Ahmad Hasan ——— CS (40294488)

Diego Palacios ——— SE (40270954)

Handika Harianto Ew Jong ——— CS (40307322)

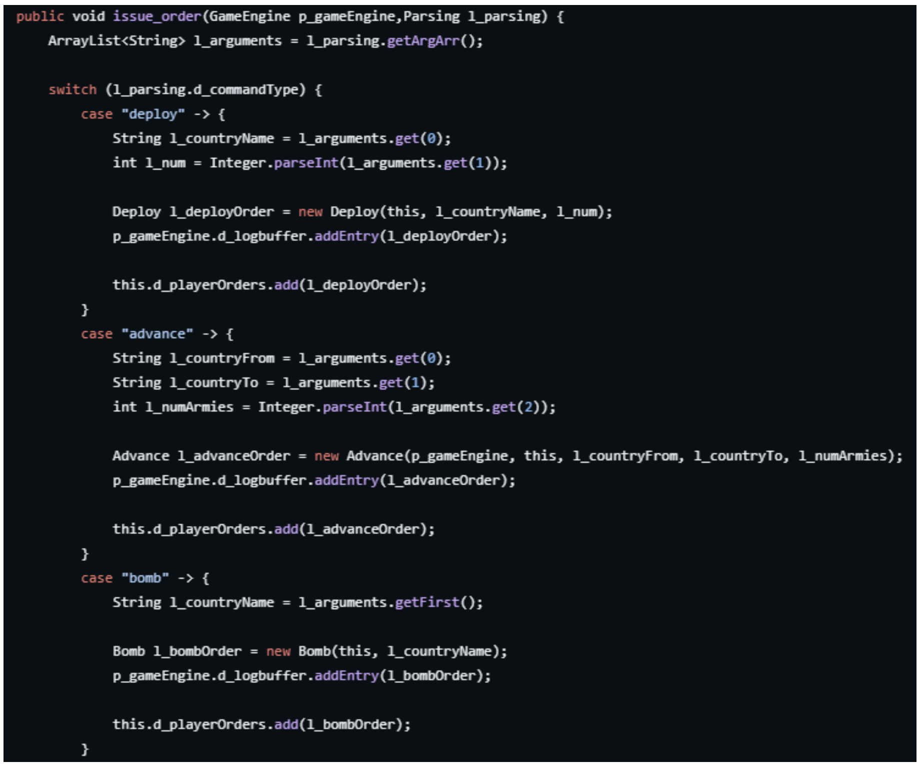
Anosh Kurian Vadakkeparampil ——— CS (40303184)

Potential Refactoring Targets

1. **Player.java**
2. **StartUp.java**
3. **Preload.java**
4. **Postload.java**
5. **MapReader.java**

Actual Refactoring Targets

- **Player.java class**
  Previously *issueOrder()* created orders based on the correctness of the input provided by the user and creating the specific order comparing the first word of the command to the orders name, everything done inside a switch case handling every option possible:



```java
public void issue_order(GameEngine p_gameEngine,Parsing l_parsing) {
    ArrayList<String> l_arguments = l_parsing.getArgArr();

    switch (l_parsing.d_commandType) {
        case "deploy" -> {
            String l_countryName = l_arguments.get(0);
            int l_num = Integer.parseInt(l_arguments.get(1));

            Deploy l_deployOrder = new Deploy(this, l_countryName, l_num);
            p_gameEngine.d_logbuffer.addEntry(l_deployOrder);

            this.d_playerOrders.add(l_deployOrder);
        }
        case "advance" -> {
            String l_countryFrom = l_arguments.get(0);
            String l_countryTo = l_arguments.get(1);
            int l_numArmies = Integer.parseInt(l_arguments.get(2));

            Advance l_advanceOrder = new Advance(p_gameEngine, this, l_countryFrom, l_countryTo, l_numArmies);
            p_gameEngine.d_logbuffer.addEntry(l_advanceOrder);

            this.d_playerOrders.add(l_advanceOrder);
        }
        case "bomb" -> {
            String l_countryName = l_arguments.getFirst();

            Bomb l_bombOrder = new Bomb(this, l_countryName);
            p_gameEngine.d_logbuffer.addEntry(l_bombOrder);

            this.d_playerOrders.add(l_bombOrder);
        }
```

