Introduction:

We've developed a functioning, object oriented, implementation of the classic game of poker. Our implementation features the ability to have multiple players, a functional GUI, and methods to check the ranking of each hand. Our design can be best described with one word: modular. Throughout the entire development process it was imperative that our classes were designed to be almost entirely self contained and functional on their own. With the exception of our GUI controller class, each of the objects we pretty much entirely contain all of their own functionality. The root of our whole project is our Card class, which is a very basic class that contains representation of card suit, rank, and a method for comparing instances of the Card class. From this class, we built objects that represent a poker hand, represent a deck of cards, and a class to compare different hands. These all come together in our main Poker class and the DeckViewController that enables our GUI.

User Stories:

- As a serious poker player, when I'm playing a game I want to be able to hide my cards from the other player.
- As a visual person, when I'm playing the game I would love a good looking interface to play on.
- The game must be able to automatically deal 2 cards to every player.
- The game must be able to compare, rank, and determine winning hands.
- As a casual player
- The user can place a bet. The user will be prompted to enter a bet of their desire.
- The user will be prompted with how many players are playing
- The computer will deal two cards to every player
- The computer will deal five cards for the table
- Each player can see their cards via JavaFx
- There will be buttons on JavaFx for the flop, turn, and river
- The program will determine which player wins and what they have

OOD:

To reference the OOD there are two example CRC cards as well as an IntelliJ generated UML and a UML diagram made in Lucidchart. To speak about the design in more detail though, we had to start from a conceptual understanding of how Texas Hold'em poker is played in order to think about the structure of our program. From the most fundamental level, poker consists of cards which are of utmost importance. These cards are used to make up your hand and come from the deck. This was the fundamental structure our program was built upon leading us to make a Card, Hand, and Deck class. These classes provide the fundamental structure that the game of poker itself was built upon so in our design we realized these lower level classes would need to be utilized with more complex functions that help the game of poker to play out in our program. We created a CompareHands class using the user's hands and the cards that made them up in order to determine who wins each round of the game. Comparing hands in poker specifically pertains to what cards the user's have attributed to their hands making the structure of this portion relatively fundamental. We then had to intertwine all of the basic functionality together in order to produce an output of the game that a user can experience through a GUI. This tied the basic elements that make up the game of poker to the rules of poker and allowed the user to simulate the playing of the game.

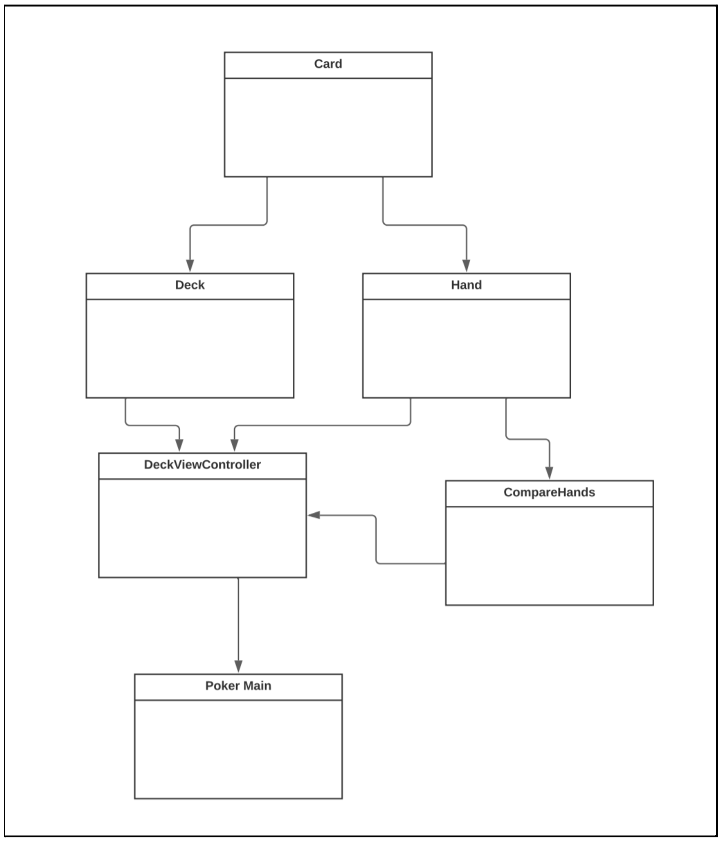
CRC Cards:

Below shows two examples of CRC cards that represent extremely important classes (DeckViewConroller and CompareHands) of our program. These two classes are extremely important for the game's functionality and rely on a bulk of the program's classes in order to be effective. As is shown in the DeckViewController, it relies on almost all of the classes within the program to be able to produce the output that the user is able to experience while running the program to play a game of poker.

