Team Members: Richard Padilla, Michael Muruthi, Francisco Alderete, Karl Dill

Assignment 4 Rationale

The goal of this assignment was to implement the Strategy Design Pattern on the original rental system code base.

Our team implemented this design pattern using a client (blockBusterRunner.java), context (Rental.java), and Strategy classes (PriceStrategy and FrequentRenterPointsStrategy.java). These Strategy classes were subclassed into concrete strategies that implemented their own version of a calculation function to determine rental price or frequent renter points.

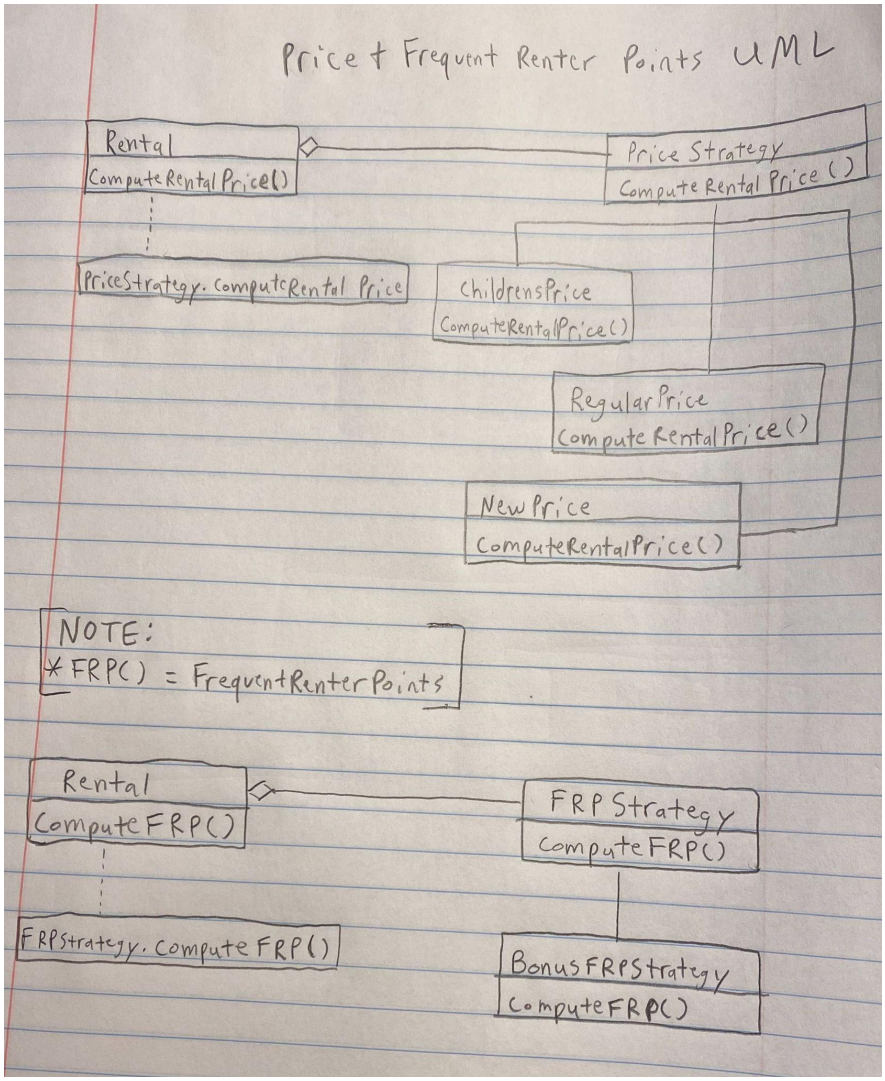
In our implementation, the context (Rental object) does not have to know ahead of time what type of strategy it will implement. It will simply call computeFrequentRenterPoints() and computeRentalPrice(), and the Strategy object will determine which algorithm to execute. The Strategy object is determined with logic in the client based on the Genre or length of time since a movie has been released.

The Genre and WeeksSinceReleaseDate fields were added to the movie class to allow the client to determine which price strategy and frequent renter point strategy to pass along into each Rental object.

The functions setPriceStrategy() and setFrequentRenterPointsStrategy() allow the Rental object to change its strategy objects dynamically. This ultimately allows the algorithm chosen to be separated from the Rental object.

Team Members: Richard Padilla, Michael Muruthi, Francisco Alderete, Karl Dill

Below is a UML diagram detailing the separation between Rental, Price/FrequentRenterPoint Strategies, and their subclasses:



References:
Dr. Nguyen Lecture Notes
In class discussions
Design Pattern Book
Derek Banas Video explaining Strategy Design Pattern: Strategy Design Pattern
Refactoring Guru blog post explaning Strategy Design Pattern:
https://refactoring.guru/design-patterns/strategy