3 INTS AND DOUBLES

Terry Marris 8 April 2001

3.1 OBJECTIVES

By the end of this lesson the student should be able to

- explain what the int data type means
- explain what the double data type means
- picture and declare variables in Java
- understand and use arithmetic operator precedence
- understand and use arithmetic operator associativity

3.2 PRE-REQUISITES

The student should be comfortable with ordinary arithmetic operations and simple algebra.

3.3 PREVIEW

We introduce the *int* and *double* Java data types. We see how to perform simple arithmetic calculations. We examine the concepts of precedence and associativity. We see how to declare and initialise variables in Java.

3.4 INTEGERS

Whole numbers such as ..., -3, -2, -1, 0, 1, 2, 3, 4, ... are known as integers. Integers do not have a decimal point or a fractional part.

The Java *int* data type represents integers between (approximately) ± 2 billion.

The usual arithmetic operations may be performed on *int* values.

+	addition
-	subtraction
*	multiplication
/	div (integer division)
%	mod (remainder after integer division)

Figure 3.1 *Integer Arithmetic Operators*

div and mod require some explanation.

Do you remember in the junior school doing something like

7 ÷ 3 = 2 remainder 1

(7 apples divided equally between 3 children means that each child gets 2 apples with one apple left over.)

Well, the ÷ corresponds to the div operator, /, and the remainder to the mod operator, %. So, in Java,

7 / 3 gives a result of 2 (the fractional part is cut off or truncated)

And 7 % 3 gives a result of 1 (the remainder after dividing 7 by 3).

Equally, 3 / 4 is 0.

And 3 % 4 is 3.

3.5 REALS

Numbers that contain a decimal point such as 3.1416, -2.7183 and 23.0 are known as real numbers.

The Java *double* data type represents real numbers between (approximately) $\pm 1.8 \times 10^{308}$ (to 15 significant figures).

The usual arithmetic operations can be performed on *double* values.

+	addition
-	subtraction
*	multiplication
/	division

Figure 3.2 *Double Arithmetic Operators*

Division is what you would normally expect. For example

7.0 / 4.0 = 1.75

3.6 PRECEDENCE

What is the result of

$$2 + 3 \times 4$$

Most students will (correctly) say 14 because they performed the multiplication before the addition. We say that × has a higher precedence than +. Precedence is the order in which operations are performed regardless of the order in which they are written.

The rules of arithmetic operator precedence are shown in the following table:

operator	precedence
( )	highest
++ --	
* / %	
+ -	
=	lowest

Figure 3.3 *Arithmetic Operator Precedence*

So, in this expression

```
5.0 / 9.0 * (f - 32.0)
```

the brackets (f - 32.0) are evaluated (i.e. worked out) first, then the whole expression is evaluated from left to right.

And in this expression

```
c = 5.0 / 9.0 * (f - 32.0);
```

the assignment operator, =, is done last of all. This means that *c* is assigned or given the value of *5.0 / 9.0 * (f - 32.0)* after it has been worked out.

The operators ++ and -- are discussed in § 3.11. (§ means Section.)

3.7 ASSOCIATIVITY

Look at

```
5.0 / 9.0 * 180.0
```

The operators / and * have equal precedence. So, which one is evaluated first? The convention is that they are evaluated from left to right, in the order as written.

So, in 5.0 / 9.0 * 180.0 the division is done before the multiplication.

But in 5.0 * 18.0 / 9.0 the multiplication is done before the division.

As a general rule, arithmetic expressions should be written so that multiplications are evaluated before divisions; this minimises rounding errors and maximises accuracy.

Associativity is the direction in which expressions are evaluated after taking precedence into account. Associativity rules are shown in the following table.

operator			associativity
( )			left to right
++	--		right to left
*	/	%	left to right
+	-		left to right
=			right to left

Figure 3.4 *Arithmetic Operator Associativity*

This is why in expressions which involve the assignment operator, =, as in

```
c = 5.0 / 9.0 * (f - 32.0);
```

the assignment is always done from right to left. In other words, you work out the answer to 5.0 / 9.0 * (f - 32.0) first, then give *c* that answer.

You can think of the assignment operator, =, as a copy operator; it copies whatever is on its right hand side into whatever is on its left-hand side.

```
c ← (5.0 / 9.0 * (f - 32.0))
```

3.8 MEMORY

A computer's memory is known as main memory. It is volatile in the sense that the data stored in it is lost when its electrical power supply is turned off.

Each item of data is stored as a sequence of binary digits. Binary digits, 0 and 1, are known as bits.

