

### Revision: How to make and run a program

- To make a Java program, you need to have the Java platform installed.
- Then you can type the program into the computer using an editor, and type `javac` followed by the filename to compile the program.
- To run the program, type `java` followed by the name of the program.
- A program usually begins with some comments, and followed by something that is always there (a line containing the program name, and a line containing the word `main`).
- A program can contain simple statements, which are always terminated by a semicolon (`;`).
- A program can use `System.out.println` to print things to the screen.

0911B L03-1

### General format of Java programs: take 1

```
public class ProgramName {
    public static void main(String[] args) {
        statement1;
        statement2;
        statement3;
        ...
    }
}
```

The statements will be executed in the order they appear in the program: sequential program.

Java is **case sensitive**: uppercase character is not the same as the lowercase character.

Convention: the name of the program should be Capitalized, with words JoinedTogetherLikeThis.

0911B L03-2

### Variables: using memory to store values

Memory is used to store intermediate results produced by the program. A programmer allocates the memory for storing them.

There are different type of things that the memory can store. The same piece of memory may store an integer, a character, a real number, a memory address, or even an instruction.

- In machine language, the programmer needs to remember what location is used for storing what type of values.
- In high level languages like Java, the programmer defines a variable. The compiler then allocates space in the memory for the value, and associates that type to that memory.
- The compiler checks that you are using the variable correctly, e.g. you're not assigning a character into an integer variable.
- In the program, the name of a variable refers to the variable. Convention: lowercase, with words\_joined\_using\_underscores.

0911B L03-3

### Using variables

```
public class TestVar {
    public static void main(String[] args) {
        double number;
        number = 2.0;
        System.out.println(number);
    }
}
```

This program defines a variable, using a *variable declaration statement*:

number

**double number;**

- "double" is the type of the variable. (It is a reserved word.)
- "number" is the name of the variable. (It identifies the variable.)
- It tells the compiler to reserve a piece of memory to be used as an "double" type value, and give it the name "number".
- Initially there is no value stored in the variable.

0911B L03-4

### Putting values into variables

The next statement is an "assignment", storing a value in the variable.

**number = 2.0;**

It reads "*number* is assigned as 2.0". It means:

- In high level language: *Assign 2.0 to the variable named "number"*.
- In machine language: *Put the value 2.0 into the memory location associated with the variable with name "number"*.

You can actually combine this with the definition of a variable:

**double number = 2.0;**

**Be careful:** the statement does not mean to use *number* to represent 2.0. This is why we do not say "number equals 2.0". We are in the business of *programming*, not *mathematical reasoning*. e.g. you can store another value to it later.

0911B L03-5

### Using a variable

The next statement actually uses the variable:

**System.out.println(number);**

To get the value of a variable, just use its name.

Since *number* has been assigned a value of 2.0, the above statement has the same effect as:

**System.out.println(2.0);**

This results in 2.0 being printed out on the screen.

Two questions:

- What are possible types in Java?
- What is really meant by 2.0? It is not an identifier since it violates the rule of identifier, but what is it?

0911B L03-6

## Various types in Java

There are a lot of different variable types in Java, and you can even define your own types (wait... we will have it by week 10).

There are some types that are "basic".  
These types form other types:

Type	Range	Usage
<code>double</code> float	$-1.80 \times 10^{308}$ – $1.80 \times 10^{308}$ $-3.40 \times 10^{38}$ – $3.40 \times 10^{38}$	Represents a real number with limited precision.
long <code>int</code> short byte	-9223372036854775808 -9223372036854775807 -2147483648–2147483647 -32768–32767 -128–127	Represents an integer exactly.
<code>boolean</code>	true/false	For truth values.
<code>char</code>	'\u0000'–'\uFFFF'	Represents a character in Unicode.

