LESSON II.2
Java syntax basics (cont.)

Trinh Thanh TRUNG (MSc)
trungtt@soict.hust.edu.vn
094.666.8608

Objectives

- Develop knowledge about the syntax and semantic of Java programming language

Content

- Operators
- Expression
- Statement
- Block
- Control flow statements

I. OPERATORS

1. Classification
2. Assignment
3. Arithmetic and substitution
4. Increment and decrement
5. Relational and logic operator
6. Ternary and instance of operator
7. Shift operator

1. Operators classification

- Operators are special symbols that perform specific operations on one, two, or three operands, and then return a result.
- Unary operators
  - Increment and decrement operators
- Binary operators
  - Assignment operator
  - Arithmetic operators
  - Substitution operators
  - Relational operators
  - Logical operators
  - Conditional operators
  - **`instanceOf`** operator
- Ternary operators

  → Used to build value expressions.

2. Assignment operator

- This operator assigns the value of the expression to the variable.

  `variable = expression;`

  - The types of the variable and expression must be compatible.
  - The value of the whole assignment expression is the value of the expression on the right → possible to chain assignment expressions:

```
int x, y, z;
x = y = z = 2;
```

3. Arithmetic and substitution

Arithmetic operators

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulo (%)
- Example:

```
System.out.println(
" i % j = " +
(i % j));
```

Substitution operators

- Store the value of right side to left side
  - Addition and assignment (+=)
  - Subtraction and assignment (-=)
  - Multiplication and assignment (*=)
  - Division and assignment (/=)

4. Increment and Decrement

- Use only one operand
- The increment operator (++) adds one to its operand
  - 2 statements count++; and count=count+1; are functionally equivalent
- The decrement operator (--) subtracts one from its operand

5. Relational and Logic operators

Relational operators

- judge the equivalence or bigness and smallness of two expressions and variables.
- == (equal to)
- <= (less than or equal to)
- >= (greater than or equal to)
- != (not equal to)
- > (greater than)
- < (less than)

Logic operators

- && (logical AND)
- & (boolean logical AND)
- || (logical OR)
- | (boolean logical inclusive OR)
- ^ (boolean logical exclusive OR)
- ! (logical NOT)

6. Ternary and instanceOf operator

Ternary operator

- Used to evaluate boolean expressions
- Decide which value should be assigned to the variable
- Syntax:

```
conditional_expression ?
     value_if_true :
     value_if_false;
```

- Example:

```
int a = 10;
boolean b = (a == 1) ?
   true : false;
// b = false, since a is
// not equal to 1
```

instanceOf operator

- Judge if an object is a product generated from a class
- Syntax:

```
object instanceOf class
```

- Example

```
String name = 'James';
boolean result =
   name instanceOf String;
// True, since name is type of
// String
```

II. Expression

- Program
  - Package
    - Class
      - Methods/block
        - » Statement
          - **Expression**
            - Token

- Task: compute values
- Feature:
  - Be made up of variables, operators, and method calls
  - Be built according to the syntax of the language
  - Evaluate to a single value. The data type of this value depends on the elements used in the expression.

```
if (value1 == value2) System.out.println("value1 == value2");
int result = 1 + 2 * 3; // result is now 7
```

Operator order in expression

- Java operators are assigned precedence order.
- When two operators share an operand, the operator with the higher precedence goes first.
  - Example: since multiplication has a higher precedence than addition, so:
    - 1 + 2 * 3 is treated as 1 + (2 * 3)
    - 1 * 2 + 3 is treated as (1 * 2) + 3
- When two operators with the same precedence the expression is evaluated according to its associativity.
  - Example:
    - x = y = z = 17 is treated as x = (y = (z = 17)), since the assignment operator has right-to-left associativity.
    - 72 / 2 / 3 is treated as (72 / 2) / 3 since the division operator has left-to-right associativity.
- Precedence rules can be overridden by explicit parentheses.

