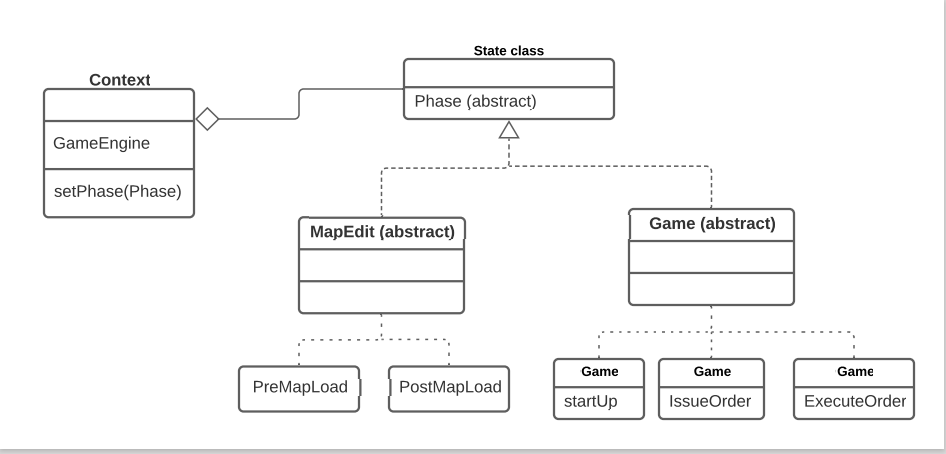


Build 1 Architecture

* In build 2 we have main **5 major changes** in our existing code which are as below: -

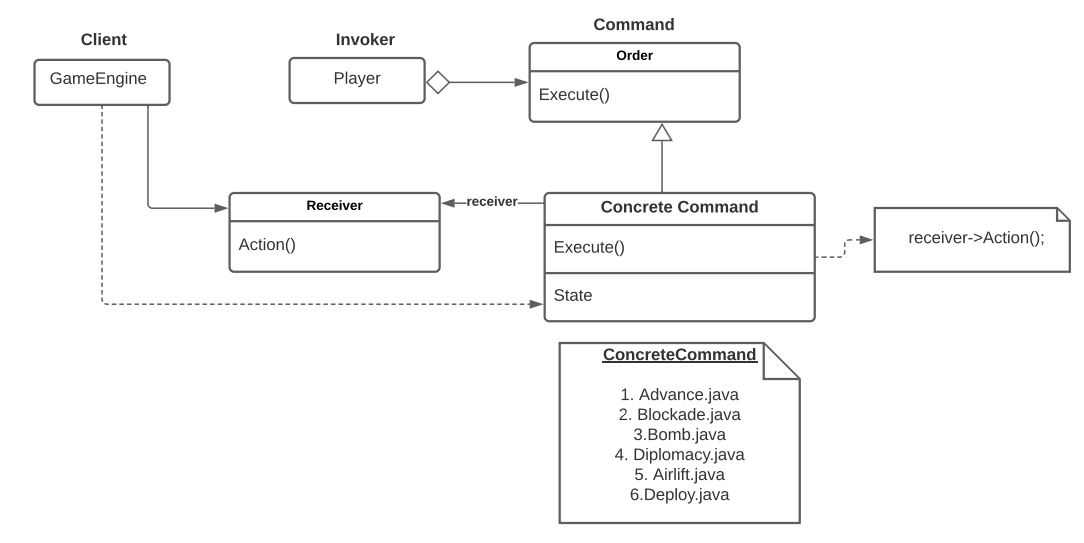
1. **Implementation of state pattern**
2. **Implementation of command pattern**
3. **Implementation of observer pattern**
4. **Implementation of different orders for player**
5. **Advance**
6. **Airlift**
7. **Bomb**
8. **Blockade**
9. **Diplomacy**
10. **Refactoring in exist code.**

**State Pattern: -**



* Here context class is GameEngine class.
* State class is Phase class.
* To implement state pattern in our existing code we mainly divide our existing MapEdit and Game classes into different phases.
* MapEdit class further divided into below phases
  + PreMapLoad
  + PostMapLoad
* Game classes divided into below phases
* StartUp
* IssueOrder
* ExecuteOrder

**Command Pattern: -**



* Here client is GameEngine class.
* Invoker is Player class.
* Command is Order class.
* To implement command pattern in our existing code we mainly make 6 different concrete command for different orders the concrete command for command pattern as below
* Advance.java
* Blockade.java
* Bomb.java
* Diplomacy.java
* Airlift.java
* Deploy.java
* The orders are created as the player executes its issue\_order () method, and the orders are executed when the GameEngine gets the Player’s orders from the Players using the next\_order () method, then executes the orders by calling the execute () method of the Order.

