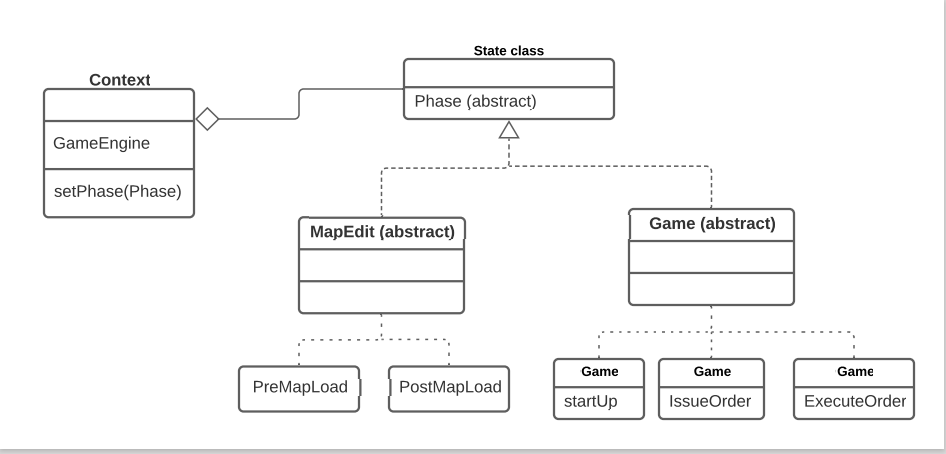
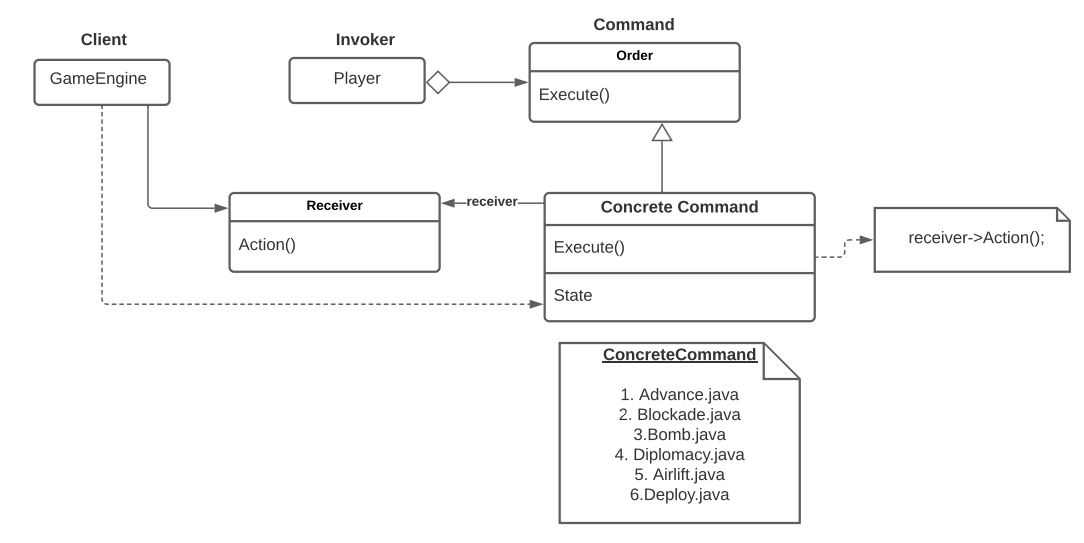
* In build 2 we have main **5 major changes** in our existing code which are as below: -

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**
11. **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

1. **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.

1. **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.

7.

8.

9.

10.

11.

12.

13.

14.

15.

1. **Refactoring: -**
2. 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.

Tests: Created tests to check the validity of execution of commands in their states. Invalid commands in a certain phase aren’t 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.

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.

Tests: Create new map if doesn’t 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

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 army’s 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.

Test: Tested Random Number Generation and Reinforcement army calculator

//Remove all the stuff below

Separated Random number generation logic from AssignCountries.

Used Random Number genaration code in AssignCard to to keep Code DRY (Don’t Repeat Yourself).

Refactored Random Number generation code to method overiding that can help to generate random number with upper bound considering lower bound to be zero or within upper bound and lower bound.

updated method signature of CreateMap to pass in map variable to keep the map model consistent and avoid creating new local copy.

Updated method signature of loadMap to pass in map varible to keep the map model consistent and avoid creating new local copy.

Updated the return type of createMap to return Map variable.

Updated the return type of loadMap to return Map variable.

Refactored Player model class to provide functionality with cards.