Java's reflection mechanism

Wikipedia ---en.wikipedia.org/wiki/Reflection_(computer_programming)--- will tell you that:

> Reflection is the ability of a computer program to examine, introspect,
> and modify its own structure and behavior at runtime.

A Java program can do that. It can get a list of fields in a class and investigate their properties. It can change the value of a private field of an object from outside the class. Information about methods, constructors, and annotations on them is available through the reflection mechanism.

There are many reasons to use Java's reflection mechanism. We once used reflection in writing a grading program for an assignment. A field in a class that students submitted was private, but the only way to write a suitable grading program was to get at that field and change it. Reflection let us do that.

We also suspect that the JUnit 4 program that calls methods with @Test before them uses reflection to process each method and its annotations. (Look at the line "Junit 4 annotations" in JavaHyperText entry "Junit testing".)

Here, we show you how to get descriptions of the fields of an object and change the value of one field. Based on that, you can study other aspects of reflection yourself.

Class java.lang.Class

We illustrate using class `Point`, shown to the right.

```
/** An instance is a point (x, y) in
  * the plane.*/
public class Point {
   private int x; // x-coordinate
   private int y; // y-coordinate

   /** Constructor: a point (x, y). */
   public Point(int x, int y)
      { this.x= x; this.y= y; }
}
```

An object of class `java.lang.Class` describes a class or interface. Both expressions below evaluate to a pointer to the object of class `Class` that describes class `Point`. It turns out that `class` is a static field of every class and function `getClass()` is in every object.

```
Point.class
(new Point(3, 5)).getClass()
```

Given `Point.class`, you can look at all aspects of `Point`. This requires using classes like `Field`, `Method`, and `Modifier`. These are in package java.lang.reflect, so you have to import them.

Looking at fields of a class

A call `printFields(new Point(3, 4));` on the procedure to the right prints this:

```
/** Print a description of the fields of ob. */
public static void printFields(Object ob) {
  Class cb= ob.getClass();
  Field[] fields= cb.getDeclaredFields();
  System.out.println(Arrays.toString(fields));
}
```

```
[private int Point.x, private int Point.y]
```

In the function, use `getFields` instead of `getDeclaredFields` to get only the public fields.

Class Field contains methods to get the name of the field, its value, it type, and its modifiers —(e.g. public, final, static). There are even methods to change the value in the field.

As an example, to the right is a method to change field x of a Point object. Thus, after execution of this:

```
/** Set the value of field x of p to v. */
public static void setX(Point p, int v)
          throws NoSuchFieldException,
                 IllegalAccessException {
   Class cp= p.getClass();
   Field f= cp.getDeclaredField("x");
   f.setAccessible(true);
   f.set(p, v);
}
```

```
Point p= new Point(3, 5);  setX(p, 1)
```

field `p.x` will contain 1.

The method first gets the `Class` object for `Point`. It then stores in local variable `f` the description of field `x`. Procedure `f.setAccessible` is called to make field `x` accessible. Without that, since `x` is private, it can't be used outside class `Point`. Finally, the call on procedure `set` changes `x`'s value to `v`.

JavaHyperText entry "reflection" contains some demo code —what we did on this page as well as functions to get field-value pairs for all fields in an object. Neat!