# SOEN 6441 - Advanced Programming Practices

Project - Warzone Risk Based game (Build #3)

Winter 2025

## Architectural Design

Submitted by **DABSV**

**Arvind Nachiappan Lakshmanan - 40310757**

**Barath Sundararaj - 40324920**

**Devasenan Murugan - 40302170**

**Swathi Priya Pasumarthy- 40322468**

Submitted to

**Prof. M. Taleb**

## Summary of build 3

---

### State Pattern

State pattern is developed and used for representing and navigating the game phase. It is used within the game model to manage the changes in the game phase. Usually comes into action when a player issues an order. It is clearly visible in the IssueOrderPhase and GameStartUpPhase allowing for a clean and organized transition into the Order state.

### Command pattern

Command pattern is used to create and execute orders passed by the players of the game. As the name suggests this pattern turns a request into a stand-alone object called command. By this we can easily pass buffers or requests which can be viewed in the encapsulation of player actions, making them easier to execute altogether at the end of a game phase, i.e after all the players have given their orders. Orders package follows the command pattern.

### Observer Pattern

This pattern is predominantly used to generate the game logs that notify and update interested observers (such as players, loggers) when a significant game event occurs, which in turn ensures a real-time record-keeping. The gameLog package visually represents the observer pattern in use.

### Adapter Pattern

The Adapter Pattern is used in this project to support loading and saving maps in two different formats: **Domination** and **Conquest**. Instead of rewriting or duplicating logic for each format, the Adapter provides a common interface (MapFileHandler) that both formats implement. This allows the game engine and handlers to interact with any map format uniformly, without knowing the underlying details. The pattern promotes **code reusability**, **extensibility**, and helps **minimize redundancy** by encapsulating format-specific logic in separate adapter classes (DominationMapFileHandler, ConquestMapFileAdapter).

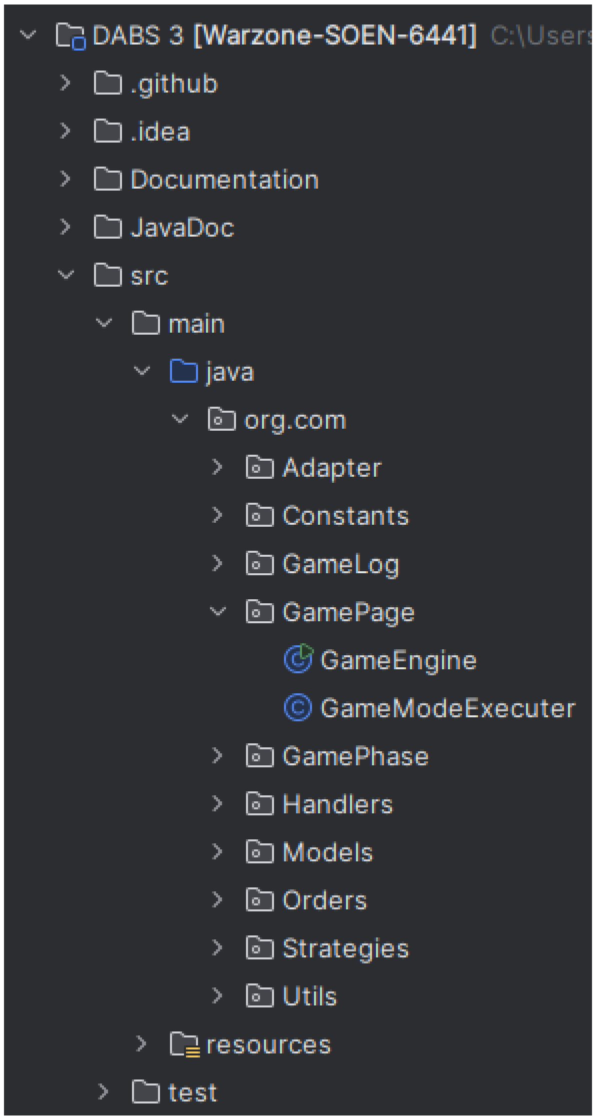
## <u>Testing</u>

The codebase is rigorously tested with JUnit5, a popular Java testing framework, to verify the correctness and reliability of individual units of code, thereby promoting software quality and robustness.

## <u>Documentation</u>

The codebase is extensively documented using JavaDoc, providing comprehensive and easily accessible documentation to assist developers and maintainers in understanding the structure, purpose, and usage of the code.