Precedence and associativity of Java operators

Operator	Description	Level	Associativity
[] . () ++, --	access array element access object member invoke a method post-increment, post-decrement	1	left to right
++, -- +, - ! ~	pre-increment, pre-decrement unary plus, unary minus logical NOT bitwise NOT	2	right to left
() new	cast object creation	3	right to left
*, /, %	multiplicative	4	left to right
+ - +	additive string concatenation	5	left to right
<<, >>, >>>	shift	6	left to right

Precedence and associativity of Java operators

Operator	Description	Level	Associativity
<, <=, >, >= instanceof	relational type comparison	7	left to right
==, !=	equality	8	left to right
&	bitwise AND	9	left to right
^	bitwise XOR	10	left to right
|	bitwise OR	11	left to right
&&	conditional AND	12	left to right
||	conditional OR	13	left to right
?:	conditional	14	right to left
=, +=, -=, *= /=, %=, &= ^=, |=, <<= >>=, >>>=	assignment	15	right to left

III. Statement

- A statement forms a complete unit of execution.
- Two kinds of statements
  - Expression statement or single statement
  - Control flow statement

Expression statement

- Syntax:

  `expression;`
- Expression can be
  - Assignment expressions

    `aValue = 8933.234;`
  - Any use of increment (++) or decrement (--) operator

    `aValue++;`
  - Method calls

    `System.out.println("Hello World!");`
  - Object creation expressions

    `String[] array = new String[5];`

IV. Block

- A block is a group of zero or more statements between balanced braces
- A block can be used anywhere a single statement is allowed

```
class BlockDemo {
public static void main(String[] args) {
   boolean condition = true;
   if (condition) { // begin block one
        System.out.println("Condition is true.");
   } // end block one
   else { // begin block 2
        System.out.println("Condition is false.");
   } // end block 2
}
}
```

V. Control flow statements

- Coding a program means expressing the proposed algorithm by writing Java statements into a source file
- Without control flow, the interpreter would execute these statements in the order they appear in the source file, left-to-right and top-down
- Control flow statements regulate the order in which statements get executed
- Control statements are divided into three groups:
  - Selection statements: allow the program choosing different parts of the execution based on the result of an expression
  - Iteration statements: enable the program execution to repeat one or more statements
  - Jump statements enable your program to execute in a non-linear fashion

1. Selection statements

- Java selection statements allow to control the flow of program's execution based upon conditions known only during run-time.
- Java provides four selection statements:
  - **`if`**
  - **`if-else`**
  - **`if-else-if`**
  - **`switch`**

The if statement

- Syntax:

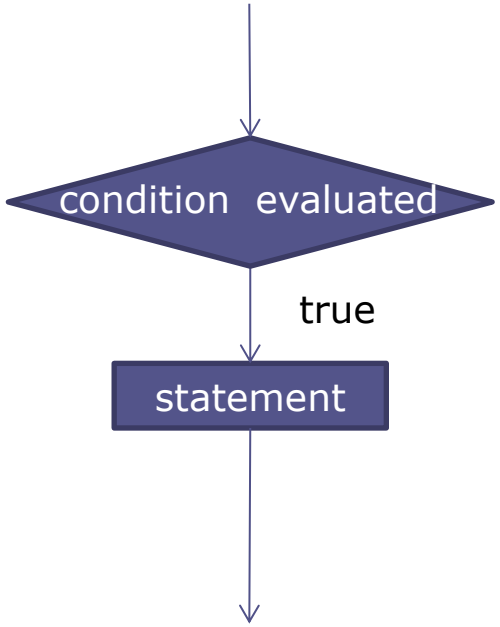
  `if (condition) statement;`

  - If **`condition`** is evaluated to **`true`**, execute **`statement`**, otherwise do nothing.
  - The **`condition`** must be of type **`boolean`**.
- The component statement may be:
  - simple:

  `if (condition) statement;`

  - compound:

