09 Conditional Statements

Instructor: Ke Wei（柯韋）

A319 ✆ Ext. 6452 ✉ wke@ipm.edu.mo

http://brouwer.ipm.edu.mo/COMP112/18/

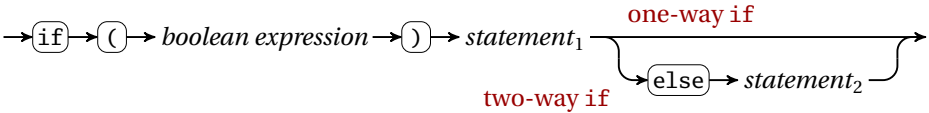
Bachelor of Science in Computing, School of Public Administration, Macao Polytechnic Institute

September 28, 2018

Outline

if Statement

- A statement can be conditionally executed by using `if` statement, it has the form shown in the syntax diagram below.



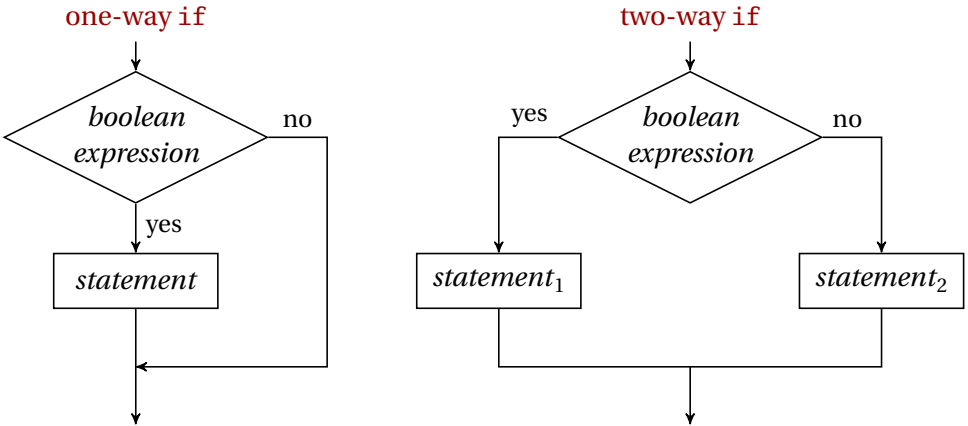
- If the boolean expression evaluates to `true`, $statement_1$ is executed, otherwise skipped.

```
if ( age >= 18 ) System.out.println("Adult.");
```

- Two statements can be selectively executed by using the alternative form of `if` statement, shown in the syntax diagram above (two-way `if`).
- If the boolean expression evaluates to `true`, $statement_1$ is executed, otherwise $statement_2$ is executed. One and only one of the statements is executed.

```
if ( mark >= 50 ) message = "Pass.";
else message = "Fail.";
```

Flowcharts of Two Forms of `if`



Finding the Min/Max

- We can use the standard two-way form to print one number or the other based on the comparison.

```
int a = scanner.nextInt(), b = scanner.nextInt();
if ( a >= b )
    System.out.println("The maximum number is: "+a);
else
    System.out.println("The maximum number is: "+b);
```

- Or, often, we can guess one, then conditionally update it only if we are wrong.

```
int a = scanner.nextInt(), b = scanner.nextInt();
int minimum = a;
if ( a > b )
    minimum = b;
System.out.println("The minimum number is: "+minimum);
```

Find the Minimum of 3

Sometimes we need to conditionally do some conditional statements, that is, to use nested conditional statements.

```
int x, y, z;
... // assign values to x, y and z.
int minOf3;
if ( x < y )
    if ( x < z ) // find min(z, x)
            minOf3 = x;
    else    minOf3 = z;
else
    if ( y < z ) // find min(z, y)
            minOf3 = y;
    else    minOf3 = z;
System.out.println("The minimum of "+x+", "+y+" and "+z+" is: "+minOf3);
```

Nested Conditional Statements

```
if ( x < y )
    if ( x < z )
        assignment
        minOf3 = x;
    else
        assignment
        minOf3 = z;
else
    if ( x < z )
        assignment
        minOf3 = y;
    else
        assignment
        minOf3 = z;
```

if statement (inner, first branch)

if statement (inner, second branch)

if statement (outer)

Incorrectly nested `if` statements:

```
int minOf3 = z;
if ( x < y )
    if ( x < z )
        minOf3 = x;
else
    if ( y < z )
        minOf3 = y;
```

You can always use braces to explicitly mark up the nested statement.

Multiple Alternative if Statements

The nested if statement can be used to implement multiple alternatives.

```
if (score >= 90.0)
  grade = 'A';
else
  if (score >= 80.0)
    grade = 'B';
  else
    if (score >= 70.0)
      grade = 'C';
    else
      if (score >= 60.0)
        grade = 'D';
      else
        grade = 'F';
```

equivalent to

```
if (score >= 90.0)
  grade = 'A';
else if (score >= 80.0)
  grade = 'B';
else if (score >= 70.0)
  grade = 'C';
else if (score >= 60.0)
  grade = 'D';
else
  grade = 'F';
```

This kind of nested if statements is called *multi-way* if statement.