## Package Structure

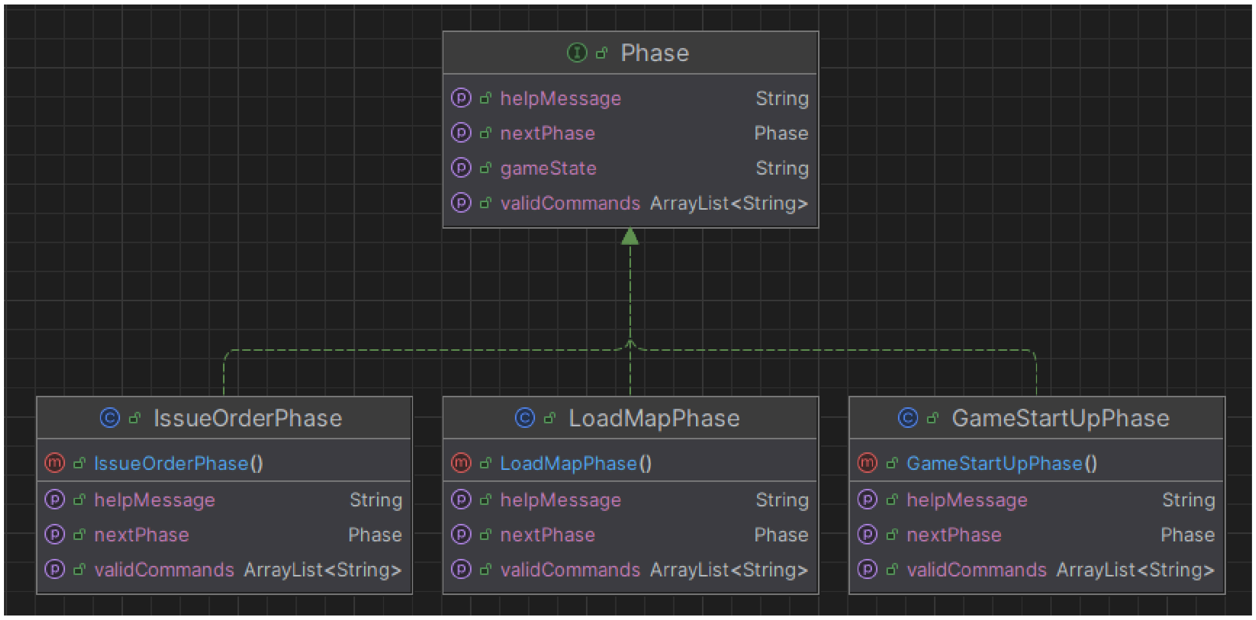


```
DABS 3 [Warzone-SOEN-6441] C:\Users
  .github
  .idea
  Documentation
  JavaDoc
  src
    main
      java
        org.com
          Adapter
          Constants
          GameLog
          GamePage
            GameEngine
            GameModeExecuter
          GamePhase
          Handlers
          Models
          Orders
          Strategies
          Utils
      resources
    test
```

## Design Description

---

### 1. State Pattern



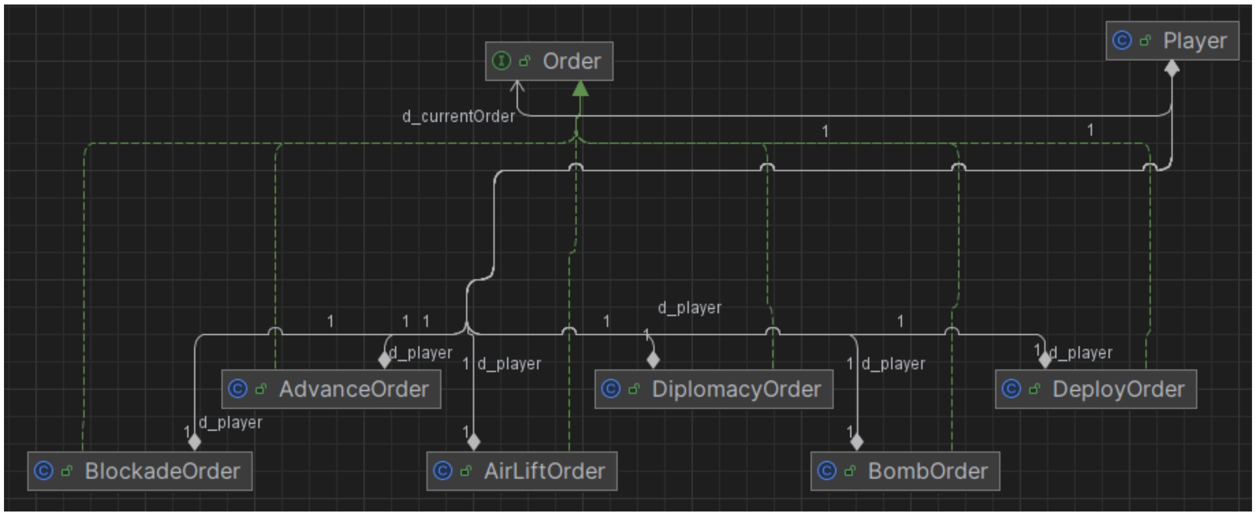
To represent each phase of the game, a new package called GamePhase was developed and introduced. The GameEngine class will serve as the context class, coordinating and transitioning between each game phase. Each phase class will encapsulate the Phase's interface class behaviour and logic.

