RISK GAME (BUILD-1) ARCHITECTURAL DESIGN

Advanced Programming Practices

SOEN 6441
Fall-2019

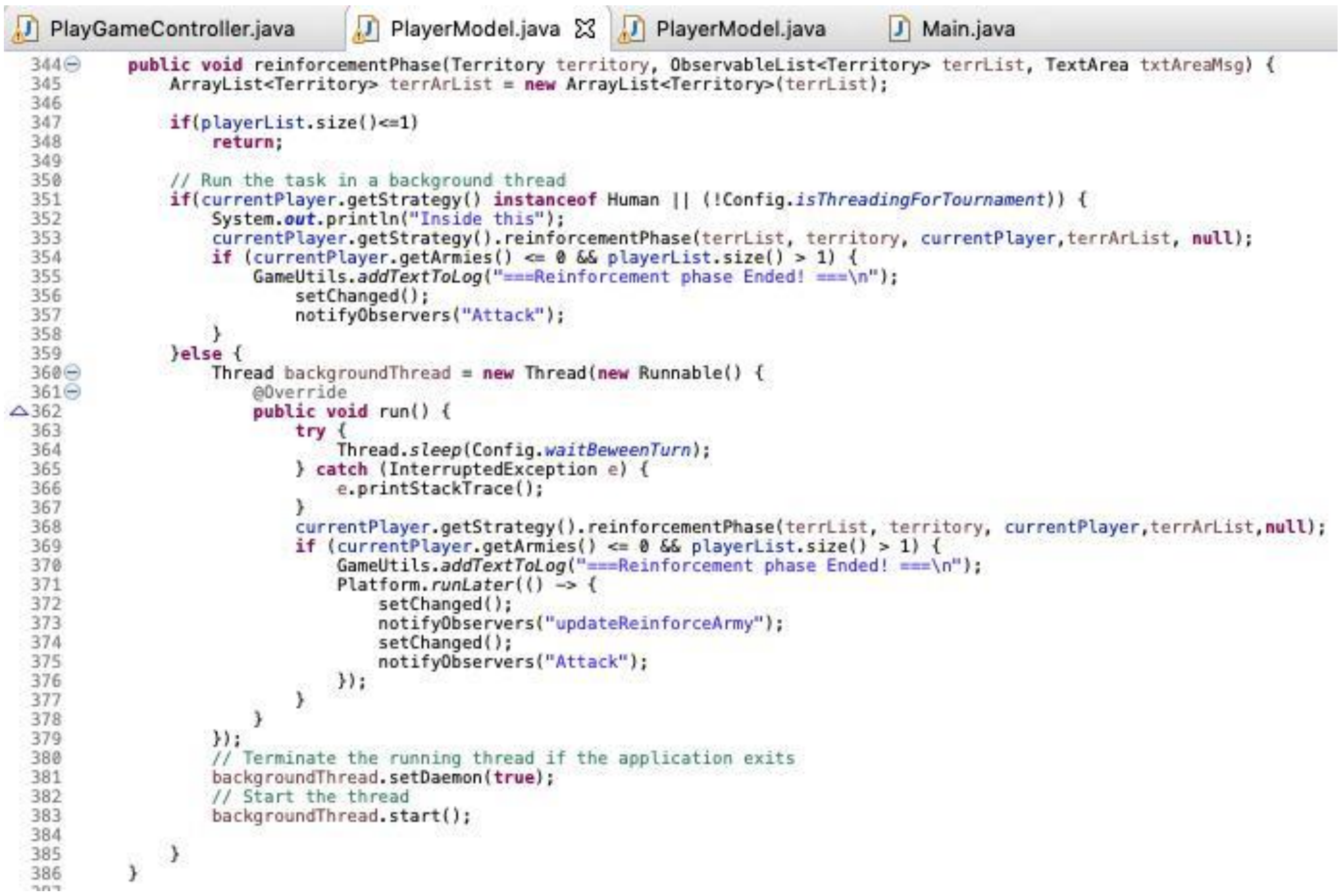
Team: Group_U_J

Mehul Prajapati	(40076930)
Komal Panchal	(40130791)
Maryam Giahi	(40016260)
Mahmoudreza Entezami	(40058782)
Parth Patel	(40081116)
Shubham Ranadive	(40083991)

