Add Test Suite for all the project folders and test files.

Actual Refactoring Targets:

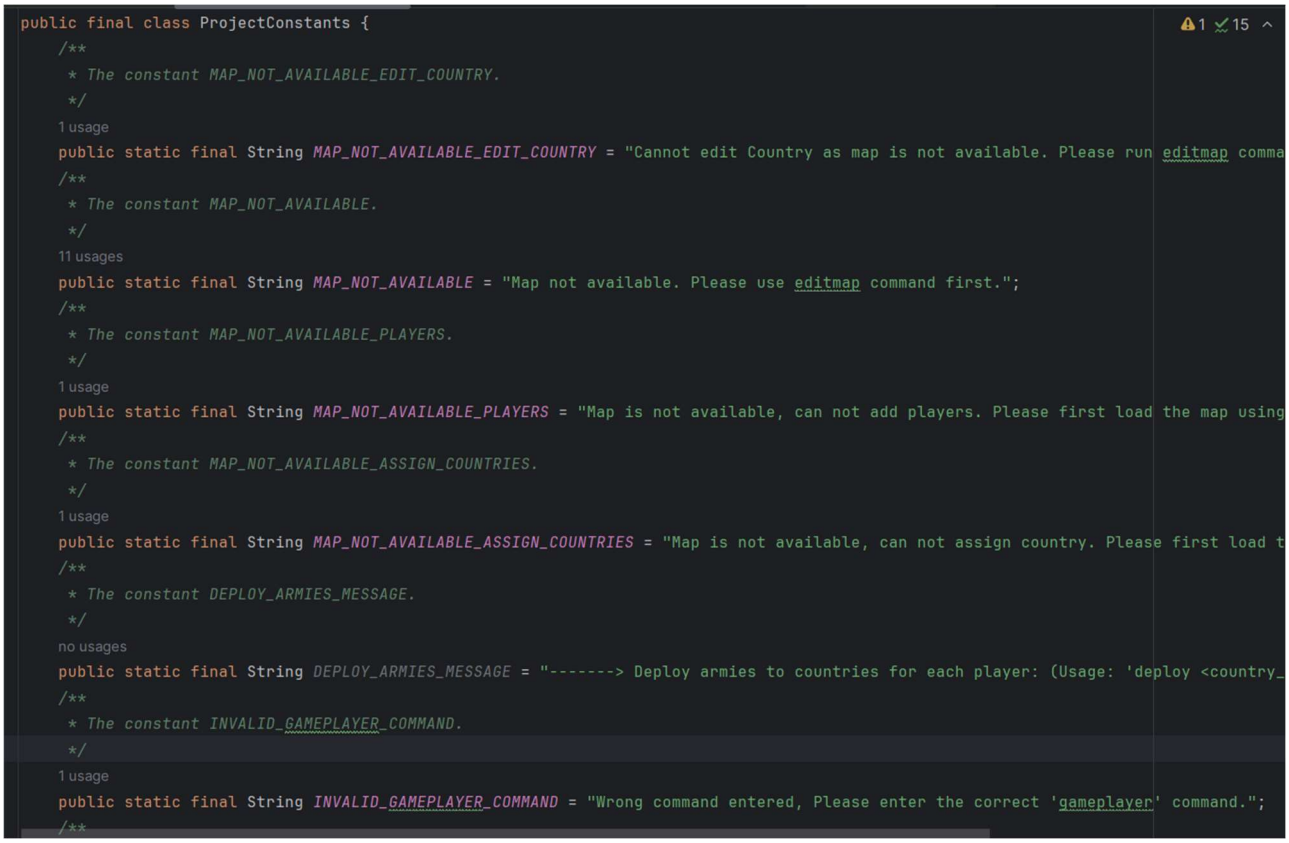
The selection of refactoring tasks for the specified target list was primarily influenced by the new requirements introduced in the second build, as well as the most significant challenges and inconsistencies experienced during the development of the initial build.

1) Replace hard-coded strings with constants all grouped together in separate class to enhance reusability.

Before Refactoring: -

```
3 usages  Jaiwant
public boolean validateCountriesAndContinents() {
    if(d_mapContinents == null || d_mapContinents.isEmpty()){
        System.out.println("Map does not have Continents");
        return false;
    }
    else if(d_mapCountries == null || d_mapCountries.isEmpty()){
        System.out.println("Map does not have Countries");
        return false;
    }
    else{
        for(Country l_eachCountry : d_mapCountries){
            if(l_eachCountry.getD_neighbouringCountriesId().isEmpty()){
                System.out.println("Country: "+l_eachCountry.getD_countryName()+" does not have any neighbours.");
                return false;
            }
        }
    }
    return true;
}
```