The new version relegates this task to an interface, liberating the Player class from a very extensive method:

```java
public void issue_order(GameEngine p_gameEngine,Parsing l_parsing) {
    Order l_order = this.d_playerStrategy.createOrder(l_parsing);

    if (l_order != null) {
        d_playerOrders.add(l_order);
        p_gameEngine.d_logbuffer.addEntry(l_order);
    }
}
```

Once any type of order is created for the player, it simply accesses the player object and adds it to the list of issued orders.

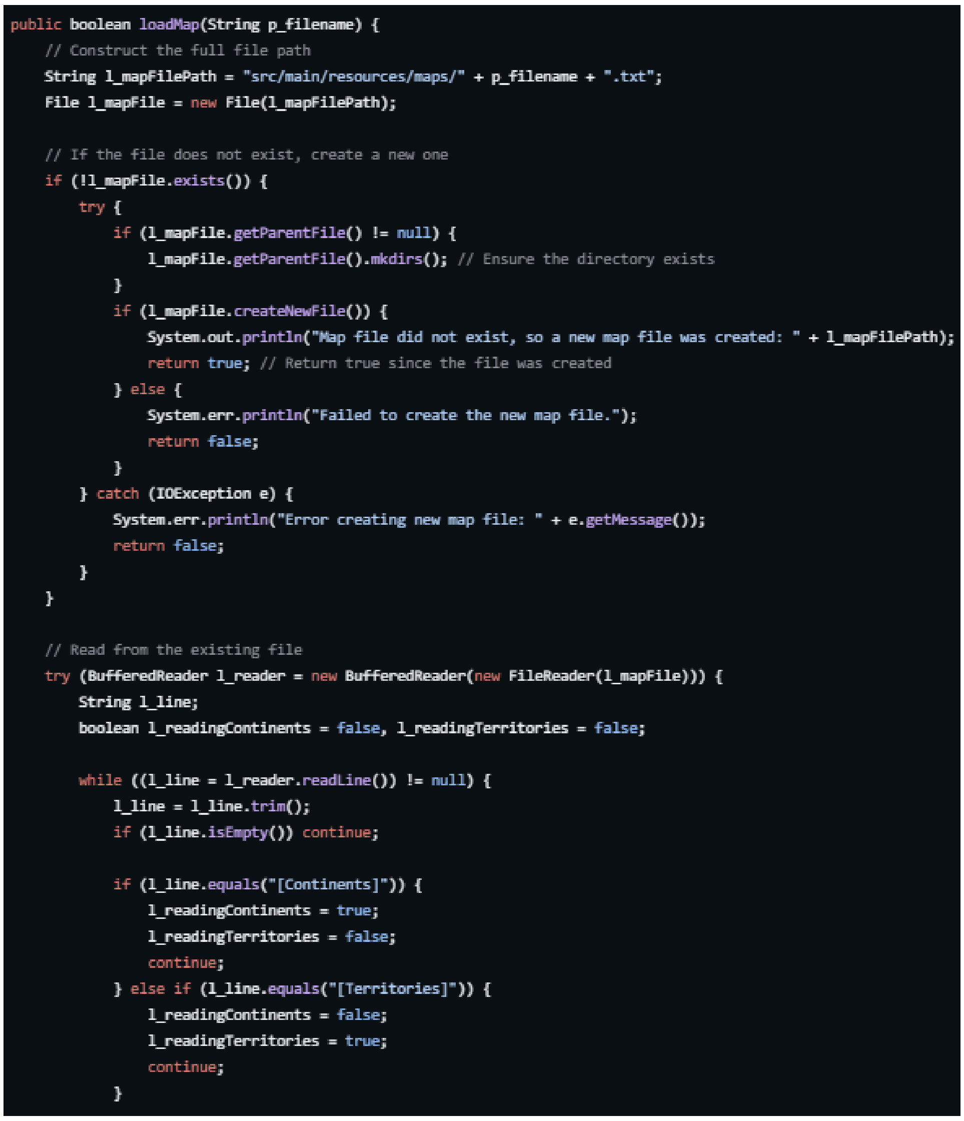
- **StartUp.java class**
  The previous *addGamePlayer()* method wasn't updated. It only maintained the information relevant to fulfil the States pattern:

```java
@Override
public void addGamePlayer(Parsing p_parsing) {
    if (p_parsing.d_argsLabeled.containsKey("-add")) {
        for (String l_playername : p_parsing.d_argsLabeled.get("-add")) {
            d_engine.d_playersList.add(new Player(l_playername));
            System.out.println("Player added: " + l_playername);
        }
    }
    if (p_parsing.d_argsLabeled.containsKey("-remove")) {
        for (String l_playername : p_parsing.d_argsLabeled.get("-remove")) {
            d_engine.d_playersList.removeIf(p -> p.getName().equals(l_playername));
            System.out.println("Player removed: " + l_playername);
        }
    }
}
```