Coding Conventions

1. Code Layout

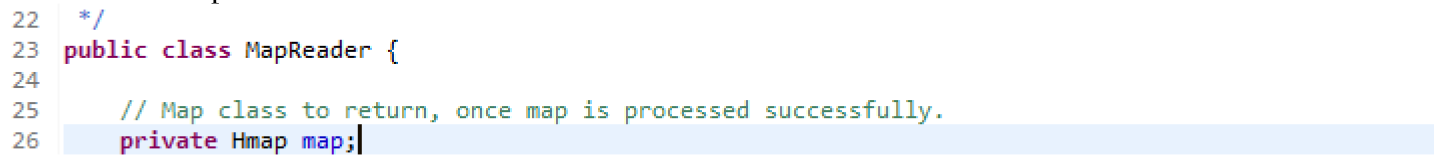
- To minimize the length and maximize the readability of the code, the curly braces are appended to the statement preceding it.
- Blank lines are added to separate sections or components to increase the readability.



```java
public void reinforcementPhase(Territory territory, ObservableList<Territory> terrList, TextArea txtAreaMsg) {
    ArrayList<Territory> terrArList = new ArrayList<Territory>(terrList);

    if(playerList.size()<=1)
        return;

    // Run the task in a background thread
    if(currentPlayer.getStrategy() instanceof Human || (!Config.isThreadingForTournament)) {
        System.out.println("Inside this");
        currentPlayer.getStrategy().reinforcementPhase(terrList, territory, currentPlayer,terrArList, null);
        if (currentPlayer.getArmies() <= 0 && playerList.size() > 1) {
            GameUtils.addTextToLog("===Reinforcement phase Ended! ===\n");
                setChanged();
                notifyObservers("Attack");
        }
    }else {
        Thread backgroundThread = new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(Config.waitBeweenTurn);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                currentPlayer.getStrategy().reinforcementPhase(terrList, territory, currentPlayer,terrArList,null);
                if (currentPlayer.getArmies() <= 0 && playerList.size() > 1) {
                    GameUtils.addTextToLog("===Reinforcement phase Ended! ===\n");
                    Platform.runLater(() -> {
                        setChanged();
                        notifyObservers("updateReinforceArmy");
                        setChanged();
                        notifyObservers("Attack");
                    });
                }
            }
        });
        // Terminate the running thread if the application exits
        backgroundThread.setDaemon(true);
        // Start the thread
        backgroundThread.start();

    }
}
```

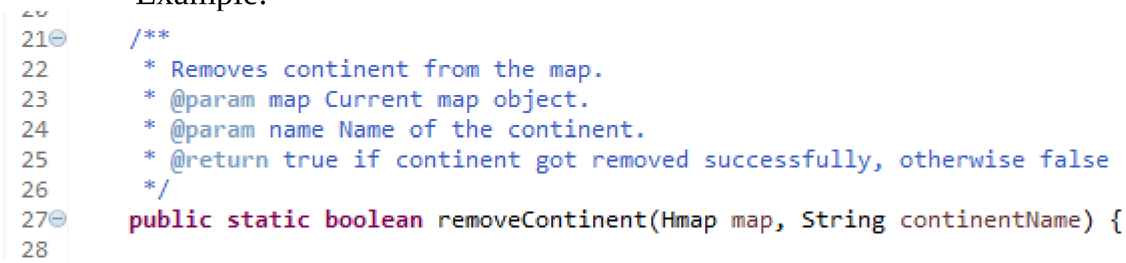
2. Naming Conventions

- Class name:
  - The class names are written as the first letter as the uppercase of each word.
  - Example:



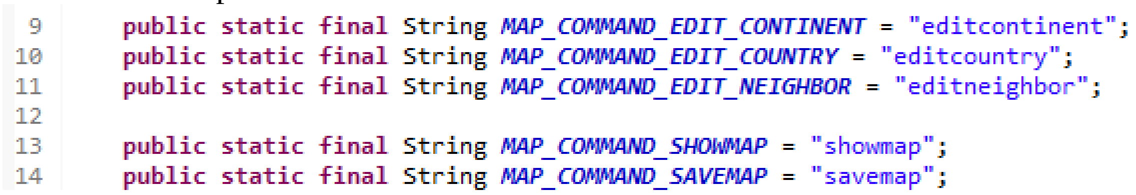
```
 */
public class MapReader {

    // Map class to return, once map is processed successfully.
    private Hmap map;
```

- Function name:
  - Each function name is started with the lower case, followed by the uppercase of first letter of other words to bifurcate between the words.
  - Example:



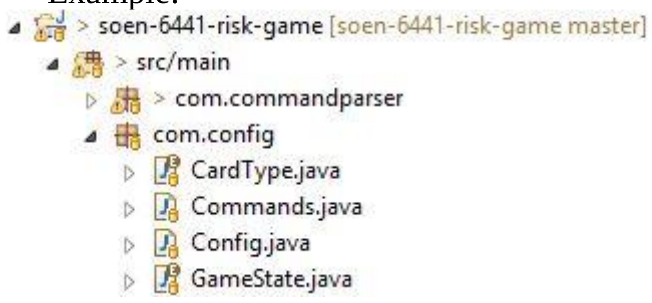
```
    /**
     * Removes continent from the map.
     * @param map Current map object.
     * @param name Name of the continent.
     * @return true if continent got removed successfully, otherwise false
     */
    public static boolean removeContinent(Hmap map, String continentName) {

```

- Variable name:
  - Constant variables is started with **all uppercase** with words separated by underscores (“_”).
  - Example:



```
    public static final String MAP_COMMAND_EDIT_CONTINENT = "editcontinent";
    public static final String MAP_COMMAND_EDIT_COUNTRY = "editcountry";
    public static final String MAP_COMMAND_EDIT_NEIGHBOR = "editneighbor";

    public static final String MAP_COMMAND_SHOWMAP = "showmap";
    public static final String MAP_COMMAND_SAVEMAP = "savemap";
```

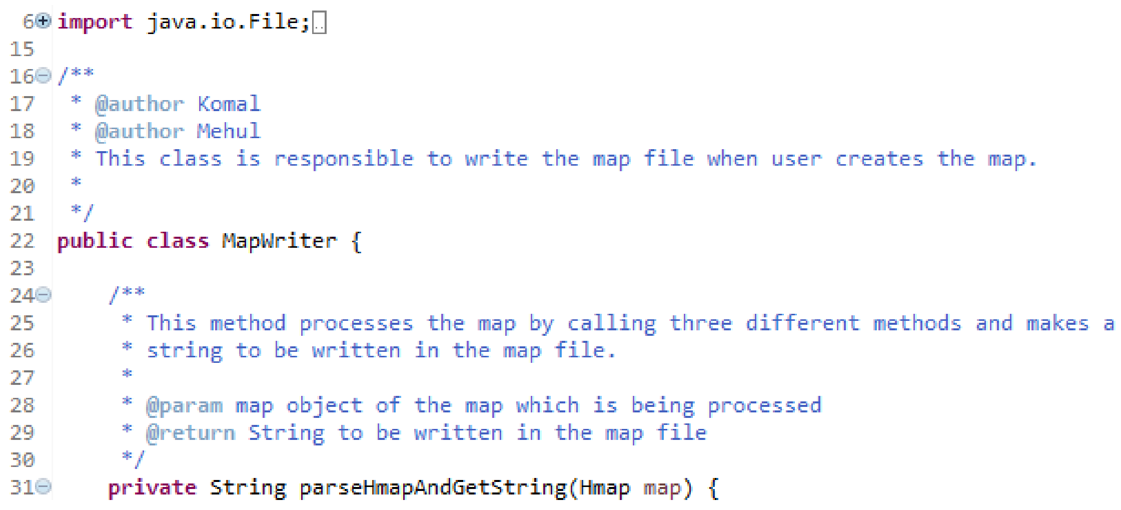
- Folder and Package names:
  - All folder and package names are written in lowercase.
  - Example:



```
▲ > soen-6441-risk-game [soen-6441-risk-game master]
  ▲ > src/main
    ▷ > com.commandparser
    ▲ com.config
      ▷ CardType.java
      ▷ Commands.java
      ▷ Config.java
      ▷ GameState.java
```

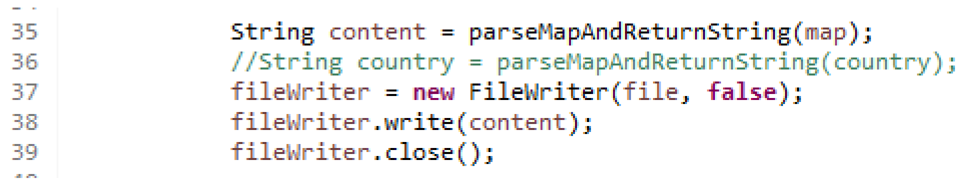
3. Comments

- Commenting is done as per conventions for Java Doc.
- Each class declaration precedes by a comment explaining what the class is for.
- Each method or function have comments explaining what it does, as well as what is the purpose of parameters and return type description if the method’s return is non-void.



```
import java.io.File;

/**
 * @author Komal
 * @author Mehul
 * This class is responsible to write the map file when user creates the map.
 *
 */
public class MapWriter {

    /**
     * This method processes the map by calling three different methods and makes a
     * string to be written in the map file.
     *
     * @param map object of the map which is being processed
     * @return String to be written in the map file
     */
    private String parseHmapAndGetString(Hmap map) {
```