**Possible Refactoring Targets :**

Listed below are 15 potential refactoring targets:

1. Implement State Pattern

2. Implement Command Pattern

3. Split EditMap functionality into CreateMap for non-existing map and LoadMap for existing map

4. Merge MapEngine And GameEngine into one GameEngine class that handles entire control of the Game.

5. Move all functions related to Game in GameUtils package.

6. Remove ContinentID from Country Class since BelongsToContinent is already present. [Country.java]

7. Remove getPlayerFromPlayerID since it is not used anywhere. [Player.java]

8. Rename getCountryFromCountryName() to getCountry(). [Country.java]

9. Rename getContinentFromContinentName() to getContinent(). [Continent.java]

10. Merge displayEditorMap() and displayGameMap() into one. [ShowMap.java]

11. Rename d\_PlayerList to Players to store all the players of the game. [Main.java]

12. Rename removeContinentFromContinentList() to removeContinent(). [Map.java]

13. Rename addContinentToContinentList() to addContinent(). [Map.java]

14. Rename removeCountryFromCountryList() to removeCountry(). [Continent.java]

15. Rename addCountryToCountryList() to addCountry(). [Continent.java]

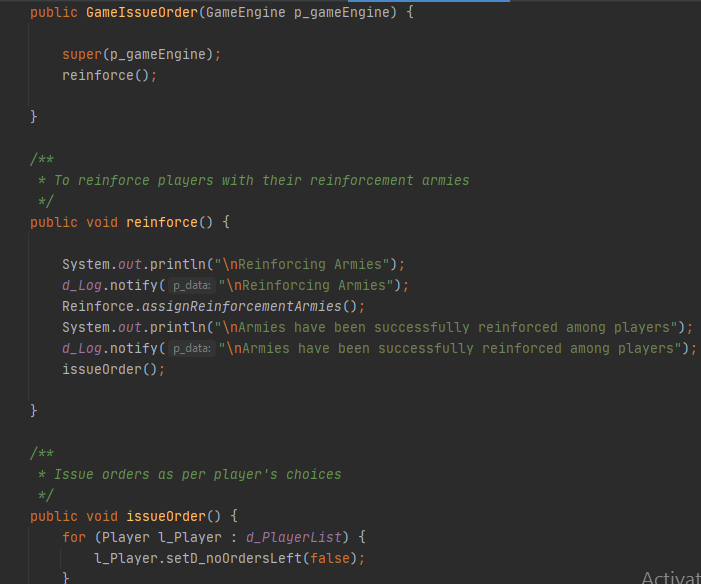
**Refactoring: -**

1. Implemented State Pattern

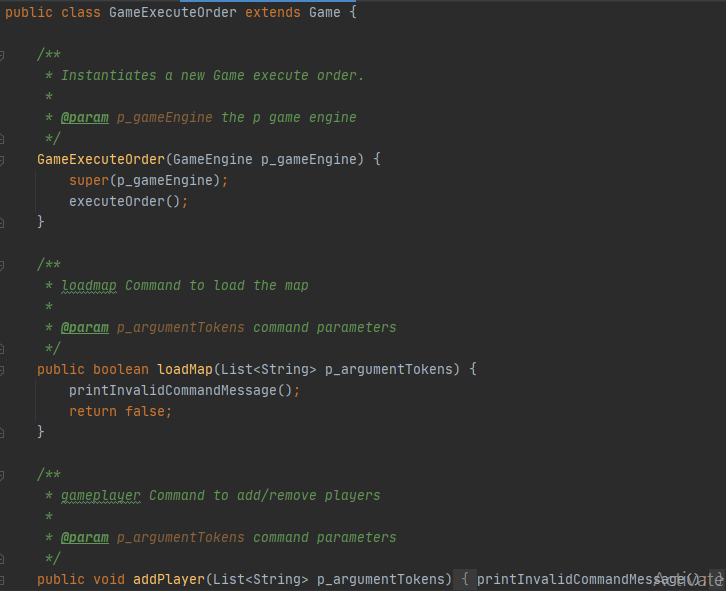
The State pattern implements the phases of the application, including the phases in the map editor, and the game play. The game play is divided into the following phases: Game Startup, Game Issue Order, and Game Execute Order phases. The context class of the State pattern is GameEngine class, and the State class is a new class named Phase.



