CS 342 **Project#2** **Fall 2024**

3 Card Poker JavaFX GUI

Due Date:
Part 1: Wireframe and Class diagram: Sunday 10/27 @11:59pm
Part 2: Working Application and code: Sunday, 11/10, @11:59pm

Description:

In this project you will implement a two player version of the popular casino game 3 Card Poker. This is a somewhat simple game to understand and play which should allow you to focus on learning GUI development in JavaFX and trying your hand at event driven programing.

This project will be completed in two parts: Part #1 is the planning stage where you will create a wireframe of your user interface and well as a class diagram with all of the classes, interfaces and relations you expect to use in your code.

This project will be developed as a Maven project using the template provided. **You may work in teams of two but do not have to.**

How the game is played:
*****Keep in mind: there are different variations of this game you will find on the web; the following is how your version will play*****

In three card poker, each player only plays against the dealers hand, not each other:

- Both players will start by placing an ante wager. We will limit the ante bet to $5 or greater, up to $25.
- There is one optional bet the players can make called the Pair Plus wager. We will also limit this bet to $5 or greater, up to $25. This is a separate bet that will win if a players hand is at least a pair of 2's. The payoff for this bet applies regardless of the dealers hand and what happens in the rest of the game. (See below for payouts).
- After all bets are made(ante and/or pair plus), the cards are dealt out. Each player and the dealer receive three cards each. The players cards are face up and the dealers hand is face down.

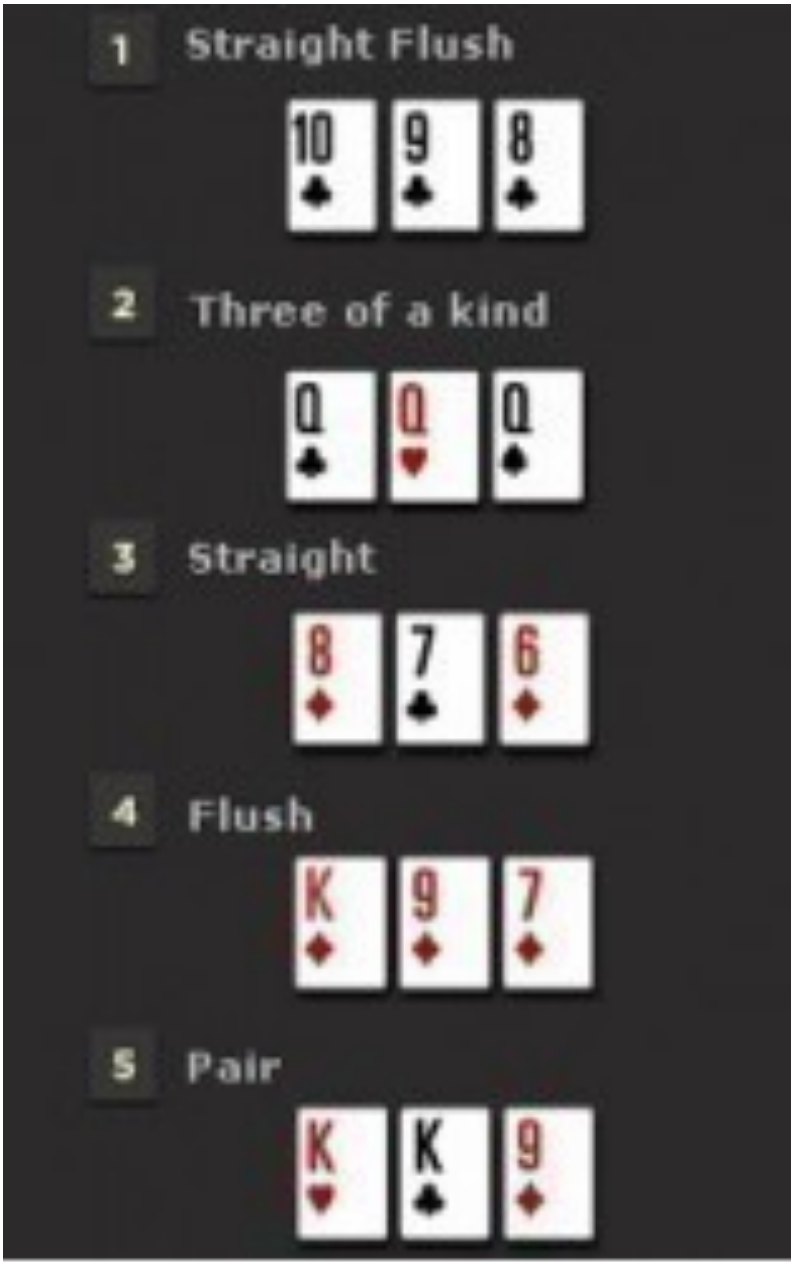
- Each player must decide if they will play or fold. If they fold, they lose their ante wager and pair plus wager(if they made one).
- If the player wants to continue, they will make a play wager (this must be equal to the amount of the ante wager).
- At this point, the dealer will show their cards. If the dealer does not have at least a Queen high or better, the play wager is returned to the players who did not fold and the ante bet is pushed to the next hand.
- If the dealer does have at least a Queen high or better, then each players hand, that did not fold, is evaluated against the dealers hand (see below for order of winning hands). If the dealer wins, the player loses both the ante and play wager. If the player wins, they get paid out 1 to 1 (they get back double what they wagered). Say the player bet $5 each for the ante and play wager and won, they would get back $20.

