DATA HIDING : Private & Public.

WHEN WE CREATE A NEW DATA TYPE THROUGH THE class CONSTRUCT IN JAVA, WE OFTEN THINK OF THIS NEW TYPE AS BEING A BLACK BOX.

THE USER OF THE NEW DATA TYPE CAN PERFORM ONLY THE OPERATIONS INTENDED BY THE DESIGNER, AND NO OTHERS.

THUS IT SHOULD NOT BE POSSIBLE FOR THE USER (i.e. THE PROGRAMMER USING THE TYPE) TO ARBITRARILY CHANGE FIELD VALUES.

e.g.

```
class myClass{
    int happy;
    int sad;
}
```

Then in some method in some other class

```
myClass a = new myClass();

a.happy = 1 ;
a.sad = 5 ;
a.happy = 3 ;
   :
```

As a Rule, member variables (i.e. fields) are always given the access level private.

```
class myClass {
    Private int happy;
    private int sad;

    myClass(int h, int s) {
        happy = h;
        sad = s;
    }

    void setHappy(int h) { happy = h; }
    void setSad(int s) { sad = s; }
    int getHappy() { return happy; }
    int getSad() { return sad; }
}
```

THEN

```
myClass a = new myClass(5,6);
System.out.println(a.getSad());
a.setHally(8);
   :
   :
```

NOTE: BY WAY OF CORRECTION! WHERE ONCE SPECIFIED ANY CONSTRUCTOR FOR A CLASS, THE DEFAULT NO-ARGUMENT CONSTRUCTOR GOES AWAY.

THUS

```
myClass a = new myClass();
```

WILL NOW GIVE A SYNTAX ERROR.

THE NEXT EXAMPLE IS A LITTLE LESS CONTRIVED.

```
// Counter.java
// A wrap-around counter

class Counter{

   // Member variables
   private int value;  //will range from 0 to (limit - 1)
   private int limit;

   // Constructor
   Counter(int limit){
      this.limit = limit;
      this.value = 0;
   }

   // Methods
   void reset(){ value = 0;}
   int getValue(){ return value; }
   void click(){ value = (value + 1)%limit; }
}



// CounterTest.java

class CounterTest{

   public static void main( String[] args ){

      Counter a = new Counter(10);
      Counter b = new Counter(100);

      a.click();
      a.click();
      System.out.println(a.getValue());
      for(int i=0; i<12; i++){
         a.click();
      }
      System.out.println(a.getValue());

      for(int i=0; i<123; i++){
         b.click();
      }
      System.out.println(b.getValue());
      b.reset();
      System.out.println(b.getValue());
   }
}



// output:
// 2
// 4
// 23
// 0
```

NOTE THAT OPERATIONS like

```
a.value = 50;
b.limit = 35;
```

GIVE A SYNTAX ERROR NOW. SINCE THE DATA FIELDS ARE NOW PRIVATE, THEY CANNOT BE ACCESSED FROM OUTSIDE class Counter.

THIS is THE MEANING OF <u>DATA HIDING</u>. THE USER OF A NEW TYPE CAN ONLY INTERACT WITH THAT TYPE THROUGH Public (OR PACKAGE) methods.

THUS THE USER is PREVENTED FROM KNOWING ABOUT, AND HENCE is ABSOLVED OF RESPONSIBILITY OF KNOWING ABOUT THE INSIDE OF THE 'BLACK BOX'.

THE ULTIMATE PURPOSE OF THIS is TO REDUCE COMPLEXITY OF VERY LARGE PROGRAMS.

NOW THAT MOST OF THE BASIC FEATURES OF CLASSES IN JAVA HAVE BEEN DISCUSSED, WE CAN BUILD MORE & USEFUL (& LESS CONTRIVED) EXAMPLES.

EACH PRIMITIVE DATA TYPE IN JAVA HAS A CORRESPONDING REFERENCE TYPE:

int	Integer
long	Long
short	Short
byte	Byte
char	Char
float	Float
double	Double
boolean	Boolean

EACH REFERENCE TYPE ABOVE CONTAINS JUST ONE DATA FIELD OF THE CORRESPONDING PRIMITIVE TYPE.

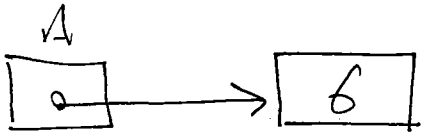
THESE CLASSES ARE SOMETIMES CALLED WRAPPER CLASSES SINCE THEY WRAP A SINGLE PRIMITIVE

TYPE IN AN OBJECT.

For instance

```
Integer A = new Integer(6);
```

CREATES



In addition to providing this object wrapping, the Integer class provides methods for String to int and int to String conversions, as well as other constants and methods useful when dealing with an int.

See Documentation for Fields, Constructors, and methods. In particular

compareTo()	MAX_VALUE
equals()	MIN_VALUE
doubleValue()	
toString()	
valueOf()	

See Also Documentation
For Type Double

compareTo()
equals()
toString()
valueOf()
etc...

Java Provides NO class To Represent Rational numbers, i.e. fractions.

