# Sprint 0 - Design Patterns

Thomas Kolman Razi Messinger
Justin Zhang Nasim Bondar Sahebi

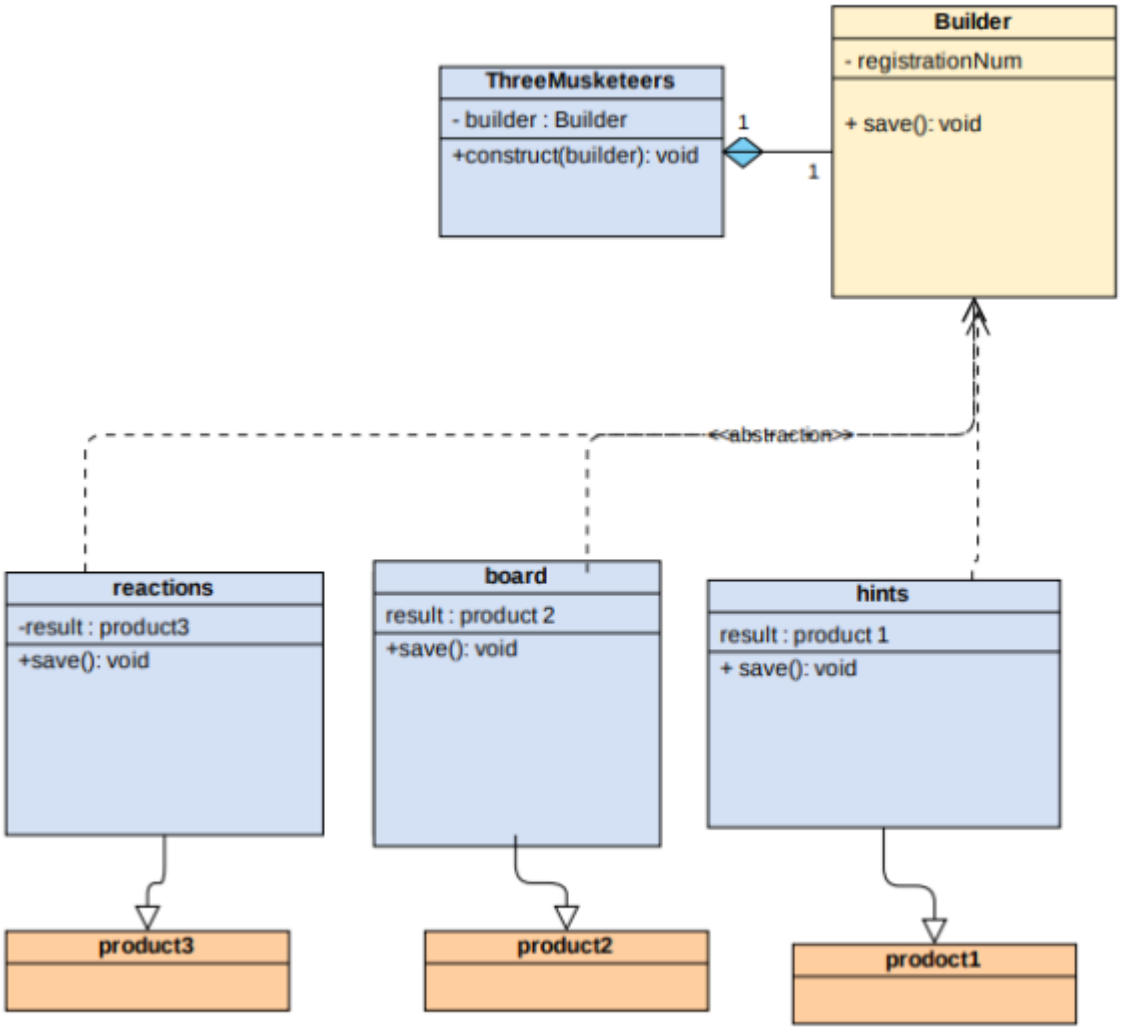
November 29, 2021

# Contents

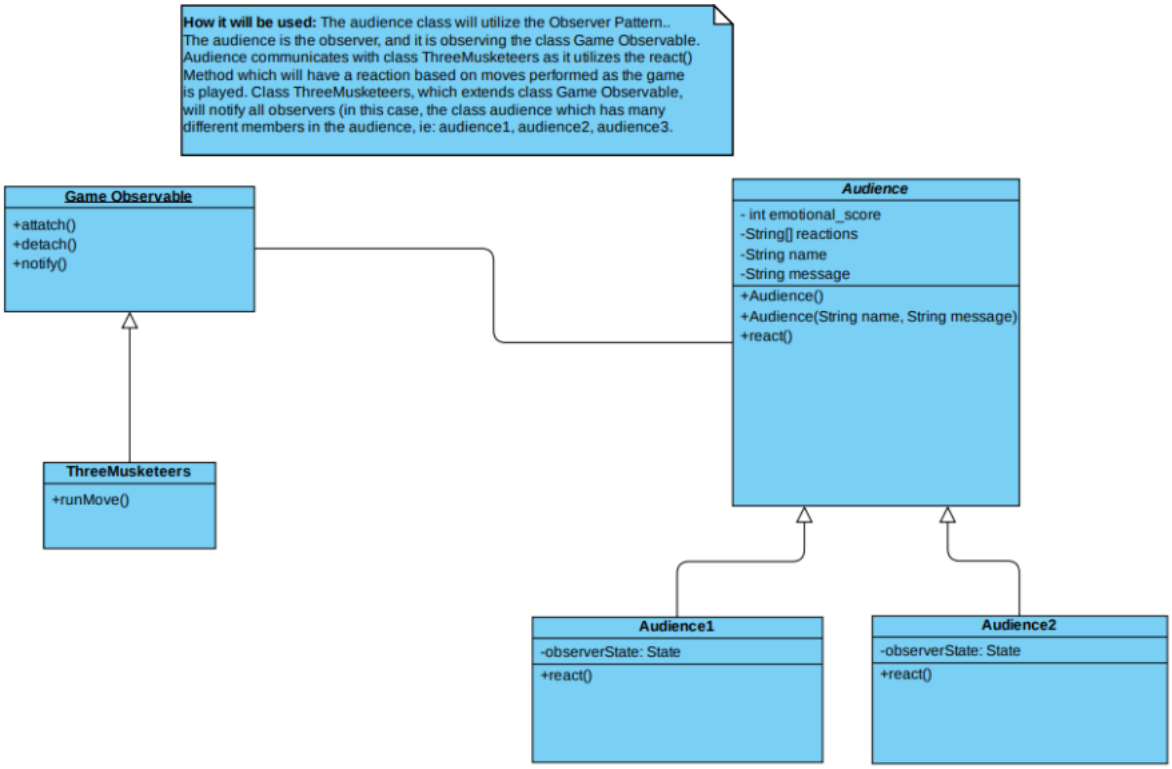


## 0.1 Builder Pattern

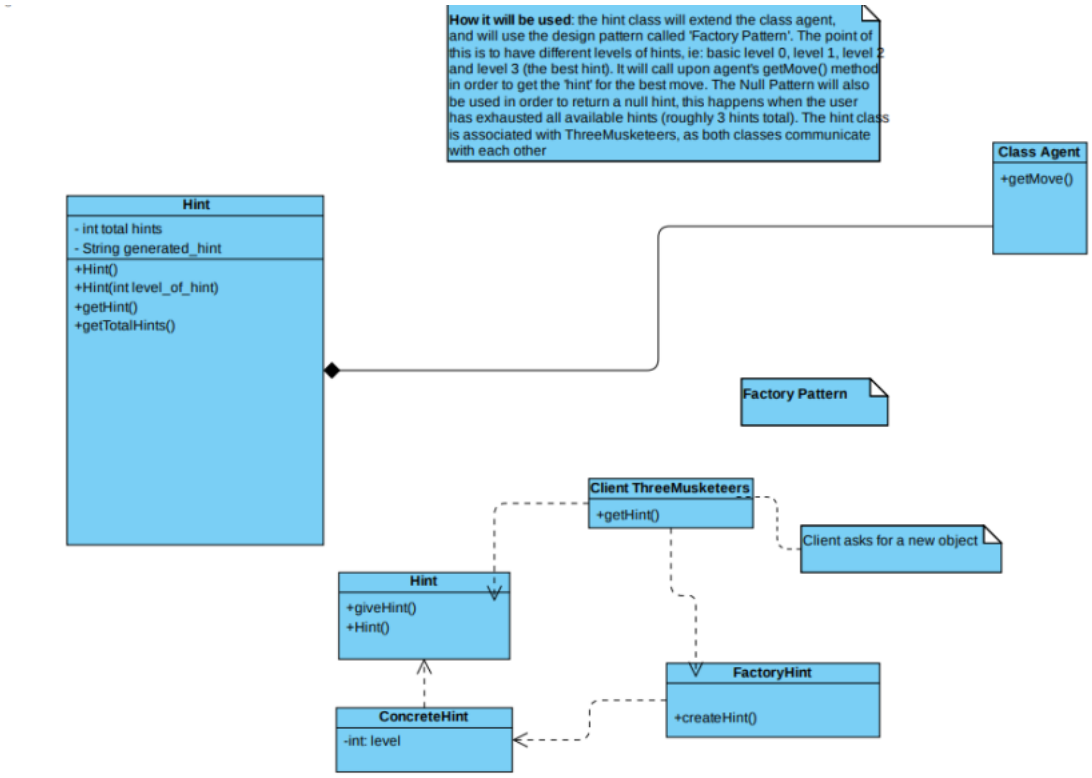
Builder pattern defines an instance for the object but lets subclasses decide which class to start. It also refers to the newly created obj through a common interface. To save the game, we use the builder interface and three classes of hints and reaction and board, which implements by the builder. each three-class shave save() method inside


