Team 17: SOEN 6441 WW Advanced Programming Practices
Architecture Document

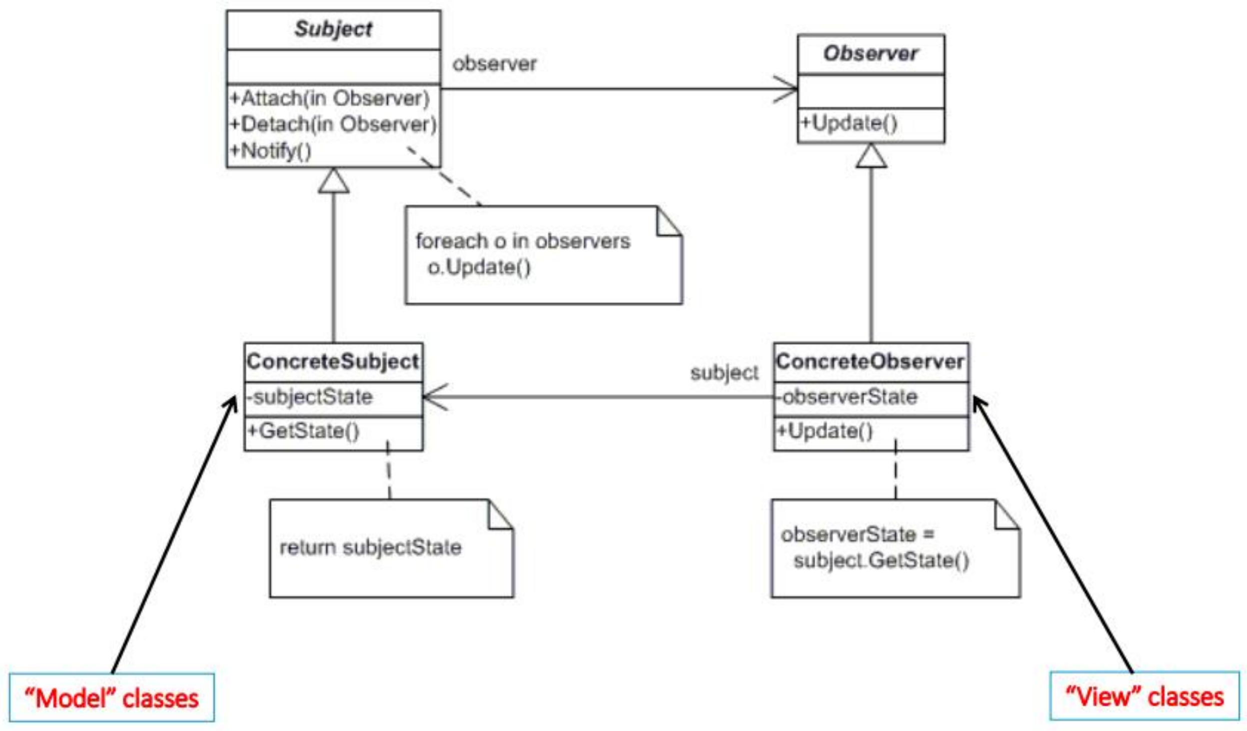
Architectural Design:

Our RISK game is designed based on MVC and Observer Design Pattern. We have divided and distributed our modules in the parts **MODEL, VIEW, CONTROLLER and OBSERVER**.

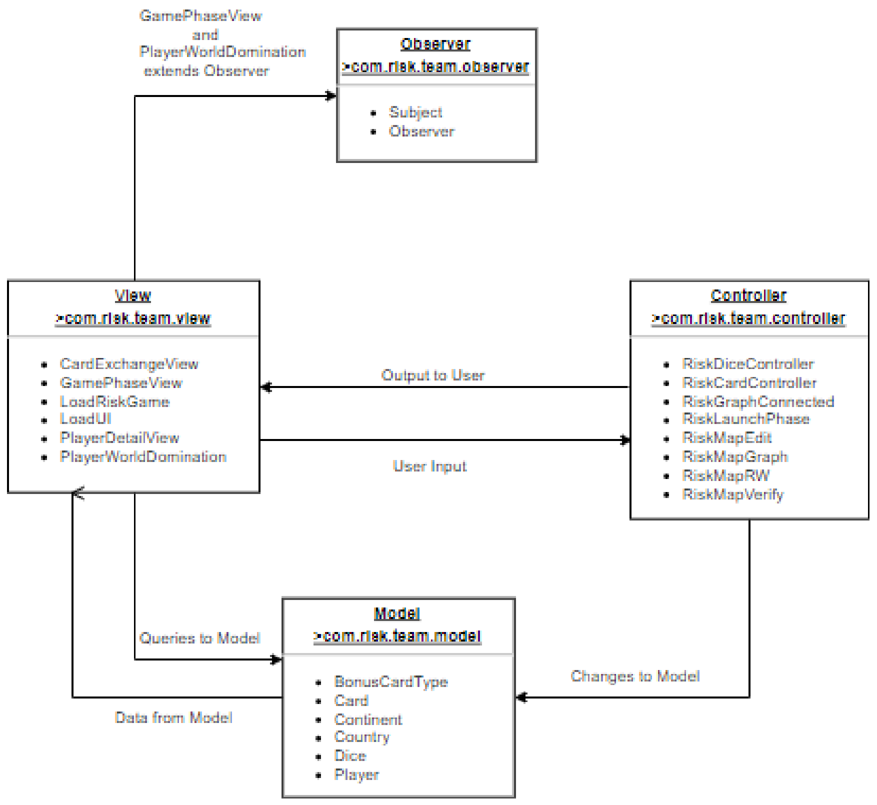
Observer Pattern:

Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.

This pattern is a cornerstone of the Model-View-Controller architectural design, where the Model implements the business logic of the program, and the Views are implemented as Observers that are as much uncoupled as possible from the Model components.



Architecture Diagram:



Modules in Model:

Model files in the project are described as below:

1. Player: Player class is used to model the actual player entities of the game as well as provides the implementation of attack, reinforcement and fortification phase.
2. Country: Country class models country entity, where each player owns few countries in the beginning and their aim is to get ownership of all the countries.

3. Continent: Continent class models Continent entity, which can be considered as a superset of countries, where each country belongs to one continent.
4. BounsCardType (Interface): It is used to model cards which are allocated after the attack phase if any player is eligible for the card.
5. Dice: Dice class is used to initialize values for attacking and defending country as well as stores the result at the end of dice roll for both attacking and defending country.
6. Card: Card class deals with operations related to trade in of cards in the game.

Modules in Controller:

1. RiskMapEditor: This class provides functionality to edit or create a map from scratch.
2. RiskMapGraph: RiskMapGraph creates a connected graph from valid map data and provides methods to modify the graph.
3. RiskGraphConnected: Checks whether a graph is connected or not. In case of fortification , it checks whether there is adjacent path through connection or not.
4. RiskMapRW: This class reads the data from a **.map** file and provides it to RiskMapEditor's object. It also gets data from RiskMapEditor's object and writes it to an empty **.map** file.
5. RiskMapVerify: It is responsible for verifying the correctness of a map which is to be loaded for game play.
6. RiskLaunchPhase: It takes data from RiskMapRW, and initializes data for Players, countries and armies. Here countries are randomly assigned to each player.
7. RiskDiceController: This class performs all the actions taking place during dice roll which in turn compares the dice values of attacker and defender and decides the winning country for each dice roll.
8. RiskCardController: It takes inputs from user for selecting the kind of card and checks the possibilities if card trade is possible for the selected cards.

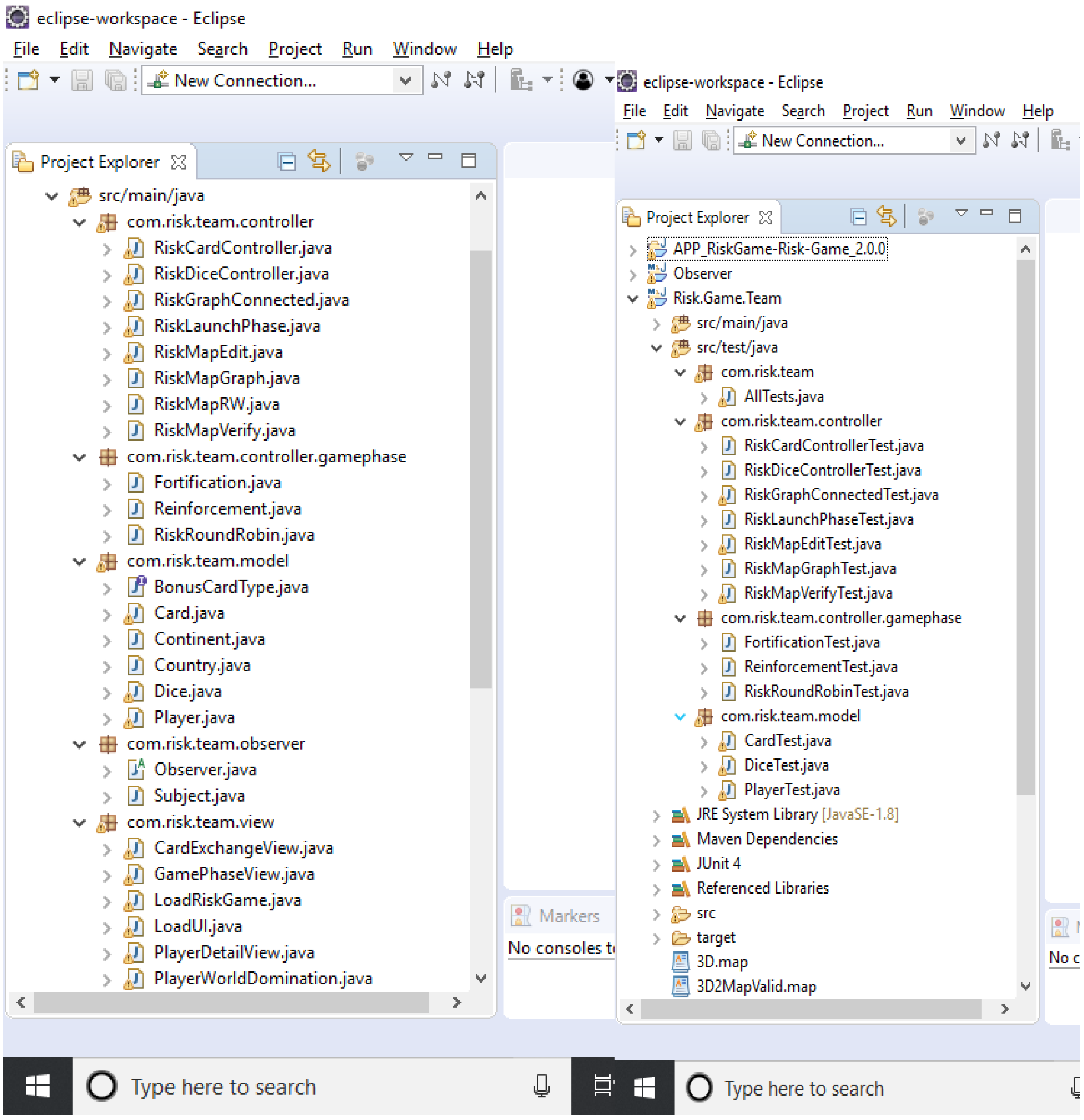
Modules in View:

1. LoadUI: Provides an interface for the user to interact with the game.
2. LoadRiskGame: This class loads LoadUI file to start game window.
3. PlayerDetailView: The initial details of the players playing the game are contained in this class file.
4. CardExchangeView: It displays the army count update before and after the card exchange phase in the game.
5. GamePhaseView: The user interactions for attack, reinforcement and fortification phase takes place with the help of this file. Moreover, this view file inherits the **Observer** class to record all the major updates during all the phases of the game.
6. PlayerWorldDomination: This file provides overall analysis of the game using Observer pattern. It is a subclass of **Observer** Class. It displays the following information:
   1). The number of armies owned by each player in the game.
   2). The continents owned by each player in the game.
   3). The amount of map occupied by each player in terms of percentage.

Modules in Observer:

Team 17: SOEN 6441 WW Advanced Programming Practices
Architecture Document

1. Subject: Subject class notifies all the observer files attached with it about the various updates in the army count, player actions and countries owned for attack phase, reinforcement phase, fortification phase and world domination analysis
2. Observer: This is an abstract class which acts as an observer and listens to changes updated by the Subject class. GamePhaseView and PlayerWorldDomination are the subclass which provides actual logic on account of updates recorded for all the phases of the game.


eclipse-workspace - Eclipse
File Edit Navigate Search Project Run Window Help
New Connection...
Project Explorer
Markers
No consoles t
Type here to search
eclipse-workspace - Eclipse
File Edit Navigate Search Project Run Window Help
New Connection...
Project Explorer
No c
Type here to search

```
src/main/java
  com.risk.team.controller
    RiskCardController.java
    RiskDiceController.java
    RiskGraphConnected.java
    RiskLaunchPhase.java
    RiskMapEdit.java
    RiskMapGraph.java
    RiskMapRW.java
    RiskMapVerify.java
  com.risk.team.controller.gamephase
    Fortification.java
    Reinforcement.java
    RiskRoundRobin.java
  com.risk.team.model
    BonusCardType.java
    Card.java
    Continent.java
    Country.java
    Dice.java
    Player.java
  com.risk.team.observer
    Observer.java
    Subject.java
  com.risk.team.view
    CardExchangeView.java
    GamePhaseView.java
    LoadRiskGame.java
    LoadUI.java
    PlayerDetailView.java
    PlayerWorldDomination.java
```

```
APP_RiskGame-Risk-Game_2.0.0
Observer
Risk.Game.Team
  src/main/java
  src/test/java
    com.risk.team
      AllTests.java
    com.risk.team.controller
      RiskCardControllerTest.java
      RiskDiceControllerTest.java
      RiskGraphConnectedTest.java
      RiskLaunchPhaseTest.java
      RiskMapEditTest.java
      RiskMapGraphTest.java
      RiskMapVerifyTest.java
    com.risk.team.controller.gamephase
      FortificationTest.java
      ReinforcementTest.java
      RiskRoundRobinTest.java
    com.risk.team.model
      CardTest.java
      DiceTest.java
      PlayerTest.java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  JUnit 4
  Referenced Libraries
  src
  target
  3D.map
  3D2MapValid.map
```

Same hierarchy is followed for **Test Folder** and **Test Classes**. There is one to one Mapping of Each Java Tested Class to Test Class.