  The new version implements the Strategy pattern and allows the user to add or remove as many players as they want in a single command line:

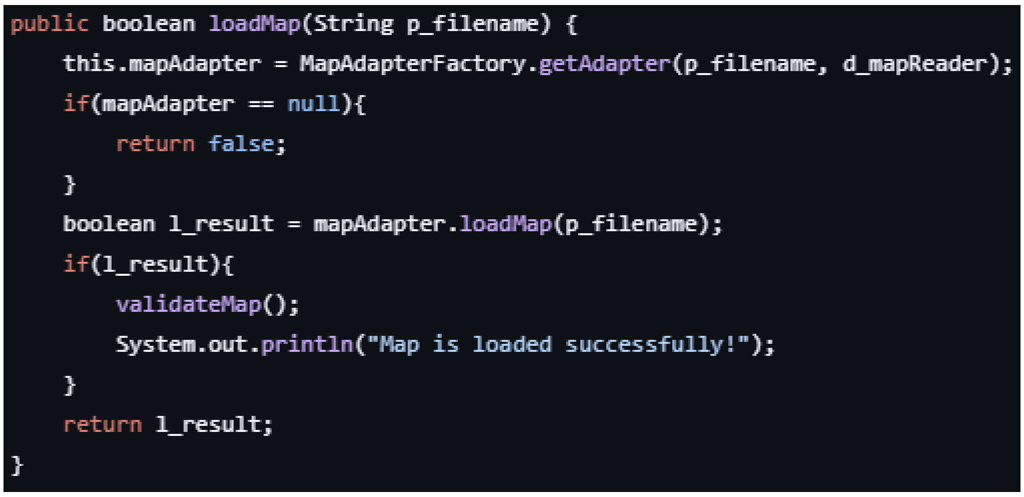
```java
@Override
public void addGamePlayer(Parsing p_parsing) {
    if (p_parsing.d_argsLabeled.containsKey("-add")) {
        for (String l_playername : p_parsing.d_argsLabeled.get("-add")) {
            Player l_player = new Player(l_playername);
            l_player.setPlayerStrategy(new HumanPlayerStrategy(this.d_engine, l_player));
            d_engine.d_playersList.add(l_player);
            System.out.println("Player added: " + l_playername);
        }
    }
    if (p_parsing.d_argsLabeled.containsKey("-remove")) {
        for (String l_playername : p_parsing.d_argsLabeled.get("-remove")) {
            d_engine.d_playersList.removeIf(p -> p.getName().equals(l_playername));
            System.out.println("Player removed: " + l_playername);
        }
    }
}
```

- **Preload.java class**
  The previous *loadMap()* method was very extensive, putting too much weight on the preload class and making it do the validations for the command itself instead of containing only the logic:



