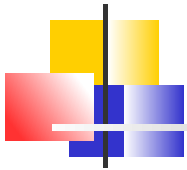
Software and Programming I

Introduction
Basic elements of Java

Roman Kontchakov / Carsten Fuhs

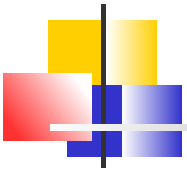
Birkbeck, University of London



Module Information

- Time: Thursdays in the Spring term
- Lectures: MAL B04 (A–H: 2–3.30pm, I–Z: 3.30–5pm)
  MAL B20 (A–L: 6–7.30pm, M–Z: 7.30–9pm)
- Labs: MAL 109 (2–5pm) and MAL 414/415 (6–9pm)
- **Optional tutorial hour**: MAL 109, 5–6pm
- web: `http://www.dcs.bbk.ac.uk/~roman/sp1`

  moodle (`http://moodle.bbk.ac.uk`)

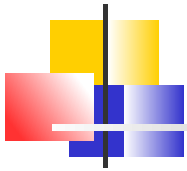


Assessment

- In-Class Tests (weeks 5 & 11): 20% (10% & 10%)
- Programming Exercises 5%

if you do not complete all exercises by week 10 then you will not be able to sit In-Class Test 2 and you will get 0 marks for the exercises

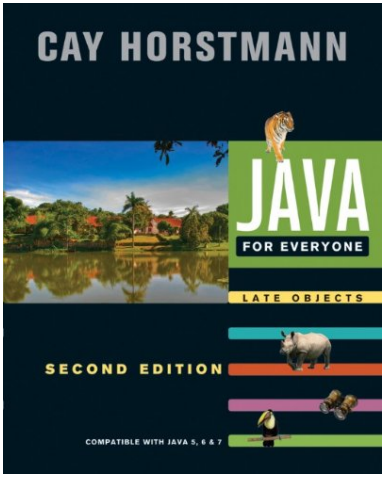
- Two-hour examination in summer 2020: 75%



Essential Textbook

Cay Horstmann
Java for Everyone
2nd edition
John Wiley & Sons, 2013

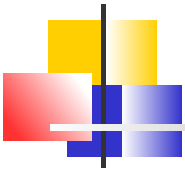
1st edition
John Wiley & Sons, 2010



book also available **online** via BBK library, see
`http://www.dcs.bbk.ac.uk/~roman/sp1/`

the module draws on Chapters 1–9 and 12

the lab classes are based on exercises suggested in JFE

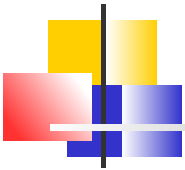


Python v Java

Introduction to Programming was in Python. Why learn Java?

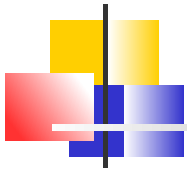
- Python particularly suitable for first steps
- Java widely used for large software systems
- **concepts** carry over
  from one programming language to another
- main difference: Java is **statically typed**
- goal: be(come) comfortable with
  more than one programming language

NB: Java is not a version of JavaScript



Syllabus

- primitive data types and strings
- branching and loops
- arrays
- objects and classes
- methods and constructors
- instance and static variables and methods
- inheritance and polymorphism
- object-oriented design
- input/output
- basic data structures and algorithms



SP1 Module Specification

Software and Programming I is a **Level 5** module

Introduction to Programming is a Level 4 module

Software and Programming I is **15 credits**

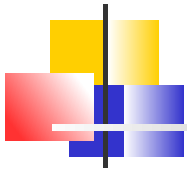
each credit is worth 10 hours of study **150 hours**

term = 11 weeks = 33 hours of classes

any difficulties $\Longrightarrow$ attend tutorials (MAL 109, 5–6pm)

Python:

```
n = "World"
print("Hello, " + n + "!")
```

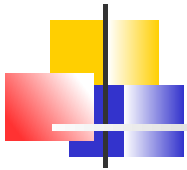


My First Program

```
1  /* HelloWorld.java
2     Purpose: printing a hello message on the screen
3  */
4  public class HelloWorld {
5      // each program is a class (week 6)
6      // almost everything in Java is an object
7      public static void main(String[] args) {
8          String n = "World";
9          System.out.println("Hello, " + n + "!");
10     }
11 }
```

NB. watch out for **semicolons** — they are compulsory

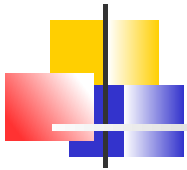
NB. names and reserved words are **case-sensitive**