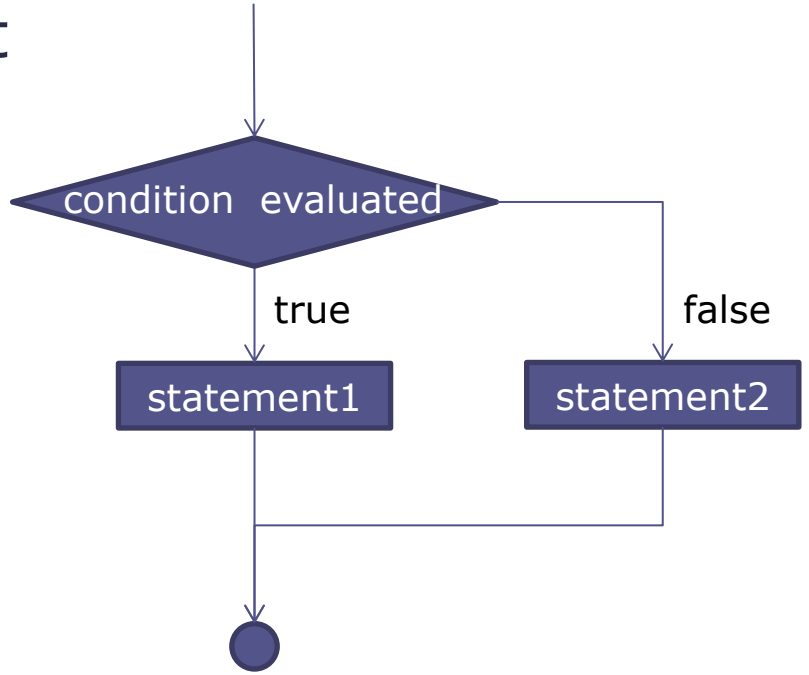
  `if (condition) {statement;}`



The if-else statement

- An else clause can be added to an if statement to make an if-else statement

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



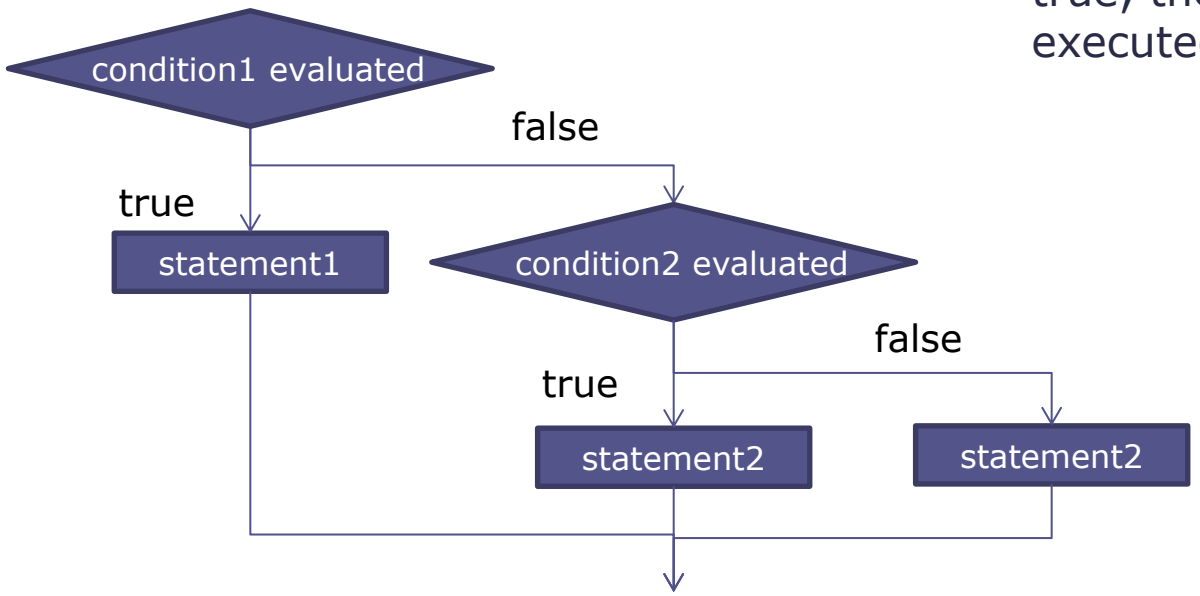

```
int max(int n1, int n2) {
   if (n1 >= n2) { return n1; }
   else { return n2; }
}
```

The if-else-if statement

- Syntax:

```
if (condition1) statement1
else if (condition2) statement2
…
else statement
```

- Semantics:
  - statements are executed top-down
  - as soon as one expressions is true, its statement is executed
  - if none of the expressions is true, the last statement is executed



