

## Precedence of Boolean operators

As operators, boolean operators have precedence and associativity: see the table. Operators higher in the table have higher precedence.

Summary: Comparisons are just lower than arithmetic operators. Equality operators are a bit lower. Logical operators are below comparison, but their precedences are too complicated to memorize.

Operators	Associativity
.	left
unary +, unary -, !	right
casting ()	right
*, /, %	left
binary +, binary -	left
<, <=, >, >=	left
==, !=	left
&	left
^	left
	left
&&	left
	left
=, +=, -=, *=, /=, %=	right

0911B L07-1

0911B L07-2

## Revision: Selection structure

- Most programs tries to be intelligent, by doing different things when the input differs.
- This usually involves executing two different section of the program, depending on whether a test succeeds or fails.
- The test is defined by a boolean expression.
- A boolean expression can be a boolean constant, a boolean variable, a relational operation or a logical operation.

## The dot operator

We have silently been using one operator called dot operator. We don't even know that it is in fact an operator.

- All the Java methods, variables and constants are organized into classes.
- Classes are organized into packages.
- The organization is hierachical, very much like the files and directories in your hard disk.
- The dot operator allow us to specify something in a particular class or package.
- For example, `Math.sqrt` (more exactly, `java.lang.Math.sqrt`) means the `sqrt` method in the `Math` class (of the `lang` package in the `java` package).

0911B L07-3

## if/else statement

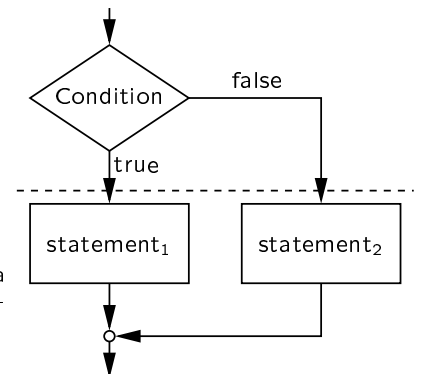
To perform a decision based on the value on a boolean expression:

Use **if/else** statement.

General form:

```
if (condition)
    statement1
else
    statement2
```

The whole construct is a statement (if-else statement).



0911B L07-4

## Example

```
import chapman.io.*;
public class Marks {
    public static void main(String[] args) {
        StdIn in = new StdIn();
        System.out.println("What is your mark? ");
        double mark = in.getDouble();
        if (mark >= 40)
            System.out.println("Pass!");
        else
            System.out.println("Fail!");
    }
}
```

0911B L07-5

## Variations

Some times, not both if-part and else-part are needed.

- If only the if-part is needed, we can skip the else part completely (the if statement). For example:

```
if (mark >= 40)
    System.out.println("Pass!");
```

- If only the else-part is needed, we can use a **null-statement**: a statement containing just a semicolon. Or better, we can reverse the test so that we want the if-part. For example:

```
if (mark >= 40);
else
    System.out.println("Fail!");

if (mark < 40)
    System.out.println("Fail!");
```

0911B L07-6

### Compound statements

At many times we want to execute more than one statements in a part of the if/else structure. However, each part is only allowed to have a statement, not a list of statements.

We can use a compound statement. It is a pair of braces enclosing zero or more statements. For example:

```
{
    System.out.print("The first solution is ");
    System.out.println((-b+Math.sqrt(determinant))/2/a);
    System.out.print("The second solution is ");
    System.out.println((-b-Math.sqrt(determinant))/2/a);
}
```

This is a single (compound) statement.

0911B L07-7

### Our quadratic example

```
/* Solves the quadratic equation */
import chapman.io.*;
public class Quadratic {
    public static void main(String[] args) {
        StdIn in = new StdIn();
        System.out.print("a? ");
        double a = in.readDouble();
        System.out.print("b? ");
        double b = in.readDouble();
        System.out.print("c? ");
        double c = in.readDouble();
        double determinant = b * b - 4 * a * c;
        if (determinant < 0)
            System.out.println("No solution.");
        else {
            System.out.print("The first solution is ");
            System.out.println((-b+Math.sqrt(determinant))/2/a);
            System.out.print("The second solution is ");
            System.out.println((-b-Math.sqrt(determinant))/2/a);
        }
    }
}
```

0911B L07-8

### Spaces in a program

You should have notice that space and lines usually do not count in a program: you can insert them everywhere. Here are the two exceptions:

- Spaces and newlines cannot occur within an operator, a numeric or character constant, a reserved word, the comments introducers or an identifier. So you cannot write `"/*` as `/ *`, or `"6.02e23"` as `"6.02 e 23"`.
- Newlines are not allowed within a string.

Usually we place spaces at strategic places to improve program readability. For example, at the beginning of each statement, we add two spaces per any un-closed brace ("indentation"). This makes sure the braces can easily be matched.

0911B L07-9