Documentation for Poker Game Project:

Programmers: Ceren Kahyaoglu, Muhammad Chaudhary and Patricia Sison.

Brainstorming ideas of what things we will need before we start working on project:

Inputs:

>Arrays to store the faces and suits
>Use shuffle/Random methods to shuffle the deck.
>Collection method to collect all data.
>Have comparator to compare hands.
>Have some sort of variables to hold pairs of cards of same rank.
>Having conditions, that satisfy different kinds of hands.
>Have for loops to run through cards to figure out what kind of hand it is.
>Have a proper functioning method that could identify more than one pairs in a hand, especially for the hands such as “2 pair” and “Full House” (possible to use Hashset or If/else Statements or maybe create new variables that would read the left over ranks after the first pair is recoded)
>Have JFrame and JLabel to show the UI.

Outputs:

>UI
>Cards
>Name of players (maybe students from class, if have time)
>Ranks of each hand
>Show the winning card.
>Display the name of the winner player

Some Ideas/thoughts on Data Structures we should be using.
“So, I've been thinking about a data structure that we can use to sort through the hand to determine winning cards. If it was only numbers that didn't repeat, then it would be easy to use an array. But since we have suits and repeating numbers it's a little more difficult, but manageable. I think we can set the card numbers from 0-51 and use a set as an entry. If we use the number of the card as the key, we can set each value to a suit i.e., <3, C>. This way each key will be different. Then we use the method entrySet which will allow us to iterate through each (entry) card using Next”. (Muhammad)

Suggestions by CeCe

Methods that we could use to count the pairs and add those pairs:

```
private static PairCount
countPairs(String s)
{
```

```
s += " "; PairCount pc = new PairCount(s);
for (int start = 0; start < s.length() - 1; start++)
{
        if ( s.charAt(start) == s.charAt(start + 1) && s.charAt(start) != s.charAt(start + 2) )
        { pc.count++; }
        }
}
```

Suggestions coming from Patricia.

Learned about how sorting works in terms of picking up a hand of cards.
Insertion sort algorithm:
Step 1: for each array element from the second index to the last.
Step 2: Insert the element at where it belonged in the array, increasing the length of the sorted subarray by 1 element.
And to accomplish Step 2, the insertion step, we need to make room for the element to be inserted by shifting all values that are larger than it, starting with the last value in the sorted subarray.

Card class:

Started working on the project by creating a Card Class, a class that contains ranks and suits variable, and a static string that will be used to represent the different ranks of cards when we move forward with our poker hand class.

We used static methods for the ranks and other strings (i.e card_rank_string) to be allied to the class as a whole.

We will be using a utility method, “card_rank_string” it will play the role of taking a number and turning it into the string of the rank. The card_rank_string will be used in the “PokerHand” class to represent the ranks of cards, again we are using a static array to represent different names of suits, and it belongs to the class as a whole. So, it can be accessed by instance method of each Card and by static method of the class.

We used short for ranks and suits because it is a converse memory and since we have a large array so it's better to hold our data safely in the memory. (But later when integrating The UI with "PokerHand" class we had to add some extra methods to take in an object "card" from the UI class and convert its value of ranks and suits to a type short.

A "toString" method, to return what rank is of what suit.

Deck Class:

In the deck class, we made it as a container for cards, we put cards "faces" in the array list of cards. We included our random generator and tested to see if it is printing cards randomly. So, our random generator takes 50 pairs of cards and switch them, which will shuffle our deck. Card1 and Card2 variables are made to generate random cards.

And lastly "drawFromDeck" method, which will return the first element of the card and then remove that card from the deck and repeat the shuffling process. This way we are not repeating cards of same rank and suit. Our drawFromDeck method is a lot simpler and less complicating since randomization of the cards is already done.

We ran a quick test, of our program and successfully were able to print out random cards from the deck.

Poker Hand class.

This class we planned to do all the dirty work. The comparing of hands, holding and recoding of pairs and ranks using different variables, giving values to hands of the cards, setting up expectation for each hand and then using the display method to tell the user what hand what is (a pair, 2 of a kind, flush, straight. etc.).

PokerHand class will extend the Deck class since we need the cards (Ranks, suits) and drawFromDeck variables to write the comparing methods.

We have an array of 5 cards and 6 Ints to represent the value of a hand. We used for loop to use the list of cards that can be used as a poke hand, and then added all the cards from the deck into an array.