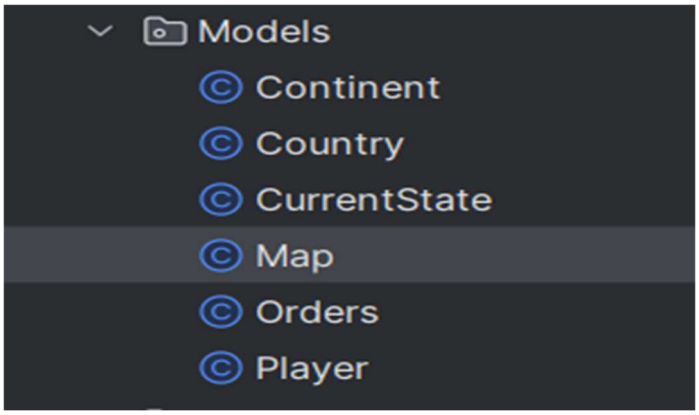
After Refactoring: -



```
public final class ProjectConstants {                                                    1  15
    /**
     * The constant MAP_NOT_AVAILABLE_EDIT_COUNTRY.
     */
    1 usage
    public static final String MAP_NOT_AVAILABLE_EDIT_COUNTRY = "Cannot edit Country as map is not available. Please run editmap comma
    /**
     * The constant MAP_NOT_AVAILABLE.
     */
    11 usages
    public static final String MAP_NOT_AVAILABLE = "Map not available. Please use editmap command first.";
    /**
     * The constant MAP_NOT_AVAILABLE_PLAYERS.
     */
    1 usage
    public static final String MAP_NOT_AVAILABLE_PLAYERS = "Map is not available, can not add players. Please first load the map using
    /**
     * The constant MAP_NOT_AVAILABLE_ASSIGN_COUNTRIES.
     */
    1 usage
    public static final String MAP_NOT_AVAILABLE_ASSIGN_COUNTRIES = "Map is not available, can not assign country. Please first load t
    /**
     * The constant DEPLOY_ARMIES_MESSAGE.
     */
    no usages
    public static final String DEPLOY_ARMIES_MESSAGE = "-------> Deploy armies to countries for each player: (Usage: 'deploy <country_
    /**
     * The constant INVALID_GAMEPLAYER_COMMAND.
     */
    1 usage
    public static final String INVALID_GAMEPLAYER_COMMAND = "Wrong command entered, Please enter the correct 'gameplayer' command.";
    /**
```

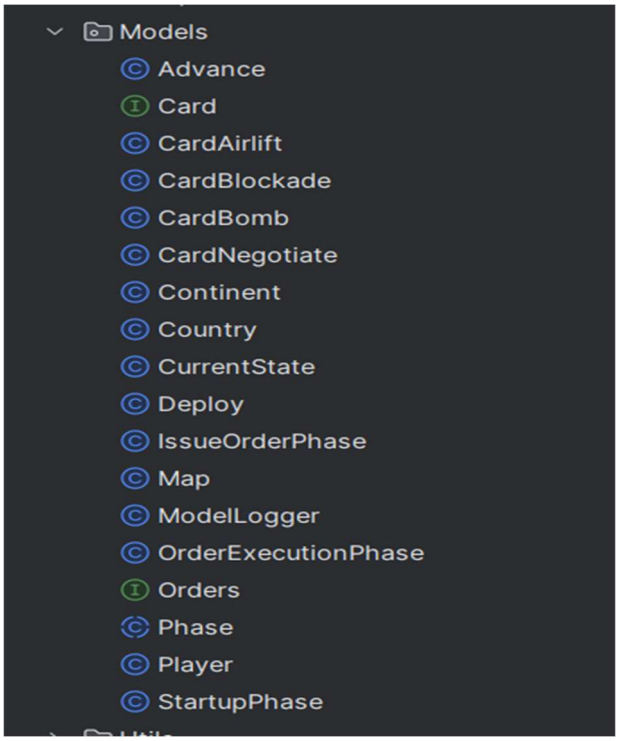
2. Implement State pattern for Phase Change: -

Before Refactoring: -



After Refactoring: -

The State pattern has been implemented according to the requirements, with separate classes created for StartupPhase, IssueOrderPhase, and OrderExecutionPhase.



3. **Command pattern for processing orders**

Before refactoring: -

In the initial Build1, the processing of deploy orders didn't strictly adhere to the command pattern. Instead, there existed a singular function named "execute" which encompassed the entire logic. However, in the subsequent Build 2, all of the commands including Deploy, Advance, Bomb, Blockade, and Airlift have been structured to implement a chain of commands, effectively adopting the command pattern. This architectural adjustment enhances modularity and flexibility by separating the commands into distinct objects, thus promoting better code organization and maintainability.

```
     */
    3 usages   Mehakveer
    public void execute(Player p_eachPlayer) {
        if(d_order.equals("deploy")){
            for(Country l_eachCountry : p_eachPlayer.getD_currentCountries()){
                if(l_eachCountry.getD_countryName().equals(this.d_targetName)){
                    l_eachCountry.setD_armies(l_eachCountry.getD_armies() + this.d_noOfArmiesToMove);
                    break;
                }
            }
        }
    }
}
```

