

SOEN6441 – Advanced Programming Practices

Refactoring – Build 2

Team 11

Potential Refactoring Targets

In response to the requirements outlined in build 2 and inconsistencies identified during the analysis of build one, the following refactoring targets have been selected to improve the codebase's structure, readability, and maintainability. These refactoring points aim to enhance the overall quality of the code and align it more closely with best practices and project goals.

1. Consolidate Duplicate Code for GameEngine class:

This refactoring simplifies the `handleCommand` method by breaking its functionality into smaller, dedicated methods. Each method handles a specific command, making the code easier to understand, maintain, and extend.

2. Create a utility class with common functionalities to avoid duplication:

Both the `MapView` and `PlayerView` classes have common functionalities for printing separators, calculating total width, displaying continent name and displaying neighbouring countries. We have extracted these methods into a utility class `ViewUtils` to avoid duplication. This will enhance code readability, maintainability and reduce redundancy.

3. Use Object Composition over Inheritance for Reusability:

Instead of having static methods in `MapView` for obtaining neighboring country names, consider moving this functionality into a separate class or interface, and then use composition to include it in both `MapView` and `PlayerView`. This adheres more closely to the principle of "favor composition over inheritance" and promotes code reuse and flexibility.

4. Use Enums Instead of String Constants in MapEditor class:

Instead of using string constants (`CONTINENTS`, `COUNTRIES`, `BORDERS`) for identifying different types of lines, consider using enums. This approach makes the code more readable, less error-prone, and easier to maintain.

5. Extract Method for Command Handling in Player class:

The `issue_order` method is responsible for handling various commands entered by the player. It contains a switch statement that handles each command separately. This method could be refactored by extracting the logic for handling each command into separate methods. This

improves the readability and maintainability of the code by making each method responsible for only one task.

6. Create Constant Error Messages in MapValidator class:

One refactoring we can implement in the MapValidator class is to encapsulate the validation error messages into constants or a message generation method. This can improve code readability and make it easier to manage and modify error messages in the future.

7. Modify File Handling in LogEntryWriter class:

One possible refactoring can be done in LogEntryWriter class is to handle file operations more gracefully, including proper exception handling and resource management. Additionally, it would be beneficial to separate concerns and improve code readability.

8. Refinement of LoadMap Functionality to Prevent Data Duplication:

In response to identified issues with the loadMap functionality in the Warzone project, the system has been refined to prevent data duplication when loading map files multiple times. Previously, executing the loadMap command for the same map file resulted in the addition of duplicate map data to the existing map, leading to an accumulation of redundant information.

9. Enhanced EditMap Functionality to Ensure Consistent Continent IDs:

In response to identified discrepancies in the editMap functionality within the Warzone project, enhancements have been implemented to ensure consistent continent IDs across map loading and saving operations. Previously, loading a map file after editing and saving resulted in inconsistent continent IDs due to the auto-generation mechanism based on the lastAssignedID. This inconsistency led to incorrect continent mappings and potential gameplay issues.

10. Introduction of GameContext for Global Game State Management:

The introduction of the GameContext class marks a significant enhancement in the Warzone project, providing a centralized mechanism for managing the global state of the game. Utilizing the singleton pattern, GameContext ensures that there is a single instance throughout the game, allowing seamless access to essential components such as the game map, players, map services, and current game phase.

11. Encapsulation for member variable in Map class:

One refactoring point for the Map class to enhance encapsulation by making the member variables d_continents and d_countries private and providing methods for accessing and modifying them, if necessary. This encapsulation helps in maintaining data integrity and enables better control over the internal state of the Map class.

12. Reduce Direct Dependency on the GameContext in AirliftOrder class:

One refactoring point for the AirliftOrder class is to improve encapsulation by reducing direct dependencies on the GameContext singleton. Instead of directly accessing the GameContext within the class, we can pass necessary dependencies through the constructor or method parameters. This enhances flexibility, testability, and reduces tight coupling.

13. Introduce Proper Error Handling in BlockadeOrder class:

One refactoring point for the BlockadeOrder class is to introduce proper error handling instead of using print statements for reporting errors. By throwing exceptions with descriptive error messages, you can provide more meaningful feedback to the caller and improve the maintainability of the code.

