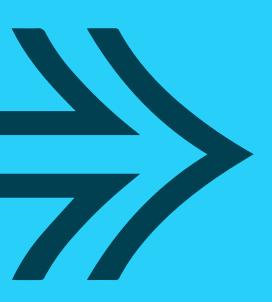
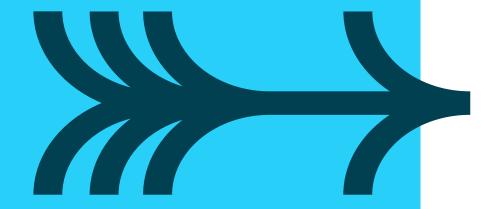


# Java Language basics



### **CONTENTS**



### Objectives

 To look at some fundamental Java language constructs

#### Contents

- Language Basics
- Declaring and initialising variables
- Literals why are they important?
- Mathematical operators (+ \* / etc.)

### Hands on Labs (2)

- Declaring variables
- Doing Maths

# **Expressions and Statements**

- An expression is anything that evaluates to a value
  - 12 + 7
  - x = 12 + 7
  - x > y
- A statement is:
  - An executable instruction to the compiler. It ends with a semicolon.
- A statement can be:
  - an expression statement, e.g. x = 12 + 7;
  - a declaration statement, e.g. int a;
  - a control statement, e.g. if, while etc.

### **Comments**

- In Java, a comment may be:
  - A block comment
  - Everything between /\* and \*/ is ignored by the compiler

```
/*
This is a multi-line comment ...
*/
```

- A line comment
- Everything to the right of // is ignored by the compiler

Comments are used to provide explanations of lines and blocks of code.

Comments are also used to "comment out" lines or blocks of code in order to temporarily inactivate them.

### Comments

- A third type of comment is used for program documentation
- This is outside the scope of the course

Please see <a href="https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html">https://docs.oracle.com/javase/8/docs/technotes/tools/windows/javadoc.html</a>
for Javadoc comments and <a href="https://docs.microsoft.com/en-us/dotnet/csharp/codedoc">https://docs.microsoft.com/en-us/dotnet/csharp/codedoc</a>
for XML Documentation in .NET

# Naming types, identifiers in Java

### Naming convention

- Start with a letter of the alphabet (recommended) or underscore (not recommended)
- Cannot start with a number
- Cannot use a reserved word
- Java is case sensitive

### Naming conventions

- Pascal<u>C</u>asing Types (classes)
- camel<u>C</u>asing methods, all variables
- packages are in <u>lowercase</u> e.g. java.util

```
String student; ✓
String student1; ✓
String Student; ✓ but 🄽
String 1student; 😕
String student; ✓
String studentName; ✓
String student_name; ✓
String public; *
String publicStudent; ✓
```

## Variables (Symbolic name for an address in memory)

- Must be declared with a type before use
- Local variables must be initialised before being read from

```
public void main(String[] args) {
  int myAge;
  boolean answer = true;
  String myName = "Samantha";
  int i = 0, j;

  myAge = 21;
  System.out.println(i);  
  System.out.println(j);  
  // not initialised
}
```

Local variables (defined inside a method) are only visible inside the method

# Pre-defined in-built primitive data types

```
byte eightBit;
short sixteenBit;
int thirtyTwoBit;
long sixtyFourBit;
```

```
float x32;
double x64;

Float limits 7 digits of precision
Double limits 16 digits of precision
```

```
char initial; // 16 bit Unicode character
boolean isActive; // true or false
initial = 'M';
isActive = true;
```

```
Integer.MIN_VALUE Integer.MAX_VALUE

Float.MIN_VALUE Float.MAX_VALUE etc.
```

# Standard Mathematical Operators

```
int x = 4;
x = x + 5;
x = x / 2;

int y = ( 20 % 7 );

double d = 9.0;
d = d / 2;

// x is 9
// x is 4

// y is 6

// d is 4.5
```

```
int x = 9;
int y = (x / 2); // y is 4
```

```
+ addition
- subtraction
* multiplication
/ division
% modulus division
```

# Compound (Mathematical) Operators

Each mathematical operator can be combined with "="

```
int z = 8;
z *= 2;  // z now 16
```

```
System.out.println(z % 5); // displays 1 but z still 16
z %= 5; // z now 1
```

```
x + 345;  // an expression not a statement x += 345;  //
```

# Pre & Post-fix ++ and -- Operators

```
int var1 = 0;
                        var1++; // var1 now 1
Identical statements
                        ++var1; // var1 now 2
                        int var1 = 0;
  varl and var2 are
                        int var2 = ++var1; // pre-fix
      both 1
                        int var1 = 0;
     var1 will be 1
     var3 will be 0
                        int var3 = var1++; // post-fix
int x = 10;
// passes the value 10 to println
print(x++); // displays 10
                               and then increments x
print(x); // displays 11
void print(object x) {
  System.out.println(x.ToString());
```

# Integer Arithmetic & Casting

### You can cast any numeric type to another type using explicit casting

But Java does have implicit casting on its own (see examples)

```
int x = 4;
                                 Implicit casting
double dbl = x;
double dbl = 4.5;
                                   An Integer cannot hold a double 🗶
int x = db1;
long lng = 5;
                                    An Integer cannot hold a long 🗶
int x = lnq:
double dbl = 4.9;
                                      Explicit casting is required
int x = (int)db1;
                                     x will be 4 (the decimal point is lost)
                                              Y will be 5
long lng = 5;
int y = (int) \ln q:
```

# Casting strings

Must use a parse method to cast strings to numeric types



```
String no = "123";
int x = (int)no;
```

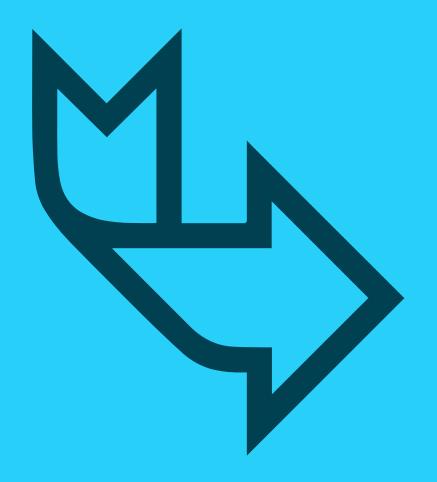
```
String no = "123.45";
double d = (double)no;
```

```
int x = Integer.parseInt(no);
double d = Double.parseDouble(no);
float f = Float.parseFloat(no);
```

### Review



- Must be initialised before being read from
- Literals
  - Numeric, boolean, character and String
- Mathematical operators
  - Compound operators
  - Casting to perform narrowing conversions
- Expressions
  - Always have a resulting type



# Hands on Lab

- Exercise 2
  - Part I 'Declaring Variables'
  - Part II 'DoingMaths'

# More on arithmetic operations

Any operation between two numeric variables that are smaller than an integer results in an integer.

```
byte b1=1, b2=2, b3;
b3 = b1 + b2;

Type mismatch: cannot convert from int to byte
2 quick fixes available:

() Add cast to 'byte'

Change type of 'b3' to 'int'
```