the indentation back. If your indentation is completely out, use BlueJ's "Auto-layout" function (find it in the editor menu!) to fix it.

- Pay attention to the scope highlighting. You will quickly get used to the way well-structured code looks. Try removing a curly bracket in the editor or adding one at an arbitrary location, and observe how the coloring changes. Get used to recognizing when scopes look wrong.

**Exercise 2.52** After a ticket has been printed, could the value in the **balance** field ever be set to a negative value by subtracting **price** from it? Justify your answer.

**Exercise 2.53** So far, we have introduced you to two arithmetic operators, **+** and **–**, that can be used in arithmetic expressions in Java. Take a look at Appendix C to find out what other operators are available.

**Exercise 2.54** Write an assignment statement that will store the result of multiplying two variables, **price** and **discount**, into a third variable, **saving**.

**Exercise 2.55** Write an assignment statement that will divide the value in **total** by the value in **count** and store the result in **mean**.

**Exercise 2.56** Write an if-statement that will compare the value in **price** against the value in **budget**. If **price** is greater than **budget**, then print the message "Too expensive"; otherwise print the message "Just right".

**Exercise 2.57** Modify your answer to the previous exercise so that the message includes the value of your budget if the price is too high.

## 2.16 Local variables

So far, we have encountered two different sorts of variables: fields (instance variables) and parameters. We are now going to introduce a third kind. All have in common that they store data, but each sort of variable has a particular role to play.

Section 2.7 noted that a method body (or, in general, a *block*) can contain both declarations and statements. To this point, none of the methods we have looked at contain any declarations. The **refundBalance** method contains three statements and a single declaration. The declaration introduces a new kind of variable:

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

What sort of variable is **amountToRefund**? We know that it is not a field, because fields are defined outside methods. It is also not a parameter, as those are always defined in the method header. The **amountToRefund** variable is what is known as a *local variable*, because it is defined *inside* a method body.

**Concept**

A **local variable** is a variable declared and used within a single method. Its scope and lifetime are limited to that of the method.

Local variable declarations look similar to field declarations, but they never have **private** or **public** as part of them. Constructors can also have local variables. Like formal parameters, local variables have a scope that is limited to the statements of the method to which they belong. Their lifetime is the time of the method execution: they are created when a method is called and destroyed when a method finishes.

You might wonder why there is a need for local variables if we have fields. Local variables are primarily used as temporary storage, to help a single method complete its task; we think of them as data storage for a single method. In contrast, fields are used to store data that persists through the life of a whole object. The data stored in fields is accessible to all of the object's methods. We try to avoid declaring as fields variables that really only have a local (method-level) usage, whose values don't have to be retained beyond a single method call. So even if two or more methods in the same class use local variables for a similar purpose, it is not appropriate to define them as fields if their values don't need to persist beyond the end of the methods.

In the **refundBalance** method, **amountToRefund** is used briefly to hold the value of the **balance** immediately prior to the latter being set to zero. The method then returns the remembered value of the balance. The following exercises will help to illustrate why a local variable is needed here, as we try to write the **refundBalance** method without one.

It is quite common to initialize local variables within their declaration. So we could abbreviate the first two statements of **refundBalance** as

```
int amountToRefund = balance;
```

but it is still important to keep in mind that there are two steps going on here: declaring the variable **amountToRefund**, and giving it an initial value.

**Pitfall** A local variable of the same name as a field will prevent the field being accessed from within a constructor or method. See Section 3.13.2 for a way around this when necessary.

**Exercise 2.58** Why does the following version of **refundBalance** not give the same results as the original?

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

What tests can you run to demonstrate that it does not?