SOEN 6441 - Advanced Programming Practices

Project - Warzone Risk Based game (Build #2)

Winter 2025

Refactoring Document

Submitted by **DABSV**

Arvind Nachiappan Lakshmanan - 40310757

Barath Sundararaj - 40324920

Devasenan Murugan - 40302170

Swathi Priya Pasumarthy- 40322468

Submitted to

Prof. M. Taleb

Potential Refactoring Targets

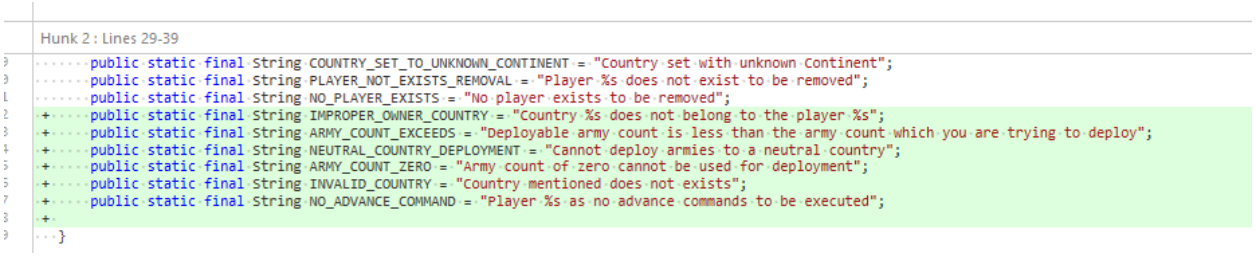
Given below is the list of potential refactoring targets:

1. **Repackaging the Orders Model into a separate module -** This was implemented to maintain code readability and isolate the order functionality from the game phase.
2. **Encapsulated the methods related to issuing order from the PlayerOperationsHandler class into separate Class**
3. **Command validation for invalid option types and param length -** Validation of commands were properly handled to overcome the errors in case the player entered the wrong option type or parameter length.
4. **Command validation for invalid and redundant game phase commands**
5. **Decomposed the processAdvanceCommand method Into processAdvanceCommand and processCommitCommand in IssueOrderHandler**
6. **Consolidate loadMap() into sub-functions -** We discovered that the MapOperationsHandler class is overly complex, handling all the map functions in one class file, making it difficult to understand and maintain.
7. **Decomposed the executeBufferedCommands method Into issueOrder and advanceTurn methods in IssueOrderHandler**
8. **Updating the access modifiers of the methods -** This was discovered to ensure that methods have appropriate levels of visibility in order to maintain encapsulation and control access to specific functionalities.
9. **Updating Unintuitive variable names**
10. **Removing dead code and unused Import statements-** We identified this by seeing code and import statements that appear to have no effect on the program's behavior.

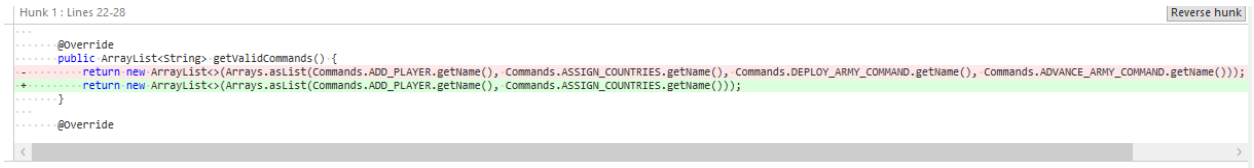
11. **Replacing if..else..if with switch..case -** This target was chosen to improve code readability and maintainability in situations where a switch statement would be preferable to a series of if-else statements

12. **Encapsulated the methods related to order deploying from the PlayerOperationsHandler class into DeployOrder class**

13. **Adding constants for the valid commands -** Constants were introduced to reduce the repetition of the same statements.



