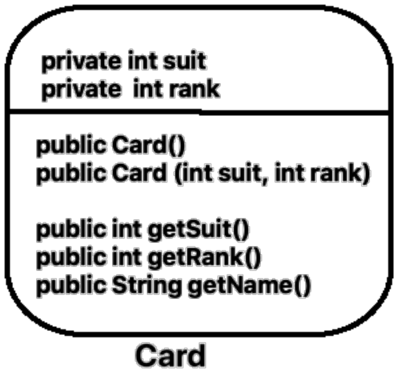
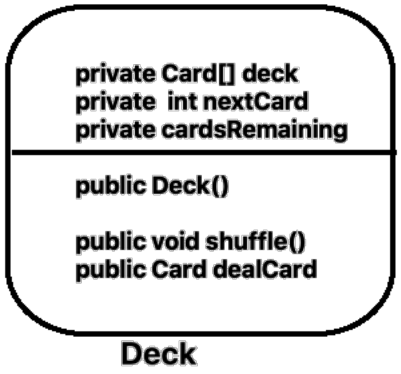
Class 5 Notes

From the last class : looked at two classes





A Card class

```
public class Card
{
        private int suit;          // 0 = Hearts, 1 = Diamonds, 2 = Clubs, 3 = Spades
        private int rank;          // 1 through 13 (Ace is 1, Jack 11, Queen 12, King 13)
        public Card()         // default constructor, sets Card to Ace of Hearts
        {
                suit = 0; // Ace
                rank = 1; //Hearts
         }
         public Card (int s, int r) // two argument constructor
        {
                suit = s;
                rank = r;
        }
        public int getSuit()
         {
                 return suit;
        }
        public int getRank()
        {
                return rank;
        }
        public String getName()
        {
                String name = "";
                switch(rank)
                {
                        case 1:  name = "Ace of "; break;
                        case 11: name = "Jack of "; break;
                        case 12: name = "Queen of "; break;
                        case 13: name = "King of "; break;
                        default :name = rank + " of "; // 2,3,4,5,6,7,8,9,10
                }
                switch(suit)
                {
                         case 0:  name = name + "Hearts"; break;
                        case 1:  name = name + "Diamonds"; break;
                        case 2:  name = name + "Clubs"; break;
                        case 3:  name = name + "Spades"; break;
                }
                return name;
        }
}
```

--------------------------------------- A deck class ---

A deck is modeled as a one dimensional array of Card objects:

Card[52] deck

index	Card	"name"	
0	Card(0,1)	**A of H**	**<-- Ordered deck** **(The third column is not part of the deck array but just gives the name of each card)** The suits are assigned arbitrary numbers: • 0 --> Hearts • 1 --> Diamonds • 2--> Clubs • 3--> Spades The ranks are numbers 1 (Ace)...13 (King) Assign each card in the deck a number (cardNum) from 0 to 51
1	Card(0,2)	**2 of H**	
2	Card(0,3)	**3 of H**	
3	Card(0,4)	**4 of H**	
4	Card(0,5)	**5 of H**	
5	Card(0,6)	**6 of H**	
6	Card(0,7)	**7 of H**	
7	Card(0,8)	**8 of H**	
8	Card(0,9)	**9 of H**	
9	Card(0,10)	**10 of H**	
10	Card(0,11)	**J of H**	
11	Card(0,12)	**Q of H**	
12	Card(0,13)	**K of H**	
13	Card(1,1)	**A of D**	
14	Card(1,2)	**2 of D**	
15	Card(1,3)	**3 of D**	
16	Card(1,4)	**4 of D**	

17	Card(1,5)	5 of D	• cardNum/13 gives the suit • cardNum%13 + 1 gives the rank Examples: Card 27 : suit = 27/13 =2 (Clubs) rank = 27%13+1 = 1 +1 = 2 --> 2 of Clubs Card 13 : suit = 13/13 = 1 (Diamonds); rank =13%13+1 = 0 + 1 = 1 --> Ace of Diamonds Card 43 : suit = 43/13 = 3 (Spades) rank= 43%13 +1 = 4 + 1 = 5 --> 5 of Spades
18	Card(1,6)	6 of D	
19	Card(1,7)	7 of D	
20	Card(1,8)	8 of D	
21	Card(1,9)	9 of D	
22	Card(1,10)	10 of D	
23	Card(1,11)	J of D	
24	Card(1,12)	Q of D	
25	Card(1,13)	K of D	
26	Card(2,1)	A of C	
27	Card(2,2)	2 of C	
28	Card(2,3)	3 of C	
29	Card(2,4)	4 of C	
30	Card(2,5)	5 of C	
31	Card(2,6)	6 of C	
32	Card(2,7)	7 of C	
33	Card(2,8)	8 of C	
34	Card(2,9)	9 of C	
35	Card(2,10)	10 of C	
36	Card(2,11)	J of C	
37	Card(2,12)	Q of C	
38	Card(2,13)	K of C	
39	Card(3,1)	A of S	
40	Card(3,2)	2 of S	
41	Card(3,3)	3 of S	
42	Card(3,4)	4 of S	
43	Card(3,5)	5 of S	
44	Card(3,6)	6 of S	
45	Card(3,7)	7 of S	
46	Card(3,8)	8 of S	
47	Card(3,9)	9 of S	
48	Card(3,10)	10 of S	
49	Card(3,11)	J of S	
50	Card(3,12)	Q of S	
51	Card(3,13)	K of S	

