**Refactoring for Build 2**

**Potiential List of Refactoring**

1. Use try-with-resource

* WriteMap.java 38

1. Remove Explicit Type Argument

* Continent.java 25
* Country.java 26
* GameEngine.java 314
* GameMap.java 244
* MapValidation.java 132
* MapValidationTest.java 144 152

1. Split Multiple Variable Declaration

* ReadMap.java 52
* WriteMap.java 37

1. Use Collection Singleton List

* GameStarterTest.java 5

1. Remove Collection :: addAll

* GameEngine.java 314

1. Reuse Random Objects

* GameEngine.java 29

1. Use Secure Random

* GameEngine.java 14

1. Remove Boxing for String Conversion

* GameMap.java 250 254

1. Reorder String Equality Check

* Commands.java 80 100 105 110 114 120 145 150 155 159 165 190 195 200 205 211 234 253 273 277 282 286 292
* GameEngine.java 110 132 154 201
* Player.java 142 145 151 155
* ReadMap.java 63 78 94
* UserCommand.java 37

1. Replace assignment with Compound Operator

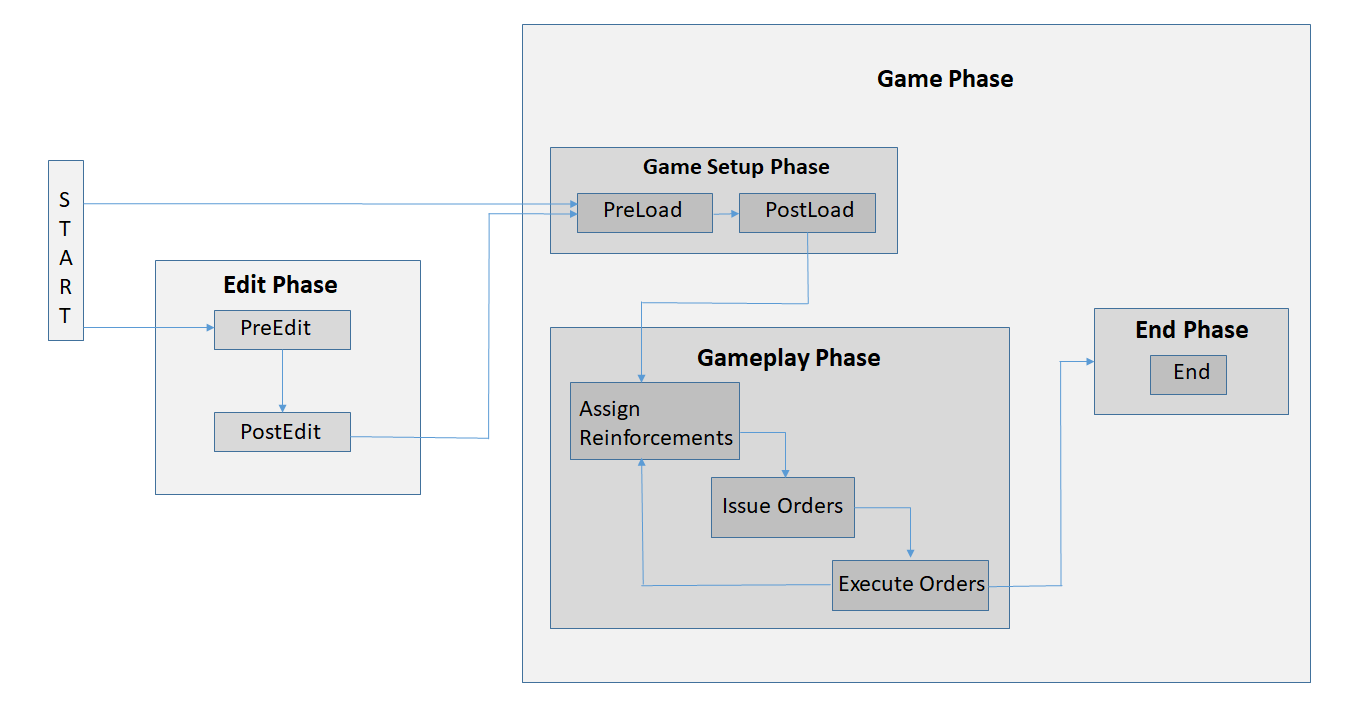
* Commands.java 109 118 154 163 199 209 281 290

1. Use StringBuilder::Append

* GameEngine.java 44 47 278
* GameMap.java 245
* GameStarter.java 49
* ReadMap.java 51
* WriteMap.java 40 50 62

1. State Pattern
2. Removed Commands.java
3. Move executeCommand() to UserCommand class from GameEngine
4. Reuse Random variable

**Refactoring Operations**

1. **State Pattern**

The state pattern in Java is a behavioural software design pattern that allows an object to alter its behaviour when its internal state changes. ... The state design pattern is generally used in cases where an object depends on its state and its behavior must be changed during run time depending on its internal state.

