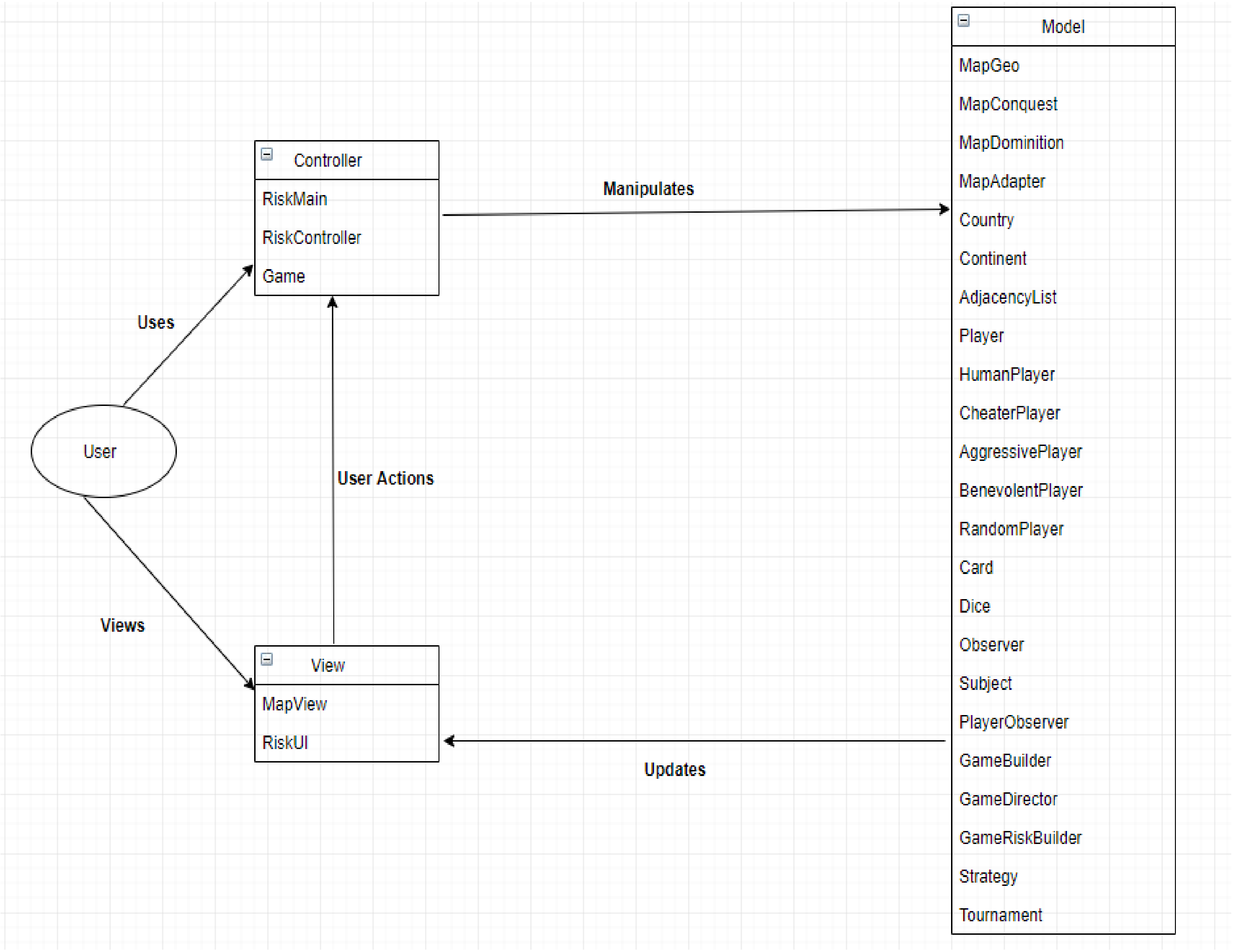
Team 17

Build Architecture 3



Model Layer

- Model layer includes classes that define the game entities, which are embedding their own state and actions.
- Model layer consists of following classes

model.MapGeo
model.MapConquest
model.MapDominition
model.MapAdapter
model.Continent
model.Country
model.AdjacencyList
model.Player
model.HumanPlayer
model.AggressivePlayer
model.BenevolentPlayer
model.CheaterPlayer
model.RandomPlayer
model.Card
model.Dice
model.Observer
model.Subject
model.PlayerObserver
model.GameBuilder
model.GameDirector
model.GameRiskBuilder
model.Strategy
model.Tournamentt

View Layer

- This layer includes all classes related to visualization of the game or interaction with the user.
- View layer consists of following classes

  view.MapView
  view.RiskUI

Controller Layer

- This layer receives user input and initiate a response by making calls on appropriate model objects.