We then used “initialize” method to assign values to different arrays, there will be 14 different ranks(2345678910JQKA). We ignore the 0 Array here that will keep our comparing methods much simpler to understand. Arrays of ints will have 13 slots, one for each rank which will then go through the cards. The first index of the array will be empty. We then incremented the appropriate index of the array and then incremented the “rankOfcards” array at the index of each card rank.

Now we already knew that we need some kind of variables to record and hold our cards of same rank, which we successfully were able to implement, but we caught ourselves into a problem where we didn’t knew how to implement those variables and make it hold 2 pairs at a time.

The “IdenticalCards” could record at least one rank pairs of cards and have the variable “LargeRankPairs” hold that pairs. We knew that it will be a successful approach if we were to only look up for hands such as “A pair” “3 of a kind” and “4 of a kind” but weren’t able to figure out what needs to be done in order to make them hold more than 2 pairs. So now we needed to implement a code that keeps a track of atleast 2 ranks of cards and the amount of cards in each rank This was important part of our code especially because it was the only way we could determine our “Full house” and “2 pair” hands.

We referred to the book “Introduction to Programming Using Java” Chapter 2. Names and Things 2.2.3 Variables in programs, it states that multiple variables hold memory and could be used and compared in a single method. It has a brief explanation of different ways variables could be used. Now we did knew that variables could be used and may or may not be a successful approach but we kept trying the If/Else method to see if it works. After our multiple tries we gave variables a try and It looked like it made more sense.

We continued with the project; we created a loop going through 13 to 1. We started setting up conditions that is Rank of cards is greater than Identical Cards we store the value in LargeRankPairs and if there is another pair found, store it in small rank pair variable. We also tried to override the identical cards, so if we find a small pair first and then a large rank pair, show the large rank pair first.

Now the only determination we hand to do was if pair is a straight or flush. We started with the flush first, the requirement was to determine if all the cards in a hand are of same suit or not. So, we hitched a ride on the loop that iterated through the cards recording their ranks. We assumed that there is a Flush and traveled through the cards using for loop to see if one of their suits doesn’t match the suit of the next cards its cant be a flush, but let’s say if it does match the suit of all other 4 cards, we call it a Flush. For Straight we needed to figure out if there are 5 cards in a row whose rank is increasing by 1. We used Boolean and assumed that there is no straight. We than used for loop to iterate through cards and see if the cards are in increasing order of 1 call it a straight. For the Royal Hand we put a condition that it must start from rank 10, and with the help of for loop we stated that if the value of r which is 10 is increasing to a point where it meets A it will be a Royal hand. Royal hand mush have same suit so we set that conditions when we were evaluating hands. We used Identical cards variables as our key

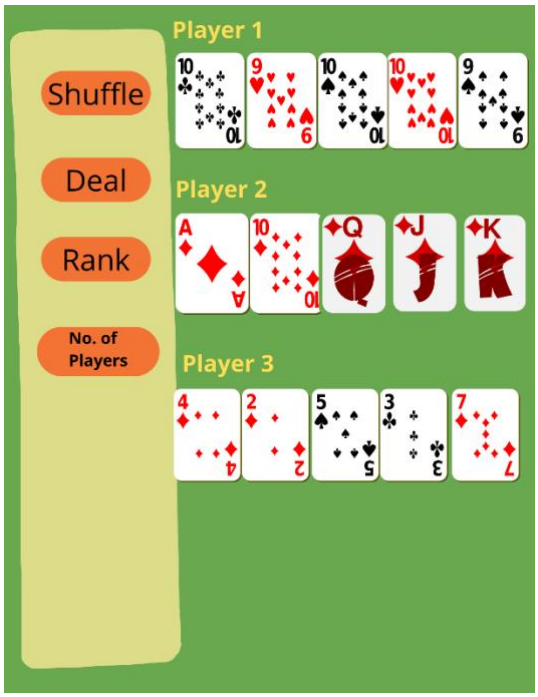
components to put conditions for most of our hands. So, let's say if Identical Cards =1 means that there is only one identical card (no pair) in a hand and all the other cards are no identical call it a high card. Similar with pair hands, we set identical cards = 2, so if there are 2 identical cards call it "a pair" and for "3 of a kind" identical cards are set to 3. For 2 pair we used "?" ternary operator, I found this term when using Oracle java API documentation. "?" is a more efficient way of using if/else statement on a single line. So, in the situation of 2 pair hands if the large Rank Pairs is greater than small Rank Pairs the large Rank Pair will be returned else small Ranked pairs will be returned and same for rank of smaller pairs, so if small rank pair is larger than LRP, LRP will be returned. We gave values to each of these hands, High Card having the value of 1, the higher we go the higher the value is, Royal flush having a value of 10.

