# Project 6
# Submitted by Ankur Sharma & Lincoln Rychecky

## Work Done:

### Lincoln

UNO Java Classes for Gameplay

- Card Objects & Factory
- Computer Player & Simple Game Play Strategy
- UNO Board Generation
- UNO Data Model
- UNO Game Controller
- Initial JUnit tests for UNO Game Elements

With all of these elements, computer players can play a game against each other.

### Ankur

- UNO card designs.
- UNO Board UI design, SelectPlay screen UI design.
- Delegated rendering of the activities and fragments to screen navigator.
- Dependency injection setup
- UNOGameFragment, SelectGameFragment and GameScreenActivity design.
- SelectPlayController and mvc setup with fragment and model.
- ComputerPlayerAdapter and HumanPlayerAdapter to interact with each player's card deck and PlayerDataModel
- Observer pattern to display Selected card by human player.

**Changes/Issues Encountered**

There have been minimal issues that have been encountered, but there have been some changes to the initially planned classes based on expected. For example, we have added a strategy class for the card object that determines how the data model is altered based on the particular card that is played. This helps to avoid a lengthy conditional statement whenever a card is played. We also have relocated any data to do with game state to the game model object with the game board just holding collections of objects that compose the game like cards and players. Additionally, we determined that the java class for the game screen fragment can function as a view for our MVC design.

**Use of Design Patterns**

Strategy: Gameplay algorithms for computer and human players. Model alteration algorithms based on cards when they are played. For example, (skip, reverse, plus two, wild, etc) all have differing effects on game state. That requires rather significant updates to the UNO game model.

MVC: We are making use of the MVC design pattern in order to implement game play logic, store/update game state, and render our views.

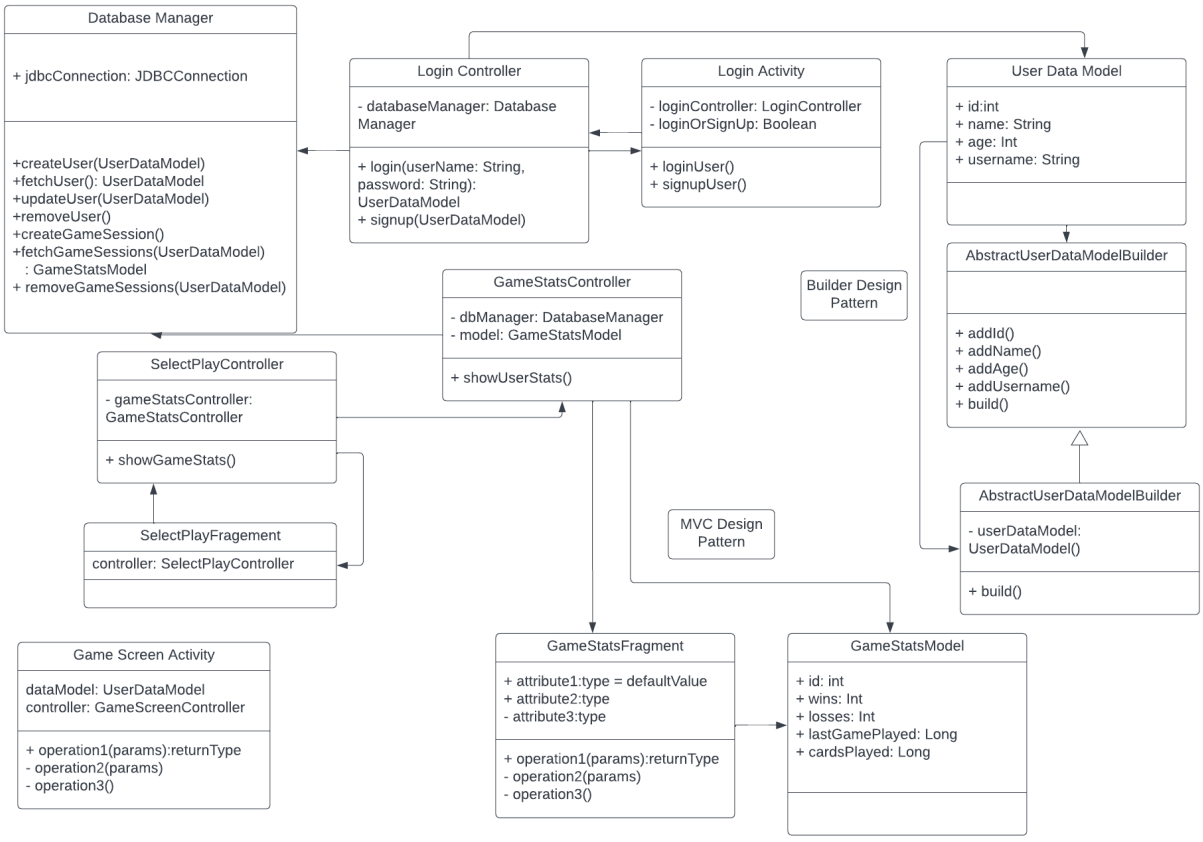
Factory: Factory is used to create instances of the card class. We will be changing this pattern to an abstract factory to help with the creation of card objects now that they will have a special strategy class object that determines their effect on the game model.