* 1. Refactoring of *GameStartup* class



* 1. Refactoring of *GameIsssueOrder* class



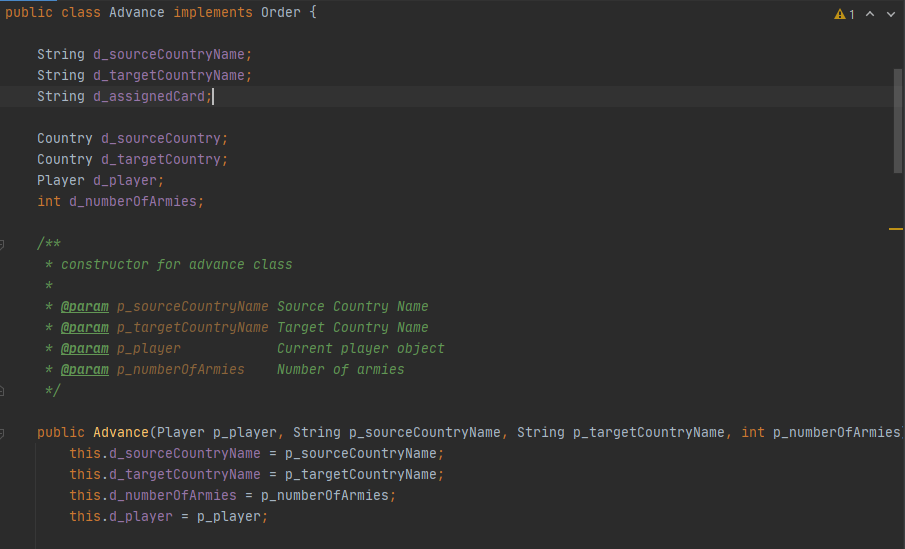
1.3 Refactoring of *GameExecuteOrder* class

Tests : Created tests to check the validity of execution of commands in their states. Invalid commands in a certain phase are not allowed to execute

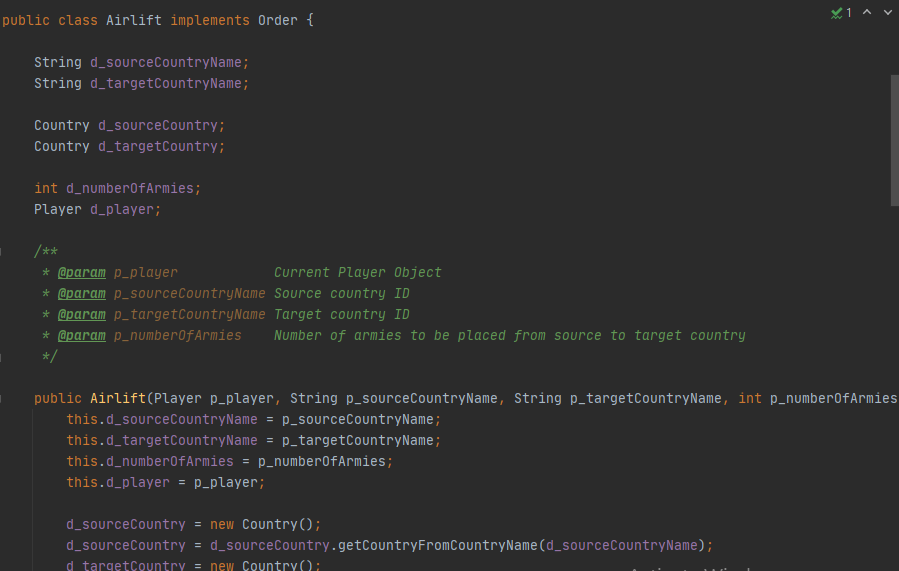
1. Implemented Command Pattern

The Command class is the Order class, the Invoker Class is the Player, and the Client class is the GameEngine. The orders are created as the player executes its issueOrder() method, and the orders are fetched by getting orders from the Players using the nextOrder() method, then executed by calling the execute() method of the Order.