- The central coordinator who manages the entire game phase and players is the GameStartUpPhase.
- The game begins in the above mentioned class and validates the commands passed down by players.
- Then the LoadMapPhase begins with players creating, editing and loading a valid map, assigning countries to players.
- The game moves to the IssueOrderPhase once the map is loaded. During this phase, each player takes turns to issue multiple orders such as deploying armies,

advancing from owned countries to neighbouring countries and other strategic actions.

- After execution of the orders, the game returns to the IssueOrderPhase in preparation of the next player's turn.

## 2. Command Pattern

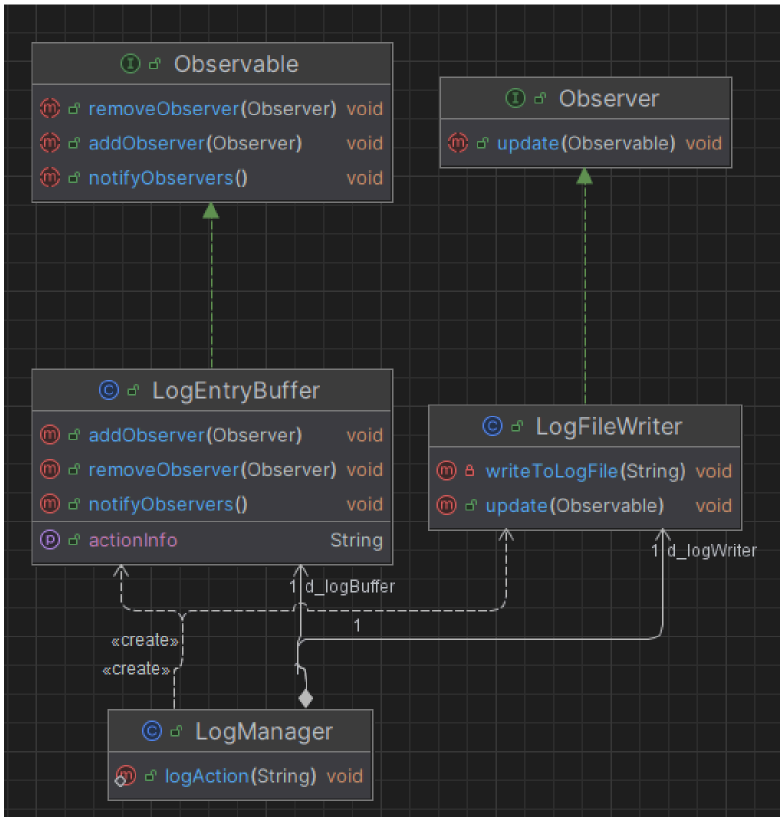


The command pattern is used in context with GameEngine to manage player orders as distinct commands. Each order is contained within an Order class (an Interface), and the Player class serves as the invoker, queuing orders as they are issued by the player. The client class is the GameEngine, which retrieves and executes orders. When the Order objects are executed, they carry out the corresponding actions.

- The DeployOrder class is used to define the operations related to deployment of armies owned by a particular player in their own turn.
- The AdvanceOrder class is used to define the operations related to advancing armies from a player's own country to a neighbouring country (either a neutral country or an enemy country or their own country).

