There are at least two significant differences between the headers of the **TicketMachine** constructor and the **getPrice** method:

```
public TicketMachine(int cost)

public int getPrice()
```

- The method has a *return type* of **int**; the constructor has no return type. A return type is written just before the method name. This is a difference that applies in all cases.
- The constructor has a single formal parameter, **cost**, but the method has none—just a pair of empty parentheses. This is a difference that applies in this particular case.

It is an absolute rule in Java that a constructor may not have a return type. On the other hand, both constructors and methods may have any number of formal parameters, including none.

Within the body of **getPrice** there is a single statement:

```
return price;
```

This is called a *return statement*. It is responsible for returning an integer value to match the **int** return type in the method's header. Where a method contains a return statement, it is always the final statement of that method, because no further statements in the method will be executed once the return statement is executed.

Return types and return statements work together. The **int** return type of **getPrice** is a form of promise that the body of the method will do something that ultimately results in an integer value being calculated and returned as the method's result. You might like to think of a method call as being a form of question to an object, and the return value from the method being the object's answer to that question. In this case, when the **getPrice** method is called on a ticket machine, the question is, What do tickets cost? A ticket machine does not need to perform any calculations to be able to answer that, because it keeps the answer in its **price** field. So the method answers by just returning the value of that variable. As we gradually develop more-complex classes, we shall inevitably encounter more-complex questions that require more work to supply their answers.

2.8 Accessor and mutator methods

We often describe methods such as the two "get" methods of **TicketMachine** (**getPrice** and **getBalance**) as *accessor methods* (or just *accessors*). This is because they return information to the caller about the state of an object; they provide access to information about the object's state. An accessor usually contains a return statement in order to pass back that information.

Concept

Accessor methods return information about the state of an object.

There is often confusion about what "returning a value" actually means in practice. People often think it means that something is printed by the program. This is not the case at all— we shall see how printing is done when we look at the **printTicket** method. Rather, returning a value means that some information is passed internally between two different parts of the program. One part of the program has requested information from an object via a method call, and the return value is the way the object has of passing that information back to the caller.

The **get** methods of a ticket machine perform similar tasks: returning the value of one of their object's fields. The remaining methods—**insertMoney** and **printTicket**—have a

Exercise 2.23 Compare the header and body of the **getBalance** method with the header and body of the **getPrice** method. What are the differences between them?

Exercise 2.24 If a call to **getPrice** can be characterized as "What do tickets cost?" how would you characterize a call to **getBalance**?

Exercise 2.25 If the name of **getBalance** is changed to **getAmount**, does the return statement in the body of the method also need to be changed for the code to compile? Try it out within BlueJ. What does this tell you about the name of an accessor method and the name of the field associated with it?

Exercise 2.26 Write an accessor method **getTotal** in the **TicketMachine** class. The new method should return the value of the **total** field.

Exercise 2.27 Try removing the return statement from the body of **getPrice**. What error message do you see now when you try compiling the class?

Exercise 2.28 Compare the method headers of **getPrice** and **printTicket** in Code 2.1. Apart from their names, what is the main difference between them?

Exercise 2.29 Do the **insertMoney** and **printTicket** methods have return statements? Why do you think this might be? Do you notice anything about their headers that might suggest why they do not require return statements?

Concept

Mutator methods change the state of an object.

much more significant role, primarily because they *change* the value of one or more fields of a ticket-machine object each time they are called. We call methods that change the state of their object *mutator methods* (or just *mutators*).

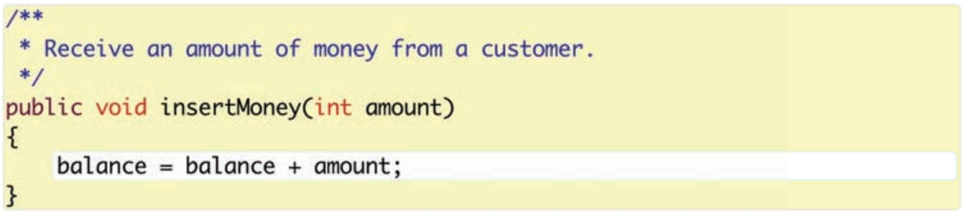
In the same way as we think of a call to an accessor as a request for information (a question), we can think of a call to a mutator as a request for an object to change its state. The most basic form of mutator is one that takes a single parameter whose value is used to directly overwrite what is stored in one of an object's fields. In a direct complement to "get" methods, these are often called "set" methods, although the **TicketMachine** does not have any of those, at this stage.

One distinguishing effect of a mutator is that an object will often exhibit slightly different behavior before and after it is called. We can illustrate this with the following exercise.

Exercise 2.30 Create a ticket machine with a ticket price of your choosing. Before doing anything else, call the **getBalance** method on it. Now call the **insertMoney** method (Code 2.6) and give a non-zero positive amount of money as the actual parameter. Now call **getBalance** again. The two calls to **getBalance** should show different outputs, because the call to **insertMoney** had the effect of changing the machine's state via its **balance** field.

The header of **insertMoney** has a **void** return type and a single formal parameter, **amount**, of type **int**. A **void** return type means that the method does not return any value to its caller. This is significantly different from all other return types. Within BlueJ, the difference is most noticeable in that no return-value dialog is shown following a call to a **void** method. Within the body of a **void** method, this difference is reflected in the fact that there is no return statement.[2]

Code 2.6
The **insert-Money** method



```
/**
 * Receive an amount of money from a customer.
 */
public void insertMoney(int amount)
{
    balance = balance + amount;
}
```

In the body of **insertMoney**, there is a single statement that is another form of assignment statement. We always consider assignment statements by first examining the calculation on the right-hand side of the assignment symbol. Here, its effect is to calculate a value that is the sum of the number in the **amount** parameter and the number in the **balance** field. This combined value is then assigned to the **balance** field. So the effect is to increase the value in **balance** by the value in **amount**.[3]

Exercise 2.31 How can we tell from just its header that **setPrice** is a method and not a constructor?

```
public void setPrice(int cost)
```

Exercise 2.32 Complete the body of the **setPrice** method so that it assigns the value of its parameter to the **price** field.

Exercise 2.33 Complete the body of the following method, whose purpose is to add the value of its parameter to a field named **score**.

```
/**
 * Increase score by the given number of points.
 */
public void increase(int points)
{
    ...
}
```

[2] In fact, Java does allow **void** methods to contain a special form of return statement in which there is no return value. This takes the form

```
return;
```

and simply causes the method to exit without executing any further code.

[3] Adding an amount to the value in a variable is so common that there is a special **compound assignment operator** to do this: +=. For instance:

```
balance += amount;
```