Control structures: if

Like a recipe tells the cook what to do, a program tells the computer what to do; in *imperative languages* like Java, lines in the program are imperative *statements* and you can imagine the computer running the program by pointing to each line in turn, like a cook follows the statements one after the other.

In a recipe, sometimes you find conditional instructions: "if you have parsley available, use three ounces of parsley; otherwise, use a branch of thyme." This instructs the cook to do different things depending on what he/she has available at the moment the recipe is used.

The corresponding construct in imperative language is the *conditional statement*, defined as follows:

Conditional statement:

Syntax:

> `if (` <expression> `)` <block-or-statement>
> [`else` <block-or-statement>]?
> (the keyword "`if`" followed by an expression between parentheses, followed by either a statement block between "`{`" and "`}`" or a single statement, followed optionally by the keyword "`else`", followed by either a statement block or a single statement.)

Semantics:

> The expression between parentheses, also called *conditional expression*, is evaluated first. If its value is equivalent to `true`, the first block-or-statement is executed, otherwise, the first block-or-statement is skipped, and if there is a "`else`" part, its block-or-statement is executed instead.

For example:

```
if (x > 0)
{
    System.out.println("x is positive");
}
```

(if x is positive, the program prints "x is positive"; otherwise, nothing is printed)
or:

```
if (x > 0)
{
    System.out.println("x is positive");
}
else
```

```
{
    System.out.println("x is zero or negative");
}
```

(if x is positive, the program prints "x is positive"; otherwise, the program prints "x is zero or negative")

Relational operators

The condition can contain any *expression*, or formula, that computes a value that is either "true" or "false". For this you can use the following operators in most programming languages, also called *relational operators*:

Operator	Description
`x == y`	tests if x is equal to y
`x != y`	tests if x is different from y
`x > y`	tests if x is strictly greater than y
`x < y`	tests if x is strictly lower than y
`x >= y`	tests if x is greater or equal to y
`x <= y`	tests if x is lower or equal to y

Note

Beware of the difference between the single "`=`" and the double "`==`". The first is the *assignment* operator, and "`x = 3`" means "change the value of the variable `x` to become 3"; the second is the *test for equality* operator, and "`x == 3`" means "test whether the value of variable `x` is equal to 3".

In Java, the relational operators are designed primarily to compare numbers; in particular you cannot use them to compare character strings (`String`). You will see later how to compare strings in Java.

Also, you must compare things that are comparable; in particular, *both sides must have comparable types*. For example it is not possible in Java to compare a value of type `int` with a value of type `double`. When in doubt, always ensure both sides have the same type.

Nested conditionals

The one or two "branches" of a `if` statement are themselves statements, so we can combine a `if` inside an `if`. This is called *nesting*: using a control structure "inside" of another. For example:

```
if (x == 0)
{
   System.out.println("x is zero");
}
else
{
   if (x > 0)
   {
      System.out.println("x is positive");
```

```
    }
    else
    {
        System.out.println("x is negative");
    }
}
```

In this example, the expression "`x == 0`" is first evaluated. If it is true, the first message is printed. Otherwise, the second `if` is executed and the expression "`x > 0`" is evaluated. If it is true, the 2nd message is printed. Otherwise, the 3rd message is printed. In this example, the 2nd `if-else` construct is *nested* within the 1st `if-else` construct; more specifically it is nested in its "else branch".

Conjunction, disjunction and negation

Say you want to write a program that prints a message "working age" if the age of a person is between 18 and 67. The relational operators seen above only have 2 operands left and right; to establish within the age falls within the range, you must thus combine two different conditions. For example:

```
if (age >= 18)
{
    if (age <= 67)
    {
        System.out.println("working age");
    }
}
```

This type of combination, called a *logical conjunction*, or "AND relation", happens so often in programs that most languages have a simplified notation; in Java this is noted "`&&`":

```
if (age >= 18 && age <= 67)
{
        System.out.println("working age");
}
```

Value of `x`	**Value of `y`**	**Value of `x && y`**
false	N/A	false
true	false	false
true	true	true

Similarly, say you want to offer a reduced price for people under 18 or above 67. For this you need a *logical disjunction*, or "OR relation". In Java, another operator exists for this and is noted "`||`":

```
if (age <= 18 || age >= 67)
{
        System.out.println("reduced price");
}
```

Value of `x`	Value of `y`	Value of `x || y`
false	false	false
false	true	true
true	N/A	true

To complete this set of logical operators, programming languages almost always also provide a general operator that "inverts" the truth value of a condition. This is called *negation*; in Java, this is noted using an exclamation mark ("`!`") followed by the condition to invert:

Example	Equivalent to
`!(x == y)`	`x != y`
`!(x != y)`	`x == y`
`!(x < y)`	`x >= y`
`!(x && y)`	`(!x) || (!y)`
`!(x || y)`	`(!x) && (!y)`

Value of `x`	Value of `!x`
true	false
false	true

Important concepts

- *conditional statement*
- syntax and semantics of `if` ... `else`
- relational operators: `==`, `!=`, `<`, `>`, `<=`, `>=`
- difference between "`=`" and "`==`"
- relational operators designed to work on numbers, in particular *cannot use on strings*
- *nesting*
- *conjunction* "`&&`", *disjunction* "`||`"
- *negation* with "`!`"

Further reading

- Think Java, sections 4.1-4.5 (pp. 39-42), section 6.6 (pp 62-63)
- Programming in Java, sections 3.1.1 (pp. 67-68), 3.1.3 (pp. 71-73), 3.5 (pp. 96-104)
- Absolute Java, section 3.1 (pp. 100-106)

Copyright and licensing