Documentation

The Java™ Tutorials

Trail: Essential Classes
Lesson: Concurrency
Section: Immutable Objects

A Strategy for Defining Immutable Objects

The following rules define a simple strategy for creating immutable objects. Not all classes documented as "immutable" follow these rules. This does not necessarily mean the creators of these classes were sloppy — they may have good reason for believing that instances of their classes never change after construction. However, such strategies require sophisticated analysis and are not for beginners.

1. Don't provide "setter" methods — methods that modify fields or objects referred to by fields.
2. Make all fields `final` and `private`.
3. Don't allow subclasses to override methods. The simplest way to do this is to declare the class as `final`. A more sophisticated approach is to make the constructor `private` and construct instances in factory methods.
4. If the instance fields include references to mutable objects, don't allow those objects to be changed:
   - Don't provide methods that modify the mutable objects.
   - Don't share references to the mutable objects. Never store references to external, mutable objects passed to the constructor; if necessary, create copies, and store references to the copies. Similarly, create copies of your internal mutable objects when necessary to avoid returning the originals in your methods.

Applying this strategy to `SynchronizedRGB` results in the following steps:

1. There are two setter methods in this class. The first one, `set`, arbitrarily transforms the object, and has no place in an immutable version of the class. The second one, `invert`, can be adapted by having it create a new object instead of modifying the existing one.
2. All fields are already `private`; they are further qualified as `final`.
3. The class itself is declared `final`.
4. Only one field refers to an object, and that object is itself immutable. Therefore, no safeguards against changing the state of "contained" mutable objects are necessary.

After these changes, we have `ImmutableRGB`:

```
final public class ImmutableRGB {

    // Values must be between 0 and 255.
    final private int red;
    final private int green;
    final private int blue;
    final private String name;

    private void check(int red,
                       int green,
                       int blue) {
        if (red < 0 || red > 255
            || green < 0 || green > 255
            || blue < 0 || blue > 255) {
            throw new IllegalArgumentException();
        }
    }

    public ImmutableRGB(int red,
                        int green,
                        int blue,
                        String name) {
        check(red, green, blue);
        this.red = red;
        this.green = green;
        this.blue = blue;
        this.name = name;
    }


    public int getRGB() {
        return ((red << 16) | (green << 8) | blue);
    }

    public String getName() {
        return name;
    }

    public ImmutableRGB invert() {
        return new ImmutableRGB(255 - red,
                       255 - green,
                       255 - blue,
                       "Inverse of " + name);
    }
}
```

Previous page: A Synchronized Class Example
Next page: High Level Concurrency Objects