In our project we refactored our code for build 1 to support state pattern design.

State Pattern consists of the following phases:

1. Edit Phase
2. Game Phase consisting of 2 phases residing in it

1) Game Setup phase

2) Game Play phase

3) End Phase

**Edit Phase**

This phase is divided into 2 phases namely PreEdit and PostEdit.

These phases allow the game to executed commands that are valid in the editing phase of the game.

editmap, editContinent, editCountry, editNeighbor, saveMap are commands valid in these phases.

**Game Setup Phase – Game Phase**

This phase is divided into 2 phases namely PreLoad and PostLoad.

These phases allow the game to executed commands that are valid in the loading phase of the game.

laodMap, gamePlayer, assignCountries, addPlayer, removePlayer are commands valid in these phases.

**Game Play Phase – Game Phase**

This phase is divided into 3 phases namely AssignReinforcements, IssueOrder and ExecuteOrder.

These phases allow the game to executed commands that are valid while playing the game.

assignArmies, issueOrder, deploy, advance, bomb, airlift, blockade, diplomacy, executeOrder are commands valid in these phases.

**End Phase**

This phases prints the name of the winner.

**List of Tests:**

GameEngineTest.java:

* testExecuteCommand
* testPhase
* testLoadMap
* testEditContinent
* testEditCountry
* testEditNeighbor
* testEditMap
* testSaveMap
* testGamePlayer
* testValidateMap
* winningTest