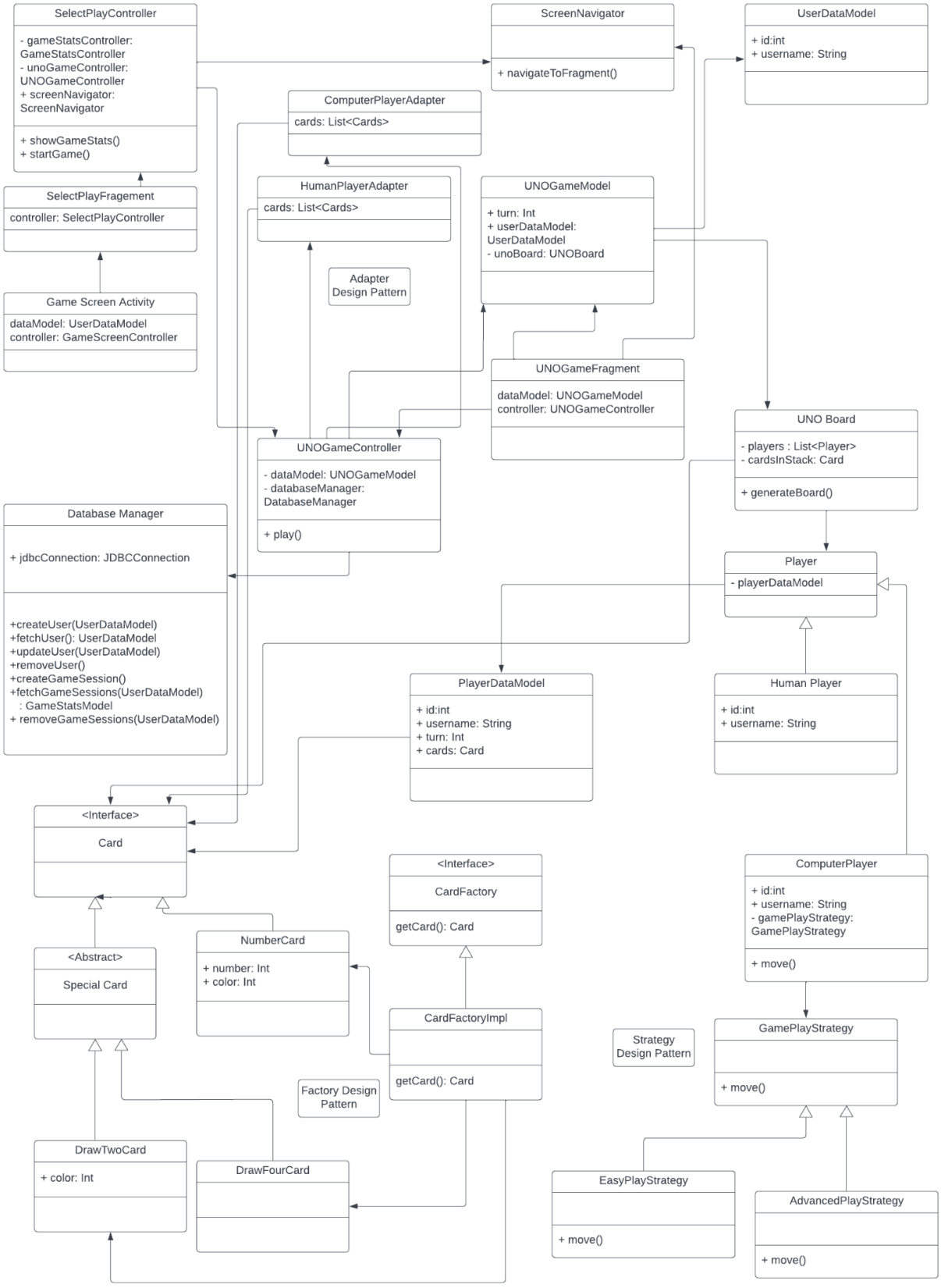
Dependency Injection: We are using the Android dagger framework to implement dependency injection.