Builder
- registrationNum
+ save(): void
ThreeMusketeers
- builder : Builder
+construct(builder): void
1
1
<<abstraction>>
reactions
-result : product3
+save(): void
board
result : product 2
+save(): void
hints
result : product 1
+ save(): void
product3
product2
prodoct1

## 0.2 Observer Pattern

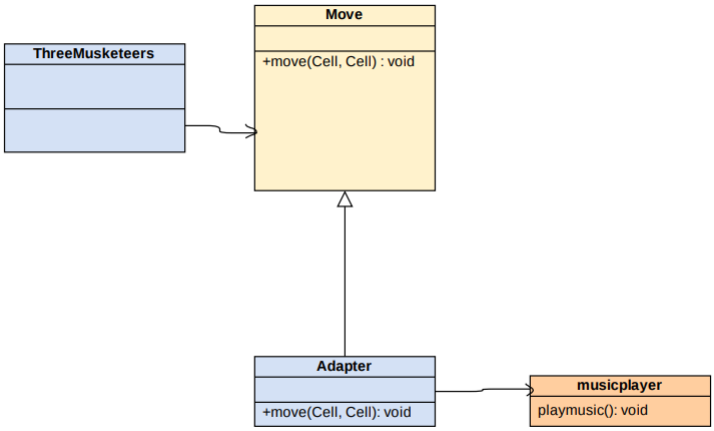


## 0.3 Factory Pattern



## 0.4 Adapter Pattern

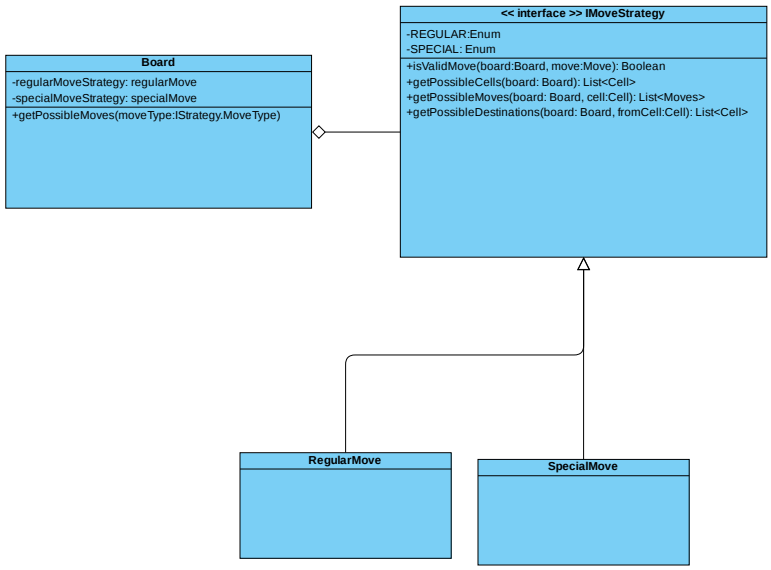
Adapter patterns: using it to adaptor to add the audio. In the adaptor pattern, An Adapter pattern acts as a connector between two incompatible interfaces that otherwise cannot be connected directly, in this case connecting music player to three musketeer classes.



## 0.5 Command Pattern

**NOTE: This Pattern has been switched from the command pattern to the strategy pattern.** The strategy pattern is used to implement the different types of moves that can be done. In our rough implementation, we have defined an interface that contains the strategy (the move) methods called IMoveStrategy, and then we have defined two concrete move implementations, called RegularMove and SpecialMove, respectively, that handle both regular movement and special moves.





## 0.6 Singleton Pattern

**How it's used:** These classes use the singleton design pattern. The singletons are the RandomAgent, HumanAgent and GreedyAgent. Agent is an abstract class, and Greedy and Random and Human Agent implement Agent.
One instance of each object (everything is static), there will be only one object of each agent. Run game essentially takes all the agent's and utilizes them. The point of this is to be able to switch agent's mid game (just like how you could state which agent at the start of the game, the user is also able to change agents mid-way through.