COMP6700/2140 Operators, Expressions, Statements

Alexei B Khorev and Josh Milthorpe

Research School of Computer Science, ANU

3 March 2017

Operators

- Assignment
  `=`
- Arithmetic
  `+ - * / %`
- Unary
  `+ - ++ -- !`
- Equality, relational, conditional and `instanceof`
  `== != > >= < <= && || instanceof`
- Bitwise
  `~ & ^ | << >> >>>`

Arithmetic and String Operators

Integer arithmetic is modular two's complement arithmetic, *ie*, if the value exceeds the range of its type (`int` or `long`), it is reduced modulo the range (it *wraps*), so that the integer arithmetic never results in an overflow or an underflow.

Integer division *truncates toward zero* (`5/2 = 2`, and `-5/2 = -2`). Division and remainder obey this rule:

```
(x/y)*y + (x%y) == x // hence 5%2 = 1 and -5%2 = -1
```

Division (or remainder) by zero is illegal for integer arithmetic and throws *ArithmeticException*.

Character arithmetic is an integer arithmetic after the `char` is implicitly converted to `int`.

Floating point arithmetic is more lengthy subject (we only scratched the surface in **J5**).

***String* concatenation**: The addition operator also applies to two *String* objects, returning a string with the joined content:

```
String s1 = "French Connection"; String s2 = " UK";
System.out.print("s1 + s2 = " + s1 + s2);
```

Unlike for primitive types, string "addition" is *not commutative*.

Other Operators

- Comparing test operators: `<` , `>`, `<=`, `>=`, `==`, `!=`
  (Non-)Equality (`!=` and `==`) operators apply to all types, others — to numbers and characters
- Logical operators:

`&` (bitwise `AND`)	`|` (bitwise inclusive `OR`)	`^` (bitwise exclusive `XOR`)
`&&` (conditional `AND`)	`||` (conditional `OR`)	`!` (logical negation)

Conditional operators "short-circuit": they do not evaluate the right-hand side if it is not required.

- Increment and decrement operators (in **J9**)
- Conditional ternary operator (in **J9**)
- Type check: `(reference) instanceof (ClassName/InterfaceName)` (evaluates to *boolean*)
- Bit manipulation operators: first three of the logical operators plus three shift operators
- Assignment operator — `x = 5;`
- Type conversion operator `(convert-to-type)var: char c = (char)x;`

Operators on Objects

Objects (more details in **O2**) need to be *created* and *operated* upon.

Object creation operator:

```
String str = new String("I love Perignon!");
```

The operator `new` allocates memory for a new object and return a reference to it,

Method invocation: Objects represent chunks of data *and* operations which involve these data. These operations are carried out by *invoking* or *calling* a corresponding method by using the object reference o — the invocation operator is a dot which follows the object reference:

```
o.methodname(x); // x is a parameter passed to the method
System.out.println("I am confused"); // "out" is output stream object
```

Field access: Data associated with the object o can be read by using the same dot operator followed by the name of the *field* representing the data:

```
int age = o.age;
System.out.println("The object name is " + o.name);
```

Expressions

- A construct that evaluates to a single value
- Made up of
  - variables
  - operators
  - method invocations
- Compound expressions follow precedence rules
  - Use parentheses (clarity, disambiguation)
- `2 * Math.PI * Math.sqrt(Math.pow(r, 3.0) / (G * (earth.mass + moon.mass)))`

$$T = 2\pi\sqrt{\frac{r^3}{G(M_1 + M_2)}}$$

Operator Precedence

Precedence	Type	Operators
1.	postfix operators	[] . (params) expr++ expr--
2.	unary operators	++expr --expr +expr -expr ~ !
3.	creation or cast	new (type)expr
4.	multiplicative	* / %
5.	additive	+ -
6.	shift	<< >> >>>
7.	relational	< > <= >= instanceof
8.	equality	== !=
9.	bitwise AND	&
10.	bitwise exclusive OR	^
11.	bitwise inclusive XOR	|
12.	conditional AND	&&
13.	conditional OR	||
14.	ternary (elvis)	?:
15.	assignment	= += -= *= /= %= >>= <<= >>>= &= ^= |=

Practice with Expressions

Which expressions are legal and what do they evaluate to?

```
3 << 2L - 1;
(3L << 2) -1;
10 < 12 == 6 > 17;
10 << 12 == 6 >> 17;
13.5e-1 % Float.POSITIVE_INFINITY;
Float.POSITIVE_INFINITY + Double.NEGATIVE_INFINITY;
Double.POSITIVE_INFINITY - Float.NEGATIVE_INFINITY;
0.0 / -0.0 == -0.0 / 0.0; // very important why the answer is what it is?
Integer.MAX_VALUE + Integer.MIN_VALUE;
Long.MAX_VALUE + 5;
(short) 5 * (byte)  10;
(i < 15) ? 1.72e3f : 0; // assuming int i is declared, but not initialised
i++ + i++ + --i;   // i = 3 before statement executed, what is i after?
```

(**hint** the table on Operator Precedence slide above can help; if perplexed, type them into a program, run and try to understand why the results are what they are)

Statements

- A complete unit of execution
- Expression statements (expressions made into statements by terminating with ';'):
  - Assignment expressions
  - Use of ++ or --
  - Method invocations
  - Object creation expressions
- Declaration statements
- Control flow statements

```
double T;
if (careAboutRelativity) {
  einstein.ask();
} else {
  T = 2 * Math.PI * Math.sqrt(Math.pow(r, 3.0) / (G * (earth.mass + moon.mass)));
}
```

Blocks

- Zero or more statements between balanced braces ('{' and '}')
- Can be used anywhere a single statement can
- Blocks define *scope*

Further Reading

- *Core Java for the Impatient*, Ch. 1.4, 1.5
- Expressions, Statements, and Blocks in *Java Tutorial*