Example

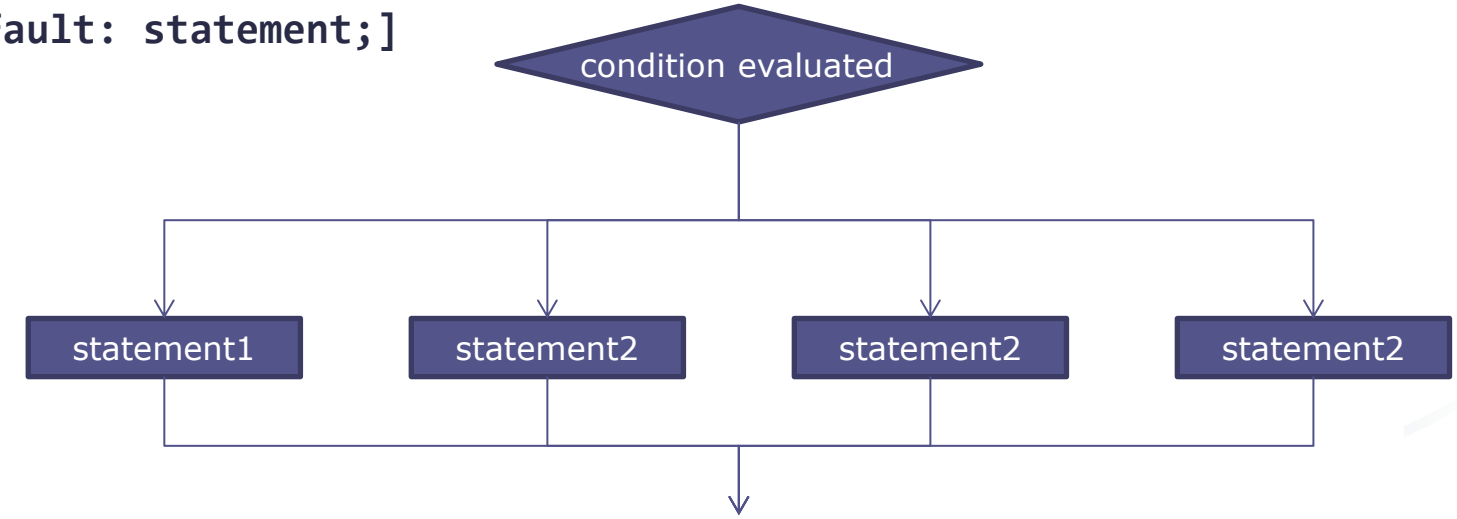
```
public class IfElseUsage {
    public static void main(String args[]) {
        int month = 12;
        String season;
        if (month == 12 || month == 1 || month == 2)
                season = "Winter";
        else if(month == 3 || month == 4 || month == 5)
                season = "Spring";
        else if(month == 6 || month == 7 || month == 8)
                season = "Summer";
        else if(month == 9 || month == 10 || month == 11)
                season = "Autumn";
        else season = "Bogus Month";
        System.out.println(
                "December is in the " + season + ".");
    }
}
```

The switch statement

- The general syntax of a switch statement is:

```
switch (condition) {
case value1: statement1; [break;]
case value2: statement2; [break;]
case value3: statement3; [break;]
…
[default: statement;]
}
```

- Assumptions:
  - condition must be of type byte, short, int or char
  - each of the case values must be a literal of the compatible type
  - case values must be unique

