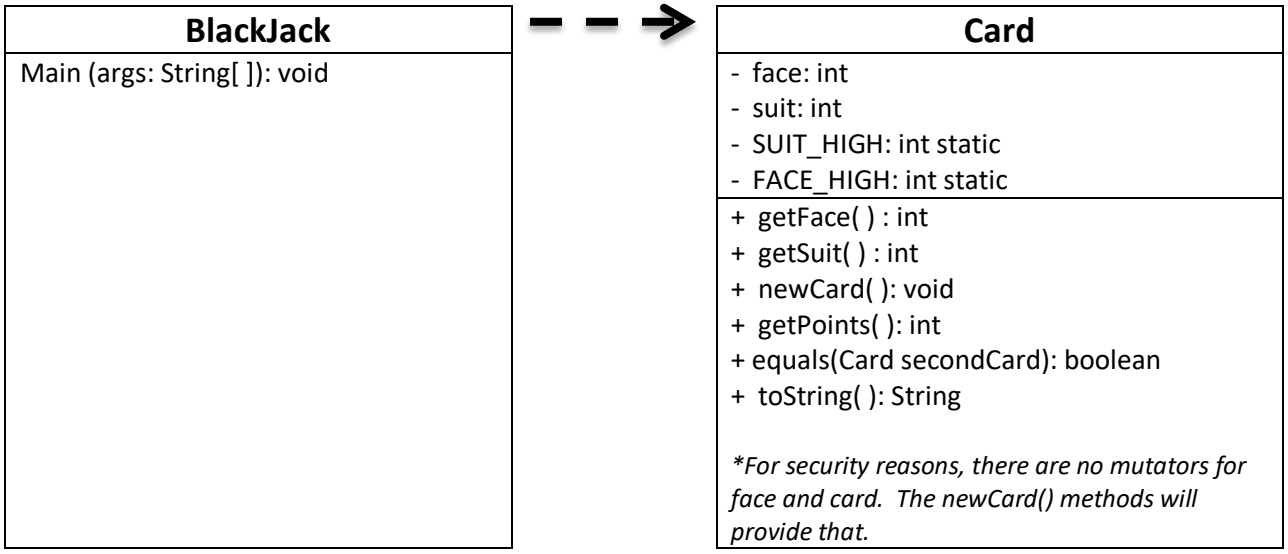
Java Programming – Card Class

Last week we began our study of user-defined Java classes. You were given a program (called a driver) which used a **Horse** class. You learned how to create a new class in Java by coding the **Horse** class.

This week we will reverse your role. You will be given Java class and you will write a program (driver) that uses that class.

Below is the UML diagram for a **Card** class that generates a card from a standard 52-card playing deck which could be used for a variety of card games. The UML diagram also illustrates a driver named **BlackJack**. **The Card class is provided** with this assignment. You will modify the **Card** class JUST a little bit and create the **Blackjack** driver. Open the **Card** class on your computer and review the code to see how it is structured. Afterwards, continuing reading below for an explanation of the **Card** class.



(+ represents the public modifier and – represents the private modifier)

Card Class

Instance Data

Refer to the UML class diagram above. You will see the class has 4 instance data items: face, suit, FACE_HIGH and SUIT_High

face: a number in the range 1-13 that represents the face of a card

1	=	Ace
2-10	=	2-10 respectively
11	=	Jack
12	=	Queen
13	=	King

suit: a number in the range 1-4

1	=	Hearts
2	=	Diamonds
3	=	Clubs
4	=	Spades

FACE_HIGH and **SUIT_HIGH** represent the high end of a range for the numbers to be generated. They are static constants -- only 1 copy of each constant is held in memory for all of the instances of the class rather than one copy for EACH card object. This saves memory.

Example: A face value of 12 and a suit value of 2 would represent the Queen of Diamonds

Required modification to the Card Class:

A **toString**( ) method is included which creates a simple display of each card. You are to rewrite this method so that the cards are displayed in a better format. Such as:

Ace of Spades
Queen of Hearts
4 of Diamonds
10 of Clubs

This is the only modification you are to make to the Card Class.

BlackJack Class (the driver program which contains a main() method)

Just a reminder of your goal:

Last week you created a new class (Horse) and a driver was provided for you to test your new Horse class. We will flip the process this week. A new class (Card) is provided by your instructor in Blackboard. You will modify the toString() method of the Card class and you will create a driver program that will use the Card class. Your driver program will simulate a modified version of BlackJack. DO NOT PANIC. Code this program a piece at a time (with testing) and you will be FINE.

Here are the Blackjack rules if you are not familiar with the game:
http://casinogambling.about.com/od/blackjack/a/Blackjack101.htm

To simplify the coding, your Blackjack game (program) will use the following modifications:

- You will deal 2 cards for the player and 2 cards for the dealer.
- The player and dealer will not draw additional cards for this assignment. They will simply play the hands as dealt (2 cards each).
- Aces will always count as 11. We will ignore the possibility of using an ace with a point value of 1.
- No splitting hands in the game. Both cards are played in a single hand.
- No betting will take place in this game. You are simply calculating the winner based upon points.
- If a person is dealt 2 Aces – they bust as the point value would be 22 -- which is greater than 21. Refer to the link above for card point values, if needed.

NOTE: For those who have taken other programming classes, you are not permitted to use arrays. We are using simple classes and variables. Use only concepts learned in the class to date. I am assessing specific features at this point. We will cover arrays immediately after Chapter 5.

Step 1: Dealing the player's cards

- "Deal" two cards for the player. This means you need to create two unique Card objects (instantiations).
- Make sure they are not the same card. **There is an equals() method in the Card class to help.** For example, if card1 and card2 are two Card objects:

```
card1.equals(card2)
```

would be true if they are the same card (suits and faces match) and false if they are not.

- If the cards are the same, generate a **new second** card for the player using the **newCard( )** method in the **Card** class. We know that there is a possibility of the same being dealt again. ***You can use a loop to check and re-deal if the cards are identical (think about why you would you use a loop instead of an if statement).***

- Display the cards for the player using the **toString()** method from the Card class.

- Calculate the point value for the player's hand using the **getPoints()** method from the **Card** class.

Step 2: Dealing the Dealer's cards

- Deal the first Dealer's card. Check to see if it is the same as the Player's first card. If so, re-deal the Dealer's first card.
- Check to see if the Dealer's first card is the same as the Player's second card. If so re-deal the Dealer's first card.
- Deal the second Dealer's card. Check to see if it is the same as the Dealer's first card. If so, re-deal the Dealer's second card.
- Check to see if the Dealer's second card is the same as the Player's first card. If so, re-deal the Dealer's second card.
- Check to see if the Dealer's second card is the same as the Player's second card. If so, redeal the Dealer's second card.
- Display the dealer's two card using the **toString()** method from the Card class.
- Calculate the point value for the dealer's hand using the **getPoints()** method from the **Card** class.

Step 3: Using if statements:

- Using the points for the player and the dealer, display the winner (the one with the most points).
- If the player or dealer goes over 21 (busts), indicate this.
- If there is a tie, indicate this.

Step 4: Using Loops

- Allow the game to be played again.

It will be easier if you do this in steps and test each step along the way.

Email your instructor if you have questions about the assignment. The main purpose of this assignment is to learn to instantiate objects from a class and to learn how to use methods from the class (like toString(), getPoints(), equals(), etc.)

Zip the **Card.java** and **BlackJack.java** files together for grading. No other files are needed.