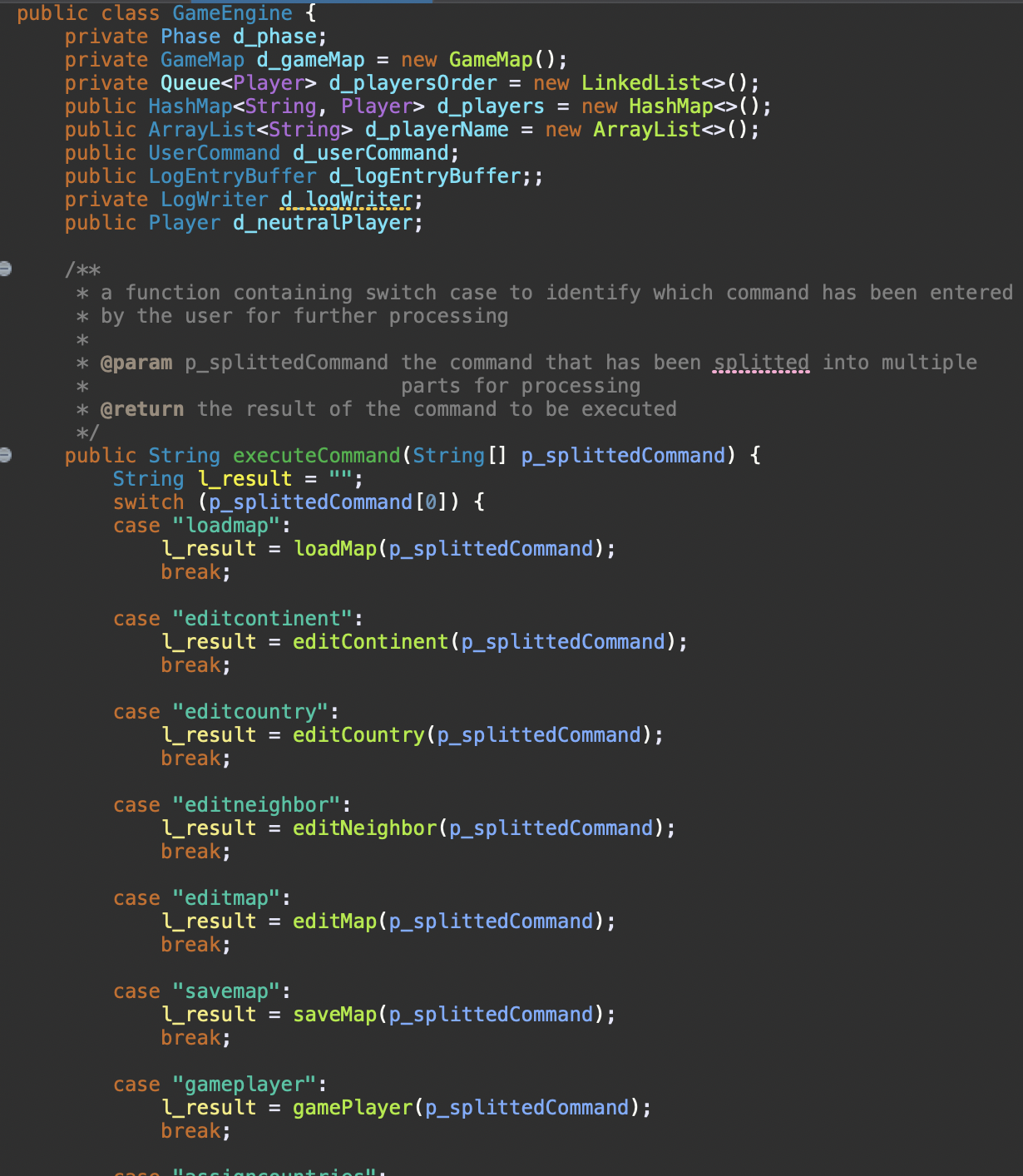
1. **Removed Commands.java**

Commands.java is removed and all its functionalities are transferred to GameEngine class, GameEngine class acts as the brain of the game. GameEngine is called by UserCommand class which takes necessary commands from user and transfer those commands to GameEngine which contains different methods to handle the user command.

**Before**:

****

**After**:



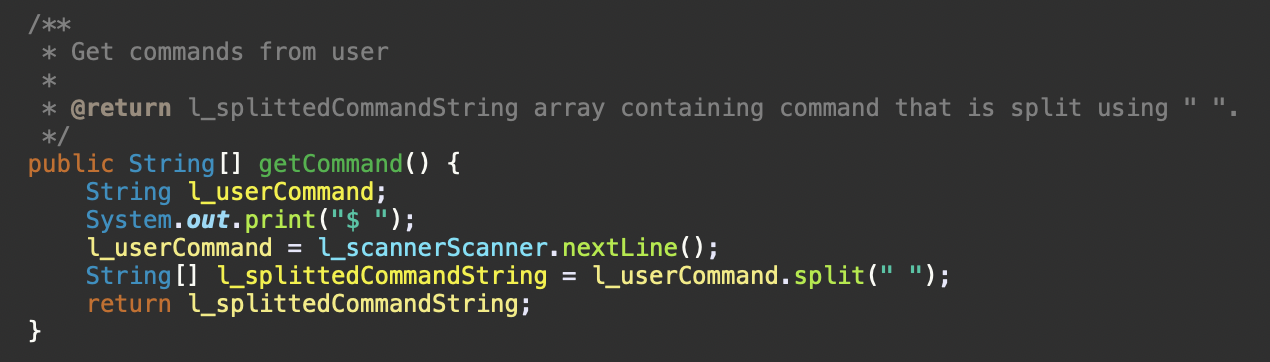
**List of Tests:**

GameEngineTest.java:

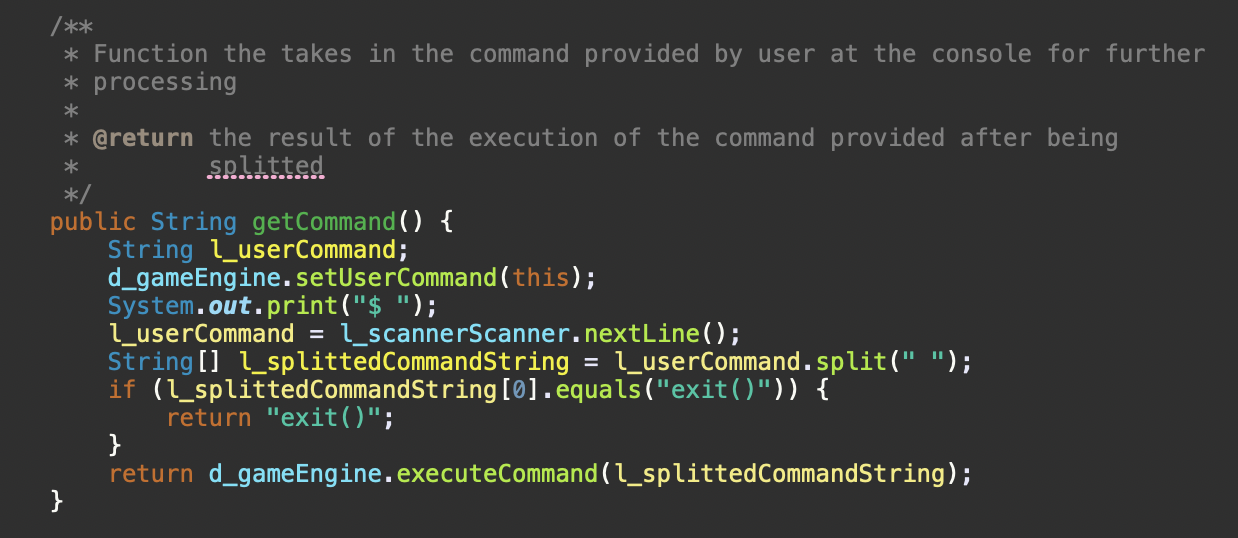
* testExecuteCommand
* testPhase
* testLoadMap
* testEditContinent
* testEditCountry
* testEditNeighbor
* testEditMap
* testSaveMap
* testGamePlayer
* testValidateMap
* winningTest