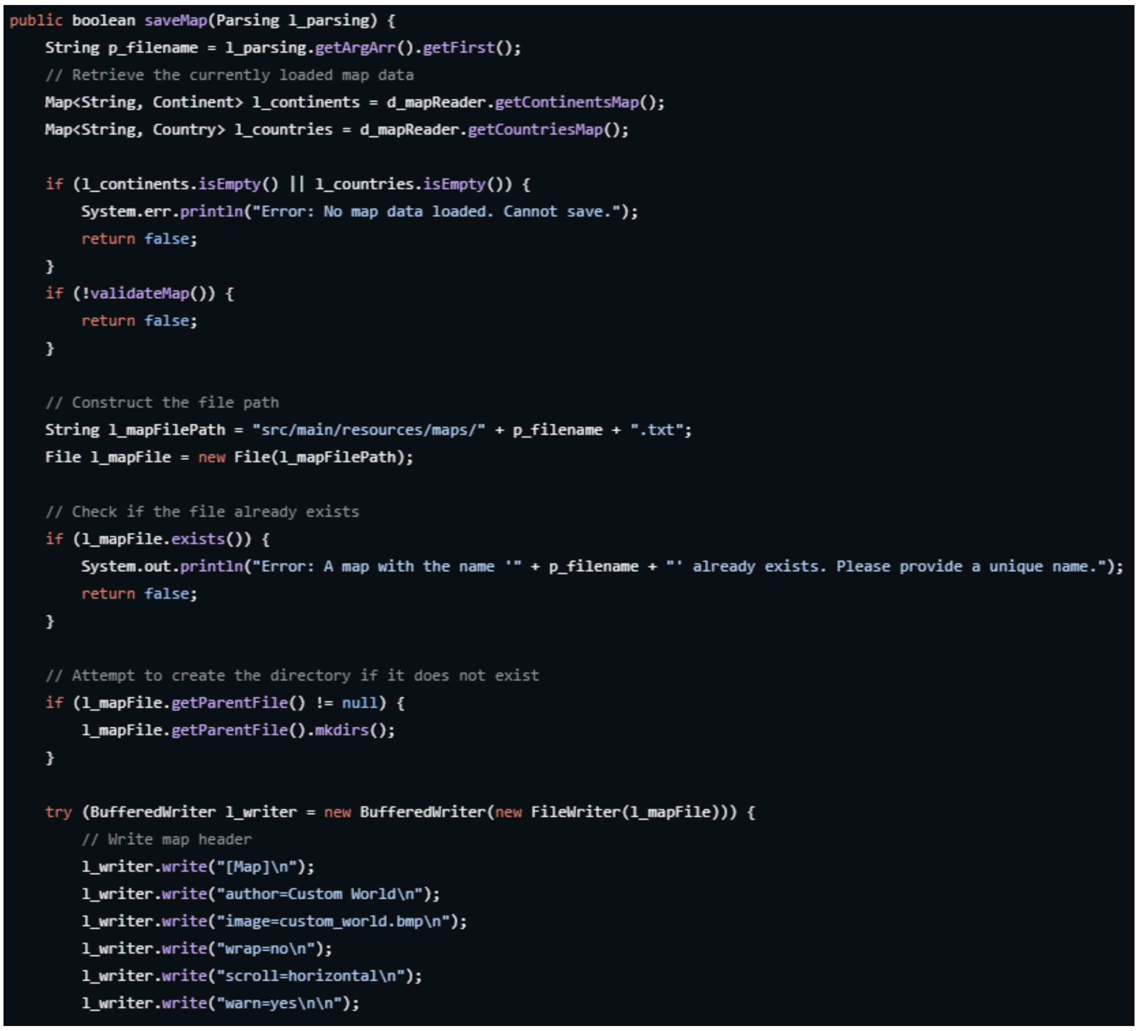
```java
public boolean loadMap(String p_filename) {
    // Construct the full file path
    String l_mapFilePath = "src/main/resources/maps/" + p_filename + ".txt";
    File l_mapFile = new File(l_mapFilePath);

    // If the file does not exist, create a new one
    if (!l_mapFile.exists()) {
        try {
            if (l_mapFile.getParentFile() != null) {
                l_mapFile.getParentFile().mkdirs(); // Ensure the directory exists
            }
            if (l_mapFile.createNewFile()) {
                System.out.println("Map file did not exist, so a new map file was created: " + l_mapFilePath);
                return true; // Return true since the file was created
            } else {
                System.err.println("Failed to create the new map file.");
                return false;
            }
        } catch (IOException e) {
            System.err.println("Error creating new map file: " + e.getMessage());
            return false;
        }
    }

    // Read from the existing file
    try (BufferedReader l_reader = new BufferedReader(new FileReader(l_mapFile))) {
        String l_line;
        boolean l_readingContinents = false, l_readingTerritories = false;

        while ((l_line = l_reader.readLine()) != null) {
            l_line = l_line.trim();
            if (l_line.isEmpty()) continue;

            if (l_line.equals("[Continents]")) {
                l_readingContinents = true;
                l_readingTerritories = false;
                continue;
            } else if (l_line.equals("[Territories]")) {
                l_readingContinents = false;
                l_readingTerritories = true;
                continue;
            }
```