14. **Update allowed commands in the startUp phase**

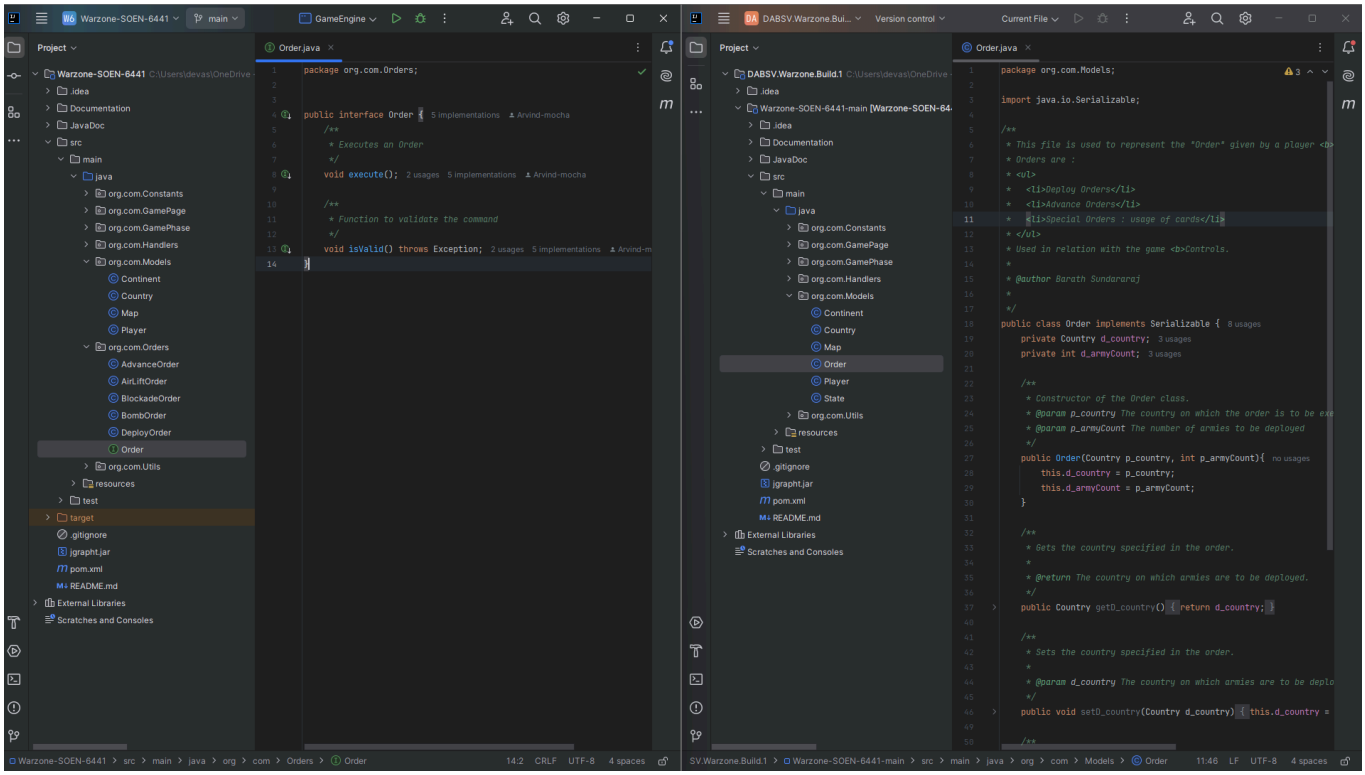


15. **Encapsulated the methods related to order advancing from the PlayerOperationsHandler class into AdvanceOrder Class**.

