

The assignment operators

We have said that the assignment is actually an operator, and thus can form part of an expression. In fact, it is one of a group of operators: the assignment operators.

- `variable = Expression` Assign the value of the Expression to the variable.
- `variable += Expression` means `variable = variable + Expression`
- `variable -= Expression` means `variable = variable - Expression`
- `variable *= Expression` means `variable = variable * Expression`
- `variable /= Expression` means `variable = variable / Expression`
- `variable %= Expression` means `variable = variable % Expression`

All of them have the value the same as that assigned to Variable.

0911B L06-1

Assignment Operators (cont)

There are two interesting things about assignment operators:

1. They don't just have a value, but also modify the value of a variable. That is, they have side-effect. Example: what will happen?

```
int a = 10, b = 20, c = 15;
int d = a + (b += 10) + b - c * (c = 5);
```

2. They are right-associative. That is, grouping are from right to left. (They also have very low precedence.) Example: what's the grouping?

```
int a = 10, b = 20, c = 15, d;

d = a = b + (c -= 15 + a);

d = a + b = b + (c -= 15 + a);
```

0911B L06-2

Most frequent forms

The following forms accounts for most of the usage of assignment operators.

1. `variable = ExpressionWithoutAssignment;`
To assign a value to a variable.
2. `variable_1 = variable_2 = ... = ExpressionWithoutAssignment;`
To assign the same value to some variables.
3. `variable += ExpressionWithoutAssignment;`
To add something to (or subtract something from, multiply something to or divide something by) a variable and also assign it back, basically for convenience.

Other usage forms are usually too complicated and should be avoided, since they are too difficult to understand.

0911B L06-3

String operations

You can concatenate (join) two strings together by using the "+" operator. The "+=" operator also works.

Both operators follow the normal rules of precedence and associativity.

If you add a string with an expression of another type, the value of that expression will be converted to a string before the concatenation.

Example:

```
string s = 5 + 2 + "Hello " + 2F;

s += "Hello " + 2F + 3F * 2
```

0911B L06-4

Named constants

At many times, you want constants to have names, like variables. For example, you might want to speak of something like `HKD_PER_USD`, instead of `7.75`.

Advantages:

1. If you mistype anything (e.g. type `HKU_PER_USD`), the compiler gives an error, saving you time to debug your programs. If you type `7.65` instead of `7.75` your program still compile correctly but will behave wrongly.
2. If suddenly you want `HKD_PER_USD` to become `7.8`, you just need to fix one place instead of many different places.
3. It makes programs more easy to read. For example, if you say `my_account += wages * HKD_PER_USD`; you clearly shows why you multiply wages by `7.8`. Compare this to `my_account += wages * 7.8`;

0911B L06-5

Named constants in Java

A named constant is made just in the same way like variables, except you add the word "final":

```
final constant_type constant_name = Expression;
```

The convention is that constant LOOKS_LIKE_THIS: ALL CAPS, and joined_by_underscores.

Therefore, to make the constant `HKD_PER_USD`, you can write something like this:

```
final double HKD_PER_USD = 7.8;
```

There are some constants already defined in Java which you might find useful:

`Math.E`: the base of natural logarithms, `2.718281828459045`.
`Math.PI`: the value of π , `3.141592653589793`.

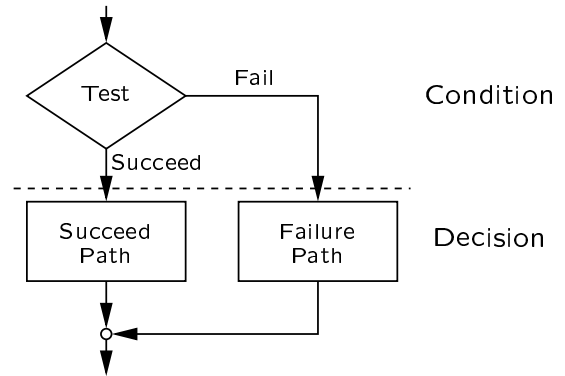
0911B L06-6

Concepts of Data Dependent Decisions

- Until now, all our programs are sequential: each program consists of a sequence of simple statements.
- These programs are not very intelligent: they always do the same operations, independent of the input.
- For example, the program solving quadratic equation tries to do the same operation no matter whether the determinant is negative. If the determinant is really negative, the program does something strange.
- A better program should try detecting such conditions and do something else in such cases (e.g. printing "No solution").
- These are called data dependent decisions: the program decides what to do depending on the data.

0911B L06-7

Ingredients of data dependent decision



0911B L06-8

Realization in Java

- There must be a way to express a condition. A condition is either true or false, so it should be something that gives a *boolean value*.
- There must be a way to express that we want to make a decision, and the two possible execution paths.

The first is done by a *boolean expression*. The second is done by the *if-then-else* construct.

Since we only execute one path out of the two possible paths, we call it the selection structure: the decision is to select one of the two possible paths.

0911B L06-9

Boolean expressions

There are two simple kinds of boolean expressions that you already know:

1. A boolean constant, i.e. `true` and `false`.
2. A boolean variable. E.g. if you have "boolean `to_go`", then `to_go` is a boolean expression.

The *relational operators* create a boolean value from non-boolean values, by performing a comparison.

The *logical operators* join two boolean values to give a new boolean value.

0911B L06-10

Relational operators

Relational operators compare two integer numbers, real numbers, booleans or characters (but not Strings!):

- `x == y` true if $x = y$, false otherwise.
- `x != y` true if $x \neq y$, false otherwise.
- `x < y` true if $x < y$, false otherwise.
- `x > y` true if $x > y$, false otherwise.
- `x <= y` true if $x \leq y$, false otherwise.
- `x >= y` true if $x \geq y$, false otherwise.

Examples: `20 == 30`, `1.9 < 5.3`, `'A' >= 'D'`.

Characters are compared according to the Unicode table.

0911B L06-11

Logical operators

Logical operators makes a boolean from boolean(s).

- `!x` true if x is false.
- `x & y` true if both x and y are true.
- `x && y` true if both x and y are true.
- `x | y` true if either x or y is true.
- `x || y` true if either x or y is true.
- `x ^ y` true if either x or y is true but not both.

`||` and `&&` will stop evaluating once the result is known. Other operators always evaluate both operands.

Example: `('A' < 'C') | (3 < 4)`, `(2 > 3) && (2.5 < 7)`, `(97 > 38) ^ (8 > 38)`.

0911B L06-12