  Now, thanks to the use of an interface, the preload can take care of only taking the logic of the method and notifying the user of its correct implementation:



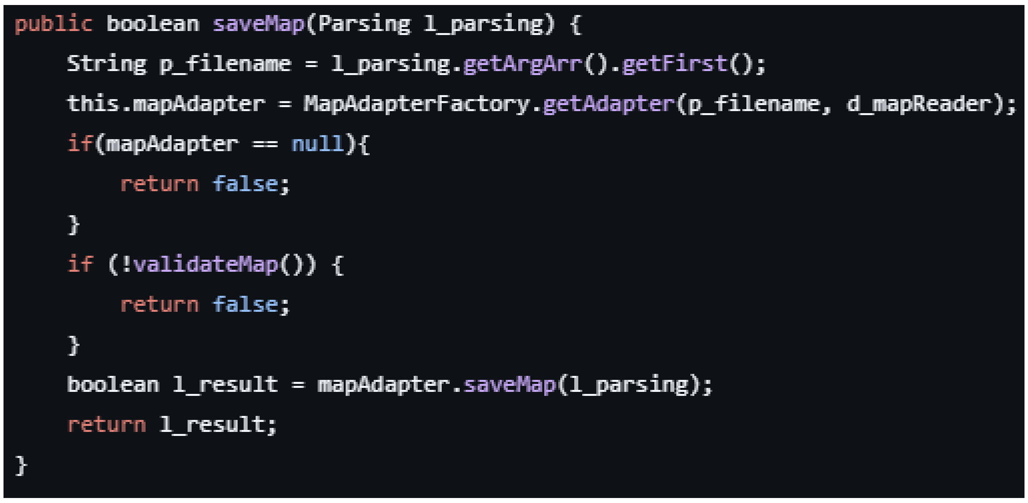
```java
public boolean loadMap(String p_filename) {
    this.mapAdapter = MapAdapterFactory.getAdapter(p_filename, d_mapReader);
    if(mapAdapter == null){
        return false;
    }
    boolean l_result = mapAdapter.loadMap(p_filename);
    if(l_result){
        validateMap();
        System.out.println("Map is loaded successfully!");
    }
    return l_result;
}
```

- **Postload.java class**
  The previous *saveMap()* method was very extensive, putting too much weight on the preload class and making it do the validations for the command itself instead of containing only the logic:



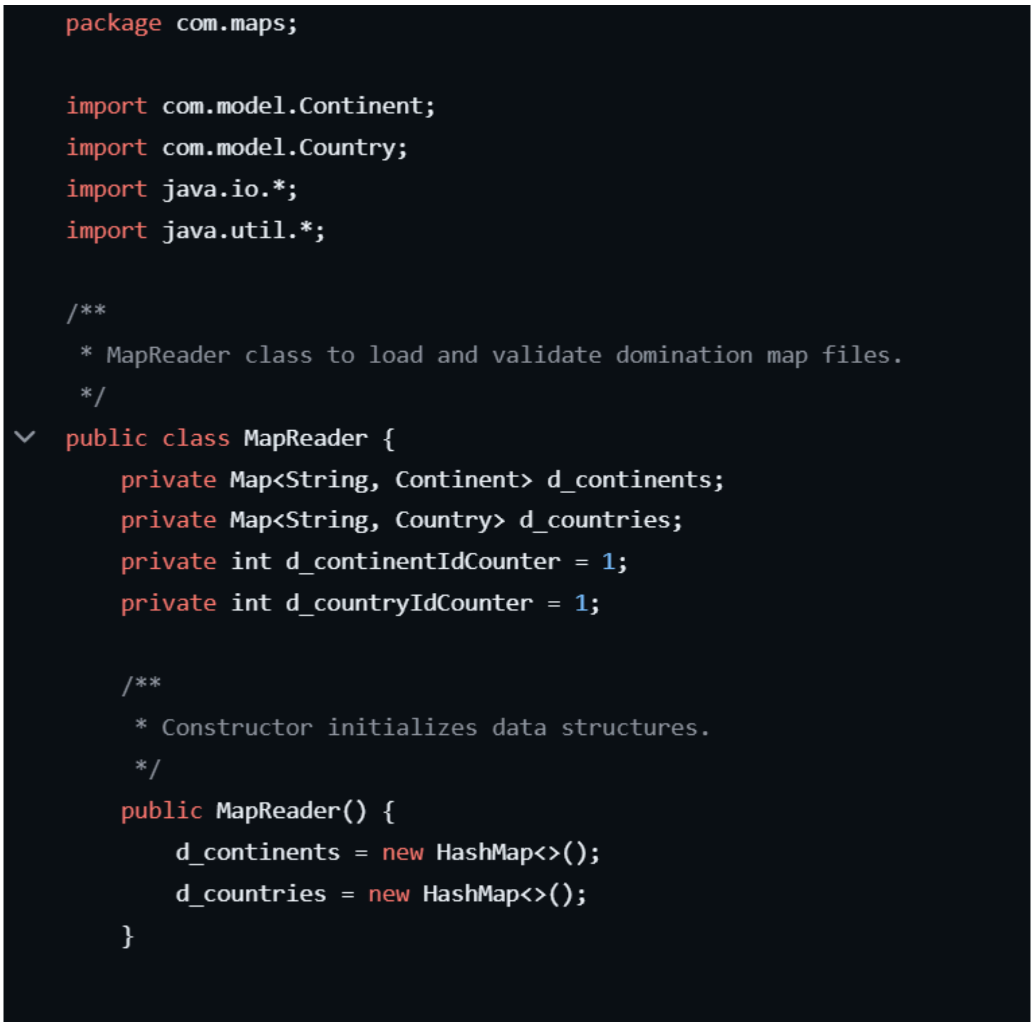
```java
public boolean saveMap(Parsing l_parsing) {
    String p_filename = l_parsing.getArgArr().getFirst();
    // Retrieve the currently loaded map data
    Map<String, Continent> l_continents = d_mapReader.getContinentsMap();
    Map<String, Country> l_countries = d_mapReader.getCountriesMap();

    if (l_continents.isEmpty() || l_countries.isEmpty()) {
        System.err.println("Error: No map data loaded. Cannot save.");
        return false;
    }
    if (!validateMap()) {
        return false;
    }

    // Construct the file path
    String l_mapFilePath = "src/main/resources/maps/" + p_filename + ".txt";
    File l_mapFile = new File(l_mapFilePath);

    // Check if the file already exists
    if (l_mapFile.exists()) {
        System.out.println("Error: A map with the name '" + p_filename + "' already exists. Please provide a unique name.");
        return false;
    }

    // Attempt to create the directory if it does not exist
    if (l_mapFile.getParentFile() != null) {
        l_mapFile.getParentFile().mkdirs();
    }

    try (BufferedWriter l_writer = new BufferedWriter(new FileWriter(l_mapFile))) {
        // Write map header
        l_writer.write("[Map]\n");
        l_writer.write("author=Custom World\n");
        l_writer.write("image=custom_world.bmp\n");
        l_writer.write("wrap=no\n");
        l_writer.write("scroll=horizontal\n");
        l_writer.write("warn=yes\n\n");
```

Now, thanks to the use of an interface, the preload can take care of only taking the logic of the method and notifying the user of its correct implementation:



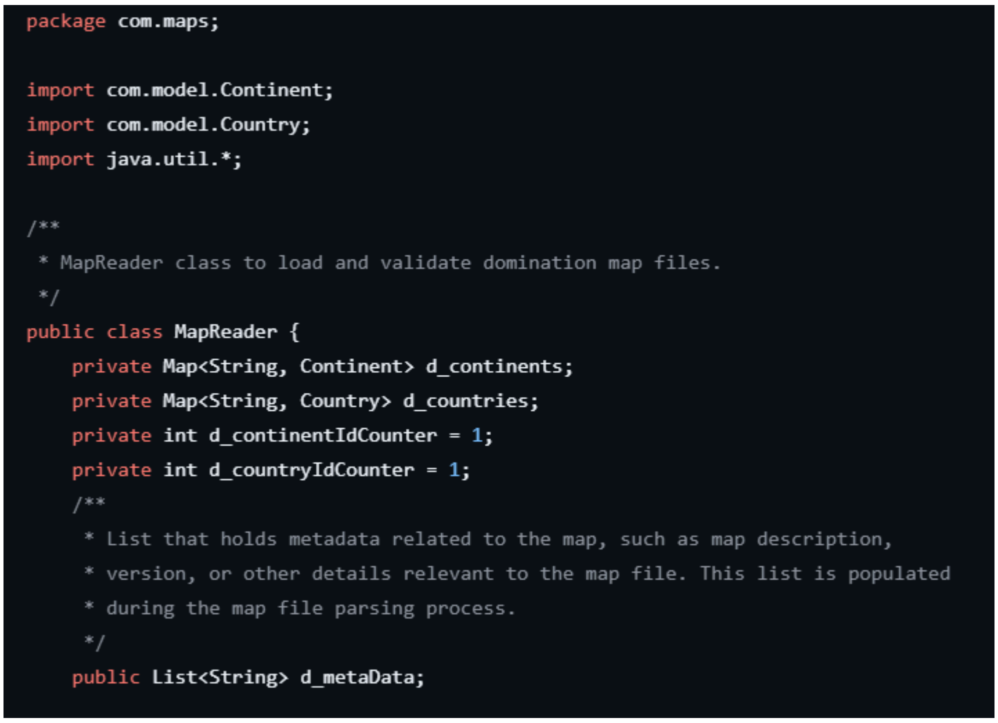
```java
public boolean saveMap(Parsing l_parsing) {
    String p_filename = l_parsing.getArgArr().getFirst();
    this.mapAdapter = MapAdapterFactory.getAdapter(p_filename, d_mapReader);
    if(mapAdapter == null){
        return false;
    }
    if (!validateMap()) {
        return false;
    }
    boolean l_result = mapAdapter.saveMap(l_parsing);
    return l_result;
}
```

- **MapReader.java class**
  The previous MapReder class did not have the capability to store the metadata.



```java
package com.maps;

import com.model.Continent;
import com.model.Country;
import java.io.*;
import java.util.*;

/**
 * MapReader class to load and validate domination map files.
 */
public class MapReader {
    private Map<String, Continent> d_continents;
    private Map<String, Country> d_countries;
    private int d_continentIdCounter = 1;
    private int d_countryIdCounter = 1;

    /**
     * Constructor initializes data structures.
     */
    public MapReader() {
        d_continents = new HashMap<>();
        d_countries = new HashMap<>();
    }
```

We updated the MapReader class to maintain an ArrayList to store the metadata of a map in case we are loading/saving a conquest map file.



```java
package com.maps;

import com.model.Continent;
import com.model.Country;
import java.util.*;

/**
 * MapReader class to load and validate domination map files.
 */
public class MapReader {
    private Map<String, Continent> d_continents;
    private Map<String, Country> d_countries;
    private int d_continentIdCounter = 1;
    private int d_countryIdCounter = 1;
    /**
     * List that holds metadata related to the map, such as map description,
     * version, or other details relevant to the map file. This list is populated
     * during the map file parsing process.
     */
    public List<String> d_metaData;
```