For the Pair Plus wager:

As long as the player does not fold, the Pair Plus wager gets evaluated regardless of if their hand beat the dealers hand; it is a separate bet based solely on the players hand. If the player does not have at least a pair of 2's, they lose this bet. Otherwise, the payouts are as follows:

- **Straight Flush 40 to 1**
- **Three of a Kind 30 to 1**
- **Straight 6 to 1**
- **Flush 3 to 1**
- **Pair 1 to 1**

Order of winning three card poker hands:



CS 342 **Project#2** **Fall 2024**

Implementation Details:

You must create the following 6 classes for this project:

public class Card

```
char suit;
int value;

Card(char suit, int value)
```

This class represents a card in a deck of 52 playing cards. The data member **suit** will be a capitalized character representing the suit of the card(clubs, diamonds, spades, or hearts) **'C', 'D', 'S', 'H'**

The data member **value** will be an integer value between 2 - 14, with the value of an ace being 14, king 13, queen 12, jack 11, ten 10…..and so on.

You will provide a two argument constructor that takes in and sets the values for **suit** and **value**.

public class Deck extends ArrayList<Card>

```
Deck();
newDeck();
```

This class represents a 52 card, standard deck, of playing cards. The constructor will create a new deck of 52 cards that have been sorted in random order. The second method will clear all the cards and create a brand new deck of 52 cards sorted in random order.

public class Dealer

```
Deck theDeck;
ArrayList<Card> dealersHand;

Dealer();
public ArrayList<Card> dealHand();
```

This class represents the dealer in the game. The no arg constructor will initialize theDeck. The data member **dealersHand** will hold the dealers hand in each game. The method **dealHand()** will return an **ArrayList<Card>** of three cards removed from theDeck. Before each game starts, the Dealer class must check to see if there are more than 34 cards left in the deck. If not, theDeck must be reshuffled with a new set of 52 cards in random order.

public class Player

```
ArrayList<Card> hand;
int anteBet;
int playBet;
int pairPlusBet;
int totalWinnings;

Player();
```

This class represents a player in the game. It keeps track of each games current hand and current bets as well as the total winnings for that player across multiple games. If the player has lost more than he/she has won, that number can be negative. Provide a no argument constructor for this class

public class ThreeCardLogic

```
public static int evalHand(ArrayList<Card> hand);
public static int evalPPWinnings(ArrayList<Card> hand, int bet);
public static int compareHands(ArrayList<Card> dealer,
ArrayList<Card> player);
```

This class represents the logic in the game. The method **evalHand** will return an integer value representing the value of the hand passed in. It will return:

- 0 if the hand just has a high card
- 1 for a straight flush
- 2 for three of a kind
- 3 for a straight
- 4 for a flush
- 5 for a pair

The method **evalPPWinnings** will return the amount won for the PairPlus bet. It will evaluate the hand and then evaluate the winnings and return the amount won. If the player lost the Pair Plus bet, it will just return 0.

The method **compareHands** will compare the two hands passed in and return an integer based on which hand won:

- 0 if neither hand won
- 1 if the dealer hand won
- 2 if the player hand won

public class JavaFXTemplate

```
Player playerOne;
Player playerTwo;
Dealer theDealer;
```

This class is given to you in the Maven template. You will need to create instances of the above dataMembers to run your game and put your event driven logic here (you can get rid of the animation example).

Note: You must implement the above just as they are described , with the exact signatures, in the project write up. We will use these data members and methods to test your projects. Failure to do so will result in significant loss of points.

Note2: You are free to add data members and methods as you see fit to implement your game.

