User Manual:

The game will start you with $10,000 and you need to make a bet on the next round against the computer. Each round starts with you getting two cards and the computer getting two cards. The goal is to get the sum of your cards closest to 21 without going over. You can ask for another card if your total is under 21 by typing H(hit/add another card to hand) or S(to stay). If the sum of your cards is closer to 21 (than the sum of the computer's cards), your bet is added to your total. If you don't get closer or go over 21, then the bet is subtracted.

Class Documentation:

The whole game consist of 4 classes: Card (superclass), Deck (subclass), Player (class), Blackjack (Abstract class)

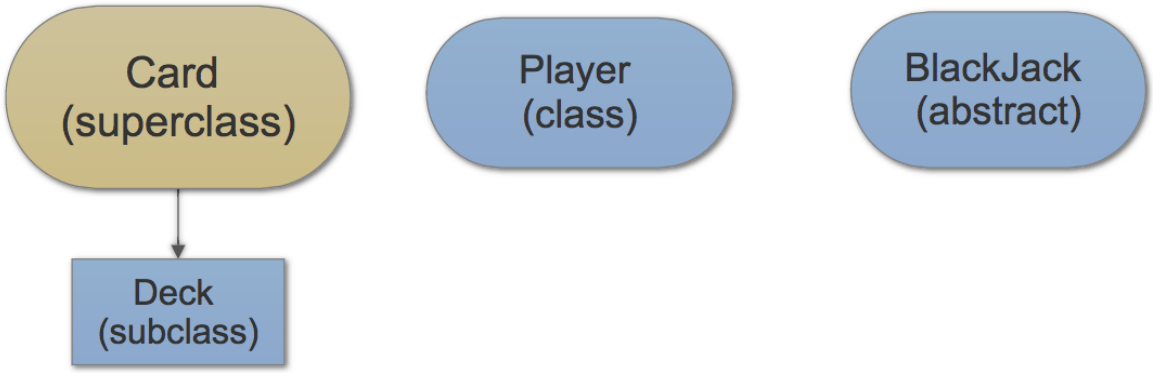
Card: It's a class that is used to define each initial element of the deck.
We have an initial constructor that accepts two arguments from the user(suit and number of the card). In the constructor we use an if-else statement to check if the number that the user doesn't enter a number below zero or a number over 13. We also added a getNumber method to get the card number that the user entered which is used in another class to check the total value of the user's hand. Finally we have a toString method so that we can print out the card that the user is dealt and any cards that they chose to draw throughout the game. We used a case statement to assign each number value to the appropriate string.

Deck: It's a class that is used to make a deck, randomize each card in it and deal a card.
In this class we define an array that holds 52 elements, one for each possible card. This process is done using the default constructor and a for loop. Of course, to play a good game of any card game we need to shuffle the cards. Using a random object and a for loop we are able to randomize the order of all the cards in the deck. Finally we have a deal method to set the initial hands for the player and the computer and for the event that either player wants another card to get closer to 21.

Player: This class is used to define a player and the different actions they can take.
In this class we have two constructors, one with no parameter that just empties the players hand and one with a parameter for the name in case the user wants to make things personalized. We have a simple empty hand method that sets ever card in the user's hand to null. Next is the addCard method that checks if adding another card sends the player's hand total over 21 so that the game can end before the player adds another card to their hand. getHandSum is a simple method the iterates through each element in the player's hand and adds it together. Finally we made a printHand method so that the user can see what is in their hand after each turn.

Blackjack: This class executes the gameplay.
This class contains the driver which gives the player an initial amount of money and the option to bet a certain amount of money. The program then uses a series of while loops and all of the previous classes and methods to follow the rules of blackjack and play out the game until the player runs out of cash or wants to give up.

Hierarchy Drawings:



Error Timeline:

(1) 5/9:

Bug: The CPU's hand was also printed

Before-

```
public void printHand() {

    System.out.printf("%s cards: \n", this.name);
    for (int i = 0; i < this.numCards; i++) {


            System.out.printf(" %s\n", this.hand[i].toString());

        }
    }
}
```

After-

```
public void printHand(boolean showFirstHand) {

    System.out.printf("%s cards: \n", this.name);
    for (int i = 0; i < this.numCards; i++) {

        if (!showFirstHand) {

            System.out.println("[hidden]");

        } else {

            System.out.printf(" %s\n", this.hand[i].toString());

        }
    }
```

(2) 5/14:

Bug: 11,12 and 13 were added for jacks, queens and kings respectively.