So now we have implemented all different types of hands that we might encounter while playing poker game. We just need to compare those hands. We used a "compareTo" method that compare these hands. So, if one hands value is greater than the other hand returns the hand with greater value as a winning hand, and if it is opposite, return the other hand as winning hand.
At the end we displayed those hands using a Show Hand method. The case 1-10 are the ranking of the hands similar to the values we gave to all these hands earlier when implementing the code for conditions.

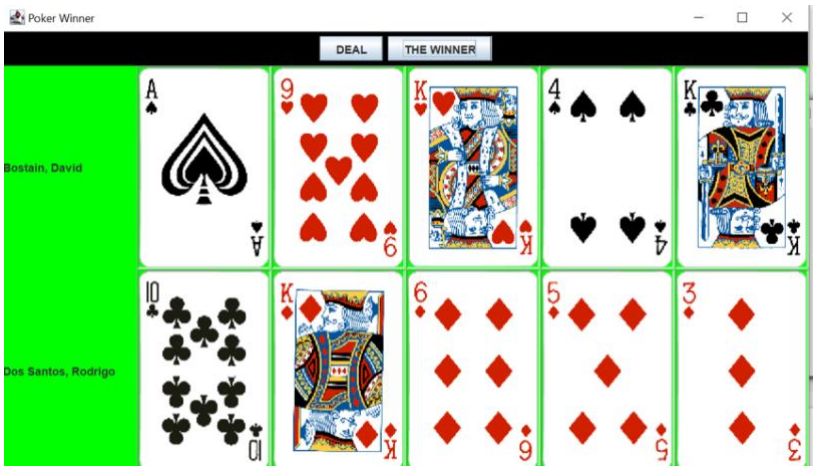
UI

Thanks to Prof. Bostain, that he let us use his deck.java program to help us take a quick start with the UI of our project. Since we were designing a poker game with a simulation several modifications were made in the UI class, to meet the requirements.

Initial Design for User Interface



Actual User Interface



The first UI we used that only had deal button running.

The initial design for the User Interface is for the control panel to be lined up on the left side of the screen and the name of the players to be on top of each hand of cards.

The additional buttons on the initial design were not included on the final design since there was a modification of the project. The project was planned to user for number of players and number of cards in hand, then it was decided to forward with fixed two players and 5 cards in hand for each player. The only buttons that are significant for the project are the deal and winner button as two players and their cards are set in the panel.

In actual design labels and panels are nested into the frames. Frame can be resizable. In frame, there is a main panel holding three other panels in it. First panel is created to hold buttons on top the frame by guiding users to start to play from top by using buttons. After buttons' panel, there are two same designed panels. Each player has their own panel to show their name tag, and their hands. They have same panel design but their hand is not same in labels. To represent players' hand, cards are being loaded into labels by creating newly each time when clicked on DEAL button.

THE WINNER button has a function to open another frame in the UI. When you click the button, it creates a new window holding the winner player's panel and a label to print the result for winning case. So, in the second window you can see who the winner is, what are the winning hand's cards and what the winning case is. For example, this hand has a pair of 2's or This hand has a 3 of a kind etc.

When you clicked the exit button on second winner window, It is set that user is being back to main window where he can keep playing the game. DEAL and THE WINNER button is still functioning to keep playing. In main window if you click to exit button on frame, you quit the came and to play it again you have to re-run the program.