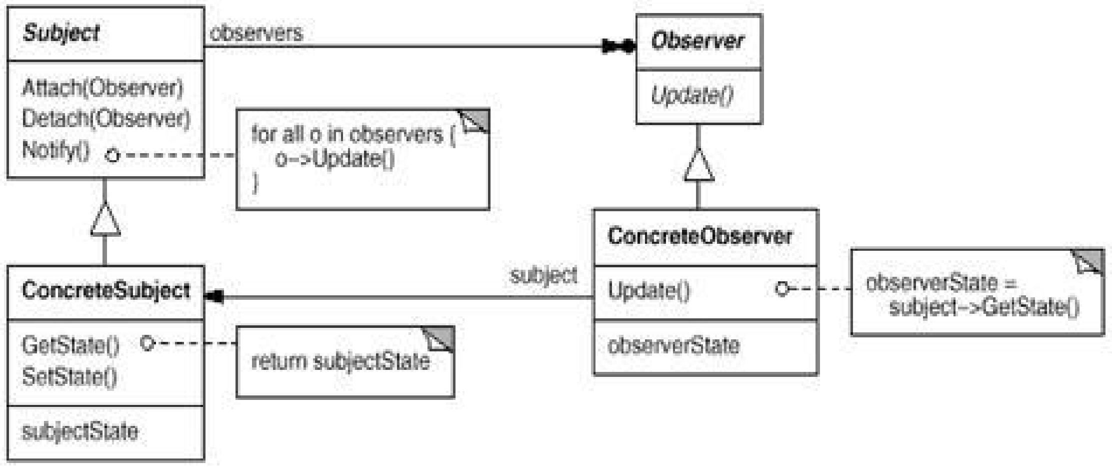
- Controller layer consists of following classes –

  controller.RiskController
  controller.RiskMain
  controller.Game

Design Patterns

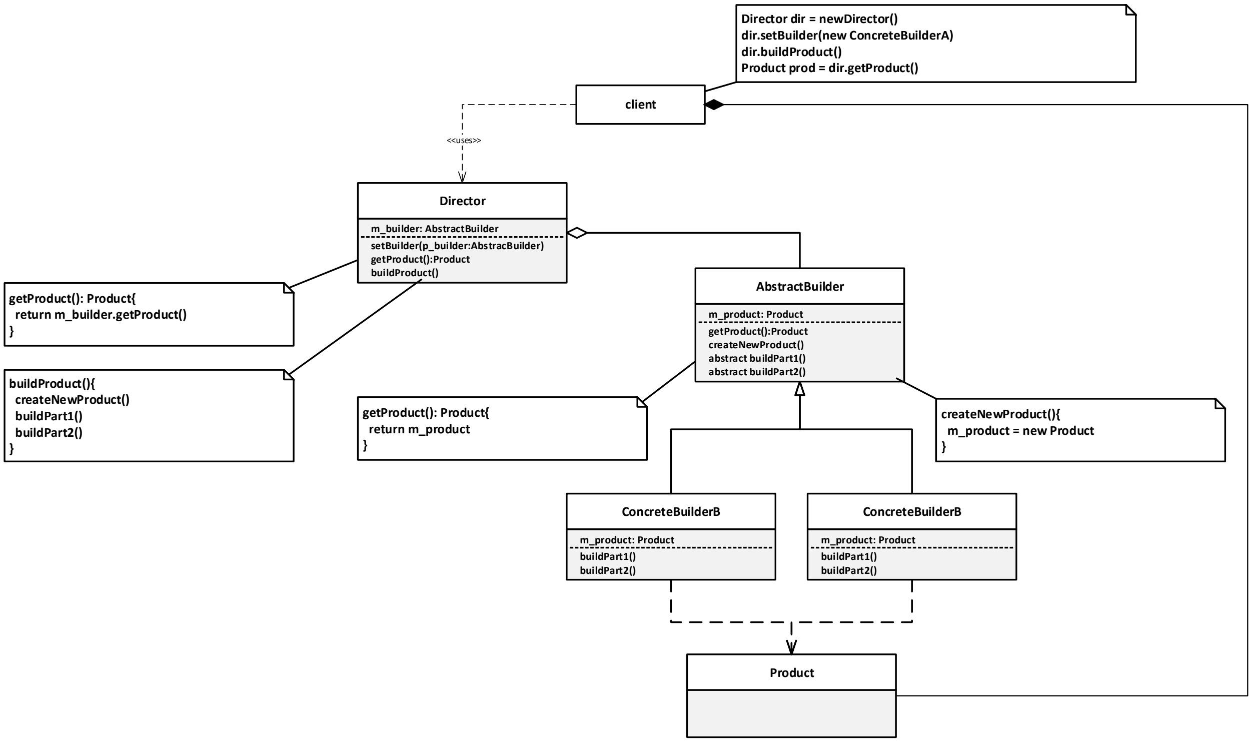
Observer

- 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.
- Example: adding or losing territories, exchanging cards, etc.