0911B L03-7

## Constants (Literals)

To write a value in our program, we use a "constant":

- `int`: a constant is written directly, e.g. 0, 42, -52.
- `long`: like `int`, but with a 'L' added at the end. e.g. -9999999999L.
- `double`: a constant is written by a decimal number optionally followed by a 'E' and then an integer. e.g. 3.1415926535, 6.02E23 (meaning  $6.02 \times 10^{23}$ ), -20., .25E-2 (i.e. 0.0025).
- `float`: like `double`, but with a 'F' added at the end. e.g. 3.14F, 6.02E23F, -20.F.
- `char`: a constant is written directly but enclosed by single quotes, e.g. 'a'. The single quote character need to be written like '\''.
- `boolean`: written as either `true` or `false`.

0911B L03-8

## Java Strings

There is a more complicated type called *String*, which is actually a collection of `char`-type values linked together.

You can specify a `String` constant by enclosing the characters by double quotes ("). e.g. "Hello World".

To have a double quote in the middle, we have to place a backslash (\) before the double quote, e.g. "She said \"Hello\"".

A `String` can contain no character, and is written like "".

A `String` is never a `char`, even if it contains only one character. You cannot say something like this:

```
char s = "a"; // Type mismatch! Should be 'a'
```

0911B L03-9

## Escape sequences

There are some characters which cannot be conveniently written out in a program, e.g. a newline character.

Instead, they can be written using a "escape sequence". We have already seen that the backslash character will modify the meaning of the next character. Here are some more:

- `\n` The newline character. Terminate a line.
- `\r` The carriage return character. Go back to the beginning of a line
- `\t` The tab character. Go to the next tab stop.
- `\\` The backslash character itself.
- `\'` Single quote.
- `\"` Double quote.
- `\u####` The character which is at the ####-th Unicode character.

For example, `System.out.println("Hello\nWorld");` actually prints out two lines.

0911B L03-10

## Operators in Java: arithmetic calculations

You can ask the computer to do arithmetic calculations by using these 5 arithmetic **operators**:

- `+`: perform addition when appeared like  $a + b$ . Do nothing when appeared like  $+a$ .
- `-`: perform subtraction when appeared like  $a - b$ . Do negation when appeared like  $-a$ .
- `*`: perform multiplication.
- `/`: for integer types, it finds the quotient ("integer division"). For real types, it performs division.
- `%`: finding remainder.

The numbers that an operator operate on are called **operands**.

0911B L03-11

## Expressions

Things like  $a + b$  is called an expression.

- Expressions have values. In this case, the value of the expression is the result when  $a$  and  $b$  are added together.
- You can use the value of an expression like the value of a variable.
- For example, you can assign it to a variable, and you can use it as operands of other operations.
- Even the assignment is actually an expression. We will come back to it in the next lecture.

0911B L03-12

### Types of results and mixed-mode computations

An expression also has a type. Computations follows the following rules.

- If one operand is of type double, then the operands are converted (promoted) to double, the operation is done in double, resulting in a double value.
- Otherwise, if one operand is of type float, then the operation is done in float.
- Otherwise, if one operand is of type long, then the operation is done in long.
- Otherwise, the operation is done in int.

Examples:  $1+2.0F \rightarrow 3.0F$ ,  $1.5*5 \rightarrow 7.5$ ,  $3L*7F \rightarrow 21F$ .

0911B L03-13

### Multiple operators in an expression

When there are more than one operators in a single expression like

$$a*b+c*-d/e\%f,$$

there need to be rules to specify the order in which the computations is done.

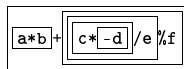
For arithmetic operators in Java, three rules:

1. Multiplications/divisions/modulus are always done before additions/subtractions. We say the former three operators have *higher precedence*.
2. For binary operators of the same precedence, they are performed from left to right. We say operators are *left associative*.
3. When doing a binary operation, the left operand is found before the right operand. We say evaluation is *left-to-right*.

0911B L03-14

### Example of precedence

$a*b+c*-d/e\%f$  has the grouping:



The order of evaluation:

1. find a.
2. find b.
3. find  $a*b$ .
4. find c.
5. find d.
6. find  $-d$ .
7. find  $c*-d$ .
8. find e.
9. find  $c*d/e$ .
10. find f.
11. find  $c*d/e\%f$ .
12. find  $a*b+c*d/e\%f$ .

