Objects and Methods

called an Object

```
public class IntPair {
  private int i1, i2;
  /**/
  public IntPair ( int i, int j ) {
    i1=i;  i2=j;
  }
  /**/
  public int getI1 ( ) {
    return i1;
  }
  /**/
  public int getI2 ( ) {
    return i2;
  }
}
```

`private` fields of the class. They can be accessed only from within `IntPair`

The Constructor initializes the fields

`public` non-`static` methods can access the fields. `static` methods cannot. These are called accessor methods, because they give outside access to the fields. `i1` and `i2` cannot be modified outside of `IntPair`. This is called data encapsulation.

● Class Work. Type in and run `TestIntPair`.

called an instance of `IntPair`. A program can contain more than one instance of an Object. A program contains only one instance of a `static` method.

calls the Constructor

non-`static` class methods are called with an instance of the Object

```
public class TestIntPair {
  public static void main ( String [] arg ) {
    IntPair ip=new IntPair (4,5);
    int a=ip.getI1();
    int b=ip.getI2();
    System.out.println();
    System.out.println("a = " + Integer.toString(a) );
    System.out.println("b = " + Integer.toString(b) );
    System.out.println("ip = " + ip);
    /**/
    return;
  }
  /**/
  public TestIntPair ( ) { }
}
```

address of `ip`

`i1`

`i2`

Class Work. Type in and run `PrimeTest`.

```
public class PrimeTest {
  public static void main ( String [] arg ) {
    int [] n=new int [2];
    n[0]=843;  n[1]=2017;
    Prime p;
    /**/
    for ( int i=0; i<2; ++i ) {
      p=new Prime(n[i]);
      System.out.println();
      if ( p.isPrime() ) {
        System.out.println(Integer.toString(n[i]) + " is prime.");
      }
      else {
        int f=p.getFactor();
        System.out.println(Integer.toString(n[i]) + " is not prime.");
        System.out.println("factor = " + Integer.toString(f));
      }
    }
    /**/
  }
}
```

```
public class Prime {
  /*
   *  private fields
   */
  private int n;
  private boolean prime;
  private int factor;
  /*
   *  Constructor --
   *  initialize the fields
   */
  public Prime (int num) {
    n=num;
    prime=false;
    factor=0;
    /**/
    calcPrime();
  }
  /**/
  public boolean isPrime( ) {
    return prime;
  }
  /**/
  public int getFactor() {
    return factor;
  }
  /*
   *  Calculates if n is prime or not --
   *  if not, then the smallest prime factor of n is calculated.
   */
  private void calcPrime ( ) {
    /**/
    double nD, maxN, test;
    int max;
    /**/
    nD=(double)n;
    maxN=Math.sqrt(nD);
    max=iPart(maxN);
    for ( int i=2; i<=max; ++i ) {
      test=nD/(double)i;
      if ( fPart(test) == 0.0 ) {
        factor=i;
        prime=false;
        return;
      }
    }
    prime=true;
    /**/
    return;
  }
  /*
   * returns the integer part of x
   */
  private int iPart ( double x ) {
    return (int)(Math.floor(x));
  }
  /*
   *  returns the fractional part of x
   */
  private double fPart ( double x ) {
    return x-Math.floor(x);
  }
}
```

public accessor methods. called with an instance of Prime

private methods. can only be called from inside class Prime

A cast
Makes the int into a double

A cast
Makes the double into an int (the fractional part is thrown out if there is one)