After refactoring: -

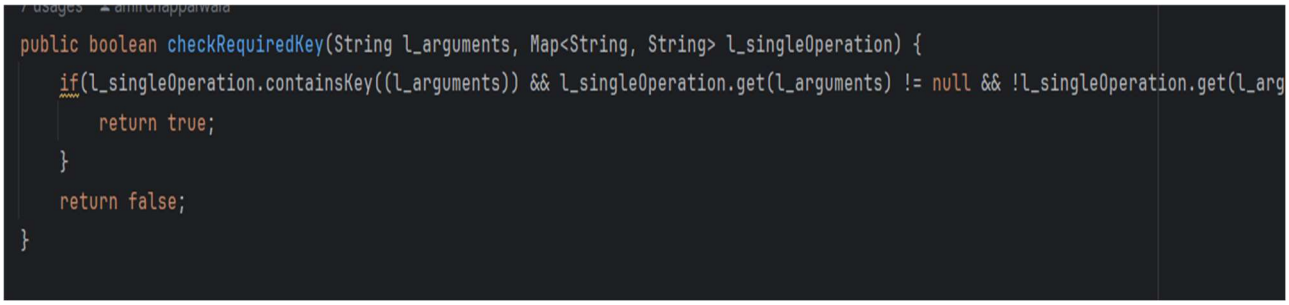
```java
public class CardBomb implements Card{

    /**
     * The D card owner.
     */
    6 usages
    Player d_cardOwner;
    /**
     * The D target country name.
     */
    8 usages
    String d_targetCountryName;

    /**
     * Instantiates a new Card bomb.
     *
     * @param p_cardOwner         the p card owner
     * @param p_targetCountryName the p target country name
     */
    3 usages  Mehakveer
    public CardBomb(Player p_cardOwner, String p_targetCountryName) {
        this.d_cardOwner = p_cardOwner;
        this.d_targetCountryName = p_targetCountryName;
    }

    /**
     * Sets d order execution log.
     *
     * @param p_orderExecutionLog the p order execution log
     * @param p_messageType       the p message type
     */
```

4. **Implement Command syntax validation:**

Before Refactoring:

In Build 1, commands weren't thoroughly checked. In Build 2, we added ValidateCommand to make sure all commands are properly validated. This helps catch errors before they cause problems, making the system more reliable and user-friendly.

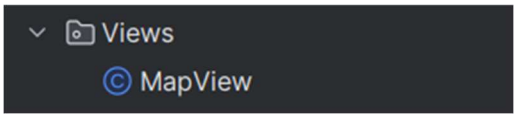
After Refactoring:



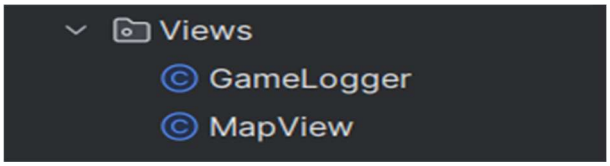
5. **Make separate view for Log Writing to Console**

Logger is added in Build 2 to store operations of the game. Logger is implemented using Observer pattern and GameLogger acts as an Observer to the ModelLogger subject.

Before refactoring**:**



After refactoring**:**



6. Javadoc Validation Using Xdoclint to ensure that Javadoc is generated for all the files: -

```
[WARNING] Generating C:\Users\jaiwa\IdeaProjects\SOEN-6441 Warzone Group 16 Winter 2024\target\site\apidocs\allpackages-index.html...
[WARNING] Generating C:\Users\jaiwa\IdeaProjects\SOEN-6441 Warzone Group 16 Winter 2024\target\site\apidocs\index-all.html...
[WARNING] Generating C:\Users\jaiwa\IdeaProjects\SOEN-6441 Warzone Group 16 Winter 2024\target\site\apidocs\search.html...
[WARNING] Generating C:\Users\jaiwa\IdeaProjects\SOEN-6441 Warzone Group 16 Winter 2024\target\site\apidocs\overview-summary.html...
[WARNING] Generating C:\Users\jaiwa\IdeaProjects\SOEN-6441 Warzone Group 16 Winter 2024\target\site\apidocs\help-doc.html...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.447 s
[INFO] Finished at: 2024-03-16T18:51:46-04:00
[INFO] ------------------------------------------------------------------------
PS C:\Users\jaiwa\IdeaProjects\SOEN-6441 Warzone Group 16 Winter 2024>
```

No Warnings for all the files when Xdoclint command is verified. Therefore, Javadoc is properly generated for all the files in our project.

Dependency added in pom.xml for Xdoclint

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-javadoc-plugin</artifactId>
    <version>3.1.1</version>
    <configuration>
        <doclint>all</doclint>
    </configuration>
    <executions>
        <execution>
            <id>attach-javadocs</id>
            <goals>
                <goal>jar</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```