

ITP20003 Java Programming

Programming Control Flow

(JIPP:Chapters 3-4)

This slide is primary taken from the instructor's resource of Java: Introduction to Problem Solving and Programming, 7th ed. by Savitch and then edited partly by Shin Hong

Programme

- Chapter 3. Flow of control
- Classwork 2
- Feedback on Classwork 1

Flow of Control



- *Control flow* is the order in which a program performs actions (i.e., executes instructions)
 - Sequential program: having one control flow
 - Concurrent program: having multiple control flows
- Control statements
 - (by default): next one (following one)
 - Jump: move the specified location, not simply the next one
 - Branching: chooses between two or more possible ways
 - Loop: repeats an action until a stopping condition occurs

The *if-else* Statement

- A branching statement that chooses between two possible actions.
- Syntax

```
if (Boolean_Expression)  
    Statement_1  
else  
    Statement_2
```

```
if (balance >= 0)  
    balance = balance + (INTEREST_RATE * balance) / 12;  
else  
    balance = balance - OVERDRAWN_PENALTY;
```

Introduction to Boolean Expressions

- The value of a *boolean expression* is either **true** or **false**.
- Examples
 - time < limit**
 - balance <= 0**

Java Comparison Operators

- Figure 3.4 Java Comparison Operators

Math Notation	Name	Java Notation	Java Examples
=	Equal to	==	<code>balance == 0</code> <code>answer == 'y'</code>
≠	Not equal to	!=	<code>income != tax</code> <code>answer != 'y'</code>
>	Greater than	>	<code>expenses > income</code>
≥	Greater than or equal to	>=	<code>points >= 60</code>
<	Less than	<	<code>pressure < max</code>
≤	Less than or equal to	<=	<code>expenses <= income</code>

Compound Boolean Expressions

- Boolean expressions can be combined using the "and" (**&&**) operator.

- Example

```
if ((score > 0) && (score <= 100))  
    ...
```

- Not allowed

```
if (0 < score <= 100)  
    ...
```

Compound Boolean Expressions

- Syntax
(Sub_Expression_1) && (Sub_Expression_2)
- Parentheses often are used to enhance readability.
- The larger expression is true only when both of the smaller expressions are true.

Compound Boolean Expressions

- Boolean expressions can be combined using the "or" (`||`) operator.

- Example

```
if ((quantity > 5) || (cost < 10))  
    ...
```

- Syntax

```
(Sub_Expression_1) || (Sub_Expression_2)
```

Compound Boolean Expressions

- The larger expression is true
 - When either of the smaller expressions is true
 - When both of the smaller expressions are true.
- The Java version of "or" is the *inclusive or* which allows either or both to be true.
- The *exclusive or* allows one or the other, but not both to be true.

Negating a Boolean Expression

- A boolean expression can be negated using the "not" (!) operator.

- Syntax

!(Boolean_Expression)

- Example

(a || b) && !(a && b)

which is the *exclusive or*

Negating a Boolean Expression

- Figure 3.5 Avoiding the Negation Operator

! (A Op B) Is Equivalent to (A Op B)	
<	>=
<=	>
>	<=
>=	<
==	!=
!=	==

Java Logical Operators

- Figure 3.6

Name	Java Notation	Java Examples
Logical <i>and</i>	&&	<code>(sum > min) && (sum < max)</code>
Logical <i>or</i>		<code>(answer == 'y') (answer == 'Y')</code>
Logical <i>not</i>	!	<code>!(number < 0)</code>

Compound Statements

- When a list of statements is enclosed in braces (`{ }`), they form a single *compound statement*.
- Syntax

```
{  
    Statement_1;  
    Statement_2;  
    ...  
}
```

Compound Statements

- A compound statement can be used wherever a statement can be used.
- Example

```
if (total > 10)
{
    sum = sum + total;
    total = 0;
}
```

Multibranch **if-else** Statements

- Syntax

```
if (Boolean_Expression_1)
    Statement_1
else if (Boolean_Expression_2)
    Statement_2
else if (Boolean_Expression_3)
    Statement_3
else if ...
else
    Default_Statement
```


Multibranch **if-else** Statements

- Equivalent code

```
if (score >= 90)
    grade = 'A';
else if ((score >= 80) && (score < 90))
    grade = 'B';
else if ((score >= 70) && (score < 80))
    grade = 'C';
else if ((score >= 60) && (score < 70))
    grade = 'D';
else
    grade = 'F';
```

The Conditional Operator

```
if (n1 > n2)
```

```
    max = n1;
```

```
else
```

```
    max = n2;
```

can be written as

```
max = (n1 > n2) ? n1 : n2;
```

- The **?** and **:** together are call the *conditional operator* or *ternary operator*.

The Conditional Operator

- The conditional operator is useful with print and print In statements.

```
System.out.print("You worked " +  
    ((hours > 1) ? "hours" ; "hour")) ;
```

The `exit` Method

- Sometimes a situation arises that makes continuing the program pointless.
- A program can be terminated normally by `System.exit(0)`.

The Type `boolean`

- The type `boolean` is a primitive type with only two values: `true` and `false`.
- Boolean variables can make programs more readable.

