the dot member operator, just like the fields. In object-oriented terminology, when we make a call to a method we are passing a message to the object.

3.2 A Simple Example

Functionality is supplied as additional members; these *methods* manipulate the object's state.

Recall that when you are designing the class, it is important to be able to hide internal details from the class user. This is done in two ways. First, the class can define functionality as class members, called *methods*. Some of these methods describe how an instance of the structure is created and initialized, how equality tests are performed, and how output is performed. Other functions would be specific to the particular structure. The idea is that the internal data fields that represent an object's state should not be manipulated directly by the class user but instead should be manipulated only through use of the methods. This idea can be strengthened by hiding members from the user. To do this, we can specify that they be stored in a *private* section. The compiler will enforce the rule that members in the private section are inaccessible by methods that are not in the object's class. Generally speaking, all data members should be private.

Public members are visible to nonclass routines; private members are not.

Figure 3.1 illustrates a class declaration for an `IntCell` object.[1] The declaration consists of two parts: public and private. The *public* section represents the portion that is visible to the user of the object. Since we expect to hide data, generally only methods and constants would be placed in the public section. In our example, we have methods that read and write to the `IntCell` object. The private section contains the data: this is invisible to the user of the object. The `storedValue` member must be accessed through the publicly visible routines `read` and `write`; it cannot be accessed directly by `main`. Another way of viewing this is shown in Figure 3.2.

```
 1 // IntCell class
 2 //  int read( )         -->  Returns the stored value
 3 //  void write( int x ) -->  x is stored
 4
 5 public class IntCell
 6 {
 7         // Public methods
 8     public int read( )         { return storedValue; }
 9     public void write( int x ) { storedValue = x; }
10
11         // Private internal data representation
12     private int storedValue;
13 }
```

Figure 3.1 A complete declaration of an `IntCell` class

1. Recall that classes must be placed in files of the same name. Thus `IntCell` must be in file *IntCell.java*.