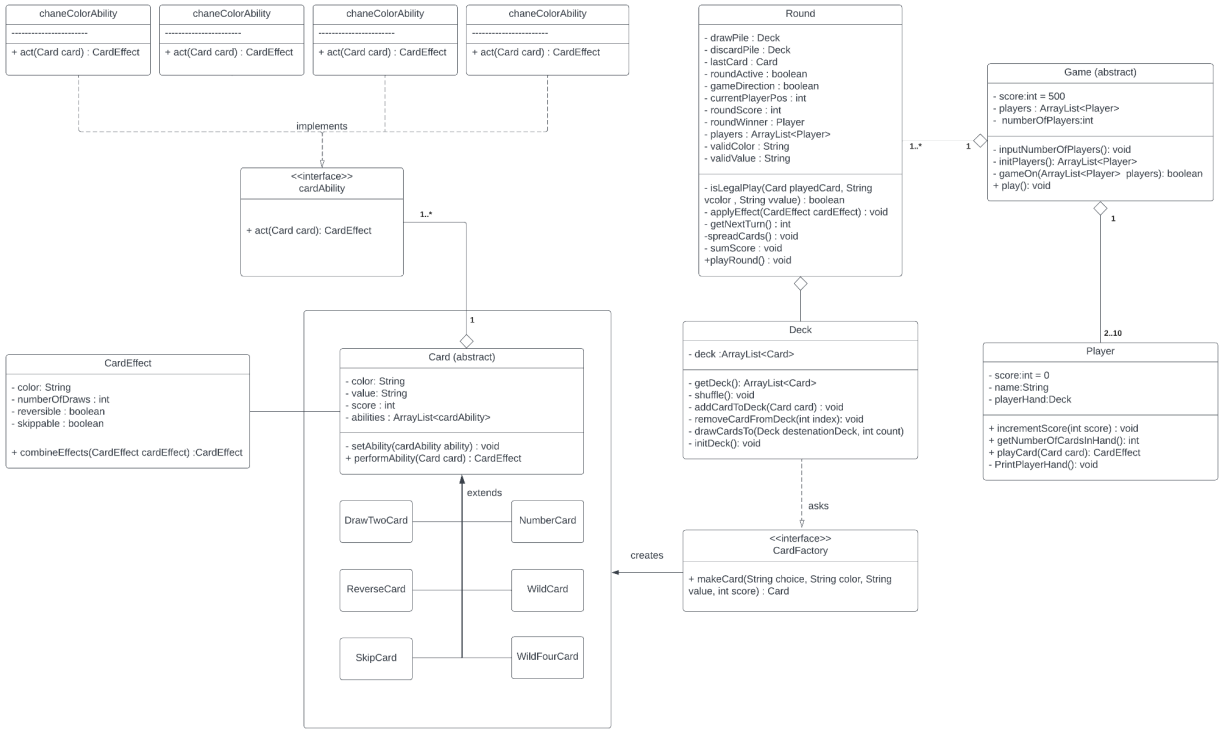
Uno Game Engine Assignment

By: Ibrahem Jaradat

In this assignment we were required to build an Uno game engine using Java and OOP to build an Uno game engine to be used by other developers in which they can build their own variation of an Uno game.