Conditional Expression

- The conditional operator (`?:`) selects one of the two expressions to evaluate based on the result of the boolean expression:

$$\rightarrow \textit{boolean expression} \rightarrow \boxed{?} \rightarrow \textit{expression}_1 \rightarrow \boxed{:} \rightarrow \textit{expression}_2 \rightarrow$$

- If the boolean expression evaluates to `true`, then $\textit{expression}_1$ that follows the (?) is evaluated as the value of the conditional expression. Otherwise, $\textit{expression}_2$ that follows the (`:`) is evaluated. Only one of the expressions is evaluated.

```
x = 1; y = 2; x = x < y ? 10 : 5; // x becomes 10.
s = 100; d = 0; s = d != 0 ? s/d : 1; // s becomes 1.
```

- The (precedence of) conditional operator is higher than all assignment operators, lower than relational and logical operators. Therefore, the statement

```
name = 1 <= day && day <= 5 ? "weekday" : "weekend";
```

makes sense.

Logical Operators and Short-circuit Evaluation

- **NOT** → `!` → *boolean expression* →
  If the *boolean expression* evaluates to `true`, the NOT-expression is `false`; otherwise `true`.
- **AND** → *boolean expression*$_1$ → `&&` → *boolean expression*$_2$ →
  If *boolean expression*$_1$ evaluates to `false`, the AND-expression is `false`, and *boolean expression*$_2$ is *not* evaluated at all; otherwise, if *boolean expression*$_1$ evaluates to `true`, *boolean expression*$_2$ is evaluated as the result of the AND-expression.
- **OR** → *boolean expression*$_1$ → `||` → *boolean expression*$_2$ →
  If *boolean expression*$_1$ evaluates to `true`, the OR-expression is `true`, and *boolean expression*$_2$ is *not* evaluated at all; otherwise, if *boolean expression*$_1$ evaluates to `false`, *boolean expression*$_2$ is evaluated as the result of the OR-expression.
- As with the conditional expression, to determine the result by partial evaluation is called *short-circuit* evaluation. Many useful expressions rely on short-circuit evaluation.

$$divisor\ \texttt{!=}\ 0\ \texttt{\&\&}\ total/divisor\ \texttt{<}\ 5 \qquad i\ \texttt{<}\ 0\ \texttt{||}\ s.charAt(i)\ \texttt{!=}\ \texttt{'A'}$$

The Minimum of 4

With the logical operations and the multi-way `if` statement, we can write clearer code.

```
int m;

if ( w < x && w < y && w < z )
    m = w;
else if ( x < y && x < z )
    m = x;
else if ( y < z )
    m = y;
else
    m = z;

System.out.println("The minimum is: "+m);
```

Leap Years

- A leap year is a year that is a multiple of 4 but not a multiple of 100, or a multiple of 400.
- We can write a boolean expression that directly reflects the definition to tell whether a *year* is a leap year.

```
(year % 4 == 0 && year % 100 != 0) || year % 400 == 0
```

- Or, since a leap year must be a multiple of 4, we can eliminate those that cannot be divided by 4 first, using negation.

```
!(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0))
```

- ✍ Try to write a multi-way `if` statement to print `true` for a leap year and `false` for an ordinary year, *without* using logical operators.

Assignment Expressions

- Assignments are usually used as statements, however, they can also be expressions.
- An **assignment statement** is actually an *assignment expression* followed by a semicolon (`;`).

→ *assignment expression* → `;` →

- The evaluation of an assignment expression has a *side effect* of changing some variable, and the value of the assignment expression is exactly the value assigned to the variable.

```
int lg = 0;
while ( (x /= 10) > 0 ) ++lg;  // computes the integer part of lg(x).
```

- The following code keeps reading user input numbers until a negative number is encountered.

```
int count = 0, total = 0;
while ( (z = scanner.nextInt()) >= 0 ) { count++;    total += z; }
```

- Usually, assignment expressions are parenthesized due to the low precedence.

Operator Precedence

Precedence Class	Operator	Associativity
postfix	*expr*++ *expr*--	
unary	++*expr* --*expr* +*expr* -*expr* !	
multiplicative	* / %	left to right
additive	+ -	left to right
relational	< > <= >=	left to right
equality	== !=	left to right
logical AND	&&	left to right
logical OR	||	left to right
conditional	$expr_1$? $expr_2$: $expr_3$	right to left
assignment	= += -= *= /= %=	right to left

Reading Homework

Textbook

- Section 3.3–3.12.
- Appendix C.

Internet

- Logical connective (`http://en.wikipedia.org/wiki/Logical_connective`).
- Assignment (`http://en.wikipedia.org/wiki/Assignment_(computer_programming)`).
- Short-circuit evaluation (`http://en.wikipedia.org/wiki/Short-circuit_evaluation`).

Self-test

- 3.1 – 3.30 (`http://tiger.armstrong.edu/selftest/selftest9e?chapter=3`).