Six Concrete Implementation of Abstract Order class are done: Advance, Airlift, Blockade, Bomb, Deploy, Diplomacy.



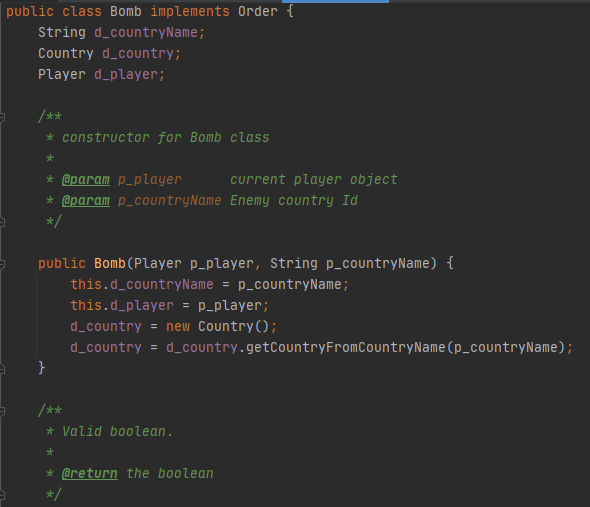
* 1. Refactoring of *Advance* class



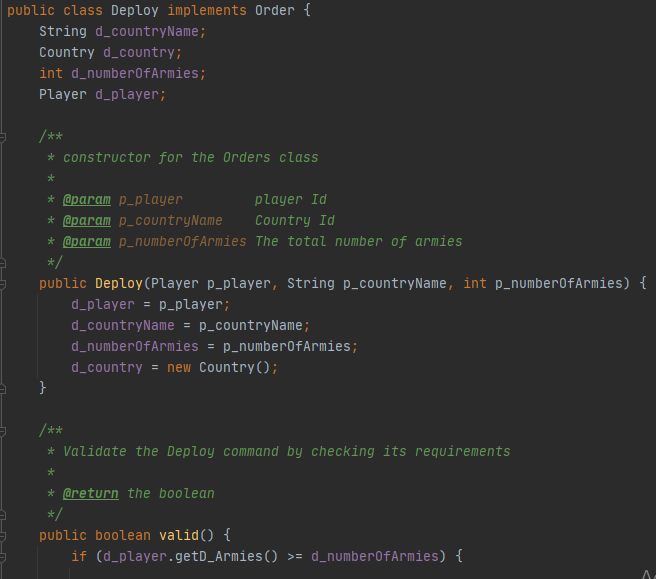
2.2 Refactoring of *Airlift* class



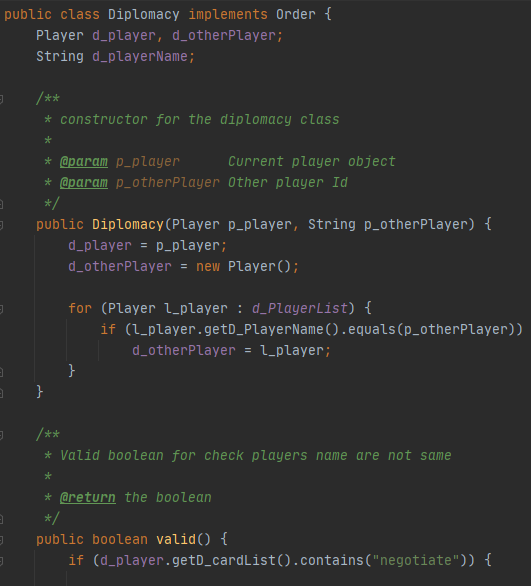
* 1. Refactoring of *Blockade* class



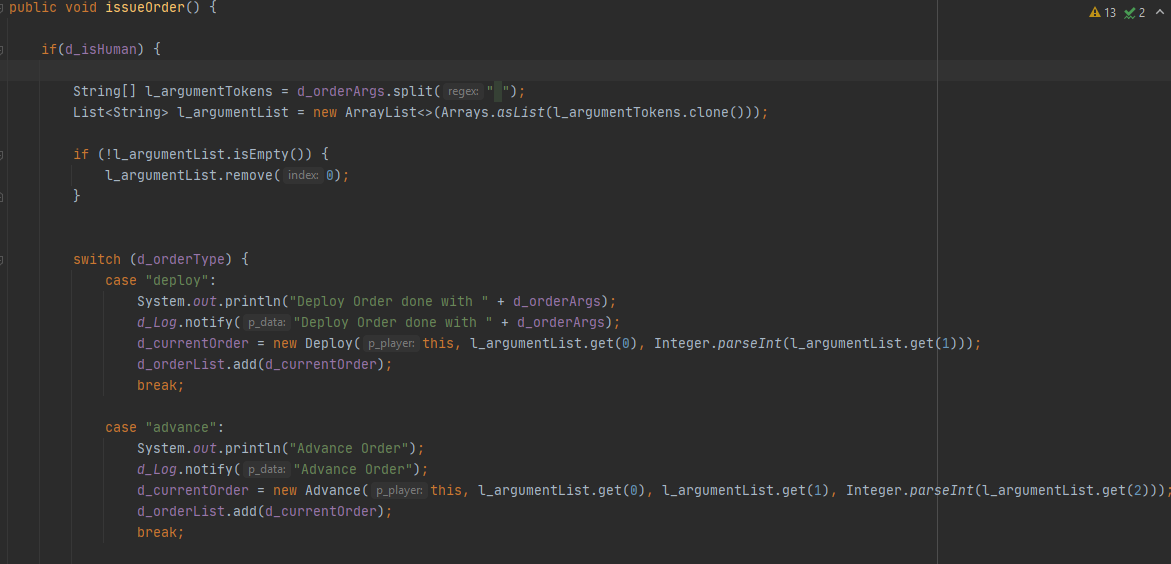
* 1. Refactoring of *Bomb* class



* 1. Refactoring of *Deploy* class



* 1. Refactoring of *Diplomacy* class