14. Add More Meaningful Behaviour in DiplomacyOrder class:

One refactoring point for the DiplomacyOrder class is to add more meaningful behavior to the execute() method. Currently, it only contains a placeholder print statement. To improve the design, we could consider implementing a proper negotiation mechanism or defining specific actions for executing a diplomacy order.

15. Encapsulate Functionality into Smaller Methods in MapValidator class:

One refactoring point for the MapValidator class could be to encapsulate some of its functionality into smaller, more focused methods to improve readability and maintainability. This can make the code easier to understand and maintain.

Actual Refactoring

After reviewing the potential refactoring targets, we have identified the following items to prioritize for implementation in build 2. These selections are based on their importance and the availability of corresponding tests, ensuring a balanced focus on critical improvements while maintaining code integrity.

1. Consolidate Duplicate Code

GameEngine Class:

This refactoring simplifies the `handleCommand` method by breaking its functionality into smaller, dedicated methods. Each method handles a specific command, making the code easier to understand, maintain, and extend.

List of Tests

The refactoring was validated by a comprehensive set of unit tests:

- `testHandleLoadMapCommand` - Verifies that the "loadmap" command is correctly processed.
- `testHandleEditMapCommand` - Ensures that the "editmap" command is correctly processed.
- `testHandleModifyMapComponentsCommand` - Confirms that commands to modify map components are handled.
- `testHandleSaveMapCommand` - Checks that the "savemap" command is handled properly.
- `testHandleShowMapCommand` - Asserts that the "showmap" command output is displayed correctly.
- `testHandleAddOrRemovePlayerCommand` - Verifies the addition or removal of players.
- `testHandleProceedCommand` - Validates the "proceed" command to move to the next phase.
- `testHandleExitCommand` - Confirms the "exit" command functionality.

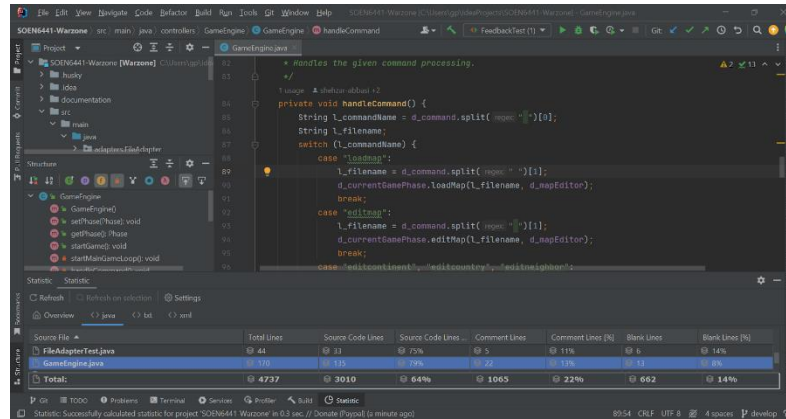
Necessity of Refactoring

Refactoring was necessary due to:

- **Complexity:** The original `handleCommand` method was long and complicated, making it difficult to understand and maintain.
- **Maintainability:** Breaking down the `handleCommand` into smaller methods made the code more maintainable.
- **Testability:** Smaller, individual methods allowed for more focused and reliable unit tests.
- **Scalability:** The new structure made it easier to add new commands and functionality to the game engine.