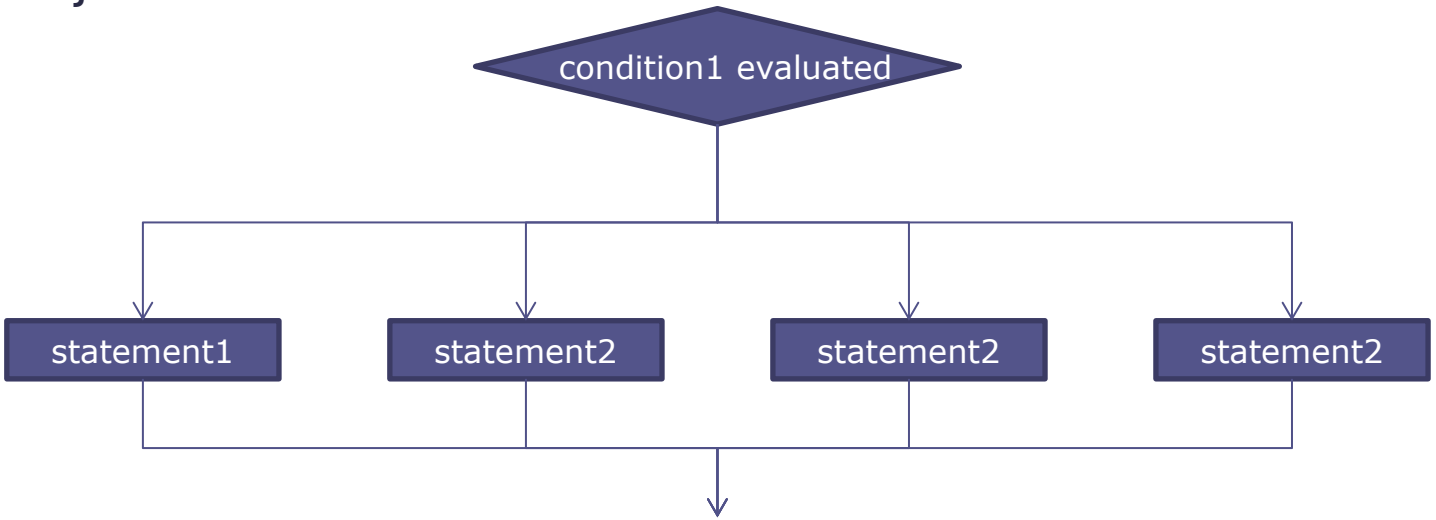


The switch statement

- The general syntax of a switch statement is:

```
switch (condition) {
case value1: statement1; [break;]
case value2: statement2; [break;]
case value3: statement3; [break;]
…
[default: statement;]
}
```

- Semantic:
  - Condition is evaluated; its value is compared with each of the case values
  - If a match is found, the statement following the case is executed
  - If no match is found, the statement following default is executed
  - The break statement terminates the enclosed iteration
  - Both default and break are optional.



Example

- What is the result of this code?

```
int month = 1;
switch(month){
   case 1:
       System.out.println("A Happy New Year!");
   case 12:
       System.out.println("Merry Christmas !");
       break;
}
```

Comparing switch and if

- Two main differences:
  - switch can only test for equality, while if can evaluate any kind of boolean expression
  - Java creates a “jump table” for switch expressions, so a switch statement is usually more efficient than a set of nested if statements

→ switch is a better alternative than if-else-if when the execution follows several branches depending on the value of an expression.

2. Iteration statements

- Java iteration statements enable repeated execution of part of a program until a certain termination condition becomes true.
- Java provides three iteration statements:
  - Iteration number is known in advance:

    `for`
  - Iteration number is not known in advance:
    - Termination condition is checked before the execution:

    `while`
    - Termination condition is checked after the execution:

    `do-while`

The for Statement

- Syntax:

```
for ( initialization ; condition ; increment )
    statement;
```

- Semantic:
  - Execute the initialization
  - Evaluate the termination condition :
    - if false, terminate the iteration
    - otherwise, continue to the next step
  - Execute the increment statement
  - Execute the statement component
  - control flow continues from the second step
- Example:

```
for (int count=1; count <= 5; count++)
    System.out.println (count);
```

The for each statement

```
for (variable : array) {
    body;
}
```

For each variable in an array, do the body statements

```
for (variable: collection) {
    body;
}
```

For each variable in a collection, do the body statements

```
public class Arithmetic {
public static void main(String[] args) {
    int data[] = {1, 80,22, 134, 0, 33,
                  93,45,33, 12};
    int sum = 0; // Total value
    float ave = 0;
    for(int num : data){
          sum += num;
    }
    ave = sum/10.0f;
    System.out.println(
          "Total sum :" + sum);
    System.out.println(
          "Mean :" + ave);
}
}
```