Before -

```
    for(int i = 0; i < this.numCards ; i++) {

        cardNum = this.hand[i][0].getNumber();

        handSum += cardNum;
```

After -

```
for(int i = 0; i < this.numCards ; i++) {

    cardNum = this.hand[i][0].getNumber();

    if (cardNum > 10) {
        handSum += 10;
    } else {
        handSum += cardNum;
    }
```

<u>(3) 5/14:</u>

Bug: Both the dealer and the player went over 21 as their final values.
Fix: Changed the if statement to see if user got over 21 as well(added && user.score > 21)
Before-

```
if ((player.getHandSum() > cpu.getHandSum() && player.getHandSum() <= 21) || (cpu.getHandSum() > 21 )
```

After -

```
if ((player.getHandSum() > cpu.getHandSum() && player.getHandSum() <= 21) || (cpu.getHandSum() > 21 && player.getHandSum() <= 21)
```

(4) 5/20:

Bug: The cash amount became negative and the game continued.

Before- `if(cash_total != 0){`

Fix: Changed the while loop to take account for less than 0 instead of just not 0.

After - `if(cash_total <= 0){`

(5) 5/22:

Bug: Even after the fix above the game continued and printed out the final cards and asked if the player wanted to play again.

Before -
```
if(cash_total <= 0){
    System.out.println("You're out of money, you can't play anymore.");
```

After -
```
if(cash_total <= 0){
    System.out.println("You're out of money, you can't play anymore.");
    option = 0;
    break;
}
```

(6) 5/22:

Bug: The bet amount entered more than total cash or is negative.

Fix: We added another while loop before the gameplay code to check if the bet amount is over zero and less than

```
while (option != 0 && cash_total > 0) {
```

(7) 5/24:

Bug: User enters a number when asked for a character and game quits

Before -
```
if (!isPlayerDone) {

    System.out.println("Hit or Stay (Hit gives you another card and stay does nothing): Enter H or S");
    ans = scan.next();


    if (ans.compareToIgnoreCase("H") == 0) {
        isPlayerDone = !player.addCard(gameDeck.deal());
        System.out.println("\n");
        player.printHand(true);
```

After -

```
if (!isPlayerDone) {

    System.out.println("Hit or Stay (Hit gives you another card and stay does nothing): Enter H or S")
    ans = scan.next();
    while (ans.compareToIgnoreCase("H") != 0 || ans.compareToIgnoreCase("S") != 0){
        System.out.println("Not a valid option. Please enter H or S");
        ans = scan.next();
    }
```

(8) 6/2:

Bug: Game kept telling user that H wasn't a valid option, even though i is.

Before -
```
while (ans.compareToIgnoreCase("H") != 0 || ans.compareToIgnoreCase("S") != 0){
    System.out.println("Not a valid option. Please enter H or S");
    ans = scan.next();
}
```

After -
```
while (ans.compareToIgnoreCase("H") != 0 && ans.compareToIgnoreCase("S") != 0){
    System.out.println("Not a valid option. Please enter H or S");
    ans = scan.next();
}
```

(9) 6/7:

Bug: Game continued even when cpu no longer needs more cards

Before-
```
while (!isPlayerDone && !isCpuDone) {
```

After -
```
while (!isPlayerDone || !isCpuDone) {
```

Testing Log:

Card(), (Card class)	The default constructor for card that defines a card with no suit and the number 0. We used this class to create a deck full of cards like this instead of using null to avoid a null pointer exception.
Card(aSuit,Number),(Card class)	Another way to create a card class. In this constructor the user can specify which suit and which number that we want. We used this class with the shuffler method to randomize a deck of cards.
getNumber(),(Card class)	This method was used to return the number

	of a specific card. We used this when we wanted to get the sum of the user's hand.
toString(),(Card class)	This method correctly formatted a string of the type of card (four of spades). We used this when we wanted to print out the user's hand.
Deck(), (Deck class)	The default constructor for the deck class. This constructor initializes a previously defined deck array with each possible card type.
shuffle(),(Deck class)	This method creates a random number and replaces the card at each index with a card at the random index. We used this after initializing the deck so that when the cards are dealt they are random.
deal(),(Deck class)	This method returns the card one index down from the top of the deck. We used this at the start of the game and when the user or CPU wanted to hit.
Player(),(player class)	The default constructor for the player class. We used this to test if a player, named user, is created, with an empty hand.
Player(name), (player class)	The other constructor for the player class. It creates a player with the specified name and an empty hand. This was used to create the cpu.
emptyhand(), (player class)	This method empties the player's hand by going through each index and setting the card in that spot to an empty card. We used this at the start of each run of the game to empty the player's hand so that the old cards don't affect the next round.
addCard(card), (player class)	This method adds another card to the player's hand. We used this method to add another card when the user selects 'H'. It also checks to see if the user will go over 21 when adding the card and will stop the user from adding more cards.
gethandsum(), (player class)	This method adds each index of the player's

	hand to give a final sum of the total hand. We use this when we check to see who wins the game.
printhand(showhand), (player class)	This method prints out the user hand or prints out hidden based on the boolean passed in for showhand. We use this when we want to show the user what cards he/she has so that they can decide if they want to hit or stay.

Overall Testing Report:

The overall idea of testing my code was to use the constructors and methods to see if the game behaved the way I wanted it to be. This way, I can add on to any of the methods if something goes wrong and enhance the game to do things at the specific time.