Description: This class contains the methods shuffle and deal cards and the panels for the frame used to display the user interface.

public static void main(String[] args)	This is main method contains an EventQueue passing in a new object of deck that is an instance of the UI class. Since swing is not thread-safe, we need to have an EventQueue and make it run there.
private JPanel shuffleDeckAndDealCards(Deck deck)	This method deals with shuffling and dealing cards. It takes a deck of type Deck as a parameter. It calls the shuffle method in the java.util.Collections to shuffle the list of cards in the UI. A panel is also created in this method that will contain the hand for one player.

private static BufferedImage load Image (String s)	This method is used to load the cards' images as Buffered Image to put into the labels later in other method. It will get the images of the cards from a file input. Since the image is too big, it needs to be scaled. A drawRenderedImage will be called passing in the buffered image and will transform it to the right scale.
public class UI	This class contains variables used to create the frame, panels and labels for the UI. The buttons are mapped to the actionListeners to call the action made by the user. • jbDeal.addActionListener • This actionListener is invoked when the user clicks on the deal button. It calls the method shuffleDeckAndDealCards(deck) and randomize the name of the players. It replaces the existing JPanels for the players with a new JPanel containing the

	updated shuffled deck and names as a new hand. • jbTheWinner.addActionListener(new ActionListener() • This actionListener is invoked when the user clicks on the winner button. It calls the method actionPerformed(ActionEvent e). In this method an arrayList of the player's cards was made to store the values of the cards each player has in their hand. It will iterate through all the cards each player has by getting them from each panel and getting the client property of the card to get its value. It creates new objects called pHandOne and pHandTwo which is a type of PokerHand. The comparator method from PokerHand class is called to compare the hand of each player. If the value for

	pHandOne is greater than 0 then it will show the hand of pHandOne and declare the first player as the winner. If the value for pHandOne is less than 0 then it will show the hand of pHandTwo and declare the second player as the winner. If the vaue is equal to 1 then there is a tie between the two players. • With an "if condition" , the button was set to open a new window to show the results. ActionListener's clicked method is used to place the new window in it. With " `if (e.getSource() == jbTheWinner)`" (if THE WINNER button is clicked), an other frame is created with the winner and hand.

Class: Card

Description:

Constructor for Card Object: This constructor will take in an object “card” from UI class and convert its rank and suit.

Methods:

- public short convertSuit
    - This method will take in an object “card” from UI class and convert its value of Suit to a type short.
- public short convertRank
    - This method will take in an object “card” from UI class and convert its value of Rank to a type short.
    - It will compare every rank of the card from UI class to the arrayList of string cards from card and deck class and return a short value of the rank if it is valid.

Class: Poker Hand

Constructor for Poker Hand

(publicPokerHand(ArrayList<edu.mccc.cos210.poker_testing2.UI.Card> cardList)

- This constructor takes the object card from the UI class and store it an array. This list of cards can be used for the hands of each player.

Problems encountered:

We had a difficult time integrating all the classes together. Since the UI class has its own card class that is built in a different structure than the card and the deck class, there are errors when trying to call Poker Hand class in the UI class. The Poker Hand class is dependent on the Card and Deck class, so we need to convert cards from the UI class into the same cards in the Card and Deck class. We made constructors for the Card object and methods that will convert its suit and rank.

During UI design process, matching our first draft-template with java design was complicated and then the project requirements were updated by professor and we decided to go simple, manual design.

Loading images into Labes to make frame looking more gambling webpage was planned. To figure out to load image into a label as banner was challenging. We import pictures into data file and tried to read it as buffered image, and load into labels, but it was not managed to put images into labels. Then we decided to forward by changing only background color of panels. We choose the green and red colors as poker game defaults.

When we load the shuffled decks into the players panel, connecting new decks with DEAL button was the challenging part of the code. We came up with the result of remove panels and put them back again after shuffling and dealing.

THE WINNER button's coding process got two phases, in first phase we focused to compare the cards ranking in hands to decide who is the winner in real time of game. Then in second part, we created another frame to show that ranking result as announcing the winner.

In our project we successfully were able to complete most of it but were not able to successfully implement a program that identifies and compares the ranks of similar hands of the deck. We created variable that would count the values of each rank in a hand and then whichever hand has the highest value of cards would win but that did not work out as expected, because having just one A and random small rank cards in one hand and having K, J, 10 cards in the other hand makes it a winning hand. It also started giving us issues with 2 pair hands. We were running out of time, so we ended up keeping our program to what it was before.

Communication within groups was effective and cooperating. Arranging a time where we all could meet together was a bit challenging, but we still managed to do that. Everyone worked on their part of the project but at the end we all helped each other with issues we were facing with our programs.

Work Cited:

Introduction to programming Using Java by David J.Eck

Oracle Java API Documentation

Wikipedia