

ADVANCED TOPIC 5.2



The switch Statement

A sequence of if/else/else that compares a single value against several constant alternatives can be implemented as a switch statement. For example,

```
int digit;
. . .
switch (digit)
{
    case 1: System.out.print("one"); break;
    case 2: System.out.print("two"); break;
    case 3: System.out.print("three"); break;
    case 4: System.out.print("four"); break;
    case 5: System.out.print("five"); break;
    case 6: System.out.print("six"); break;
    case 7: System.out.print("seven"); break;
    case 8: System.out.print("eight"); break;
    case 9: System.out.print("nine"); break;
    default: System.out.print("error"); break;
}
```

This is a shortcut for

```
int digit;
. . .
if (digit == 1) System.out.print("one");
else if (digit == 2) System.out.print("two");
else if (digit == 3) System.out.print("three");
else if (digit == 4) System.out.print("four");
else if (digit == 5) System.out.print("five");
else if (digit == 6) System.out.print("six");
else if (digit == 7) System.out.print("seven");
else if (digit == 8) System.out.print("eight");
else if (digit == 9) System.out.print("nine");
else System.out.print("error");
```

Using the switch statement has one advantage. It is obvious that all branches test the same value, namely digit.

The `switch` statement can be applied only in narrow circumstances. The test cases must be constants, and they must be integers, characters, or enumerated constants. You cannot use a `switch` to branch on floating-point or string values. For example, the following is an error:

```
switch (name)
{
    case "one": . . . break; // Error
    . . .
}
```

Note how every branch of the `switch` was terminated by a `break` instruction. If the `break` is missing, execution falls through to the next branch, and so on, until finally a `break` or the end of the `switch` is reached. For example, consider the following `switch` statement:

```
switch (digit)
{
    case 1: System.out.print("one"); // Oops—no break
    case 2: System.out.print("two"); break;
    . . .
}
```

If `digit` has the value 1, then the statement after the `case 1:` label is executed. Because there is no `break`, the statement after the `case 2:` label is executed as well. The program prints "onetwo".

There are a few cases in which this fall-through behavior is actually useful, but they are very rare. Peter van der Linden [1, p. 38] describes an analysis of the `switch` statements in the Sun C compiler front end. Of the 244 `switch` statements, each of which had an average of 7 cases, only 3 percent used the fall-through behavior. That is, the default—falling through to the next case unless stopped by a `break`—was wrong 97 percent of the time. Forgetting to type the `break` is an exceedingly common error, yielding incorrect code.

We leave it to you to decide whether or not to use the `switch` statement. At any rate, you need to have a reading knowledge of `switch` in case you find it in the code of other programmers.