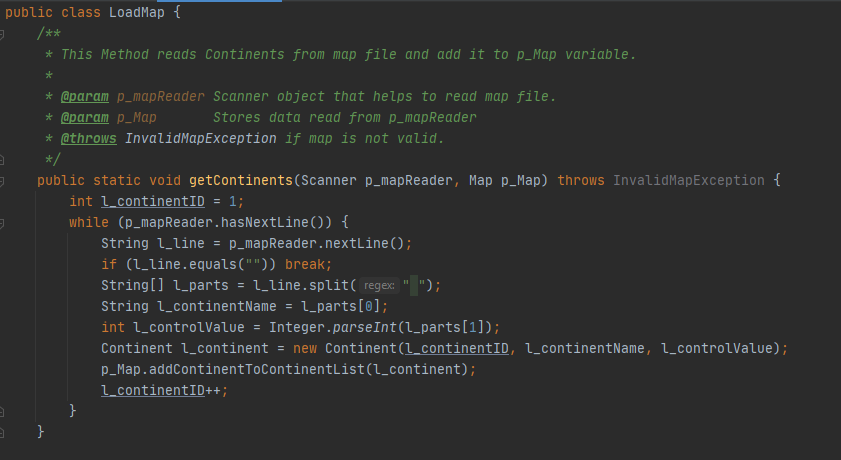


2.6 Refactoring of *issueOrder()* method from *Player* class

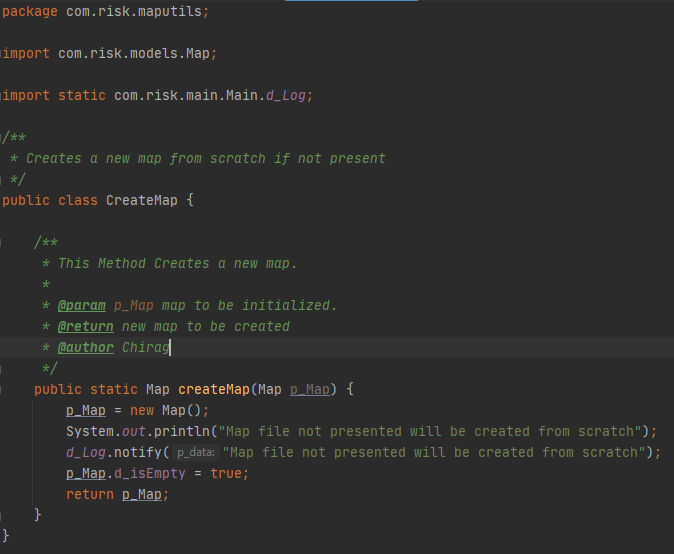
Tests: Individual tests to check validity for each command type

1. Splitting EditMap for better modularity.

Created new LoadMap class to load the existing map and new CreateMap class to create a new empty map.



* 1. Refactoring of *LoadMap* class

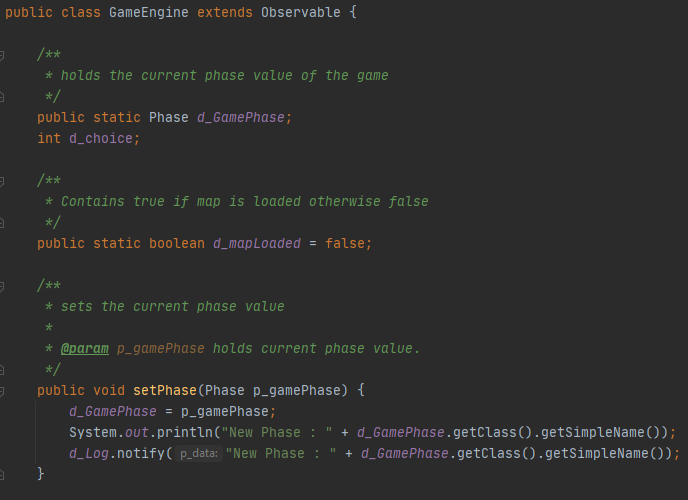


3.2 Refactoring of *CreateMap* class

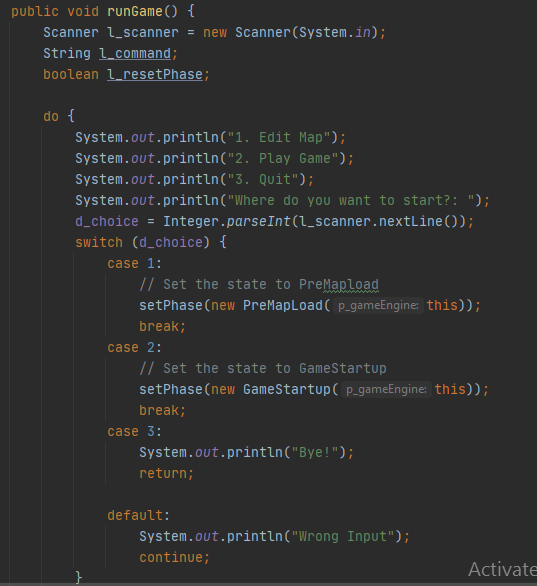
Tests : Create new map if does not exists, Load an existing map

1. Merged all game and map related command controls into a single GameEngine Class

Previously, two classes namely GameEngine and MapEngine class existed. Main method would transfer control to GameEngine for game related commands and to MapEngine for map related commands. Too many calls to and from main method were there. So, both the class were merged into single GameEngine Class to directly access all the game and mapedit commands from one class.