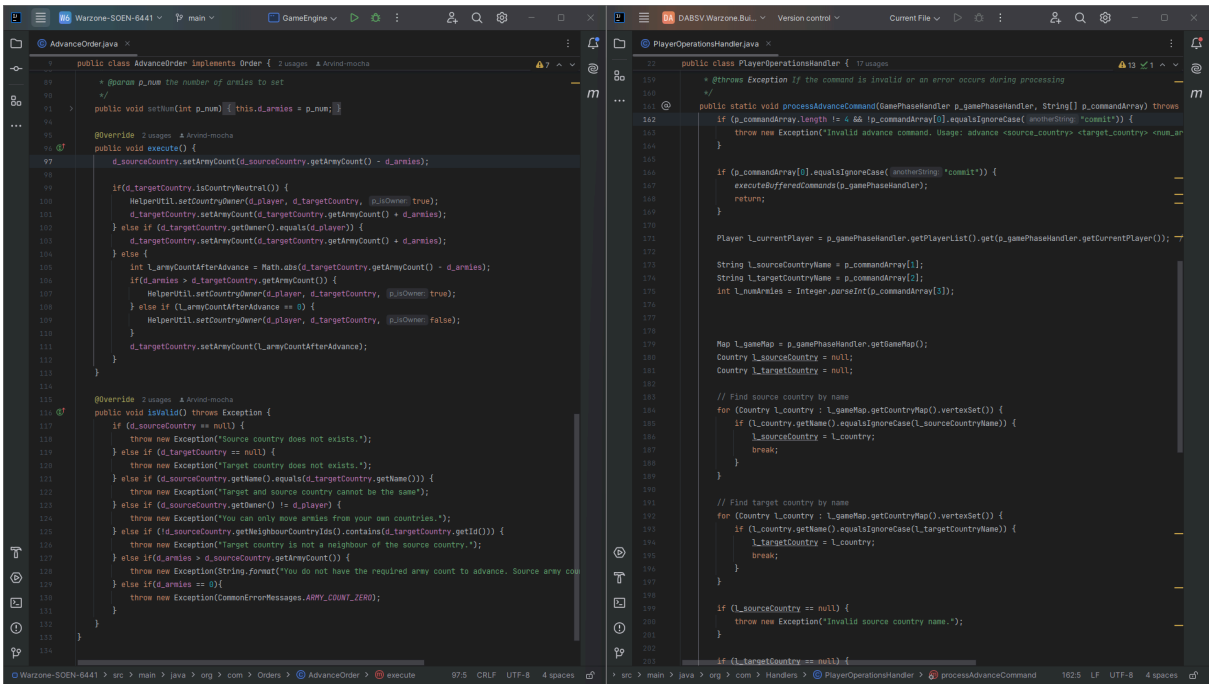
Actual Refactoring Targets

Below are the 5 refactoring targets chosen from the above mentioned list of all potential targets, based on the requirements established in Build 2.

