Refactoring of Build 1

We identified some refactor operations which are needed before starting a new Build. For identification of refactoring targets we used tools like sonarlint, sonarqube and also went through the file project and identified some of the targets which are listed below in the table.

The List of Refactoring which we found in our project

Sr. No.	**File and LineNumber**	**Issue**	**Priority [high, low , medium]**
1	GameEngine.java: 129,132	Multiple switch conditions, 3 can be combined into 1	High
2	GameEngine.java:329 startGameLoop()	Cognitive Complexity of methods should not be too high. Method can be refactored into multiple ones	Medium
3	GameEngine.java : 73 startGame()	While condition doesn't have any false condition only system exit statement. Code can be refactored to have this exit command condition as condition for while loop and exit statement and print 'thank you' after loop.	Medium
4	Constants.java : 5	Constant class should not have implicit public constructor, only private constructor so its object cannot be created by anyone and only static members can be accessed.	High
5	MapHelper.java: 43 [Multiple lines]	String literals should not be duplicated. Move common strings to constants folder and use them here	Medium
6	GameEngine.java: 195 [Strings in multiple exceptions]	String literals should not be duplicated. Move common strings to constants folder and use them here	Medium
7	MapHelper.java: 169, 208, 245	Checking the same conditions multiple times. Can eliminate duplication of code by merging it into one function.	Medium
8	Map.java: 408, 423	Merge the functionalities of adding and removing the neighbours, rather than checking for same conditions multiple times.	Low
9	MapHelper.java: 52	Created continent objects are not assigned to the map using the setter methods	Very high
10	MapHelper.java: 364, 371	Multiple initialization of the string object and overwriting the unnecessary initialization. Redundancy can be removed	Low
11	Playerhelper.java: 272	Math.round function should round the number after	Low

		dividing it to three, and not the size as it is already a integer value.	
12	MapHelper.java: 124, 286	Variable name c shall be renamed into l_country to following the naming convention of our coding guidelines	High
13	MapHelper.java: 310	Else statement is not needed since the conditional if statement on line 307 has a return statement on line 307	Low
14	MapHelper.java: 401	Building a file path using concatenated strings is not recommended in java as the string might not be os agnostic. It is better to use java.nio.file.Paths	Low
15	GameState,java: 14, 19, 24, 29,	The data members of the GameState class shall be encapsulated	High

Refactors which are implemented

1. Refactoring: Removing Duplicate String Literals in MapHelper Class and Adding to Constant Class

   List of test cases which are involved:
   - testAddContinent()
   - testRemoveContinent()
   - testAddCountry()
   - testRemoveCountry()

   The reason to choose this refactor was that there was multiple use of constants string literals which can be done easily by just creating constants and to avoid the human error while using the same string in future.

2. Refactoring : Multiple switch conditions, [3 conditions can be combined into 1]
   List of test cases involved to check the refactoring:
   - testAddContinent()
   - testRemoveContinent()
   - testAddCountry()
   - testAddNeighbors()

Code change ss:

```
case "editcontinent":
    d_gameEngine.commonCommandExecutorWithArgumentsAndOperations(l_playerCommand, l_firstCommand);
    break;

case "editneighbor":
    d_gameEngine.commonCommandExecutorWithArgumentsAndOperations(l_playerCommand, l_firstCommand);
    break;
case "editcountry":
case "editcontinent", "editneighbor", "editcountry":
    d_gameEngine.commonCommandExecutorWithArgumentsAndOperations(l_playerCommand, l_firstCommand);
    break;
```

This refactoring was picked because these commands reused the same code in switch statements and so can be combined into one to prevent multiple changes in future for same methods.

3. Refactoring : Eliminated multiple instances of checking same conditions by implementing a check function.
4. Refactoring: While loading the map, update the continents of the map after linking countries with the respective continents.
   Test cases involved:
   - testIsValidMap()
   - testNoContinent()
5. Refactoring: The data members of the GameState class shall be encapsulated. They were modified into private encapsulation level to prevent other classes from directly accessing its data types. Here are the testcases related to the functionality:
   - testIsValidMap()
   - testCountriesConnectedSuccess()
   - testDiplomacyExecution()

This refactoring was picked to be complaint with cybersecurity principles. Exposing a class data member presents some high risk of vulnerability.