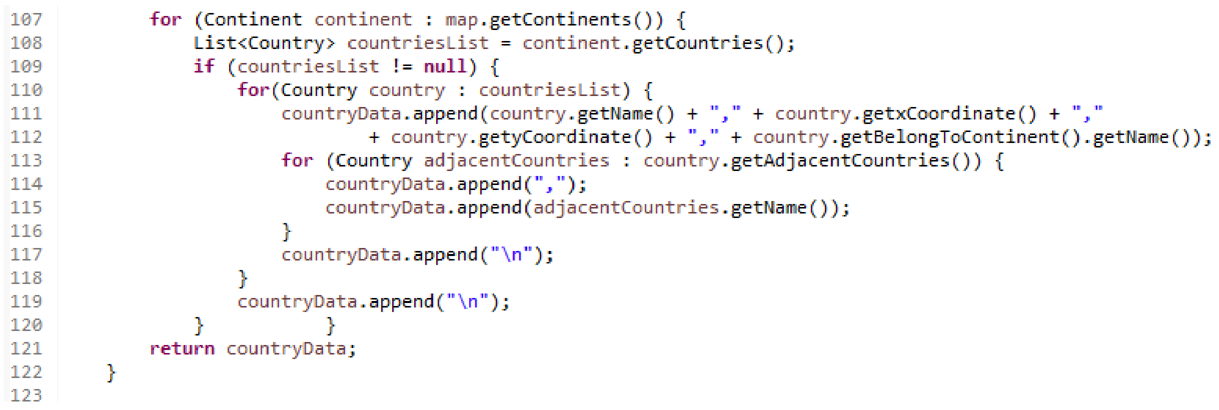
- “commented out” code:



```
            String content = parseMapAndReturnString(map);
            //String country = parseMapAndReturnString(country);
            fileWriter = new FileWriter(file, false);
            fileWriter.write(content);
            fileWriter.close();
```

4. Indentation

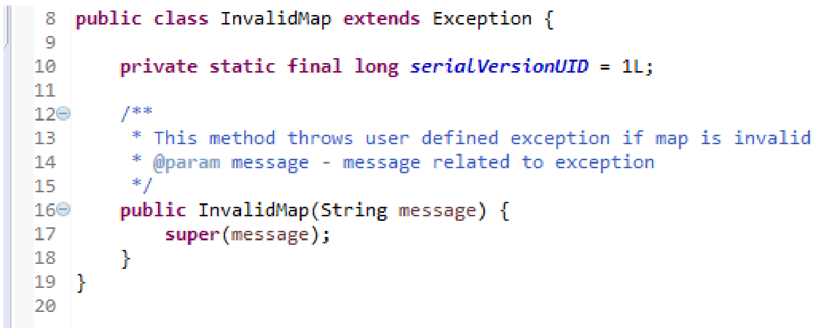
- Code is indented according to its nesting level to improve code readability. Indentation of the body of the function is done with respect to its header. Similarly, for the for, while, switch, if and other statements, it is done with respect to its first line.



```
        for (Continent continent : map.getContinents()) {
            List<Country> countriesList = continent.getCountries();
            if (countriesList != null) {
                for(Country country : countriesList) {
                    countryData.append(country.getName() + "," + country.getxCoordinate() + ","
                            + country.getyCoordinate() + "," + country.getBelongToContinent().getName());
                    for (Country adjacentCountries : country.getAdjacentCountries()) {
                        countryData.append(",");
                        countryData.append(adjacentCountries.getName());
                    }
                    countryData.append("\n");
                }
                countryData.append("\n");
            }           }
        return countryData;
    }
```

5. Exception Handling

- Exception handling is done using InvalidMap class file
- User defined exception can be defined using this class.



```java
public class InvalidMap extends Exception {

    private static final long serialVersionUID = 1L;

    /**
     * This method throws user defined exception if map is invalid
     * @param message - message related to exception
     */
    public InvalidMap(String message) {
        super(message);
    }
}

```

References:

1. https://www.geeksforgeeks.org/java-naming-conventions/
2. https://google.github.io/styleguide/javaguide.html#s6.2-caught-exceptio