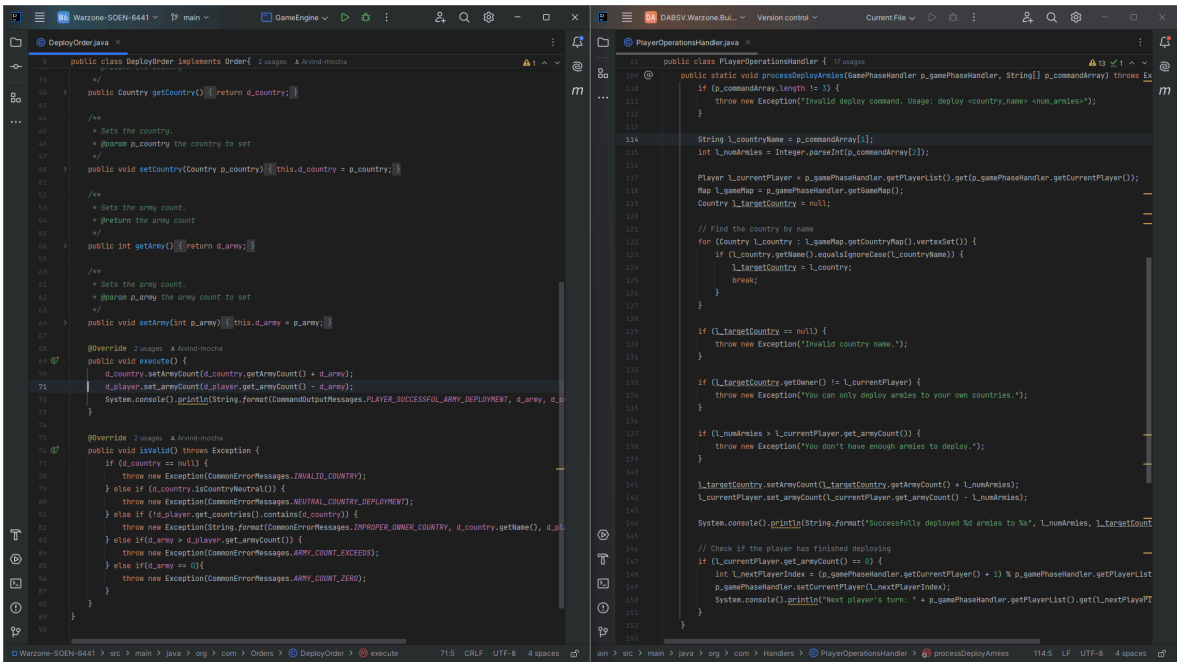
1. Repackaging the Orders Model into a standalone package



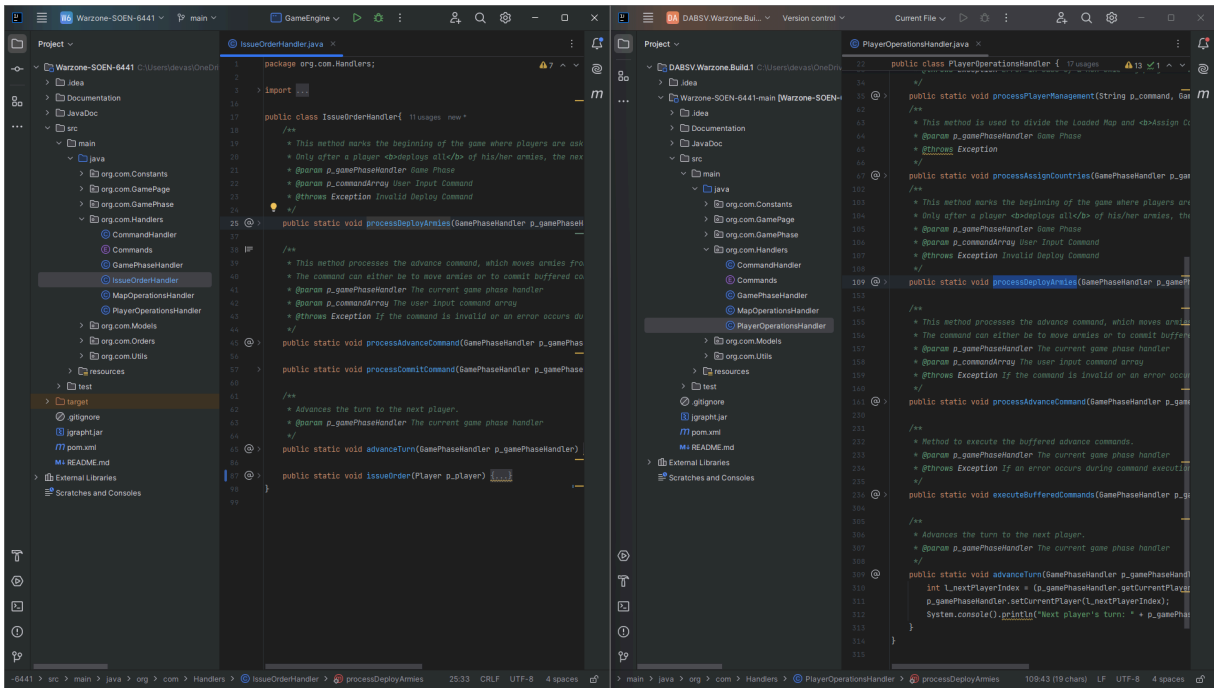
2. Encapsulated the methods related to order advancing from the PlayerOperationsHandler class into AdvanceOrder Class