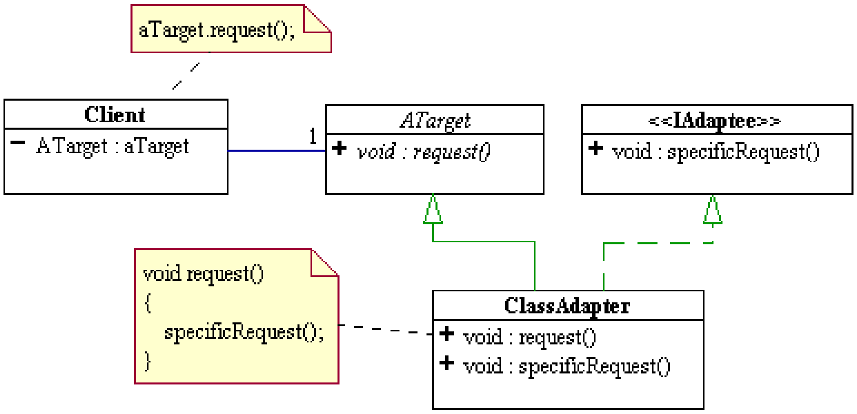
Builder

- Builder pattern separates the construction of a complex object from its representation so that the same construction process can create different representations.
- Example: saving an ongoing game, loading a saved game etc.



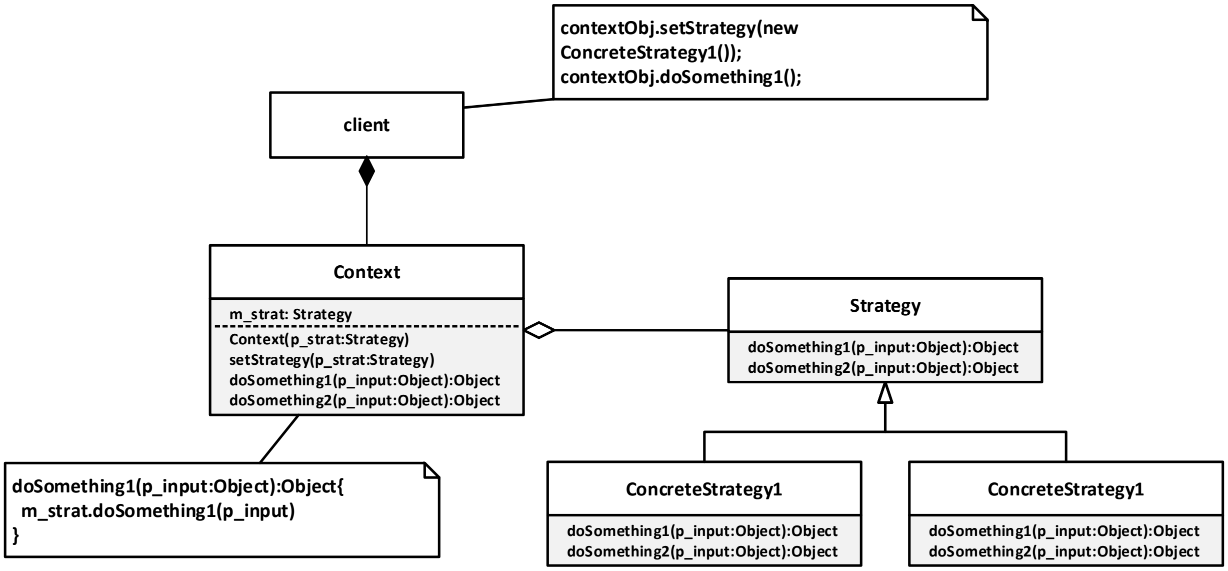
Adapter

- Adapter pattern makes classes work together which might not be able to use components from each other due to incompatible interfaces.
- Example: Reading a conquest map file, writing a new map file etc.



Strategy

- Strategy pattern allows some specific algorithm implemented by a method vary without affecting the classes using it.
- Example: Different implementation of different game phases based on type of player


contextObj.setStrategy(new ConcreteStrategy1());
contextObj.doSomething1();
client
Context
m_strat: Strategy
Context(p_strat:Strategy)
setStrategy(p_strat:Strategy)
doSomething1(p_input:Object):Object
doSomething2(p_input:Object):Object
Strategy
doSomething1(p_input:Object):Object
doSomething2(p_input:Object):Object
doSomething1(p_input:Object):Object{
m_strat.doSomething1(p_input)
}
ConcreteStrategy1
doSomething1(p_input:Object):Object
doSomething2(p_input:Object):Object
ConcreteStrategy1
doSomething1(p_input:Object):Object
doSomething2(p_input:Object):Object