0911B L03-15

### One more example

$a-b/c*d+-e\%f$  has the grouping:

### The parentheses

**Q:** What if that is not the order I want?

**A:** Use parentheses. Only parentheses (works), not brackets [won't work], and not braces {won't work}.

**Example:** To convert from Fahrenheit to Celsius:

```
double temp_in_c = (temp_in_f - 32.0) * 5.0 / 9.0;
```

Parentheses acts to modify the grouping,

from  $temp\_in\_f - \boxed{32.0 * 5.0} / 9.0$

to  $\boxed{temp\_in\_f - 32.0} * 5.0 / 9.0$

0911B L03-17

### Mathematical methods

Other mathematical functions can be used in the following form:

$Math.function(operand)$

Many functions are provided. These have the intuitive meaning:

$abs$  (absolute value),  $acos$ ,  $asin$ ,  $atan$ ,  $cos$ ,  $sin$ ,  $\tan$ ,  $sqrt$ .

Others needs some explanation.

- $exp(x)$ ,  $log(x)$ : find  $e^x$  and  $\ln x$ .
- $pow(x,y)$ : find  $x^y$ .
- $random()$ : generate a random number between 0 and 1.
- $floor(x)$ ,  $ceil(x)$ : find the floor and ceiling of  $x$ .
- $max(x,y)$ ,  $min(x,y)$ : the maximum and minimum of two numbers.

0911B L03-18

### Example: solving a particular quadratic equation

```
/* Solves 2 x^2 + 3 x + 1 = 0 */
public class ParticularQuad {
    public static void main(String[] args) {
        double a = 2.0, b = 3.0, c = 1.0;
        double determinant = b * b - 4.0 * a * c;
        System.out.print("The first solution is ");
        System.out.println((-b+Math.sqrt(determinant))/2.0/a);
        System.out.print("The second solution is ");
        System.out.println((-b-Math.sqrt(determinant))/2.0/a);
    }
}
```

Note 1: we can have more than one variable defined in a single variable declaration statement, in the above form.

Note 2: this program is not very useful as it only deal with one equation.

0911B L03-19

### Precedence and types

The precedence and result types can interplay and surprise you.

Q: What is the result of  $1/2$ ?

A: Both operands are integers, so we do integer division, giving 0.

Q: What is the result of  $2+1/2$ ?

A:

Q: What is the result of  $2.0+1/2$ ?

A:

Q: What is the result of  $2.0+1.0/2$ ?

A:

Q: What is the result of  $2+1.0/2$ ?

A:

0911B L03-20

### Conversion between types

When an expression of the wrong type is assigned to a variable, the following occurs:

- If the range of the type of expression is narrower than the resulting type, an *automatic type conversion* occurs.
- Otherwise, the compiler complains.

You can explicitly ask the compiler to do a conversion by writing something like `(int)3.14`. This convert (cast) 3.14 into an int.

- For conversion from real numbers to integers, the fractional part is discarded.
- For conversion from real number types, if the number is too large or too small, it gives the largest or smallest integer value in the resulting type.
- For conversion among integers, overflows causes "wrap around".

0911B L03-21

### Imprecise real numbers

Real numbers are not precisely represented.

- Real numbers are represented by a "mantissa-exponent" form in the memory, very much like the scientific notation.
- A fixed amount of memory is used to store the fractional (mantissa) and the exponent parts of the number.
- That means the number stored cannot be too large or too small (the exponent part can only hold a number of some fixed size).
- That also means the number of significant digits is limited (the fractional part also can only hold a number of fixed size).
- A float value holds about 7 significant digits, and a double value holds about 15.
- If you need things to be exact, you should not use real numbers.

0911B L03-22

### Example

```
public class Imprecision {
    public static void main(String[] args) {
        float v = 10000F;
        v = v + 0.0001F;
        v = v - 10000F;
        System.out.println(v);
    }
}
```

What the program prints?

Answer: try to run the program!

0911B L03-23

### Another example

```
public class Imprecision2 {
    public static void main(String[] args) {
        double v = 1.0/2.0;
        v = v + 1.0/3.0;
        v = v + 1.0/6.0;
        System.out.println(v);
    }
}
```

What the program prints?

Answer: try to run the program, of course!

0911B L03-24