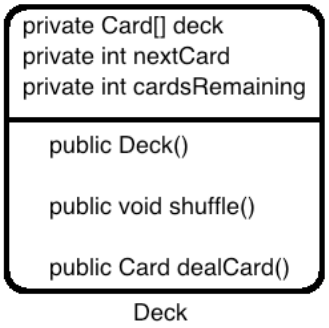
The array, deck, stores Card **REFERENCES.**
For example

deck[48] ----> Card(3,10)

When modeling a deck of cards we will need

1. an array to hold the cards --> **Card[] deck**
2. the position in the deck from which we deal a card -- > **int nextCard**
3. We must also keep track of how many cards remain in the deck after each card is dealt. If the deck is depleted we must shuffle again.

The methods --> shuffle the deck and deal the next card


private Card[] deck
private int nextCard
private int cardsRemaining

public Deck()

public void shuffle()

public Card dealCard()

Deck

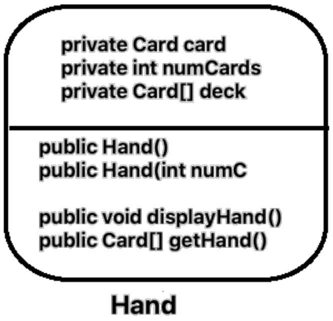
```
import java.util.*;
public class Deck
{
            private Card[] deck;
            private int cardsRemaining; // after a card is dealt
            private int nextCard; // to be dealt
            public Deck()  // default constructor
             {
                        deck = new Card[52];

                       // cardNum/13  is a number from 0 to 3→ the suit
                        // cardNum%13 + 1 is a number from 1 to 13 → the rank

                       for (int cardNum = 0; cardNum < 52; cardNum++)
                                   deck[i] = new Card(cardNum/13,   cardNum%13+1);  //Card(suit, rank)

                       cardsRemaining = 52;
                       nextCard = 0;
                       shuffle();
            }
             public void shuffle()
             {
                        Random r = new Random ();
                        for (int I = 0; I < 52; i++)
                        {
                                    int randomPlace = r.nextInt(52); // find a random place in the deck
                                    //swap deck[i] with deck[randomPlace]
                                    Card temp = deck[i];
                                    deck[i] = deck[randomPlace];
                                   deck[randomPlace] = temp;
                       }
                       cardsRemaining = 52;
                       nextCard = 0;
            }
```

```
     public Card dealCard()
    {
        // returns the card at the top of the deck (nextCard)
         if (cardsRemaining == 0)
        {
             System.out.println(“Deck was re-shuffled”);
            shuffle();
         }
        Card c = deck[nextCard];
         nextCard++;
        cardsRemaining--;
        return c;
    }
}
```

A Hand object is an array of and number of Card objects.	 private Card card private int numCards private Card[] deck public Hand() public Hand(int numC public void displayHand() public Card[] getHand() Hand

```
Public  class Hand
{
  // always dealt from a shuffled deck
  private Card[] hand ;
  private int numCards;
  private Deck deck;
  public Hand()  // default constructor sets hand to 5 cards
  {
   numCards = 5;
   deck = new Deck();
   hand = new Card[numCards];
   for (int I = 0; I < numCards; i++)
     hand[i] = deck.dealCard();
  }
  public Hand (int numC)  // one argument constructor
  {
   numCards = numC;
   deck = new Deck();
   hand = new Card[numCards];
    for (int I = 0; I < numCards; i++)
      hand[i] = deck.dealCard();
  }
  public void displayHand()  // prints the hand
  {
   for (int I =  0; I <numCards; i++)
     System.out.println(hand[i].getName());
   System.out.println();
  }
  public  Card[] getHand()
 {
    Return hand;
 }
}
```

```
import java.util.*;
public class TestCards
{
 public static void main(String[] args)
 {
  Scanner input = new Scanner(System.in);

  System.out.print(“How may cards in the hand: “);
  int numCards = input.nextInt();
  while (numCards >0)
  {
   Hand hand = new Hand(numCards);

   Card[] cards = hand.getHand();  // an array of the cards in the hand
   for(int I = 0; I < numCards; i++)
      System.out.println(cards[i].getName());
   // hand.displayHand(); will do the same thing

   System.out.print(“How may cards in the hand, enter 0 to exit: “);
   numCards = input.nextInt();
   hand = new Hand(numCards);
  }
 }
}
```

Output:

```
       > java TestCards

       How may cards in the hand:  5
       10 of Hearts
       8 of Diamonds
       5 of Diamonds
       10 of Spades
       3 of Diamonds

       How may cards in the hand, enter 0 to exit: 4
       King of Diamonds
       4 of Hearts
       10 of Hearts
       Ace of Hearts

       How may cards in the hand, enter 0 to exit:   0
```

The keyword static
What does it mean?
When should we use it?

