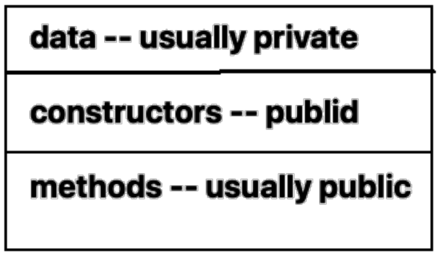
Designing your own classes

The structure of a class:



data -- usually private
constructors -- publid
methods -- usually public

Private

The data in a class is usually labeled ***private***.
The keyword *private* means that only the methods of the class have direct access to the data.If data is public, any method, either in the class or not can access the data.

Example

```
        public class Student
        {
                private String name;
                private double gpa;
               .......
         }
```

- Private data is hidden inside the the “capsule.”
- The class methods manipulate the data.
- A class method can manipulate the class variables directly. You do not have to pass them as parameters.

Constructors

The “new” operator creates or INSTANTIATES an object
Example:

```
        Card c = new Card();
        Scanner input = new Scanner(System,in)
        String name = new String(“Mary”)
```

When an object is created a special method is automatically called.

This special method is the ***constructor*.**

- The constructor is used to initialize the variable of the class but can do other things as well.
- The name of the constructor is the same as the name of the class.
- There is no return value not even void

Constructors

- The name of the constructor is the name of the class
- A constructor does not have a return value, not even void
- The constructor is called using the word "new"
- The *default constructor* has no parameters or arguments: public Rectangle()
- Other constructors can accept arguments: public Rectangle (int len, int wid)
- If you define no constructors at all then Java provides a default constructor.
- If you define a constructor with arguments then Java will not provide a default constructor. Once you make any constructor, Java stays out of the constructor business.

```
public class Rectangle
{
        private int length, width;
        public Rectangle()    // default constructor
        {
                length = 1;
                width = 1;
        }
        public Rectangle(int len, int wid)    // two- argument constructor
        {
                length = len;
                width = wid;
        }
       // methods of the class such as area() go here
}
```

Create or instantiate two Rectangle objects:

```
public static void main(String[] args)
{
        Rectangle r = new Rectangle(3,4);      //OK
        //This creates or instantiates a  3 x 4 Rectangle object

        Rectangle s = new Rectangle() ;   // call to the default constructor
        //Creates a 1 x 1 Rectangle object
}
```

You can put any statement inside a constructor and when the constructor is called all statements will be executed

```
public Rectangle() // a DEFAULT  constructor with  print statements
{
        length = 1;
        width = 1;
        System.out.println(“I am your default constructor “)
        System.out.println("I set length and width equal to 1");
}
```

Or

```
public Rectangle() // a DEFAULT  constructor that takes input from the
user
{
      Scanner input = new Scanner(System.in);
       System.out.println(“Enter length and width”);
        length = input.nextInt();
       width = input.nextInt();
}
```

Here is a call to the default constructor

```
public static void main(String[] args)
{
      Rectangle r = new Rectangle();      // calls default constructor
}
```

Output:

```
I am the default constructor
I set length and width equal to 1
```

// No Constructors defined – So Java provides a default constructor

```
public class Rectangle
{
   private int length;
   private int width;


  public int getLength()
  {
    return length;
  }


  public int getWidth()
  {
    return width;
  }

  public static void main(String[] args)
  {
    Rectangle r = new Rectangle();  // calls the Java-supplied default constructor
                                    // length and width are set to 0

    System.out.println("Length is  "+ r.getLength());
    System.out.println("Width is  "+ r.getWidth());
  }
}
```

Output:

```
        Length is  0
        Width is  0
```

- **Here a one-argument constructor is defined but not a default constructor**
- **Java does not provide a default constructor in this case**
- **So there is no default constructor and a call to a default constructor will be an error**

```
public class Rectangle
{

  private int length;
  private int width;

  public Rectangle(int n)  // One argument constructor
  {
    length = width = n;

  }

  public static void main(String[] args)
  {

   Rectangle2 r = new Rectangle2();       // attempt to call the default – error
   Rectangle2 s = new Rectangle2(5);     // this is OK
  }
}
```

Here is the syntax error message you get from trying to call the non-existent default constructor:

```
File: C:\ Rectangle2.java  [line: 16]
Error: The constructor Rectangle2() is undefined
```

Methods

You should no longer use the keyword static in a method heading. Soon, we will look more closely at static methods and what static means and when we should use it.

Example

```
 import java.util.*; // for Random class
 public class Dice
 {

   private int numDice;      // how many dice

   // Constructors

   public Dice()  // default constructor
   {
     numDice = 1;
   }

   public Dice (int n)  // one-argument constructor
   {
     numDice= n;
   }

    public int getNumDice()  // getter
    {
          return numDice;
    }

   Public void setNumDice(int n)  // setter
    {
          numDice = n;
    }



    public int rollDice()
   // rolls numDice dice and returns the sum of the spots
   {
      Random r = new Random();
     int sum = 0;

     for (int i = 1; i <= numDice; i++)
       sum = sum + r.nextInt(6) + 1; // random int 1..6

         return sum;
   }


}
```

```
// A separate class/program that TESTS the Dice class

import java.util.*;

public class TestNumDice
{
  public static void main(String[] args)
  {
    Scanner input = new Scanner(System.in);
    String answer;


    System.out.print("How many dice would you like to toss ?");
    int n = input.nextInt();

       Dice d = new Dice(n); // calls one-argument constructor

    System.out.println("Five rolls of "+ d.getNumDice() +" dice: ");
    for (int i = 1; i <= 5; i++)
           System.out.println(d.rollDice());
   }
}
```

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 a one dimensional array of Card objects:

Card[52] deck

index	Card	"name"
0	Card(0,1)	**A of H**
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**
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**

<-- 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

- 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

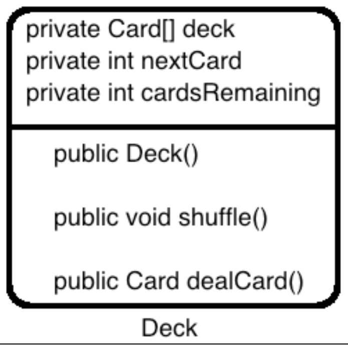
34	Card(2,9)	9 of C	
35	Card(2,10)	10 of C	
36	Card(2,11)	J of C	
37	Card(2,12)	Q of C	
38	Card(2,13)	K of C	
39	Card(3,1)	A of S	
40	Card(3,2)	2 of S	
41	Card(3,3)	3 of S	
42	Card(3,4)	4 of S	
43	Card(3,5)	5 of S	
44	Card(3,6)	6 of S	
45	Card(3,7)	7 of S	
46	Card(3,8)	8 of S	
47	Card(3,9)	9 of S	
48	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 stores Card REFERENCES

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



```
import java.util.*;
public class Deck
{
        private Card[] deck;
        private int cardsRemaining; // afte a card is dealt
        private int nextCard; // to be dealt

        public Deck()
         {
                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.

```
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();
   }
 }
```

```
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);
      hand.displayHand();
      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
```