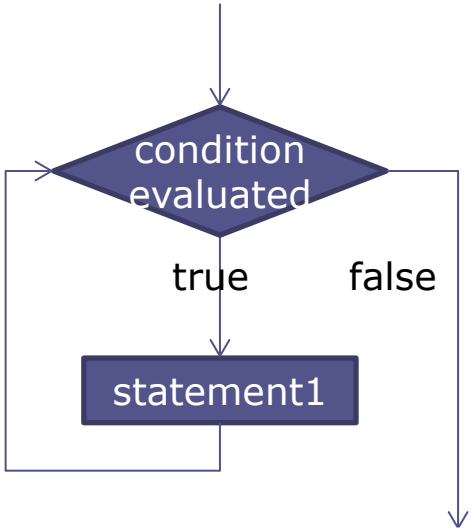
While-do and do-while statement

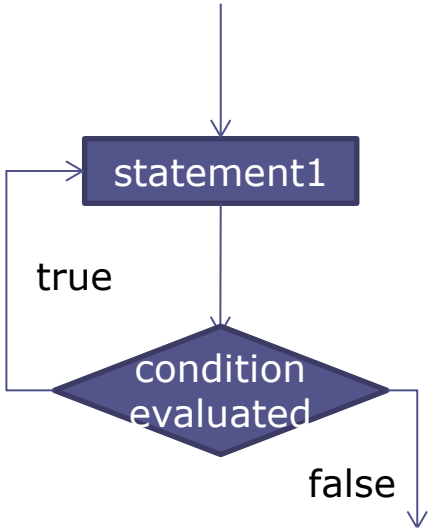
- While-do statement

```
while (condition) {
   body;
}
```

- Do-while statement

```
do {
   body;
} while (condition);
```





3. Jump statements

`break;`

- The break statement jumps to end and out of the enclosed compound statement → **`break`** must be the last statement in each compound statement.
- It transfers the control to the next statement outside the compound statement.

`break label;`

`label: { statements }`

- It transfers the control to the block of statements that is identified by **`label`**

`continue;`

- The continue statement immediately jumps to the head of the next iteration (if any) of the enclosed loop: **`for`**, **`while-do`** and **`do-while`**

  → **`continue`** does not apply to **`switch`** statement or block Statement.

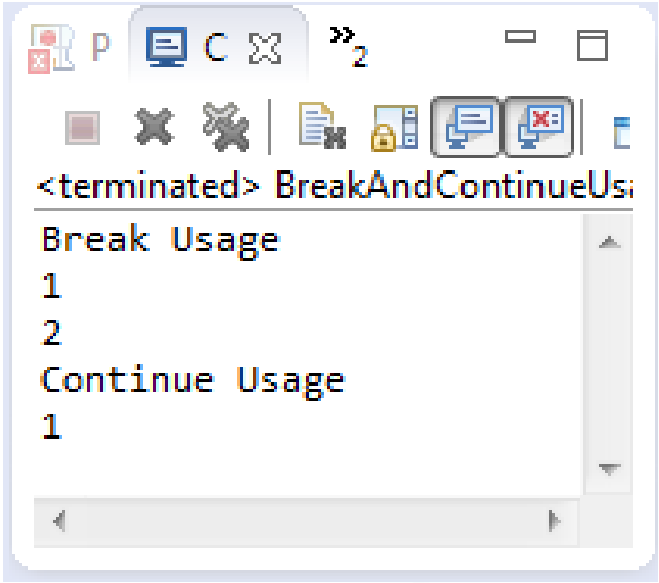
`continue label;`

- The continue statement immediately jumps to the head of the enclosing loop that is identified by **`label`**

Example

- What is the output of the following code?

```
public class BreakAndContinueUsage {
   public static void main(String[] args) {
        System.out.println("Break Usage");
        for(int i = 1; i < 100; i++){
                System.out.println(i);
                if(i > 1) break;
        }
        System.out.println("Continue Usage");
        for(int i = 1; i < 100; i++){
                if(i > 1) continue;
                System.out.println(i);
        }
   }
}
```



The return statement

- The return statement is used to return from the current method: it causes program control to transfer back to the caller of the method.

`return;`

  - return without value

`return expression;`

  - Return with the result of the expression
  - The type of returned value must match with the declared return type

