



National University of Ireland, Galway  
*Ollscoil na hÉireann, Gaillimh*

# CT 229

## Java Syntax

# Lab Assignments

- Assignment Due Date: **Oct 1<sup>st</sup>**
  
- Before submission make sure that the name of each .java file matches the name given in the assignment sheet!!!!
  
- Remember: Electronic Submission @  
<http://ecrg-vlab01.it.nuigalway.ie/uploader/Submissions.html>
  
- Where does Eclipse put my .java files????
  - By default Eclipse will put your .java files in your F:\ drive.
  - Look in your **F:\My Documents\workspace\<name of project>**

# Tutorials

□ **Tutorial Commence on the Tues 3<sup>rd</sup> of October and Thurs 5<sup>th</sup> of Oct.**

□ **Tutorials 1hr/Week**

- BE(EE) Tues AC213 (12-1)
- BE(E&CE) Tues AC213 (12-1)
- BSc(IT) Thurs IT203 (12-1)
- BSc(P&A) Thurs IT203 (12-1)

# Lab Issues

- How to use the Java API?
- Code Completion in Eclipse
- Concatenation of Strings and Variables

# Operands and Operators??

## □ Operand:

- A quantity upon which a mathematical operation is performed
- Can be a variable or expression

## □ An operator performs a function on 1, 2 or 3 operands

- Unary (`num1++`) , binary (`num1+num2`) or ternary operator (`grade = (assignmentMark>40)?"pass":"fail" )`)

## □ Operator types:

- Arithmetic
- Relational: relationship between operands (compare size, etc.)
- Boolean: **and**, **or**
- Assignment: assign value to variable
- Bitwise: manipulate individual binary bits
- Others: miscellaneous

# Arithmetic Operators

- Addition  $5 + 2$  is 7
- Subtraction  $5 - 2$  is 3
- Multiplication  $5 * 2$  is 10
- Division  $5 / 2$  is 2
- Remainder  $5 \% 2$  is 1
- Increment  $++x$  (adds 1 to value of  $x$ )
- Decrement  $--x$  (subtracts 1 from  $x$ )
  - Have pre-increment and post-increment (and decrement)
  - Pre  $++x$  carried out before expression is evaluated
  - Post  $x++$  carried out after expression is evaluated

# Pre and Post Increment Example

## Example

```
int x = 5;  
System.out.println("Value of x is "+ (++x) );  
System.out.println("Value of x is "+ (x++) );  
System.out.println("Value of x is "+ (x) );
```

## Output

```
The value of x is 6  
The value of x is 6  
The value of x is 7
```

# Relational Operators

- Compare two numerical operands; return true or false
- Used with if and other decision-making statements

Operator	Use	Returns true if
>	op1 > op2	op1 is greater than op2
>=	op1 >= op2	op1 is greater than or equal to op2
<	op1 < op2	op1 is less than op2
<=	op1 <= op2	op1 is less than or equal to op2
==	op1 == op2	op1 and op2 are equal
!=	op1 != op2	op1 and op2 are not equal

# Boolean Operators

- Used with relational operators; normally combine results of relational expressions to form more complex expressions

Operator	Use	Returns true if
&&	op1 && op2	op1 and op2 are both true [and]
	op1    op2	either op1 or op2 is true [or]
!	! op	op is false [not]
&	op1 & op2	op1 and op2 are both true [and]
	op1   op2	either op1 or op2 is true [or]
^	op1 ^ op2	Just one of op1 and op2 is true [xor]

# Boolean Operators: Summary

<b>a</b>	<b>b</b>	<b>a &amp;&amp; b</b>	<b>a    b</b>	<b>a ^ b</b>	<b>! a</b>
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

Note: C uses 0 and 1 rather than true and false for Boolean operations

# Boolean Operators

- What is the difference between
  - && and &
  - || and |
- The operators && and || perform short circuit logical expressions. That is they only evaluate the second operand if required.