3. Encapsulated the methods related to order deploying from the PlayerOperationsHandler class into DeployOrder class



4. Encapsulated the methods related to issuing order from the PlayerOperationsHandler class into IssueOrderHandler



5. Command validation for invalid and redundant game phase commands

```
@Override  Arvind-mocha *
public void isValid() throws Exception {
    if (d_sourceCountry == null) {
        throw new Exception("Source country does not exists.");
    } else if (d_targetCountry == null) {
        throw new Exception("Target country does not exists.");
    } else if (d_sourceCountry.getName().equals(d_targetCountry.getName())) {
        throw new Exception("Target and source country cannot be the same");
    } else if (d_sourceCountry.getOwner() != d_player) {
        throw new Exception("You can only move armies from your own countries.");
    } else if (!d_sourceCountry.getNeighbourCountryIds().contains(d_targetCountry.getId())) {
        throw new Exception("Target country is not a neighbour of the source country.");
    } else if(d_armies > d_sourceCountry.getArmyCount()) {
        throw new Exception(String.format("You do not have the required army count to advance. " +
                "Source army count: %s", d_sourceCountry.getArmyCount()));
    } else if(d_armies == 0){
        throw new Exception(CommonErrorMessages.ARMY_COUNT_ZERO);
    } else if (d_armies == d_sourceCountry.getArmyCount()) {
        throw new Exception(String.format("At least one army my be present in source country. " +
                "Source army count: %s", d_sourceCountry.getArmyCount()));
    }
}
```

Reasons for choosing the Actual Refactoring Targets

The following elements were chosen based on the aspects of code quality, maintainability and better functionality of the game in whole. This has helped in improving the internal structure of the program without breaking the external behaviour of the game observed in the initial build.

1. **Repackaging the Orders Model into a standalone package -** In the initial build, the orders passed by a player were being handled in the PlayerOperationsHandler file which belonged to the Handlers package. It was working fine and readable for the first build consisting of only the "deploy" command. But for the upcoming build with various commands such as "deploy", "advance", "airlift" etc, we decided to create "Orders" Model as a standalone package to provide better code readability and efficient management of pieces of code.

2. **Encapsulated the methods related to order advancing from the PlayerOperationsHandler class into AdvanceOrder Class -** Having all the methods related to player operations performed in one class file would affect the code readability and subsequently would also affect the ability to debug the program in case of any error. Hence, we came up with a solution to encapsulate the player operation methods in a different class file.

3. **Encapsulated the methods related to order deploying from the PlayerOperationsHandler class into DeployOrder class -** This was done to reduce the complexity "PlayerOperationsHandler" class and to create a more focused, single-responsibility class for deployment operations. We maintained the existing deployment logic and error handling and also preserved the original method signatures and functionality.

4. **Encapsulated the methods related to issuing order from the PlayerOperationsHandler class into IssueOrderHandler -** Decomposed the processAdvanceCommand method into processAdvanceCommand and processCommitCommand in IssueOrderHandler and also, decomposed the executeBufferedCommands method into issueOrder and advanceTurn methods in IssueOrderHandler.

5. **Command validation for invalid and redundant game phase commands**
   Validation of game phase commands were introduced to restrict the program from crashing in certain cases where the player passes multiple deploy and/or advance commands, without taking army count into consideration. In these cases, once the army count exhausts, the program tends to crash. To overcome this we introduced validation of commands wherever possible. By having a validation method and a buffer (which stores all the commands given by a player at a particular turn), the method executes orders present in the buffer until one of the conditions fails.