- Inside a method, statements after the return statement are not executed

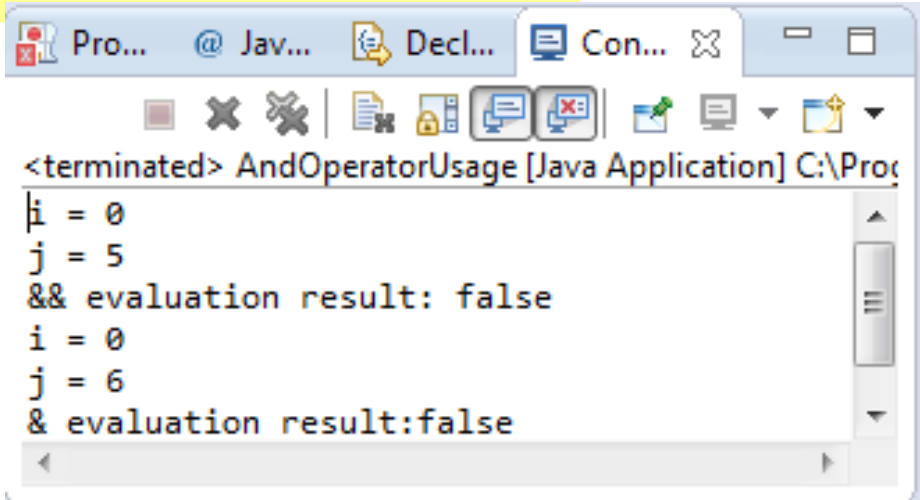
Quiz - Operator

1. Compile and run the program AndOperatorUsage.
2. Explain the difference between two AND operators &&, &

```
public class AndOperatorUsage {
     public static void main(String[] args) {
          int i = 0; int j = 5;
          boolean test = false;
          //&& operator
          test = (i > 5) && (j++ > 4);
          System.out.println("i = " + i); System.out.println("j = " + j);
          System.out.println("&& evaluation result: " + test);
          //& operator
          test = (i > 5) & (j++ > 4);
          System.out.println("i = " + i); System.out.println("j = " + j);
          System.out.println("& evaluation result:" + test);
     }
}
```

Quiz 1 - Solution

```
public class AndOperatorUsage {
     public static void main(String[] args) {
          int i = 0; int j = 5;
          boolean test = false;
          //&& operator
          test = (i > 5) && (j++ > 4);
          System.out.println("i = " + i); System.out.println("j = " + j);
          System.out.println("&& evaluation result: " + test);
          //& operator
          test = (i > 5) & (j++ > 4);
          System.out.println("i = " + i); System.out.println("j = " + j);
          System.out.println("& evaluation result:" + test);
     }
}
```



Quiz 2 - Solution

Logical AND operator (&&)

- support partial evaluations (short-circuit evaluations)
- **exp1 && exp2**
  - Evaluate the expression exp1
  - If exp1 est false: immediately return a false value
    - The operator never evaluates exp2, because the result will be false regardless of the exp2 value

Boolean logical AND operator (&)

- Does not support partial evaluations
- **exp1 & exp2**
  - Evaluate the expression exp1
  - Evaluate the expression exp2
  - Return the result of the operator

Quiz – control flow

3. Using switch statement instead of if-then-else, write the class SwitchUsage to perform the same operations as the following class:

```
public class IfElseUsage {
     public static void main(String args[]) {
          int month = 12;
          String season;
          if (month == 12 || month == 1 || month == 2)
                    season = "Winter";
          else if(month == 3 || month == 4 || month == 5)
                    season = "Spring";
          else if(month == 6 || month == 7 || month == 8)
                    season = "Summer";
          else if(month == 9 || month == 10 || month == 11)
                    season = "Autumn";
          else season = "Bogus Month";
          System.out.println("December is in the " + season + ".");
     }
}
```

Quiz 3 - solution

```
public class SwitchUsage {
     public static void main(String args[]) {
          int month = 12;
          String season;
          switch (month) {
          case 12: case 1:
          case 2: season = "Winter"; break;
          case 3: case 4:
          case 5: season = "Spring"; break;
          case 6: case 7:
          case 8: season = "Summer"; break;
          case 9: case 10:
          case 11: season = "Autumn"; break;
          default: season = "Bogus Month";
          }
          System.out.println("December is in " + season + ".");
     }
}
```

Quiz – control flow

4. Compile and run the following program. Explain the result.
5. Modify the program so that break statement terminates the outer iteration just after the first passing.