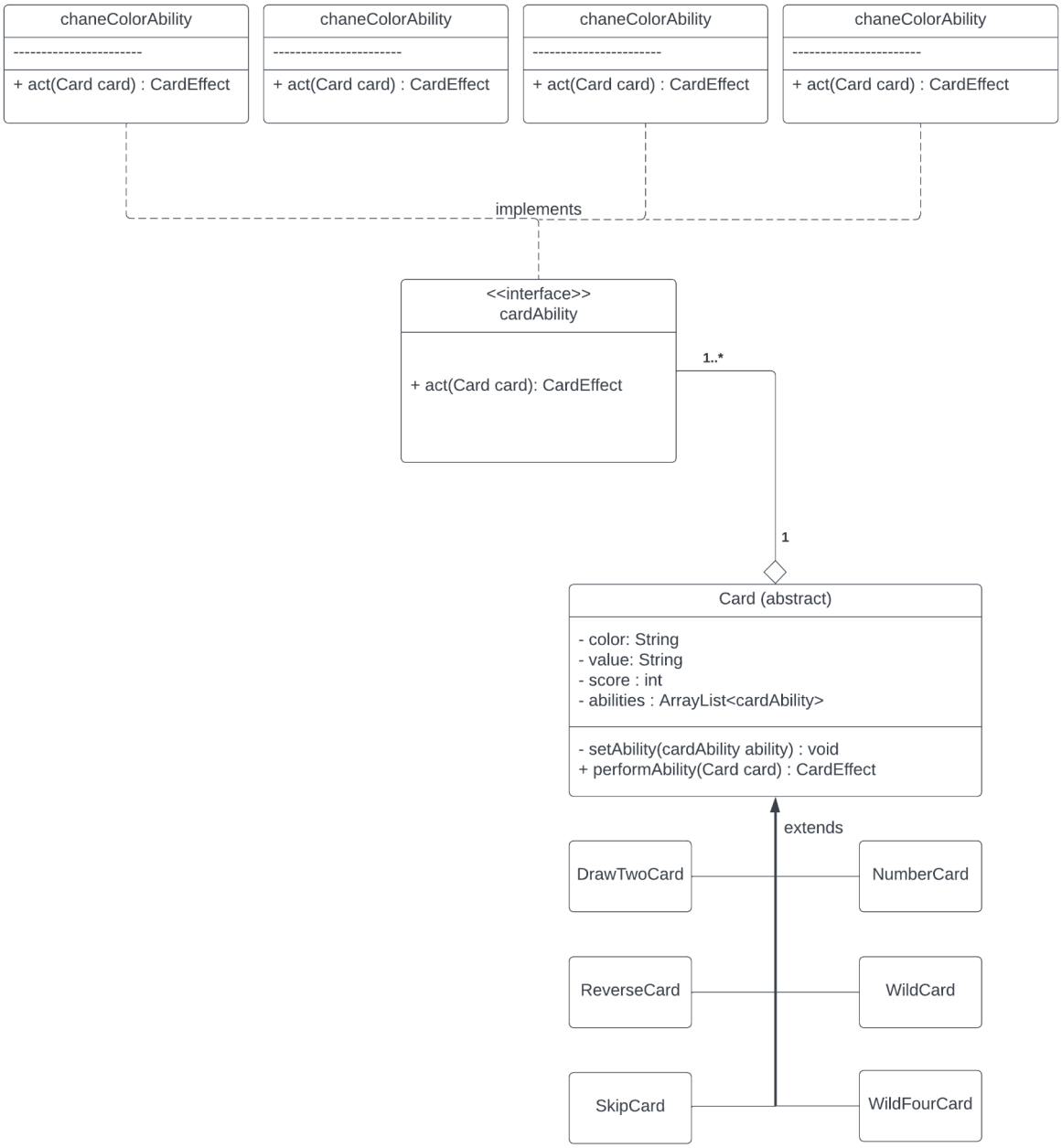
My Approach:



The Cards:

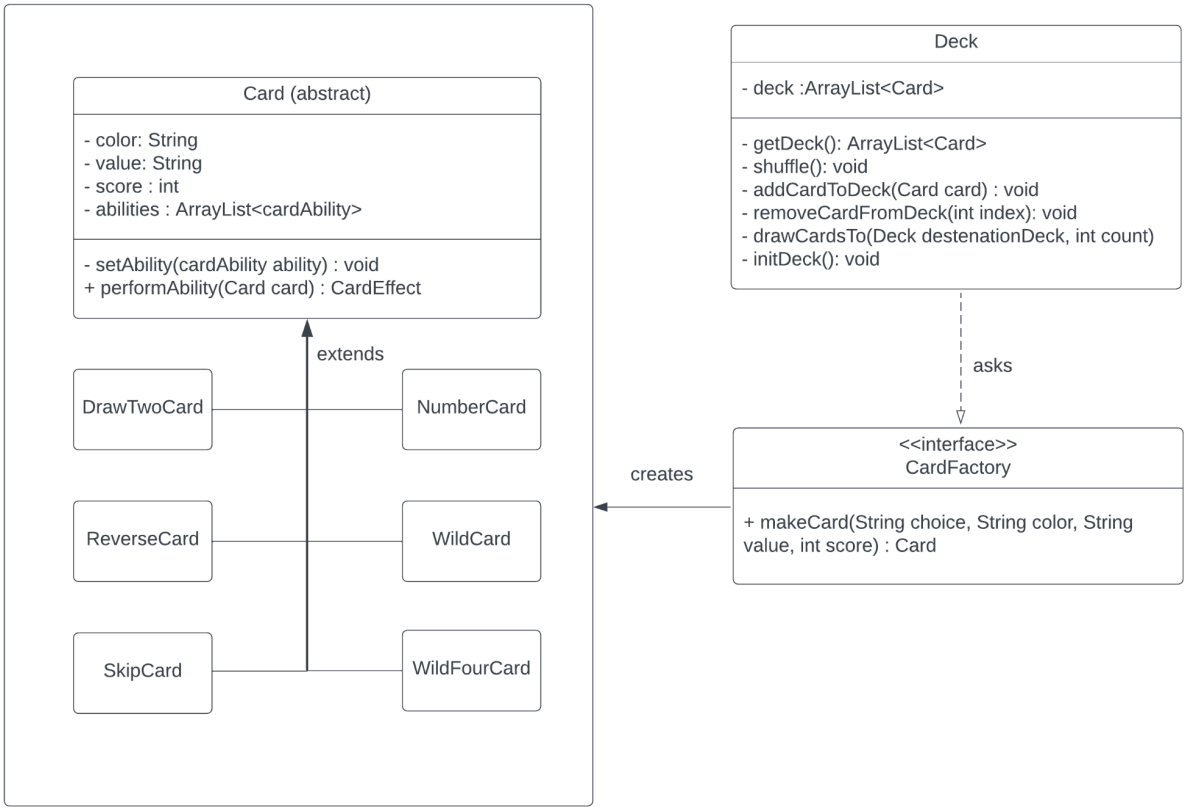
The first and the most important component of the game is the Cards, where I tend to represent the Cards with an abstract class called “Card”.
Each Card in the game has its own abilities that affects either the whole game or the subsequent player, taking in consideration that the Cards have similar specs in common, so for the purpose of isolating the program logic and extracting the behavior of the ability rules I used the Strategy Design Pattern.

Cards Strategy:



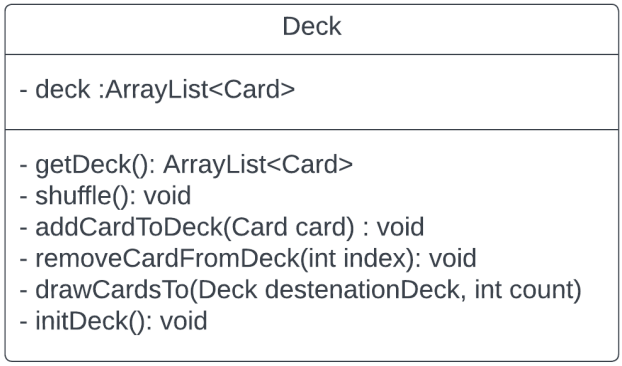
Having high Encapsulation where the logic and details are hidden from the client and given low profile methods to use, leading to minimize the accessibility like what effective java recommends.

Cards Factory:



As mentioned previously, each Card in the game has its own specs so in order to ease the process of creating and adding new cards, I used the Creational Design Pattern (Factory).

