**Code 2.8 continued**
A more sophisticated `TicketMachine`

```
    /**
     * Print a ticket if enough money has been inserted, and
     * reduce the current balance by the ticket price. Print
     * an error message if more money is required.
     */
    public void printTicket()
    {
        if(balance >= price) {
            // Simulate the printing of a ticket.
            System.out.println("##################");
            System.out.println("# The BlueJ Line");
            System.out.println("# Ticket");
            System.out.println("# " + price + " cents.");
            System.out.println("##################");
            System.out.println();

            // Update the total collected with the price.
            total = total + price;
            // Reduce the balance by the price.
            balance = balance - price;
        }
        else {
            System.out.println("You must insert at least: " +
                               (price - balance) + " more cents.");

        }
    }

    /**
     * Return the money in the balance.
     * The balance is cleared.
     */
    public int refundBalance()
    {
        int amountToRefund;
        amountToRefund = balance;
        balance = 0;
        return amountToRefund;
    }
}
```

## 2.13 Making choices: the conditional statement

Code 2.8 shows the internal details of the better ticket machine's class definition. Much of this definition will already be familiar to you from our discussion of the naïve ticket machine. For instance, the outer wrapping that names the class is the same, because we have chosen to give this class the same name. In addition, it contains the same three fields to maintain object state, and these have been declared in the same way. The constructor and the two **get** methods are also the same as before.

The first significant change can be seen in the **insertMoney** method. We recognized that the main problem with the naïve ticket machine was its failure to check certain conditions. One of those missing checks was on the amount of money inserted by a customer, as it was possible for a negative amount of money to be inserted. We have remedied that failing by making use of a *conditional statement* to check that the amount inserted has a value greater than zero:

```
if(amount > 0) {
    balance = balance + amount;
}
else {
    System.out.println("Use a positive amount rather than: " +
                        amount);
}
```

**Concept**

A **conditional statement** takes one of two possible actions based upon the result of a test.

Conditional statements are also known as *if-statements*, from the keyword used in most programming languages to introduce them. A conditional statement allows us to take one of two possible actions based upon the result of a check or test. If the test is true, then we do one thing; otherwise, we do something different. This kind of either/or decision should be familiar from situations in everyday life: for instance, if I have enough money left, then I shall go out for a meal; otherwise, I shall stay home and watch a movie. A conditional statement has the general form described in the following *pseudo-code*:

```
if(perform some test that gives a true or false result) {
    Do the statements here if the test gave a true result
}
else {
    Do the statements here if the test gave a false result
}
```

Certain parts of this pseudo-code are proper bits of Java, and those will appear in almost all conditional statements–the keywords `if` and `else`, the round brackets around the test, and the curly brackets marking the two blocks–while the other three italicized parts will be fleshed out differently for each particular situation being coded.

Only one of the two blocks of statements following the test will ever be performed following the evaluation of the test. So, in the example from the `insertMoney` method, following the test of an inserted amount we shall only either add the amount to the balance or print the error message. The test uses the *greater-than operator*, ">", to compare the value in `amount` against zero. If the value is greater than zero, then it is added to the balance. If it is not greater than zero, then an error message is printed. By using a conditional statement, we have, in effect, protected the change to `balance` in the case where the parameter does not represent a valid amount. Details of other Java operators can be found in Appendix C. The obvious ones to mention at this point are "<" (less-than), "<=" (less-than or equal-to), and ">=" (greater-than or equal-to). All are used to compare two numeric values, as in the `printTicket` method.

**Concept**

**Boolean expressions** have only two possible values: true and false. They are commonly found controlling the choice between the two paths through a conditional statement.

The test used in a conditional statement is an example of a *boolean expression.* Earlier in this chapter, we introduced arithmetic expressions that produced numerical results. A boolean expression has only two possible values (`true` or `false`): the value of `amount` is either greater than zero (`true`) or it is not greater (`false`). A conditional statement makes use of those two possible values to choose between two different actions.