2.5.1 Choosing variable names

One of the things you might have noticed is that the variable names we use for fields and parameters have a close connection with the purpose of the variable. Names such as **`price`**, **`cost`**, **`title`**, and **`alive`** all tell you something useful about the information being stored in that variable. This, makes it easier to understand what is going on in the program. Given that we have a large degree of freedom in our choice of variable names, it is worth following this principle of choosing names that communicate a sense of purpose rather than arbitrary and meaningless combinations of letters and numbers.

2.6 Assignment

In the previous section, we noted the need to copy the short-lived value stored in a parameter variable into somewhere more permanent—a field variable. In order to do this, the body of the constructor contains the following *assignment statement*:

```
price = cost;
```

Concept

Assignment statements store the value represented by the right-hand side of the statement in the variable named on the left.

Assignment statements are used frequently in programming, as a means to store a value into a variable. They can be recognized by the presence of an assignment operator, such as "=" in the example above. Assignment statements work by taking the value of what appears on the right-hand side of the operator and copying that value into the variable on the left-hand side. This is illustrated in Figure 2.4 by the arrow labeled (B). The right-hand side is called an *expression*. In their most general form, expressions are things that compute values, but in this case, the expression consists of just a single variable, whose value is copied into the **`price`** variable. We shall see examples of more-complicated expressions later in this chapter.

One rule about assignment statements is that the type of the expression on the right-hand side must match the type of the variable to which its value is assigned. We have already met three different, commonly used types: **`int`**, **`String`**, and (very briefly) **`boolean`**. This rule means that we are not allowed to store an **`int`**-type expression in a **`String`**-type variable, for instance. This same rule also applies between formal parameters and actual parameters: the type of an actual-parameter expression must match the type of the formal-parameter variable. For now, we can say that the types of both must be the same, although we shall see in later chapters that this is not the whole truth.

Exercise 2.21 Suppose that the class **`Pet`** has a field called **`name`** that is of the type **`String`**. Write an assignment statement in the body of the following constructor so that the **`name`** field will be initialized with the value of the constructor's parameter.

```
public Pet(String petsName)
{
}
```

Exercise 2.22 *Challenge exercise* The following object creation will result in the constructor of the **`Date`** class being called. Can you write the constructor's header?

```
new Date("March", 23, 1861)
```

Try to give meaningful names to the parameters.