8.13 `final` Instance Variables

The *principle of least privilege* is fundamental to good software engineering. In the context of an app's code, it states that code should be granted only the amount of privilege and access that it needs to accomplish its designated task, but no more. This makes your programs more robust by preventing code from accidentally (or maliciously) modifying variable values and calling methods that should *not* be accessible.

Let's see how this principle applies to instance variables. Some need to be *modifiable* and some do not. You can use the keyword `final` to specify that a variable is *not* modifiable (i.e., it's a *constant*) and that any attempt to modify it is an error. For example,

```
private final int INCREMENT;
```

declares a `final` (constant) instance variable `INCREMENT` of type `int`. Such variables can be initialized when they're declared. If they're not, they *must* be initialized in every constructor of the class. Initializing constants in constructors enables each object of the class to have a different value for the constant. If a `final` variable is *not* initialized in its declaration or in every constructor, a compilation error occurs.

Common Programming Error 8.8

Attempting to modify a `final` *instance variable after it's initialized is a compilation error.*

Error-Prevention Tip 8.4

Attempts to modify a `final` *instance variable are caught at compilation time rather than causing execution-time errors. It's always preferable to get bugs out at compilation time, if possible, rather than allow them to slip through to execution time (where experience has shown that repair is often many times more expensive).*

Software Engineering Observation 8.11

Declaring an instance variable as `final` *helps enforce the principle of least privilege. If an instance variable should not be modified, declare it to be* `final` *to prevent modification. For example, in Fig. 8.8, the instance variables* `firstName`, `lastName`, `birthDate` *and* `hireDate` *are never modified after they're initialized, so they should be declared*

`final` *(Exercise 8.20). We'll enforce this practice in all programs going forward. Testing, debugging and maintaining programs is easier when every variable that can be* `final` *is, in fact,* `final`*. You'll see additional benefits of* `final` *in Chapter 23, Concurrency.*

Software Engineering Observation 8.12

A `final` *field should also be declared* `static` *if it's initialized in its declaration to a value that's the same for all objects of the class. After this initialization, its value can never change. Therefore, we don't need a separate copy of the field for every object of the class. Making the field* `static` *enables all objects of the class to share the* `final` *field.*