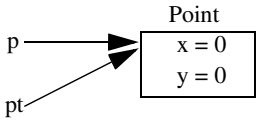
fyPoint are not reflected in main because there are two different objects. Right? Wrong.

Actually, modifyPoint is working with a copy of a *reference* to the Point object, not a copy of the Point object. Remember that p is an object reference and that Java passes parameters by value. More specifically, Java passes object *references* by value. When p is passed from main to modifyPoint, a copy of the value of p, the reference, is passed. Therefore, the modifyPoint method is working with the same object but through the alias, pt. After entering modifyPoint, but before the execution of the code at //1, the object looks like this:



Therefore, after the code at //1 is executed, the Point object has changed to (5, 5). What if you want to disallow changes to the Point object in methods such as modifyPoint? There are two solutions for this:

- Pass a clone of the Point object to the modifyPoint method. See PRAXES 64 and 66 for more details on cloning.
- Make the Point object immutable. See PRAXIS 65 for a discussion of immutable object techniques.

PRAXIS 2: Use `final` for constant data and constant object references

Many languages offer a notion of *constant data*, which is data that does not change and cannot be changed. Java provides the keyword final to specify constant data. For example:

```
class Test
{
  static final int someInt = 10;
  //...
}
```

This code declares a static class variable named someInt and sets its value to 10.