The GUI:

You are welcome to use/discover any widget, pane, node, layout or other in JavaFX to implement your GUI. **For this project, you must implement the user interface using FXML layout files and CSS**. The following elements are required:

CS 342 **Project#2** **Fall 2024**

1) A welcome screen to the game: This will display when the app opens with some sort of greeting and then the user will choose to either play the game or exit. If the user exits, the app will shut down. If the user wants to play, this screen will transition to the game play screen.

2) The game play screen:

- There should be an area to display both Players cards and Dealers cards with each clearly labeled. You may use images or text to display the cards.
- Each player must have some way to make all of the available game wagers.
- Each player should have a separate area to display the Ante, Pair Plus and Play wager.
- Each player should have a separate area to display total winnings.
- There should be an area that displays info for the game. For example:

  "Player one loses Pair Plus"
  "Player one beats dealer"
  "Player two loses to dealer"
  "Player two wins Pair Plus"
  "Dealer does not have at least Queen high; ante wager is pushed"

- You will need to have a menu bar in your program with one tab: **Options**

Under options you will have **Exit, Fresh Start and NewLook**. **Exit** will go to another screen giving the user the option to quit or return to the game. Quit will end the program and continue will return to the game play screen as it was before Exit was chosen. **Fresh Start** will reset each players current winnings to zero and allow the user to

start a new game. **NewLook** will change the look of the GUI; such as new colors, fonts, images….etc. This must be done using CSS. While there is no minimum for elements to change, the new look must be noticeable to the average user.

Playing the game in your Program:

Your game must play and feel like the user is actually playing in real time. You must include pause transitions or add buttons like "continue" to control the flow of the game. If

you did not, the program would move too fast and not allow the user to understand what is happening.

Testing Code:

You are required to include JUnit 5 test cases for your program. Add these to the src/test/java directory of your Maven Project. Keep in mind that you will not be able to test any of the graphical elements with JUnit, only the logic of your program. At a minimum you must test:

- ThreeCardLogic class, at least 20 test cases
- Deck and Dealer class, at least 10 test cases

How to Start:

Some of you are used to just starting to code with no real plan for what you are doing. This project will be very painful with that approach. You must be systematic and thoughtfully plan out how various events will drive your program. You must also thoughtfully plan out how the user is allowed to interface with your program and how the user will know what to do next.

- If you are not familiar with this game, play a few hands with some friends or play it online. Note that the online version you find might be slightly different than what we are doing here.
- Design the wireframe for your user interface. Think through how the user will interact with your program and how they will know what to do and what has happened. Think through all of the events that can happen and how each event leads to the next event and how the user interface will change.
- Design your Class diagram. Think through all of the classes, interfaces, data members and methods you will need to complete your application.
- Implement the Card, Deck and Dealer classes first and test them.
- Next implement the ThreeCardLogic class and test it.
- Think about how your user interface will control the action of the game.
- Map out your user interface and what happens as the user interacts with the different widgets.
- Add event handler methods to your controller class/classes, one by one, and test the functionality as you go.
- Play through your game and ask yourself if the user experience is good. A bad user experience ensures that no one will ever play your game and your time was wasted no matter how clever you were programatically.

CS 342 Project#2 Fall 2024

Electronic Submission:

If you worked in a group:

- only one of you needs to submit for part one and part two.
- You must include a PDF file called Collaboration.pdf. In that document, put both of your names and netIds as well as a description of who worked on what in the project.
- One of you will submit the collaboration PDF to the link provided on BB, do this after part two is complete.
- If you worked alone, no need for the Collaboration.pdf.

PART #1:

submit your wireframe and Class Diagram as a single PDF (multiple pages is ok) to the link provided. Name it netid + Project2WireframeClassDiagram.

PART #2:

Zip the Maven project and name it with your netid + Project2: for example, I would have a submission called mhalle5Project2.zip, and submit it to the link on Blackboard course website.

Assignment Details:

Late work is accepted. You may submit your code, part 2, up to 24 hours late for a 10% penalty. Anything later than 24 hours will not be graded and result in a zero. There is NO late turn in for part 1.

We will test all projects on the command line using Maven 3.6.1. You may develop in any IDE you chose but make sure your project can be run on the command line using Maven commands. Any project that does not run will result in a zero. If you are unsure about using Maven, come see your TA or Professor.

Unless stated otherwise, all work submitted for grading *must* be done individually. While we encourage you to talk to your peers and learn from them, this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is available here:

https://dos.uic.edu/conductforstudents.shtml.

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between

students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing

your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. It is also considered academic dishonesty if you click someone else's iClicker with the intent of answering for that student, whether for a quiz, exam, or

class participation. Academic dishonesty is unacceptable, and penalties range from a letter grade drop to expulsion from the university; cases are handled via the official student conduct process described at https://dos.uic.edu/conductforstudents.shtml.