Exercise 2.59 What happens if you try to compile the **TicketMachine** class with the following version of **refundBalance**?

```
public int refundBalance()
{
    return balance;
    balance = 0;
}
```

What do you know about return statements that helps to explain why this version does not compile?

Exercise 2.60 What is wrong with the following version of the constructor of **TicketMachine**?

```
public TicketMachine(int cost)
{
    int price = cost;
    balance = 0;
    total = 0;
}
```

Try out this version in the *better-ticket-machine* project. Does this version compile? Create an object and then inspect its fields. Do you notice something wrong about the value of the **price** field in the inspector with this version? Can you explain why this is?

2.17 Fields, parameters, and local variables

With the introduction of **amountToRefund** in the **refundBalance** method, we have now seen three different kinds of variables: fields, formal parameters, and local variables. It is important to understand the similarities and differences between these three kinds. Here is a summary of their features:

- All three kinds of variables are able to store a value that is appropriate to their defined types. For instance, a defined type of **int** allows a variable to store an integer value.
- Fields are defined outside constructors and methods.
- Fields are used to store data that persist throughout the life of an object. As such, they maintain the current state of an object. They have a lifetime that lasts as long as their object lasts.
- Fields have class scope: they are accessible throughout the whole class, so they can be used within any of the constructors or methods of the class in which they are defined.

- As long as they are defined as **`private`**, fields cannot be accessed from anywhere outside their defining class.
- Formal parameters and local variables persist only for the period that a constructor or method executes. Their lifetime is only as long as a single call, so their values are lost between calls. As such, they act as temporary rather than permanent storage locations.
- Formal parameters are defined in the header of a constructor or method. They receive their values from outside, being initialized by the actual parameter values that form part of the constructor or method call.
- Formal parameters have a scope that is limited to their defining constructor or method.
- Local variables are defined inside the body of a constructor or method. They can be initialized and used only within the body of their defining constructor or method. Local variables must be initialized before they are used in an expression—they are not given a default value.
- Local variables have a scope that is limited to the block in which they are defined. They are not accessible from anywhere outside that block.

New programmers often find it difficult to work out whether a variable should be defined as a field or as a local variable. Temptation is to define all variables as fields, because they can be accessed from anywhere in the class. In fact, the opposite approach is a much better rule to adopt: define variables local to a method unless they are clearly a genuine part of an object's persistent state. Even if you anticipate using the same variable in two or more methods, define a separate version locally to each method until you are absolutely sure that persistence between method calls is justified.

Exercise 2.61 Add a new method, **`emptyMachine`**, that is designed to simulate emptying the machine of money. It should reset **`total`** to be zero, but also return the value that was stored in **`total`** before it was reset.

Exercise 2.62 Rewrite the **`printTicket`** method so that it declares a local variable, **`amountLeftToPay`**. This should then be initialized to contain the difference between **`price`** and **`balance`**. Rewrite the test in the conditional statement to check the value of **`amountLeftToPay`**. If its value is less than or equal to zero, a ticket should be printed; otherwise, an error message should be printed stating the amount left to pay. Test your version to ensure that it behaves in exactly the same way as the original version. Make sure that you call the method more than once, when the machine is in different states, so that both parts of the conditional statement will be executed on separate occasions.