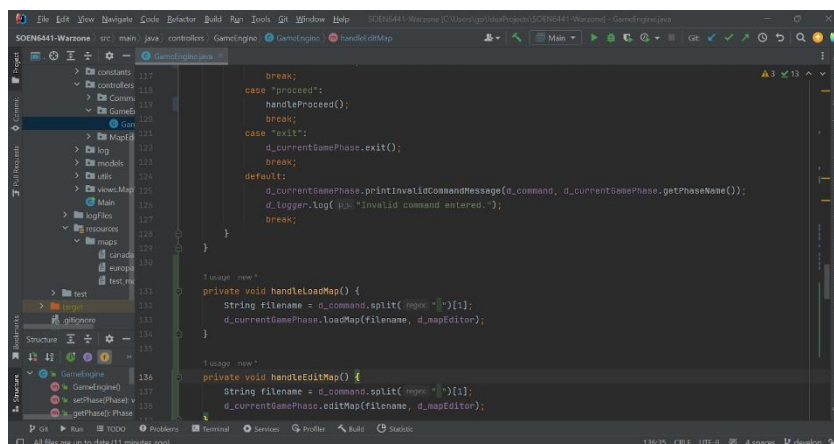
Before/After Depiction

Before refactoring, the `handleCommand` method consisted of a large switch statement that included all the logic for handling commands. This was not ideal for readability or maintainability.



After refactoring, the `handleCommand` method delegates to smaller, more focused methods:

- `handleLoadMap`
- `handleEditMap`
- `handleModifyMapComponents`
- `handleSaveMap`
- `handleShowMap`
- `handleAddOrRemovePlayer`
- `handleStartGame`
- `handleProceed`



Each method is responsible for a specific command, simplifying the overall structure and making the code more readable and easier to maintain.

2. Create a utility class with common functionalities to avoid duplication

MapView and PlayerView class:

Both the MapView and PlayerView classes have common functionalities for printing separators, calculating total width, displaying continent name and displaying neighbouring countries. We have extracted these methods into a utility class ViewUtils to avoid duplication. This will enhance code readability, maintainability and reduce redundancy.

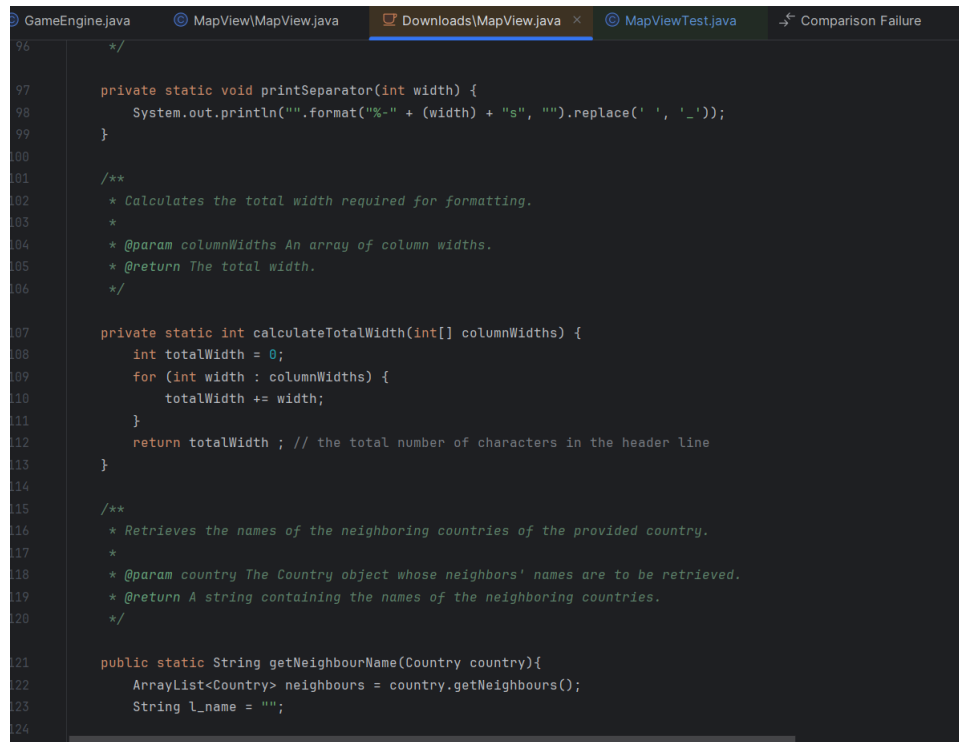
Test

The refactoring was validated by the following unit test:

- testDisplayMapInformation: Verifies the display of the map is correct

Before/After Depiction

Before refactoring the MapView and PlayerView classes have the following functions.



```
96      */
97      private static void printSeparator(int width) {
98          System.out.println("".format("%-" + (width) + "s", "").replace(' ', '_'));
99      }
100
101      /**
102       * Calculates the total width required for formatting.
103       *
104       * @param columnWidths An array of column widths.
105       * @return The total width.
106       */
107
108      private static int calculateTotalWidth(int[] columnWidths) {
109          int totalWidth = 0;
110          for (int width : columnWidths) {
111              totalWidth += width;
112          }
113          return totalWidth ; // the total number of characters in the header line
114      }
115
116      /**
117       * Retrieves the names of the neighboring countries of the provided country.
118       *
119       * @param country The Country object whose neighbors' names are to be retrieved.
120       * @return A string containing the names of the neighboring countries.
121       */
122
123      public static String getNeighbourName(Country country){
124          ArrayList<Country> neighbours = country.getNeighbours();
125          String l_name = "";
```

After refactoring the common functions are moved to ViewUtils class.

```
GameEngine.java  MapView/MapView.java  ViewUtils.java x  MapViewTest.java  -> Con
13  */
    6 usages  Tanzina Nasrin
14  public static void printSeparator(int width) {
15      System.out.println("".format("%-" + (width) + "s", "").replace(oldChar: ' ', ne
16  }
17  /**
18   * Calculates the total width required for formatting.
19   *
20   * @param columnWidths An array of column widths.
21   * @return The total width.
22   */
23  @ 2 usages  Tanzina Nasrin
    public static int calculateTotalWidth(int[] columnWidths) {
24      int totalWidth = 0;
25      for (int width : columnWidths) {
26          totalWidth += width;
27      }
28      return totalWidth;
29  }
30  /**
31   * Retrieves the names of the neighboring countries of the provided country.
32   *
33   * @param country The Country object whose neighbors' names are to be retrieved.
34   * @return A string containing the names of the neighboring countries.
35   */
36  @ 4 usages  Tanzina Nasrin
    public static String getNeighbourName(Country country){
37      ArrayList<Country> neighbours = country.getNeighbours();
38      String l_name = "";
39
40      for (Country neighbour : neighbours) {
41          l_name = l_name.concat(neighbour.getName());
42          l_name = l_name.concat(" ");
43      }
44      return l_name;
45  }
```

3. Refinement of LoadMap Functionality to Prevent Data Duplication:

To address this issue, the loadMap functionality has been modified to create a new map instance each time a map file is loaded. This ensures that the map data from the file is populated into a fresh map object, preventing the accumulation of duplicate data. By adhering to this approach, the integrity of the map data is preserved, and each map file is loaded into a separate map instance, reflecting a distinct game world.

Necessity of Refinement:

The refinement was necessary to rectify the following shortcomings:

- Data duplication: Previously, executing the loadMap command for the same map file resulted in the addition of duplicate map data to the existing map, leading to data redundancy and potential inconsistencies.
- Inefficient resource utilization: The accumulation of redundant data consumed unnecessary memory and resources, impacting system performance and scalability.

Overall, the refinement ensures that each map file is loaded into a separate map instance, maintaining data integrity and promoting efficient resource utilization. By preventing data duplication, the loadMap functionality provides a more robust and accurate representation of the game world, enhancing the overall gameplay experience.

The first line highlights the change where we are creating a new map whenever we execute a loadmap command so that we populate that new map instead of overwriting the previous one.

```
Usage shenzai-abbasi
public void loadMap(File p_file) throws FileNotFoundException, IOException {
    d_map=new Map();
    //reset the continent last assigned id.
    Continent.lastAssignedID=1;
    BufferedReader READER = new BufferedReader(new FileReader(p_file));
    String l_line = READER.readLine();
    boolean l_startReading = false;
    LineType l_lineType = LineType.CONTINENT; //initializing to remove error.
    while (l_line != null) {
        if (l_line.startsWith(CONTINENTS)) {
            l_startReading = true;
            l_lineType = LineType.CONTINENT;

        } else if (l_line.startsWith(COUNTRIES)) {
            l_startReading = true;
            l_lineType = LineType.COUNTRY;

        } else if (l_line.startsWith(BORDERS)) {
            l_startReading = true;
            l_lineType = LineType.NEIGHBOR;

        } else if (l_startReading) {
            processMapLine(l_line, l_lineType);
        }
        l_line = READER.readLine();
    }
    //when done loading set the global state
    d_ctx.setMap(d_map);
    d_map=null;
}
```