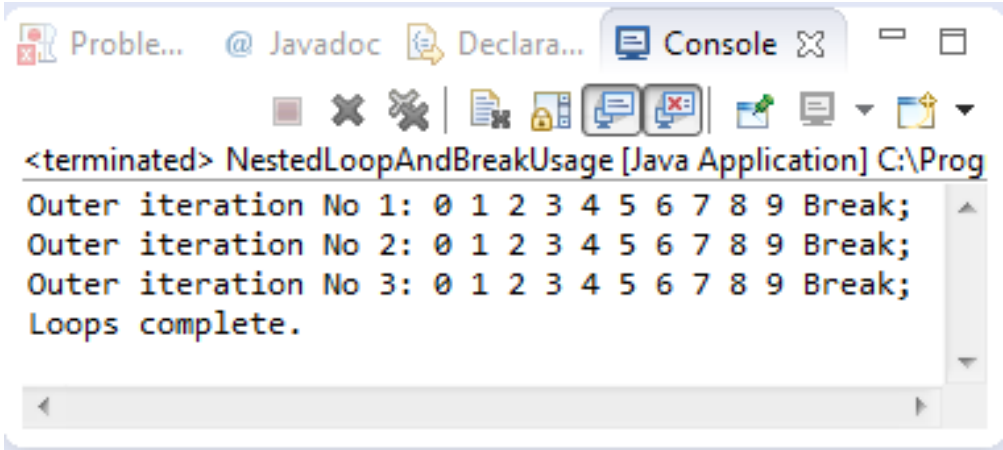
```
public class NestedLoopAndBreakUsage {
     public static void main(String args[]) {
          for (int i = 0; i < 3; i++) {
               System.out.print("Outer iteration No " + (i + 1) +
               ": ");
               for (int j = 0; j < 100; j++) {
                    if (j == 10) {
                         System.out.println("Break;");
                         break;
                    }
               System.out.print(j + " ");
             }

          }
          System.out.println("Loops complete.");
     }
}
```

Quiz 4 – solution

- The break statement terminates the inner iteration after 10 passing.
- The outer iteration is performed normally.

→ If a break statement is used inside nested loops, break will only terminate the innermost iteration

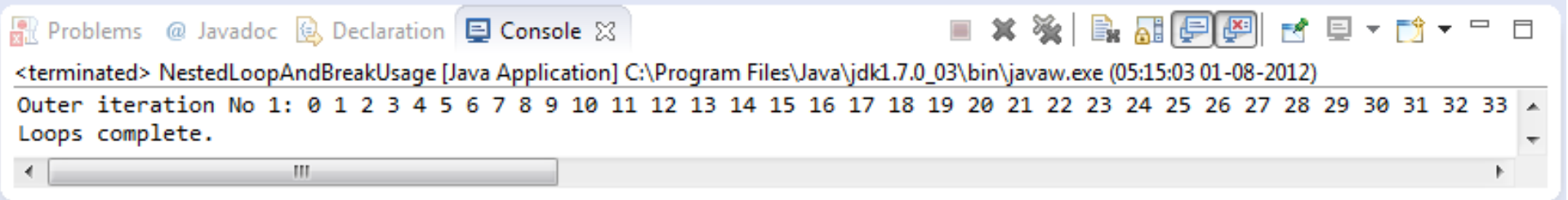
```
public class NestedLoopAndBreakUsage {
      public static void main(String args[]) {
            for (int i = 0; i < 3; i++) {
                  System.out.print("Outer iteration No " + (i + 1) + ": ");
                  for (int j = 0; j < 100; j++) {
                        if (j == 10) {
                              System.out.println("Break;");
                              break;
                        }
                  System.out.print(j + " ");
                }

            }
            System.out.println("Loops complete.");
      }
}
```



Quiz 5 – solution

- Move the break statement out of the inner iteration.
- Place it as the last statement of the outner iteration.
- Now the inner iteration terminates after 100 passings, the outer iteration is terminated after the first passing.

```
public class NestedLoopAndBreakUsage {
     public static void main(String args[]) {
          for (int i = 0; i < 3; i++) {
               System.out.print("Outer iteration No " + (i + 1) + ": ");
               for (int j = 0; j < 100; j++) {
               System.out.print(j + " ");
            }
            System.out.println("Break;");
            break;
          }
          System.out.println("Loops complete.");
     }
}
```



Review

- Operators: perform specific operations
  - Unary
  - Binary
  - Ternary
- Expression, statement and block
- Control flow structures: transfer the program flow
  - Sequence
  - Selection
  - Iteration
  - Jump