Object Oriented Programming

Group Project(The github link to the project)

Build-UNO

Project Analysis

Group 8:

Tanishq Harish Duhan

2019B5A70636P

Sushrut Patwardhan

2019B5A70382P

OOP Principles:

1) Encapsulate what varies:

   Most important data that keeps on changing in our project are the resultOfChance, card, nextCard and alreadyDrawn properties. These actually act like callbacks too, to notify the result of a player's chance. These can be encapsulated into a reactive observable.
   Plenty of private methods in both Game and Player class do encapsulate the variable data.

2) Favour composition over inheritance:

   Our code does favour inheritance in the Normal and Special Card classes as they both inherit the PlayingCard class. Instead of this, using an interface Card for both directly would have been better.

3) Program to an interface not implementation:

   The Card interface in our code could have supported this concept. Instead of inheriting a PlayingCard class, both Normal and Special Card classes could have implemented Card interface with a card factory class returning appropriate Card class objects.

4) Classes should be open for extension and closed for modification:

It is difficult to make entire systems open for extension and closed for modification, but small classes can achieve this goal. The card class is open for extension as adding new methods inside the Normal/Special Card class won't create any disturbances in the driving code. Also for modification, suppose adding a new power in the special card class can be achieved by just adding the new power in the Power enum. We do not have to change the existing code of the class yet we achieve the results.

5) Depend on abstraction, do not depend on concrete classes:

   Adding an abstract action layer between Player's play method and the main loop in the startgame method could have provided more freedom for the player action. Then, integrating more player actions like calling UNO or calling caught would have been easier compared to now. Currently we will have to change the driver code as well as the player's run method for adding these actions.

Observer Pattern:

Our code tried to have the ideology of observer pattern, but lacked it's precise implementation. The properties of the player class like resultOfChance, card, nextCard and alreadyDrawn which are also used in the Game class as well as the drawpile in the game class whose next card is required in the player class could have been abstracted out into observables. Using that would have been easier for the manipulation in the player class's play method as well as the game class's main driving while loop in the startgame method. Currently, our code is notifying changes to and fro buy the HashMap and the properties altered by setParameters method. If there was an Observable keeping track of the draw and discarded piles as well as the player (who is currently playing his/her turn) then the cluster of if-else inside the loop could have been encapsulated away. One more way would have been to register a player as an observer observing the state of draw, discard pile and turn index. There could have been an eventToState mapper continuously yielding states in the state of the game in a state stream depending on the event supplied in the event stream. This would have helped even more out when integrating the GUI. Observer pattern would have greatly improved the ability to separate out the actual business logic from the user interface logic.