1. **UserCommand.java**

The execute command has been moved to UserCommand.java so that whenever a command is given by user suitable method is called to handle the command from executeCommand method.

**Before**:

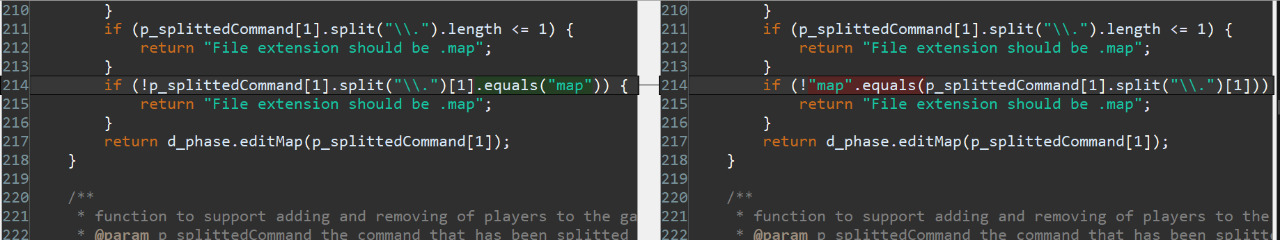
**After**:



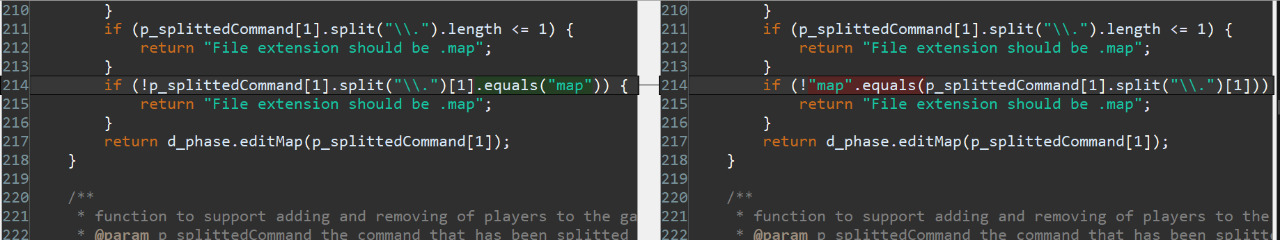
1. **NullPointerExceptions**

Null pointer exceptions were removed during the refactoring phase.

These were removed by introducing a specific syntax for .equals method all throughout our project.

**Before:**

**After:**

****

**List of Tests:**

GameEngineTest.java:

* testExecuteCommand
* testPhase
* testLoadMap
* testEditContinent
* testEditCountry
* testEditNeighbor
* testEditMap
* testSaveMap
* testGamePlayer
* testValidateMap
* winningTest

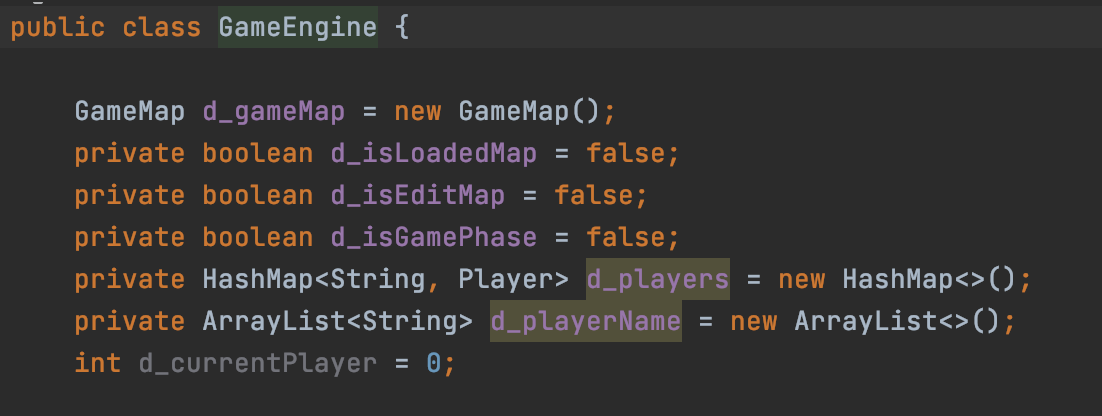
ReadMapTest.java:

* testReadFullMap1
* testReadFullMap2

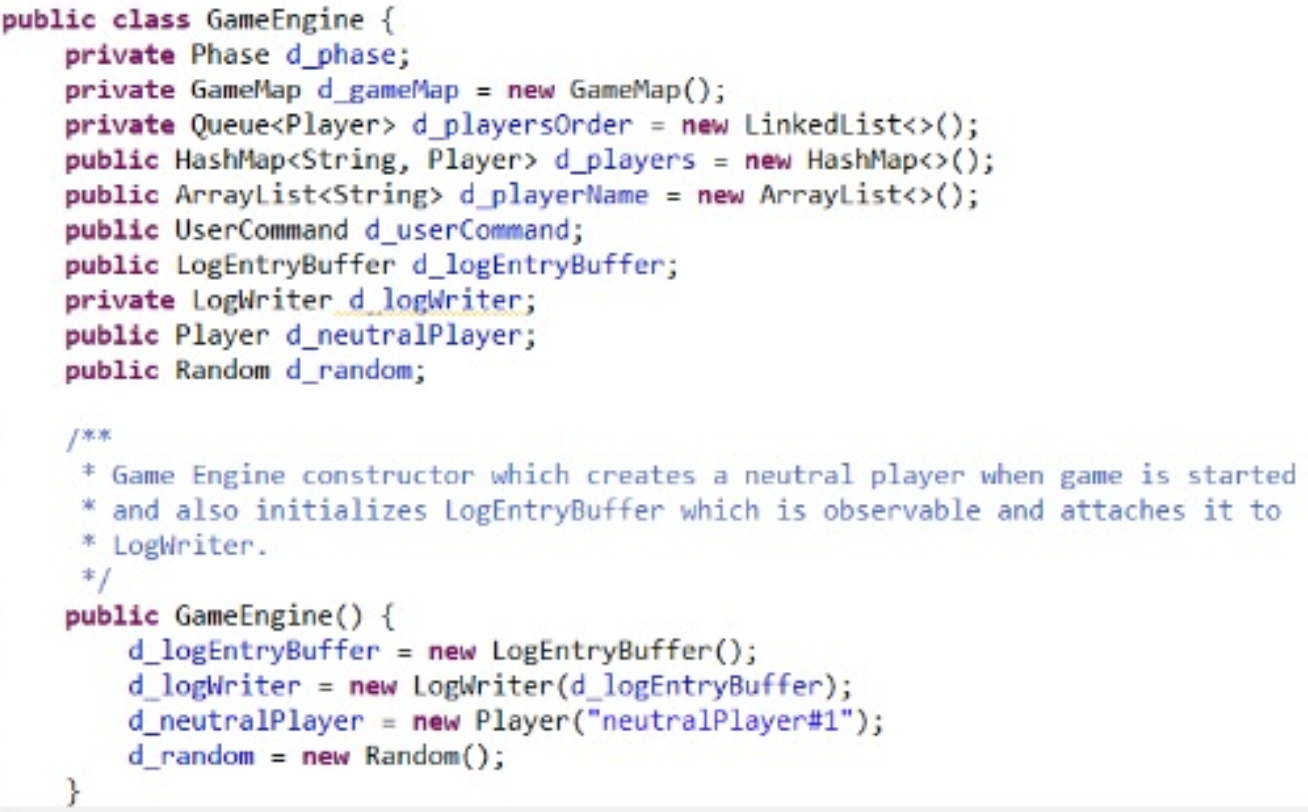
1. **Reuse Random Objects**

Creating a new random object each time a random value is needed is insufficient and produce numbers which are not random. A random variable data member is created in GameEngine which is used by other classes having GameEngine class object.

**Before:**



**After:**

****

**List of Tests:**

AdvanceTest.java:

* testExecuteOrder2
* testExecuteOrder6