4. Enhanced EditMap Functionality to Ensure Consistent Continent IDs:

To address this issue, the editMap functionality has been refined to reset the lastAssignedID to 1 whenever a new map file is loaded. By resetting the ID counter, each continent within the loaded map is assigned a unique ID starting from 1, ensuring consistency and alignment with the map's structure. This approach prevents discrepancies in continent IDs and ensures accurate continent mappings during gameplay.

Necessity of Enhancement:

The enhancement was necessary to mitigate the following shortcomings:

- Inconsistent continent IDs: Previously, loading a map file after editing and saving resulted in discrepancies in continent IDs, leading to incorrect continent mappings and potential gameplay issues.
- Data integrity: The inconsistency in continent IDs compromised the integrity of the map data, impacting gameplay accuracy and reliability.
- Gameplay consistency: Inaccurate continent mappings could adversely affect gameplay mechanics such as reinforcement calculations and territory ownership, leading to an inconsistent gaming experience.

A screenshot of a code editor showing a Java method named LoadMap. The code is as follows:

```
1 usage  shehzar-abbasi
public void LoadMap(File p_file) throws FileNotFoundException, IOException {
    d_map=new Map();
    //reset the continent last assigned id.
    Continent.lastAssignedID=1;
    BufferedReader READER = new BufferedReader(new FileReader(p_file));
    String l_line = READER.readLine();
```

5. Introduction of GameContext for Global Game State Management:

Necessity of Enhancement:

The enhancement was necessary to mitigate the following shortcomings:

- Centralized Game State Management: GameContext consolidates essential game elements such as the map, players, and map services.
- Elimination of Redundancy: Removed the need for separate mapHolder and playerHolder classes, reducing redundancy.
- Improved Code Organization: Encapsulated game components within GameContext, leading to cleaner and more maintainable code.

```

> import ...

± shehzar-abbasi
public class GameContext {
    3 usages
    private static GameContext d_instance= new GameContext();
    3 usages
    private Map d_map;
    3 usages
    private MapEditor d_mapEditor;
    5 usages
    private ArrayList<Player> d_players = new ArrayList<>();
    2 usages
    private final LogEntryBuffer d_logger= LogEntryBuffer.getInstance();
    2 usages
    private Phase d_currentGamePhase;
    2 usages ± shehzar-abbasi
    private GameContext(){
        d_map = new Map();
        d_logger.clear();
    }

    ± shehzar-abbasi
    public static GameContext getInstance(){
        if(d_instance==null) return d_instance=new GameContext();
        else return d_instance;
    }

    ± shehzar-abbasi
    public void setPhase(Phase new_state) { d_currentGamePhase = new_state; }

```

```

± shehzar-abbasi
> public Map getMap() { return d_map; }

± shehzar-abbasi
> public void setMap(Map p_map) { this.d_map = p_map; }
22 usages ± shehzar-abbasi
    public MapEditor getMapService() {
        if (d_mapEditor == null) {
            d_mapEditor = new MapEditor( ctx: this);
        }
        return d_mapEditor;
    }
21 usages ± shehzar-abbasi
> public ArrayList<Player> getGamePlayers() { return d_players; }
5 usages ± shehzar-abbasi
> public void setGamePlayers(ArrayList<Player> new_players) { this.d_players=new_players; }
2 usages ± shehzar-abbasi
> public void addPlayer(Player new_player) { this.d_players.add(new_player); }
2 usages ± shehzar-abbasi
    public Player removePlayer(String p_playerName){
        for (Player player : d_players) {
            if (player.getName().equals(p_playerName)) {
                d_players.remove(player);
                return player;
            }
        }
        return null;
    }
12 usages ± shehzar-abbasi
> public void updateLog(String p_log) { d_logger.log(p_log); }

```