* 1. Refactoring of *GameEngine* class



4.2 Refactoring of *runGame* method from *GameEngine* class

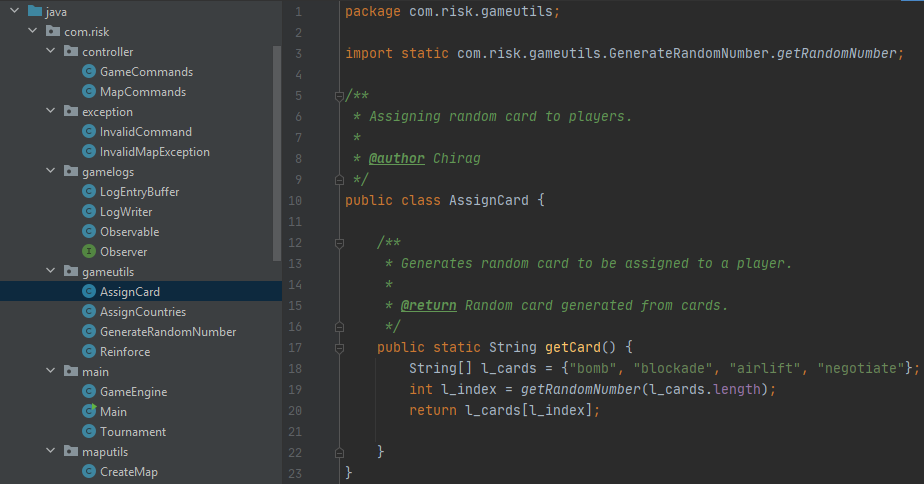


4.3 Refactoring of *compareCommand* method from *GameEngine* class

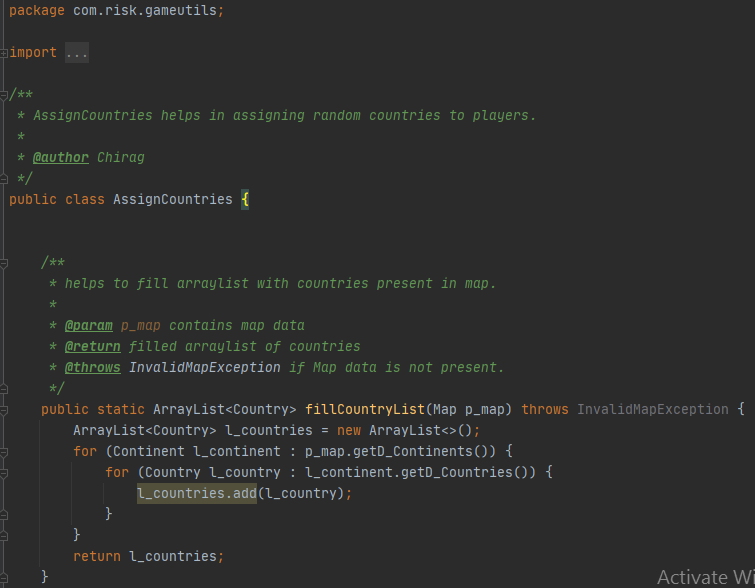
Test: Tested whether correct phases are set in the GameEngine Class or not

1. Migrated all game related methods to GameUtils package to remove clutter inside GameEngine

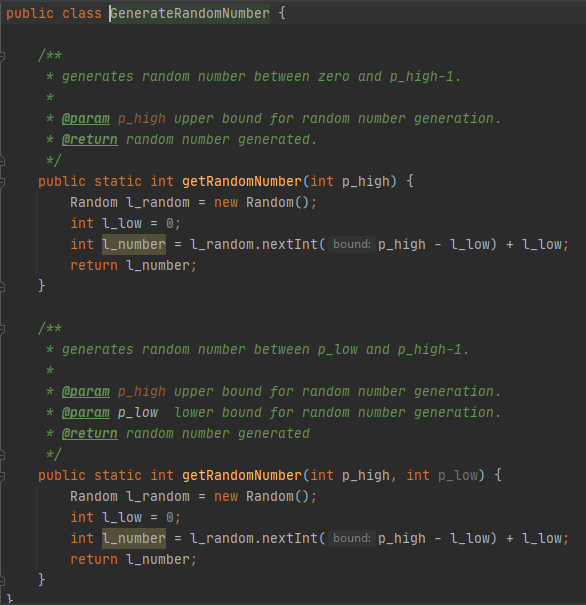
Functions including Reinforcement armies assignment, Random Number Generator for assigning countries to players randomly, assign cards to players on winning a battle all were refactored into separate class and moved under GameUtils package.



* 1. Moved methods to *gameutils* and refactored *AssignCard* class



5.2 Refactoring of *AssignCountries* class



5.3 Refactoring of *GenerateRandomNumber* class

Test : Tested Random Number Generation and Reinforcement army calculator