How does Java pass arguments into a method? The short answer is “by value,” but the details are subtle. Recall that in C, C++, and other languages you can choose to pass either a “value” or to “pass a pointer to a value.” The first option is called “pass-by-value.” The second option is called “pass-by-reference.” In Java, primitive variables are always passed by value. In other words, a copy of the variable is passed into the method. In other words, if you make a change to the variable within the method, then that change is not visible anywhere else. For example, suppose we have

```java
double d = 2.0;
changeMe(d);
System.out.println(d);  //prints 2.0
```

The printed value will be 2.0 no matter what happens inside the method changeMe.

```java
public void changeMe(double d)
{
    //this has no effect on d outside of this method!
    d = 345.0;
}
```

On the other hand, Java objects appear to be passed by reference, but it’s a ruse. All Java objects are pointers (we just don’t have to use the annoying pointer symbol * when we create them), so the pointer gets passed by value (oooh, subtle). For example, suppose I create an object called Car that has methods called setSpeed() and getSpeed().

```java
Car ferrari = new Car();
ferrari.setSpeed(65.0);
changeParameters(ferrari);
System.out.println("The speed = "+ferrari.getSpeed());
```

So what gets printed out? That depends on what happens inside the changeParameters method. Suppose

```java
public void changeParameters(Car c)
{
    //changes the car’s speed outside this method!
    c.setSpeed(200.0);
}
```

Then the code prints out 200.0. Why? Because the ferrari was a pointer to the object Car. The ferrari got passed by value into the method. So a copy of the pointer got passed
into the method. But a copy of a pointer still points to the same object, the ferrari! So changes to the ferrari in the method are felt outside the method as well.

Here is one more subtlety. Suppose we have

```java
public void changeCar(Car c)
{
    Car chevy = new Car();

    //does not affect c outside this method
    c = chevy;
}
```

And now we call

```java
Car ferrari = new Car();
changeCar(ferrari);
```

Is the original ferrari object now a chevy (outside of the method)? No! Inside of changeCar we changed the original pointer to be a new pointer. But the original pointer was passed by value, so changing the pointer within the method has no impact on the original pointer outside of the method.

Phew!

For those who doubt ☺, try the following code. Recall that arrays are pointers in Java. First let’s show that either a pointer or a copy of the pointer must be passed into a method. We’ll resolve which one in a minute. (The next example shows that it is actually a copy.)

```java
public class Example
{
    public static void main(String[] args)
    {
        //"test" is a pointer.
        int[] test = {1, 2, 3, 4};

        //The original hex value of the pointer
        System.out.println(test);

        //Let’s pass test’s pointer into a method.
        //The method changes some array values.
        Example e = new Example();
        e.modifyArray(test);

        //This prints out “9 2 3 4”, not “1 2 3 4”
        for(int i=0; i<test.length; i++)
```

```
public void modifyArray(int[] a)
{
    a[0] = 9;

    //Because “a” is a pointer to the array
    //called “test”, this changes the first
    //element of “test”.
}

See? We passed in a pointer to the array, and then used that pointer to modify an element
in the array. (But it was really a copy of the pointer! See below.)

The following code shows that a copy of the array’s pointer is passed into a method.

    public class Example
    {
        public static void main(String[] args)
        {
            //”test” is a pointer.
            int[] test = {1, 2, 3, 4};

            //The original hex value of the pointer
            System.out.println(“test’s pointer: “+test);

            //Let’s pass test’s pointer into a method.
            //The method attempts to change test’s
            //pointer.
            Example e = new Example();
            e.modifyPointer(test);

            //The method failed to change the pointer!
            System.out.println(“test’s pointer: “+);

            //This prints out “1 2 3 4”, not “9 9 9 9”
            for(int i=0; i<test.length; i++)
            {
                System.out.print(test[i]+” “);
            }
        }
    }

    public void modifyPointer(int[] a)
{  
  //“b” is a new pointer.
  int[] b = {9, 9, 9, 9};

  //The hex value of b’s pointer
  System.out.println("b’s pointer: "+b);

  //“a” now points to the same thing as “b”.
  a = b;

  //The hex value of a’s pointer
  System.out.println("a’s pointer: "+a);

  //But this affects nothing outside the
  //method. “test” does not point to “b”!
  //Only “a” points to “b”.
}

If the actual pointer had been passed into the method, then “test” should have been
changed to point to “b”. That didn’t happen!