`if (systemsAreOK)`

instead of

`if ((temperature <= 100) && (thrust >= 120
00) && (cabinPressure > 30) && ...)`

Boolean Expressions and Variables

- Variables, constants, and expressions of type **boolean** all evaluate to either **true** or **false**.
- A boolean variable can be given the value of a boolean expression by using an assignment operator.

```
boolean isPositive = (number > 0);
```

```
...
```

```
if (isPositive) ...
```

Short-circuit Evaluation

- Sometimes only part of a boolean expression needs to be evaluated to determine the value of the entire expression.
 - If the first operand associated with an `||` is `true`, the expression is `true`.
 - If the first operand associated with an `&&` is `false`, the expression is `false`.
- This is called *short-circuit* or *lazy* evaluation.

Short-circuit Evaluation

- Short-circuit evaluation is not only efficient, sometimes it is essential!
- A run-time error can result, for example, from an attempt to divide by zero.

```
if ((number != 0) && (sum/number > 5))
```

- *Complete evaluation* can be achieved by substituting `&` for `&&` or `|` for `||`.

The **switch** Statement

- The **switch** statement is a multiway branch that makes a decision based on an *integral* (integer or character) expression
 - Java 7 allows String expressions
- A **switch** statement begins with the keyword **switch** followed by an integral expression in parentheses and called the *controlling expression*.

The **switch** Statement

- A list of cases follows, enclosed in braces.
- Each case consists of keyword **case** followed by
 - A constant called the *case label*
 - A colon
 - A list of statements
- The list is searched for a case label matching the controlling expression.

The **switch** Statement

- The action associated with a matching case label is executed.
- If no match is found, the case labeled **default** is executed
 - The **default** case is optional, but recommended, even if it simply prints a message.
- Repeated case labels are not allowed.

The **switch** Statement

- Syntax

```
switch (Controlling_Expression)
{
    case Case_Label:
        Statement(s) ;
        break;
    case Case_Label:
    ...
    default:
    ...
}
```

The **switch** Statement

- The action for each case typically ends with the word **break**.
- The optional **break** statement prevents the consideration of other cases.
- The controlling expression can be anything that evaluates to an integral type.

Enumerations

- Consider a need to restrict contents of a variable to certain values
- An enumeration lists the values a variable can have
- Example

```
enum MovieRating {E, A, B}  
MovieRating rating;  
rating = MovieRating.A;
```

Enumerations

- Now possible to use in a **switch** statement

```
switch (rating)
{
    case E: //Excellent
        System.out.println("You must see this movie!");
        break;
    case A: //Average
        System.out.println("This movie is OK, but not great.");
        break;
    case B: // Bad
        System.out.println("Skip it!");
        break;
    default:
        System.out.println("Something is wrong.");
}
```

Enumerations

- An even better choice of descriptive identifiers for the constants

```
enum MovieRating {EXCELLENT, AVERAGE, BAD}
```

```
rating = MovieRating.AVERAGE;
```

```
case EXCELLENT: ...
```


Java Loop Statements

- A portion of a program that repeats a statement or a group of statements is called a *loop*.
- The statement or group of statements to be repeated is called the *body* of the loop.
- A loop could be used to compute grades for each student in a class.
- There must be a means of exiting the loop.

The **while** Statement

- Also called a **while** loop
- A **while** statement repeats while a controlling boolean expression remains true
- The loop body typically contains an action that ultimately causes the controlling boolean expression to become false.

The **while** Statement

- Syntax

```
while (Boolean_Expression)  
    Body_Statement
```

or

```
while (Boolean_Expression)  
{  
    First_Statement  
    Second_Statement  
    ...  
}
```

The **do-while** Statement

- Also called a **do-while** loop
- Similar to a **while** statement, except that the loop body is executed at least once

- Syntax

do

Body_Statement

while (Boolean_Expression);

- Don't forget the semicolon!

Infinite Loops

- A loop which repeats without ever ending is called an *infinite loop*.
- If the controlling boolean expression never becomes false, a **while** loop or a **do-while** loop will repeat without ending.

Nested Loops

- The body of a loop can contain any kind of statements, including another loop.
- In the previous example
 - The average score was computed using a **while** loop.
 - This **while** loop was placed inside a **do-while** loop so the process could be repeated for other sets of exam scores.

The **for** Statement

- A **for** statement executes the body of a loop a fixed number of times.
- Example

```
for (count = 1; count < 3; count++)  
    System.out.println(count) ;
```

The **for** Statement

- Syntax

- **for** (**Initialization** ; **Condition** ; **Update**)
 Body_Statement
- **Body_Statement** can be either a simple statement or a compound statement in **{ }**
- <https://docs.oracle.com/javase/specs/jls/se8/html/jls-14.html#jls-14.14.1>

- Corresponding **while** statement

Initialization
while (**Condition**)
 Body_Statement_Including_Update

The **for** Statement

- Possible to declare variables within a **for** statement

```
int sum = 0;  
for (int n = 1 ; n <= 10 ; n++)  
    sum = sum + n * n;
```

- Note that variable **n** is local to the loop

The **for-each** Statement

- Possible to step through values of an enumeration type
- Example

```
enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}  
for (Suit nextSuit : Suit.values())  
    System.out.print(nextSuit + " ");  
System.out.println();
```

The **break** Statement in Loops

- A **break** statement can be used to end a loop immediately.
- The **break** statement ends only the **innermost** loop or switch statement that contains the **break** statement.
- **break** statements make loops more difficult to understand.
- Use **break** statements sparingly (if ever).

The **continue** Statement in Loops

- A **continue** statement
 - Ends current loop iteration
 - Begins the next one
- Text recommends avoiding use
 - Introduce unneeded complications