Passing Primitive Data Type Arguments

Primitive arguments, such as an int or a double, are passed into methods by value. This means that any changes to the values of the parameters exist only within the scope of the method. When the method returns, the parameters are gone and any changes to them are lost. Here is an example:

```java
public class PassPrimitiveByValue {
    public static void main(String[] args) {
        int x = 3;
        // invoke passMethod() with
        // x as argument
        passMethod(x);

        // print x to see if its
        // value has changed
        System.out.println("After invoking passMethod, x = " + x);
    }

    // change parameter in passMethod()
    public static void passMethod(int p) {
        p = 10;
    }
}
```

When you run this program, the output is:

After invoking passMethod, x = 3

Passing Reference Data Type Arguments

Reference data type parameters, such as objects, are also passed into methods by value. This means that when the method returns, the passed-in reference still references the same object as before. However, the values of the object’s fields can be changed in the method, if they have the proper access level.

For example, consider a method in an arbitrary class that moves Circle objects:

```java
public void moveCircle(Circle circle, int deltaX, int deltaY) {
    // code to move origin of circle to x+deltaX, y+deltaY
    circle.setX(circle.getX() + deltaX);
    circle.setY(circle.getY() + deltaY);

    // code to assign a new reference to circle
    circle = new Circle(0, 0);
}
```

Let the method be invoked with these arguments:

moveCircle(myCircle, 23, 56)
Inside the method, circle initially refers to myCircle. The method changes the x and y coordinates of the object that circle references (i.e., myCircle) by 23 and 56, respectively. These changes will persist when the method returns. Then circle is assigned a reference to a new Circle object with \( x = y = 0 \). This reassignment has no permanence, however, because the reference was passed in by value and cannot change. Within the method, the object pointed to by circle has changed, but, when the method returns, myCircle still references the same Circle object as before the method was called.