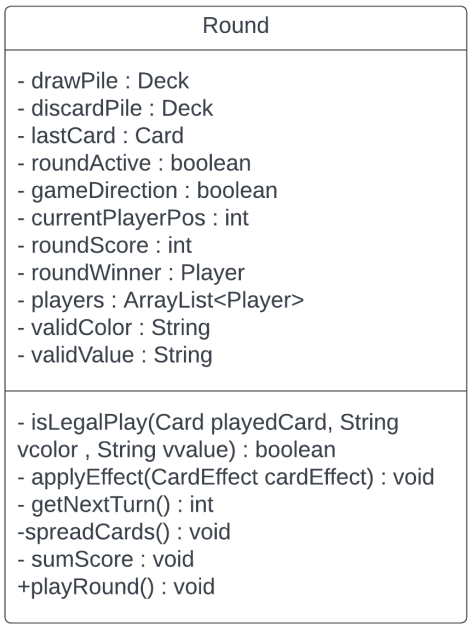
The Deck:



The Deck is basically a collection of cards, where we can shuffle cards, take cards from and/or add cards to, what that exactly means is that the draw pile, the discard pile and the player hand are all considered as decks.

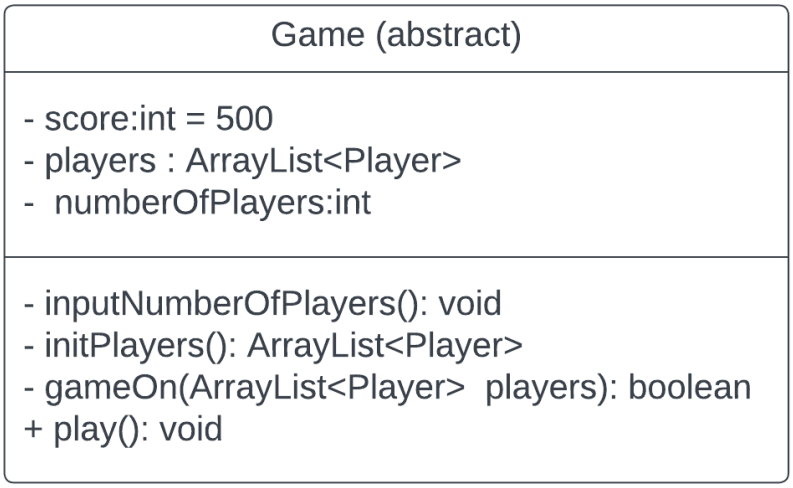
The design ensures that the system has a high Abstraction by breaking the design down into simplified classes with easy and straightforward usage, using Inheritance and Polymorphism to create and consume various types of cards.

The Round:



At the start of each round we initiate the deck and spread the cards , and at the end of the round we sum the score and add it to the winner's score.
The round is where the game is really played, and invoked only by using the (playRound) method.

The Game:



It's an abstract class where In order to create a new game variation, the developer will only need to create a new class that extends the Game class, in order to create an Uno Game.

Conclusion:

The design follows the OOP pillars in every part, with highly abstracted and encapsulated classes and methods, and taking huge advantage of the generalization and polymorphism while dealing with the different types of Cards and their different abilities.

Following appropriate design patterns making the code modular and easy to be extended in the future.

Taking in consideration that, clean code was an important aspect to follow where I tried my best to clean the code following uncle Bob's instructions by making my code DRY, solving the data clumps and long parameters smells that may appeared when using the strategy design pattern and making sure to make every class and method

highly cohesive and loosely coupled while reducing the number of method's arguments as possible.

The effective java items also was a major aspect when it comes to my design decisions, like the item of Minimizing the accessibility of classes and members and using accessor methods, and of course, minimizing the mutability of most of the classes as much as possible.

SOLID principles was the most important to follow, making every class have one responsibility and concern, of course by using good patterns and simple solutions in general leads to having an open system for extension by the ability to add new Cards and new card abilities very easily, in addition following Liskov's principle to avoid misusing the inheritance, and obviously segregating the interfaces, making the solution dependent on them with high-level classes and interfaces to follow of the Dependency Inversion Principle.