Type boolean

In some programming languages, for example Matlab and C, integers are used to represent the logical values **`true`** and **`false`**. Generally, 0 is used for **`false`**, and any other integer can be used for **`true`**.

Java handles boolean values differently. There is a (primitive) type **boolean**, whose values are **`true`** and **`false`** (that's it). This type has five operations whose operands are booleans:

Type **`boolean`**
  Values: **`true`**, **`false`**
  Operations: **`!`** (*not*), `&&` (*and*, or *conjunction*), || (*or*, or *disjunction*)
    == (equality, or equivalence), **`!`**= (inequality, or inequivalence)

Here is a table that defines the five operations.

b	**c**	**!b**	**b && c**	**b || c**	**b == c**	**b != c**
false	**false**	**true**	**false**	**false**	**true**	**false**
false	**true**	**true**	**false**	**true**	**false**	**true**
true	**false**	**false**	**false**	**true**	**false**	**true**
true	**true**	**false**	**true**	**true**	**true**	**false**

We evaluate a few expressions in the interactions pane and discuss the operations.

1. !, which is read "not", is unary logical negation **`!false`** is **`true`**, and **`!true`** is **`false`**.

2. `&&` is read *and* because `b && c` is true iff both `b` and `c` are true.

3. || is read *or* because `b || c` is true iff either `b` or `c` (or both) is true.

4. == is used for equality**:** `b == c` is true iff `b` and `c` have the same value.

5. **`!`**= is used for inequality**:** `b != c` is true iff `b` and `c` have different values.

Relations

Six relations operate on the numeric types to yield boolean values.

```
b == c, b != c, b < c, b <= c, b > c, b >= c
```

You have probably seen these relations in other programming languages, so we don’t go into full details here. The only strange point is that == is used for equality, and not =. Here are examples.

`5 < 6` is **`true`**
`5 >= 6` is **`false`**
`5 <` **`true`** is illegal because one operands is an **`int`** and the other a **`boolean`**.

These relational operators work for all the number types —**int**, **double**, **char**, etc. For example, we can test whether 5 < 6.2 is true, or whether 6.0 == 7 is true. If the two operands are not of the same type, one is converted to the other type so that the operation can be carried out. More on such conversions later.

Short circuit evaluation

Evaluation of

```
5/0 == 3  &&  false
```

results in an error, because of the division by 0. This is to be expected. But evaluation of the same expression with the operands reversed,

```
false  &&  5/0 == 3
```

does not produce an error message —it yields the value **false**. This is because evaluation of `&&` is done in *short-circuit* mode: as soon as the answer is known, evaluation stops. Since **`false`** `&& b` is always false, no matter what b is, there is no need to evaluate b.

Another way to look at the evaluation of b && c is to say that it is equivalent to an if-expression **if** b **then** c **else** **false**, which can actually be written in Java using the expression

```
b ? c : false  // equivalent to b && c
```

You will see this conditional expression later. Get used to it; it is useful.

In the same way, **`true || c`** is **`true`** no matter what the value of `c` is, so c is not evaluated in this case. The expression `b || c` is equivalent to

> **`if`** `b` **`then true else`** `c`, or the Java expression `b ?` **`true :`** `c`

As you will see in several assignments, short-circuit evaluation is a useful tool in writing boolean expressions.