

JAVA

FOR
Students




Java for Students

Visit the *Java for Students*, sixth edition Companion Website at www.pearsoned.co.uk/bell to find valuable student learning material including:

- How to download Java 6.0
- Programs from the book
- An extra chapter on Java network programming

```
private void drawSign() {  
    if (open) {  
        textField.setText("Open");  
    }  
    else {  
        textField.setText("Closed");  
    }  
    if (!on) {  
        textField.setText("");  
    }  
}
```



In the above program, one of the **if** statements is as follows, because the variable **open** is either true or false and can be tested directly:

```
if (open) {
```

This is the neater way of testing the value of a **boolean** variable. It can be rewritten less concisely as:

```
if (open == true) {
```

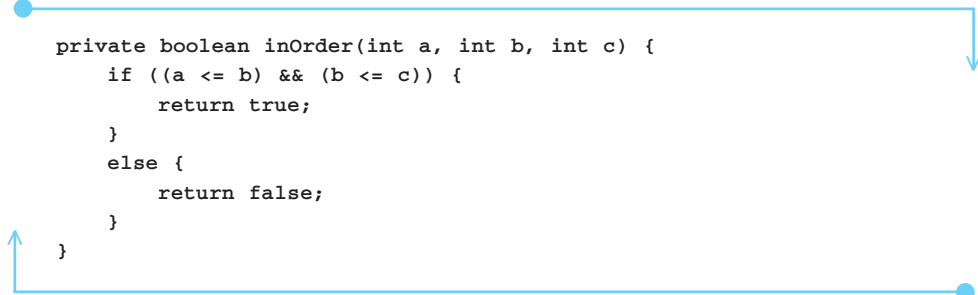
To summarize, **boolean** variables are used in programming to remember whether something is true or false, perhaps for a short time, perhaps for the whole time that the program is running.

SELF-TEST QUESTION

7.12 The shop owner needs an additional sign that says ‘SALE’. Can we still use a **boolean** variable?

Methods can use **boolean** values as parameters and as return values. For example, here is a method that checks whether three numbers are in numerical order:

```
private boolean inOrder(int a, int b, int c) {  
    if ((a <= b) && (b <= c)) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```



Comparing strings

Thus far, we have looked at programs that use the comparison operators (such as `>`) to compare numbers. However, many programs need to compare strings. The comparison operators are not appropriate and, instead, we use the `equals` method.

The safe program above required the user to enter a numeric code. Suppose instead that the code is alphabetic. In this case, the program needs to compare the string entered with the correct code (say 'Bill'). The appropriate `if` statement is:

```
String code;
code = codeField.getText();
if (code.equals("Bill")) {
    outcomeTextField.setText("unlocked");
}
```

The method `equals` is called. The parameter is the string 'Bill'. The method returns true or false. Then the `if` statement acts accordingly.

Programming principles

The computer normally obeys instructions one by one in a sequence. An `if` statement instructs the computer to test the value of some data and then take one of a choice of actions depending on the result of the test. This choice is sometimes called selection. The test of the data is called a condition. After an `if` statement is completed, the computer continues obeying the instructions in sequence.

Programming pitfalls

Brackets

The condition within an `if` statement must be enclosed in brackets. For example:

```
if (a > b) etc.
```

Equals

If you want to test for equality, use the `==` operator (not a single equals sign, `=`). So this is correct:

```
if (a == b) etc.
```



*Programming pitfalls continued*

Unfortunately, a program that uses a single `=` will compile correctly but work incorrectly.

Comparing strings

If you want to compare two strings, you must use the `equals` method, like this:

```
if (string1.equals(string2)) etc.
```

Braces

Next, we look at braces. This statement is entirely correct:

```
if (code == 123)
    outcomeTextField.setText("unlocked");
```

even though the braces that surround the statement are missing. The Java rule is that if there is only a **single** statement to be done, then the braces are not necessary. However, this can lead to nuisance programming problems, and the overwhelming advice is to insert the braces at all times. There is an exception to this suggestion when you use nested `if` statements in the `else if` style, explained above.

Compound conditions

You might find that you have written an `if` statement like this:

```
if (a > 18 && < 25)
```

which is wrong. Instead, the `&&` must link two complete conditions, preferably in brackets for clarity, like this:

```
if ((a > 18) && (a < 25))
```

switch

The `switch` statement is very useful, but unfortunately it is not as flexible as it could be. Suppose, for example, we want to write a piece of program to display two numbers, with the larger first, followed by the smaller. Using `if` statements, we would write:

```
if (a > b) {
    textField.setText(Integer.toString(a) + " is greater than "
        + Integer.toString(b));
}
```

```
if (b > a) {
    textField.setText(Integer.toString(b) + " is greater than "
                      + Integer.toString(a));
}
if (a == b) {
    textField.setText("they are equal");
}
```

We may be tempted to rewrite this using a **switch** statement as follows:

```
switch (?) { // beware! illegal Java
    case a > b:
        textField.setText(Integer.toString(a) + " is greater than"
                          + Integer.toString(b));
        break;
    case b > a:
        textField.setText(Integer.toString(b) + " is greater than"
                          + Integer.toString(a));
        break;
    case a == b:
        textField.setText("they are equal");
        break;
}
```

but this is not allowed because, as indicated by the question mark, **switch** only works with a single integer variable as its subject and **case** cannot use the operators **>**, **==**, **<**, etc.

Grammar spot

The first type of **if** statement has the structure:

```
if (condition) {
    statements
}
```

The second type of **if** statement has the structure:

```
if (condition) {
    statements
}
else {
    statements
}
```



*Grammar spot continued*

The **switch** statement has the structure:

```
switch (variable) {  
    case value1:  
        statements  
        break;  
    case value2:  
        statements  
        break;  
    default:  
        statements  
        break;  
}
```

The default section is optional.

New language elements

- Control structures for decisions:

```
if, else  
switch, case, break, default
```

- The comparison operators **>**, **<**, **==**, **!=**, **<=** and **>=**.
- The logical operators **&&**, **||** and **!**.
- Variables declared as **boolean**, which can take either the value **true** or the value **false**.

Summary

if statements allow the programmer to control the sequence of actions by making the program carry out a test. Following the test, the computer carries out one of a choice of actions. There are two varieties of **if** statement:

- **if**
- **if...else**

The **if** statement can be used to identify the source of a GUI event. The method **getSource** returns the object that caused the event. This object is compared with each of the objects that could have caused the event, using the **==** comparison operator.

The **switch** statement provides a convenient way of carrying out a number of tests. However, the **switch** statement is restricted to tests on integers.

A **boolean** variable can be assigned the value **true** or the value **false**. A **boolean** variable can be tested with an **if** statement. A **boolean** variable is useful in situations when a variable has only two meaningful values.

Exercises

- 7.1 Movie theatre (cinema) price** Write a program to work out how much a person pays to go to the cinema. The program should input an age from a slider or a text field and then decide on the following basis:
- under 5, free;
 - aged 5 to 12, half price;
 - aged 13 to 54, full price;
 - aged 55, or over, free.
- 7.2 The elevator** Write a program to simulate a very primitive elevator. The elevator is represented as a filled black square, displayed in a tall, thin, white panel. Provide two buttons – one to make it move 20 pixels up the panel and one to make it move down. Then enhance the program to make sure that the elevator does not go too high or too low.
- 7.3 Sorting** Write a program to input numbers from three sliders, or three text fields, and display them in increasing numerical size.
- 7.4 Betting** A group of people are betting on the outcome of three throws of a die. A person bets \$1 on predicting the outcome of the three throws. Write a program that uses the random number method to simulate three throws of a die and displays the winnings according to the following rules:
- all three throws are sixes: win \$20;
 - all three throws are the same (but not sixes): win \$10;
 - any two of the three throws are the same: win \$5.
- 7.5 Digital combination safe** Write a program to act as the digital combination lock for a safe. Create three buttons, representing the numbers 1, 2 and 3. The user clicks on the buttons, attempting to enter the correct numbers (say 331121). The program remains unhelpfully quiet until the correct buttons are pressed. Then it congratulates the user with a suitable message. A button is provided to restart.
- Enhance the program so that it has another button which allows the user to change the safe's combination, provided that the correct code has just been entered.
- 7.6 Deal a card** Write a program with a single button on it which, when clicked on, randomly selects a single playing card. First use the random number generator in the library to create a number in the range 1 to 4. Then convert the number to a suit (heart, diamond, club and spade). Next, use the random number generator to create a random number in the range 1 to 13. Convert the number to an ace, 2, 3, etc., and finally display the value of the chosen card.
- Hint: use **switch** as appropriate.
- 7.7 Rock, scissors, paper game** In its original form, each of the two players simultaneously chooses one of rock, scissors or paper. Rock beats scissors, paper beats rock and scissors beats paper. If both players choose the same, it is a draw. Write a program to play the game. The player selects one of three buttons, marked rock, scissors or paper. The