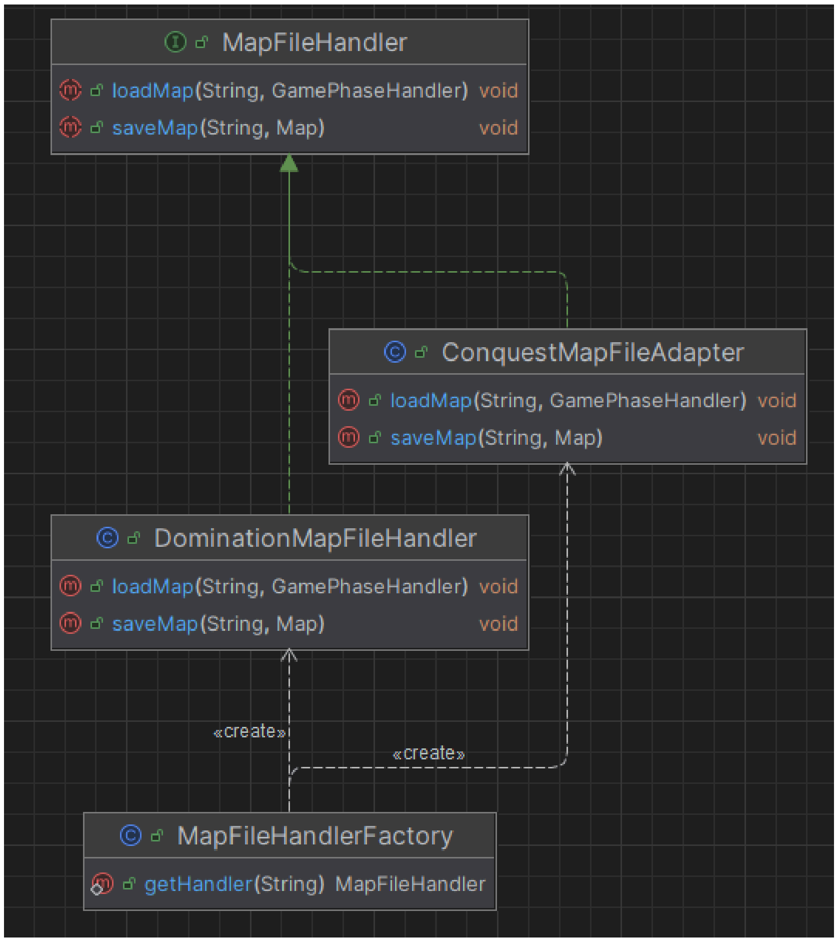
- The AirLiftOrder class is used to allow a player to transfer armies long distances. The source and target country need not be neighbouring countries. But the player should own at least one AirLift card.
- The DiplomacyOrder class when invoked enforces peace between two players. While peace is enforced neither player will be able to attack the other. Also requires one of the players to have a Diplomacy card.
- The BombOrder class allows a player to target an enemy or neutral territory and kill half of the armies on that territory. This also requires at least one Bomb card.
- The BlockadeOrder class will triple the army count on one of the current player's territories and make it a neutral country. This also requires a Blockade card.

## 3. **Observer Pattern**



The Observer pattern is used in the context of implementing a game log file to establish a one-to-many relationship between objects, allowing the state of an object (the LogEntryBuffer) to be observed and monitored by multiple other objects (the log file writer). The LogEntryBuffer class serves as the subject (Observable) and stores information about game actions. When this information changes, it alerts its observers (the log file writer) so that the changes can be captured and recorded in a log file.

## 4. <u>Adapter Pattern</u>


MapFileHandler
loadMap(String, GamePhaseHandler) void
saveMap(String, Map) void
ConquestMapFileAdapter
loadMap(String, GamePhaseHandler) void
saveMap(String, Map) void
DominationMapFileHandler
loadMap(String, GamePhaseHandler) void
saveMap(String, Map) void
«create»
«create»
MapFileHandlerFactory
getHandler(String) MapFileHandler

The diagram illustrates the use of the Adapter Design Pattern to support loading and saving maps in both Domination and Conquest formats.

- The MapFileHandler interface defines a common contract with loadMap and saveMap methods.
- Two concrete classes — DominationMapFileHandler and ConquestMapFileAdapter — implement this interface, each handling the specific logic for their respective map formats.
- The MapFileHandlerFactory acts as a factory to return the appropriate handler implementation based on the file extension (e.g., .map or .conquest).
- This design allows the rest of the application (like CommandHandler and MapOperationsHandler) to work with maps in a format-agnostic way, promoting extensibility, decoupling, and code reusability.