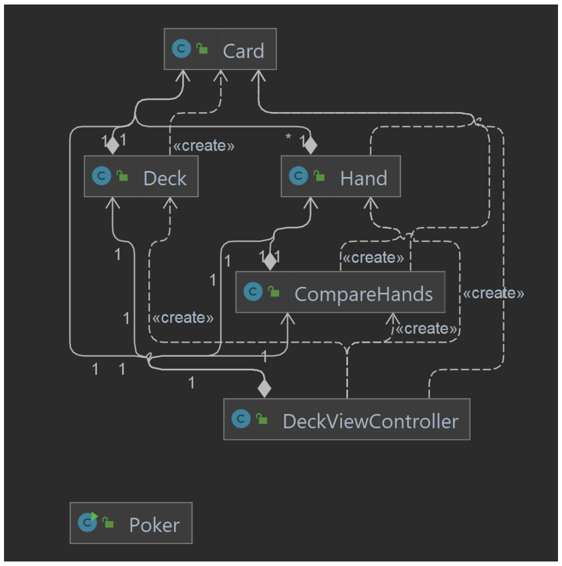
DeckViewController	
Set up the GUI for the program and produce the functionality of Texas Hold'em gameplay	Card Hand Deck CompareHands

CompareHands	
Ranks the two hands to determine which player has the better Poker hand with Royal Flush being the best all the way down to high card	Hand Card

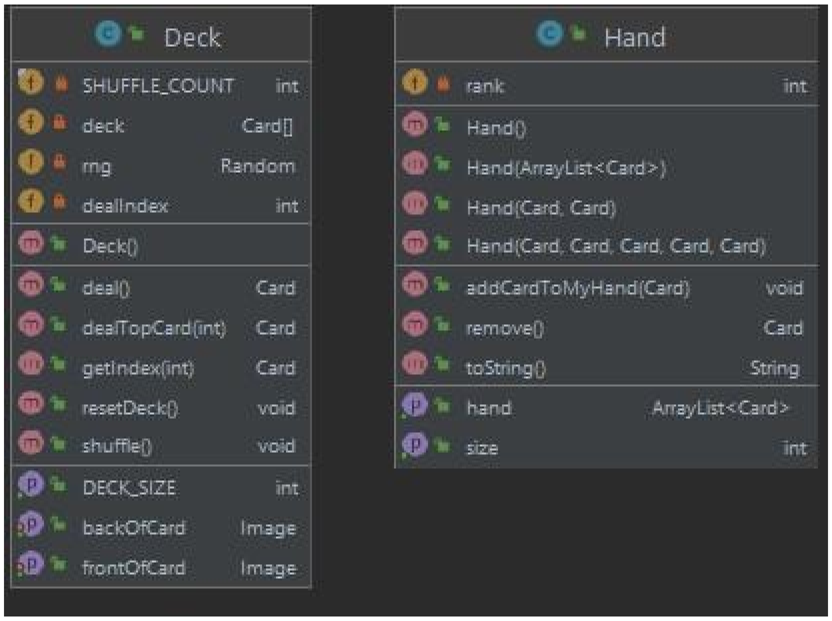
UML Diagrams:



This is a UML diagram for the entire architecture of our poker implementation. It clearly demonstrates the flow of class dependencies from the root Card class to the rest of the objects needed to build a poker game.

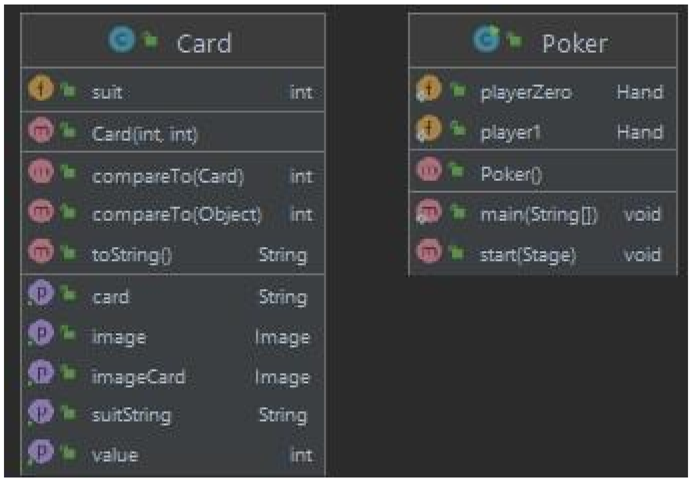


This is an IntelliJ created UML diagram similar top the one above. It is a far more in depth view of the dependencies and cross dependencies between our classes.



Deck	
SHUFFLE_COUNT	int
deck	Card[]
rng	Random
dealIndex	int
Deck()	
deal()	Card
dealTopCard(int)	Card
getIndex(int)	Card
resetDeck()	void
shuffle()	void
DECK_SIZE	int
backOfCard	Image
frontOfCard	Image

Hand	
rank	int
Hand()	
Hand(ArrayList<Card>)	
Hand(Card, Card)	
Hand(Card, Card, Card, Card, Card)	
addCardToMyHand(Card)	void
remove()	Card
toString()	String
hand	ArrayList<Card>
size	int



Card	
suit	int
Card(int, int)	
compareTo(Card)	int
compareTo(Object)	int
toString()	String
card	String
image	Image
imageCard	Image
suitString	String
value	int

Poker	
playerZero	Hand
player1	Hand
Poker()	
main(String[])	void
start(Stage)	void

These are closeups of the internal structure of our classes. Each object is built with methods meant to represent their real life counterparts as closely as possible. Take the deck class for example. We chose to contain all of the methods necessary for dealing, shuffling, resetting, etc. in the deck class rather than incorporating it into the game's main Poker class. This is necessary to maintain the modular functionality of our game.