Observer: We will be using the java observable interface in order to trigger changes to the view when the game model is updated.

Adapter: We are using the adapter pattern for connecting the UI of each player's deck with PlayerDataModel.

## Class Diagram:




SelectPlayController
- gameStatsController: GameStatsController
- unoGameController: UNOGameController
+ screenNavigator: ScreenNavigator
+ showGameStats()
+ startGame()
ScreenNavigator
+ navigateToFragment()
UserDataModel
+ id:int
+ username: String
ComputerPlayerAdapter
cards: List<Cards>
HumanPlayerAdapter
cards: List<Cards>
SelectPlayFragement
controller: SelectPlayController
Game Screen Activity
dataModel: UserDataModel
controller: GameScreenController
Adapter Design Pattern
UNOGameModel
+ turn: Int
+ userDataModel: UserDataModel
- unoBoard: UNOBoard
UNOGameFragment
dataModel: UNOGameModel
controller: UNOGameController
UNO Board
- players : List<Player>
- cardsInStack: Card
+ generateBoard()
UNOGameController
- dataModel: UNOGameModel
- databaseManager: DatabaseManager
+ play()
Database Manager
+ jdbcConnection: JDBCConnection
+createUser(UserDataModel)
+fetchUser(): UserDataModel
+updateUser(UserDataModel)
+removeUser()
+createGameSession()
+fetchGameSessions(UserDataModel)
: GameStatsModel
+ removeGameSessions(UserDataModel)
Player
- playerDataModel
PlayerDataModel
+ id:int
+ username: String
+ turn: Int
+ cards: Card
Human Player
+ id:int
+ username: String
<Interface>
Card
<Interface>
CardFactory
getCard(): Card
ComputerPlayer
+ id:int
+ username: String
- gamePlayStrategy: GamePlayStrategy
+ move()
<Abstract>
Special Card
NumberCard
+ number: Int
+ color: Int
CardFactoryImpl
getCard(): Card
Factory Design Pattern
Strategy Design Pattern
GamePlayStrategy
+ move()
DrawTwoCard
+ color: Int
DrawFourCard
EasyPlayStrategy
+ move()
AdvancedPlayStrategy
+ move()

## Plan for Next Iteration:

Implementing mySQL database (Lincoln)

Implement Human Player (Lincoln)

Connection Game logic to pre designed UI (Ankur)

Adding database operations to database manager and injecting it into screens that require user data. (Ankur)

Design and implement the Stats Screens (Lincoln)

Implement login screen for Users ( Ankur)