Design Patterns

We have not implemented any interface in our code so the **strategy design pattern** is not applicable. We could have made use of this concept while designing the Community Chest and Chance Cards. Instead of using a switch case, an interface could have been written implementing the Cards and it could have classes for each card. This approach would have been very useful if we had created multiple players sharing the board instance.

The **observer design pattern** is also not used. It could have been used by making the board object as the subject and the players and the bank as the observers. The board can notify all the members involved about the change happening in it. Currently, we are allowing each player to interact with the board (and other players) on a sequential basis and displaying the interaction to the user.

OOP Principles

1. **Encapsulate what varies:** Encapsulation in Java is a process of wrapping code and data together into a single unit. We can create a fully encapsulated class in Java by making all the data members of the class private. In all our classes, the data members are private or protected and can be accessed only by the methods of the class itself. Wherever we need to access or modify the value of data members, we have used getter and setter.

2. **Classes should be open for extension and closed for modification:** In our Game class, we stored the instance to players in an ArrayList such that any number of players can participate in the game. We have not hardcoded the number of players. Thus, the game could be played with a varying number of players without any modifications.

3. **Strive for loose coupling between objects that interact:** All the instances of Player and the bank are loosely coupled. The field members of all of them are of private access specifier thus they cannot be reached directly from the outside. All the interaction between the players as well as with the bank are done by means of public methods.

4. **Depend of abstraction, do not depend on concrete classes:** We have made use of Abstract classes wherever possible. A lot of features like properties/utilities/stations owned, cash left etc are common to both players and the bank however, their implementation differs, thus these features are stored in abstract class Actor. Similarly, the abstract class Square is used by all the classes which implement different types of squares on the monopoly game board.