A computer's memory is made up of consecutive storage locations. A number identifies each storage location. This number is known as the storage location's address.

address	memory location
...	
6501	
6502	
6503	
6504	
6505	
6506	
6507	
6508	
6509	
...	

Figure 3.5 *Data values are stored in memory locations*

When data is stored in a memory location, the data that was previously held in that location is overwritten.

When data is read (or retrieved) from a memory location, the data stored there remains unchanged.

3.9 VARIABLES

A variable is a location in a computer's memory. It has

- an address (i.e. a number which uniquely identifies it)
- a name (chosen by the programmer)
- a type (e.g. int, double) and
- a value

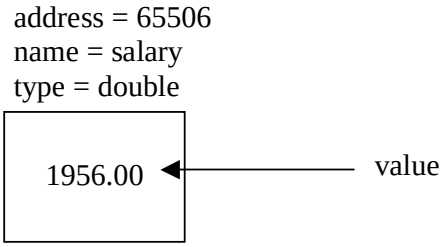


Figure 3.6 *A Variable*

In Java we create a variable named *salary*, of type *double*, with value *1956.0*, like this

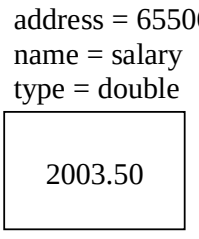
```
double salary   =   1956.0;
```

type — variable name — assignment operator — initial value

A variable can hold just one value at a time. So the Java statement

```
salary = 2003.50;
```

would replace (i.e. overwrite) the original value stored in *salary* with *2003.50*.



We say that a variable contains a value, stores a value or, sometimes, is the value.

The convention for naming variables is

- they are descriptive e.g. *temperature* rather than *x*
- they always start with a lower case letter e.g. *month* not *Month*
- they are singular e.g. *day* not *days*.

3.10 INT OR DOUBLE

Use *ints* when only integer values are involved and loss of accuracy in division is not a problem or when simplicity is required.

Use *double* when floating-point values are involved and maximum accuracy is required e.g. financial accounts or when very large numbers are involved.

For integers outside the range ±2 billion you could use the Java *long* int data type. But we shall not go into that here.

3.11 INCREMENT AND DECREMENT OPERATORS

Adding one to an integer or subtracting one from an integer is a common operation. Java has a special notation for doing just this.

```
int n = 12;
n++;
```

leaves *n* containing 13. And

```
int n = 12;
n--;
```

leaves *n* containing 11.

++ adds one to an integer value stored in a variable.

-- subtracts one from an integer value stored in a variable.

++ is known as the increment operator.

-- is known as the decrement operator.

You cannot use the increment and decrement operators on *double* values.

3.12 FURTHER READING

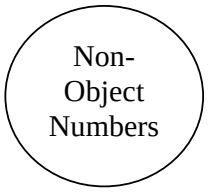
HORSTMANN & CORNELL *Core Java 2 Volume 1 - Fundamentals* pp 48

3.13 REVIEW

Integers

whole numbers

represented by int



int operations

-, +, *, /, %, ++, --

/ div truncates
% mod - remainder
++ increment, adds 1
-- decrement, subtracts 1

Reals

contain a decimal point

represented by double

Precedence

( ), ++, --, *, /, %, +, - (bodmas)

Associativity

( ), *, /, % from left to right
++, --, = from right to left

Memory

a sequence of consecutively numbered storage locations

Variables

a location in memory

in Java

```
int n = 12;
double d = 3.1416;
```

Increment & decrement operators

n++ adds 1 to integer n
n-- subtracts one from integer n

3.14 EXERCISES

1 Explain what is meant by each of the terms

(a) int
(b) double
(c) precedence
(d) associativity
(e) variable

2 Write Java statements to

(a) Declare an *int* variable named *day*, with initial value *19*
(b) Declare an *int* value named *week*, with initial value *day* div *7*
(c) Replace the original value in *day* with *day* mod *7*

3 Write Java statements to convert pence into pounds and pence. Choose your own initial value for pence.

4 For each of the fragments of Java shown below, state the value of each variable alongside each operation.

(a)
```
int n = 10;
n++;
n = n * 2;
```

(b)
```
int m = 4;
int n = 6;
m--;
n--;
int d = m * m + n * n;
```

(c)
```
int x = 3;
x = x + 2;
```

(d)
```
int y = 5;
y = y - 2;
```

(e)
```
int a = 2;
int b = 3;
int c = (a + b++) % 2;
```

5 Explain what is wrong with the Java fragment shown below

```
double g;
42.0 / 21.0 * 27.0 = g;
```