We will include some methods similar to those found in the Integer and Double classes, As well As methods for doing BASIC ARITHMETIC OF RATIONAL NUMBERS, i.e. Add, Subtract, Multiply, divide, And riciprocate.

```
//   Rational.java
//   Represents rational numbers as a pair of ints

import java.util.StringTokenizer;

class Rational{

   // Data fields
   ************************************************************************/
   private int numerator;
   private int denominator;

   // Constructors
   ************************************************************************/

   Rational(int n, int d){
      if( d==0 ){
         throw new ArithmeticException("Cannot create Rational with zero
         denominator");
      }
      if( d<0 ){
         d = -d;
         n = -n;
      }
      int g = gcd(Math.abs(n), d);
      numerator = n/g;
      denominator = d/g;
   }

   Rational(String s){
      StringTokenizer t = new StringTokenizer(s, "/");
      int n = Integer.valueOf(t.nextToken());
      int d = Integer.valueOf(t.nextToken());
      if( d==0 ){
         throw new ArithmeticException("Cannot create Rational with zero
         denominator");
      }
      if( d<0 ){
         d = -d;
         n = -n;
      }
      int g = gcd(Math.abs(n), d);
      numerator = n/g;
      denominator = d/g;
   }

   // Private methods
   ************************************************************************/

   // gcd()
   // Private function for computing the Greatest Common Divisor of two numbers.
   // Precondition: arguments are non-negative, and at most one argument is zero.
   private int gcd(int a, int b){
      int r, temp;
      if( a<b ){ temp = a; a = b; b = temp; }
      while( b!=0 ){ r = a%b; a = b; b = r; }
      return a;
   }

   // Public methods
   ************************************************************************/

   // toString()
   public String toString(){
      return( String.valueOf(numerator) + "/" + String.valueOf(denominator) );
   }

   // equals()
```

```
    public boolean equals(Object x){
       Rational r = (Rational) x;
       return( this.numerator==r.numerator && this.denominator==r.denominator );
    }

    // compareTo()
    int compareTo(Rational r){
       int comp = this.numerator*r.denominator - this.denominator*r.numerator;
       return( comp>0 ? 1 : comp==0 ? 0 : -1 );
    }

    // doubleValue()
    double doubleValue(){
       return( this.numerator/(double)this.denominator );
    }

    // valueOf()
    static Rational valueOf(int n, int d){
       return( new Rational(n, d) );
    }

    // valueOf()
    static Rational valueOf(String s){
       return( new Rational(s) );
    }

    // add()
    Rational add(Rational r){
       int n = this.numerator*r.denominator + this.denominator*r.numerator;
       int d = this.denominator*r.denominator;
       return( new Rational(n, d) );
    }

    // sub()
    Rational sub(Rational r){
       int n = this.numerator*r.denominator - this.denominator*r.numerator;
       int d = this.denominator*r.denominator;
       return( new Rational(n, d) );
    }

    // mult()
    Rational mult(Rational r){
       int n = this.numerator*r.numerator;
       int d = this.denominator*r.denominator;
       return( new Rational(n, d) );
    }

    // recip()
    Rational recip(){
       int n = this.denominator;
       int d = this.numerator;
       if( d==0 ){
          throw new ArithmeticException("Cannot create reciprocal of zero");
       }
       return( new Rational(n, d) );
    }

    // divide()
    Rational divide(Rational r){
       if( r.numerator==0 ){
          throw new ArithmeticException("Cannot divide by zero");
       }
       int n = this.numerator*r.denominator;
       int d = this.denominator*r.numerator;
       return( new Rational(n, d) );
    }
}
```

```
//   RationalTest.java
//   Tests the Rational data type

class RationalTest{

   public static void main(String[] args){
      Rational a = new Rational("6/4");
      Rational b = new Rational(8, 10);
      System.out.println(a);
      System.out.println(b);
      System.out.println(a.add(b));
      System.out.println(a.sub(b));
      System.out.println(a.mult(b));
      System.out.println(a.divide(b));
      System.out.println(a.recip());
      Rational c = new Rational(12, 8);
      System.out.println(a.equals(b));
      System.out.println(a.equals(c));
      System.out.println(a.compareTo(b));
      System.out.println(b.compareTo(c));
      System.out.println(a.compareTo(c));
      System.out.println(c.doubleValue());
      System.out.println(Rational.valueOf("9/-2"));
      System.out.println(Rational.valueOf(9,-2));
      try{
         Rational d = new Rational(1,0);
      }
      catch(ArithmeticException e){
         System.out.println(e.getMessage());
      }
      try{
         Rational d = Rational.valueOf(0,1);
         System.out.println(d.recip());
      }
      catch(ArithmeticException e){
         System.out.println(e.getMessage());
      }
      try{
         System.out.println(a.divide(Rational.valueOf(0,1)));
      }
      catch(ArithmeticException e){
         System.out.println(e.getMessage());
      }
   }

}

/*
Output:
3/2
4/5
23/10
7/10
6/5
15/8
2/3
false
true
1
-1
0
1.5
-9/2
-9/2
Cannot create Rational with zero denominator
Cannot create reciprocal of zero
Cannot divide by zero
*/
```