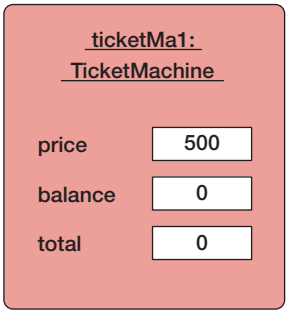

Figure 2.3
A `TicketMachine` object after initialization (created for 500-cent tickets)

2.5 Parameters: receiving data

Constructors and methods play quite different roles in the life of an object, but the way in which both receive values from outside is the same: via *parameters*. You may recall that we briefly encountered parameters in Chapter 1 (Section 1.4). Parameters are another sort of variable, just as fields are, so they are also used to hold data. Parameters are variables that are defined in the header of a constructor or method:

```
public TicketMachine(int cost)
```

This constructor has a single parameter, **`cost`**, which is of type **`int`**—the same type as the **`price`** field it will be used to set. A parameter is used as a sort of temporary messenger, carrying data originating from outside the constructor or method, and making it available inside it.

Figure 2.4 illustrates how values are passed via parameters. In this case, a BlueJ user enters the external value into the dialog box when creating a new ticket machine (shown on the left), and that value is then copied into the **`cost`** parameter of the new machine's constructor. This is illustrated with the arrow labeled (A). The box in the **`TicketMachine`** object in Figure 2.4, labeled "TicketMachine (constructor)," represents additional space for the object that is created only when the constructor executes. We shall call it the *constructor space* of the object (or *method space* when we talk about methods instead of constructors, as the situation there is the same). The constructor space is used to provide space to store the values for the constructor's parameters. In our diagrams, all variables are represented by white boxes.

Concept

The **scope** of a variable defines the section of source code from which the variable can be accessed.

We distinguish between the parameter *names* inside a constructor or method, and the parameter *values* outside, by referring to the names as *formal parameters* and the values as *actual parameters*. So **`cost`** is a formal parameter, and a user-supplied value such as 500 is an actual parameter.

A formal parameter is available to an object only within the body of a constructor or method that declares it. We say that the *scope* of a parameter is restricted to the body of the constructor or method in which it is declared. In contrast, the scope of a field is the whole of the class definition–it can be accessed from anywhere in the same class. This is a very important difference between these two sorts of variables.

Concept

The **lifetime** of a variable describes how long the variable continues to exist before it is destroyed.

A concept related to variable scope is variable *lifetime*. The lifetime of a parameter is limited to a single call of a constructor or method. When a constructor or method is called, the extra space for the parameter variables is created, and the external values copied into that space. Once that call has completed its task, the formal parameters disappear and the values they held are lost. In other words, when the constructor has finished executing, the whole constructor space is removed, along with the parameter variables held within it (see Figure 2.4).

In contrast, the lifetime of a field is the same as the lifetime of the object to which it belongs. When an object is created, so are the fields, and they persist for the lifetime of the object. It follows that if we want to remember the cost of tickets held in the `cost` parameter, we must store the value somewhere persistent—that is, in the `price` field.

Just as we expect to see a close link between a constructor and the fields of its class, we expect to see a close link between the constructor's parameters and the fields, because external values will often be needed to set the initial values of one or more of those fields. Where this is the case, the parameter types will closely match the types of the corresponding fields.

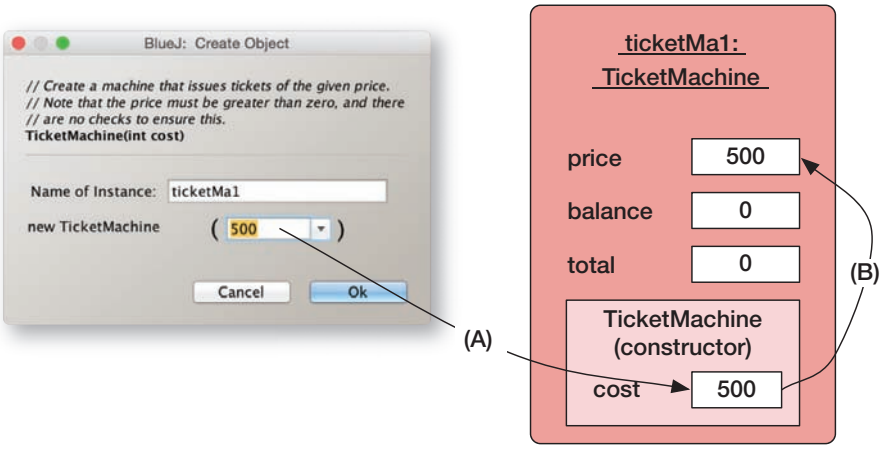

Figure 2.4
Parameter passing (A) and assignment (B)

Exercise 2.18 To what class does the following constructor belong?

```
public Student(String name)
```

Exercise 2.19 How many parameters does the following constructor have, and what are their types?

```
public Book(String title, double price)
```

Exercise 2.20 Can you guess what types some of the **Book** class's fields might be, from the parameters in its constructor? Can you assume anything about the names of its fields?