*Information hiding* is the principle. Design a program in modules in a way that hides the implementation of data in each one as much as possible, in order to make it easy to change implementations when necessary.

*Information hiding* is the principle; in Java, *encapsulation* is the way to achieve it. To *encapsulate* means to “surround, encase, or protect in or as if in a capsule” (Merriam-Webster Unabridged online dictionary). In Java and other OOP languages, the class (or each object created from it) is the capsule.

But not every class is written in a way that ensures information hiding. Here, we discuss some rules and principles to follow in writing a class so that information hiding is ensured. We do this using a class Fraction. You will see fragments of the class. The whole class can be found in JavaHyperText entry *encapsulation*.

**Rule 1. Make fields private** —unless there is a good reason to make them public.

/\*\* An instance represents a

rational number. \*/

public class Fraction {

/\*\* Numerator. \*/

private int num;

/\*\* Denominator. != 0. \*/

private int den;

The fields describe the state of the object, and they provide the information that is supposed to be hidden. Therefore, make them private. Note that the com­ment on a private field is for the *implementer*, not the client.

In our running example, a rational number is given by a numerator and den­om­inator, so there are fields for them, and we make them private. Note that the constraint requires the denominator to be different from 0. We can’t allow 0 denomin­ator because, for example, 8/0 is undefined. (Here’s a joke to play on your friends. If 8/0 = ∞, what is 5/0? If they don’t know, tell them it’s .)

5

**Rule 2. Don’t wantonly add getter and setter methods for each field**.

One is often tempted to add getter and setter methods for each field. In fact, Eclipse has a way for you to add them automatically when creating a class. Resist this temptation. The fields are private for a reason: information hiding. Add getter and setter methods for each field and that information is no longer hidden. We will see later principles for determining what information to make available through public methods.

**Rule 3. Don’t expose the difference between stored and calculated data**.

Suppose you write a method that returns the numerator in reduce form. Thus, if the fraction is 8/6, the reduced form is of the fraction is 4/3 and the reduced from of the numerator is 4. Two possible names for this method are getReducedNum and calculateReducedNum. The first name suggests that there is a field named reducedNum. The second name suggests that the reduced numerator is calculated. Neither is good because of what they suggest. Instead, just call the method reducedNumerator. That name gives no indication to the client about the implemen­ta­tion of fields in class Fraction.