Static METHODS

When we use a String methods we must first create (or instantiate) a String object.

Example

```
String s = new String("Dopey");  // created a string   object
Int num = s.length();
```

We called length() via an object, s. A method of the String class can be called only via an object.

Last semester you used the Math class.
Some of the methods of the Math class are:

- random()
- sqrt(double x) // square root
- abs(int x) // absolute value
- sin(double x) //sin(x)

There are many more.

You do not need an object to use these methods.
You can call them with the name of the class:

```
double rn = Math.random():
double x = Math.sqrt(23467.5) ;
```

If you look at Java's documentation, all the methods of the Math class are labeled *static*.

A static method

- may be called whether or not an object of the class exists.
- can be called using the name of the **class**. For example **Math**.random()
- cannot call an instance (non-static) method except via an object.

First, let's look at the third bullet point:

A static method cannot call an instance method (non-static) except via an object.

```
public class Sum  // very simple class
{
    public int add(int a, int b)  // not static
    {
        return a+b;
    }
}
```

public class Sum // very simple class
{
public int add(int a, int b) **// not static**
{
return a+b;
}
 	
// notice main(…) is static
 	
public **static** void main(String[] args)
{
int sum = add(3,4);
System.out.println("The sum is "+ sum);
}
 	
}
	public class Sum // very simple class
	{
	public int add(int a, int b) // not static
	{
	return a+b;
	}
	
	public static void main(String[] args)
	{
	Sum s = new Sum(); // make objec
	int sum = **s.**add(3,4);
	System.out.println("The sum is "+ sum);
	}
	
	}

The compiler complained: 1 error found: Error: Sum.java:10: non-static method add(int,int) cannot be referenced from a static context	Runs now and output is The sum is 7

A class can have both Static and non-static (instance) methods

```
public class Demo
{
 public void notStatic()
 {
   System.out.println( " I am not static");
 }

 public static void yesStatic()
 {
   System.out.println( " I am  static");
 }

 public static void main(String[] args)
 {
  Demo d = new Demo();
  d.notStatic();  // needs an object
  yesStatic();//does not need an object

 }
}
```

If you use a static method in any other class except the class where it is defined, call the method with the class name. For example Math.random()

```
public class UseDemo  // a separate class
{
   public static void main(String [] args)
   {
        Demo d = new Demo();
         d.notStatic();
        Demo.yesStatic();  // called with the class name
    }
}
```

Question: Why is main(…) static?

Static methods are usually used as UTILITY methods : Math.Random() or Math.sqrt(..).
Static methods do not use the class variables (attributes)

Example:

```
import java.util.*;
public class Bubblesort
{
 public static void sort(int[] x, int n)
      // n is the number of data
 {
        boolean swap = true;
        int pass = 1;

    while (pass <= n-1 && swap)
    // stop if no swaps made
    {
     swap = false;
     for (int i = 0; i < n - pass;  i++)
     {
       if (x[i] > x[i+1])
      {
         // swap x[i] and x[i+1]
         int temp = x[i];
         x[i] = x[i+1];
         x[i+1] = temp;

         swap = true;
       }
     }
     pass++;
    }
  }
}
```

```
// A  separate class that uses Bubblesoort

public class SortDemo
{
 public static void main(String[] args)
 {
  int[] list = {2,5,4,6,8,1,0,6,3,12,54,11,31};

  Bubblesort.sort(list, list.length);
  // calls with class name

  for (int i = 0; i < list.length; i++)
   System.out.println(list[i]);
 }
}

// sort is static and is called with the class
// name just as Math.random()
// is called with the class name

// Bubblesort.sort() is a utility method
```

Again, static methods do not require an object.

Variables can be labeled static but in general we will have little or no use for static variables, except possibly as a static constant. A constant is a variable that cannot be changed. It has the labeled with the keyword final.

Example: Here is a utility class with two static methods and a **static constant**
The class contains nothing else except three constants.

```
public class Constants
{
 public static final double PI = 3.14159;    // final means it cannot be changed
 public static final e = 2.71828
 public static final Root2 = 1.4142
}
```

You can use the static constants in any other class as Circle.PI

Example:

```
public class Stupid
{
public static void main(String[] args)
{
    System.out.println(“The area of a circle of radius 34 is “ + Constants.PI* 34*34);
    // that is  area = $\pi r^2$

    // here is e$^2$
     System.out.println(“The area of a circle of radius 34 is “ +Constants.e * Constants.e *);
}
}
```

Like static methods, any class can access static constants with the class name.

BTW The Math class defines Math.Pi and Math.E. This program prints their values:

```
        public class MathConstants
        {
           public static void main(String[] args)
           {
             System.out.println("Pi = "+ Math.PI);
             System.out.println("e = "+ Math.E);
           }
        }
```

```
Pi = 3.141592653589793
e = 2.718281828459045
```