## Example

```
boolean condition1 = false;
```

```
boolean condition2 = true;
```

```
if ( (condition1) && (condition2) )
```

```
if ( (condition1) & (condition2) )
```

Example: booleanExample.java

# Assignment Operators

- **Several assignment operators apart from simple =**
  - All are abbreviated forms of simple assignment
  - One corresponding to each binary arithmetic operator
- **Left side of assignment must always be a variable (l-value)**

Operator	Use	Equivalent To
<code>+=</code>	<code>op1 += op2;</code>	<code>op1 = op1 + op2;</code>
<code>-=</code>	<code>op1 -= op2;</code>	<code>op1 = op1 - op2;</code>
<code>*=</code>	<code>op1 *= op2;</code>	<code>op1 = op1 * op2;</code>
<code>/=</code>	<code>op1 /= op2;</code>	<code>op1 = op1 / op2;</code>
<code>%=</code>	<code>op1 %= op2;</code>	<code>op1 = op1 % op2;</code>

# Bitwise Operators

- Operate on int (or long or byte) variables
  - Treat number as sequence of binary digits (bits)

Op.	Use	Equivalent To
&	op1 & op2	bitwise <b>and</b>
	op1   op2	bitwise <b>or</b>
^	op1 ^ op2	bitwise <b>xor</b>
~	~op2	bitwise complement
>>	op1 >> op2	shift bits of <b>op1</b> right by distance <b>op2</b>
<<	op1 << op2	shift bits of <b>op1</b> left by distance <b>op2</b>
>>>	op1 >>> op2	shift bits of <b>op1</b> right by distance <b>op2</b> (unsigned)

**Note:** &, | and ^ can be boolean or bitwise operators, depending on whether operands are boolean or integer

# Bitwise Operators

Like boolean operations, except  
with 0 for false, 1 for true

- Integers expressed in binary form
- Calculation done bit-by-bit

<b>op1</b>	<b>op2</b>	<b>op1 &amp; op2</b>	<b>op1   op2</b>	<b>op1 ^ op2</b>	<b>~op1</b>
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

# Example Bitwise Operations

Example: bitwise.java

## Byte-Sized Example

**NOT**

```
~10110010
-----
01001101
~178 = 77
```

**AND**

```
10110010
& 11011011
-----
10010010
178 & 219 = 146
```

**XOR**

```
10110010
^ 11011011
-----
01101001
178 ^ 219 = 105
```

**OR**

```
10110010
| 11011011
-----
11111011
178 | 219 = 251
```

# Bitwise Right Shift Operators

## □ Arithmetic or signed right shift ( $\gg$ ) is used as follows:

– First operand is divided by 2 raised to the power of the second operand.

–  $128 \gg 1$  returns  $128 / 2^1 = 64$   $\longrightarrow$

–  $256 \gg 4$  returns  $256 / 2^4 = 16$

–  $-256 \gg 4$  returns  $-256 / 2^4 = -16$

*10000000  $\gg$  1*

*Answer: 1000000*

• The sign is copied during the shift

## □ A logical or unsigned right-shift operator ( $\ggg$ ) is:

– Used for bit patterns

– The sign bit is not copied during the shift

–  $178 \ggg 2 \Rightarrow 10110010 \ggg 2 == 101100$

# Bitwise Left-Shift Operator

□ The left-shift bitwise operator ( $\ll$ ) is used as follows:

– The first operand is multiplied by two and raised to the

–  $128 \ll 1$  returns  $128 * 2^1 = 256$

•  $10000000 \ll 1 \Rightarrow 100000000$

–  $16 \ll 2$  returns  $16 * 2^2 = 64$

# Other Operators

Operator	Description
<b>+</b>	Concatenates (joins) two String operands <code>name = firstName + surName;</code>
<b>? :</b>	Shortcut <b>if-else</b> statement <code>result = (mark &gt;= 40) ? "Pass" : "Fail";</code>
<b>[ ]</b>	Used to declare arrays, create arrays, and access array elements
<b>.</b>	Used to form qualified names
<b>( type )</b>	Casts (converts) a value to the specified type
<b>instanceof</b>	True if first operand (object name) is instance of second operand (class name)