My First Program: Layout Style 2

```
1  /* HelloWorld.java
2     Purpose: printing a hello message on the screen
3  */
4  public class HelloWorld
5  { // opening curly brackets on the new line
6      // each program is a class
7      public static void main(String[] args)
8      {
9          String n = "World";
10         System.out.println("Hello, " + n + "!");
11     }
12 } // closing curly brackets directly below
```

NB. different styles of curly bracket layout **(indentation is irrelevant)**



My First Program: No Style

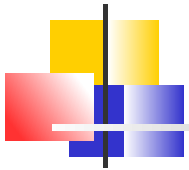
```
1 /* HelloWorld.java Purpose: printing a hello message
  on the screen */ public class HelloWorld  { public
  static void main(String[] args)  {  String n = "
  World"; System.out.println("Hello, " + n + "!"); }
  } // all in a single line
```

the Java compiler is happy, but . . .

NB: `println` prints the argument and moves the cursor to the **new line** (`ln` comes from 'line')

`print` simply prints the argument

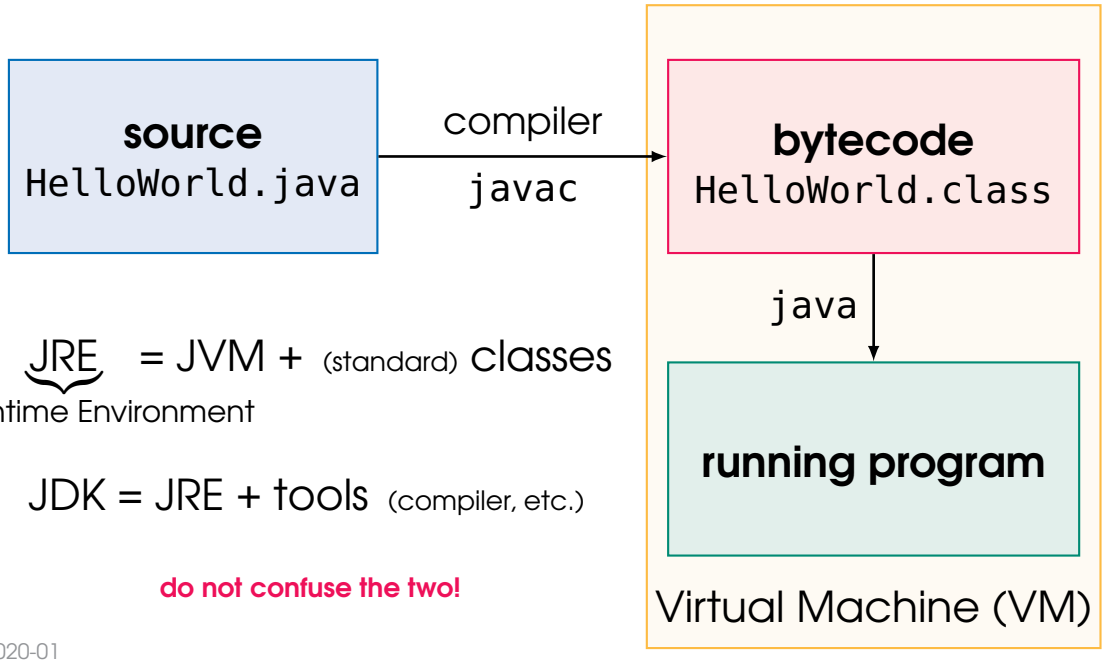
see http://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/io/PrintStream.html



Java Development Environments

- Java Development Kit (JDK), Java SE 8 /13
  Standard Edition
- BlueJ
  (a public project to make programming in Java easier)
- IntelliJ
  (extensible,
  free software with a proprietary commercial edition)
- Eclipse
  (multi-language and extensible,
  free and open source software)

Java Compilation and JRE



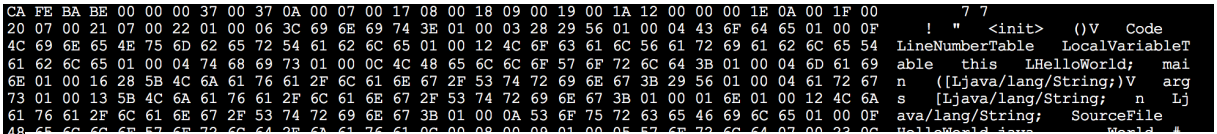
JRE = JVM + (standard) classes

Java Runtime Environment

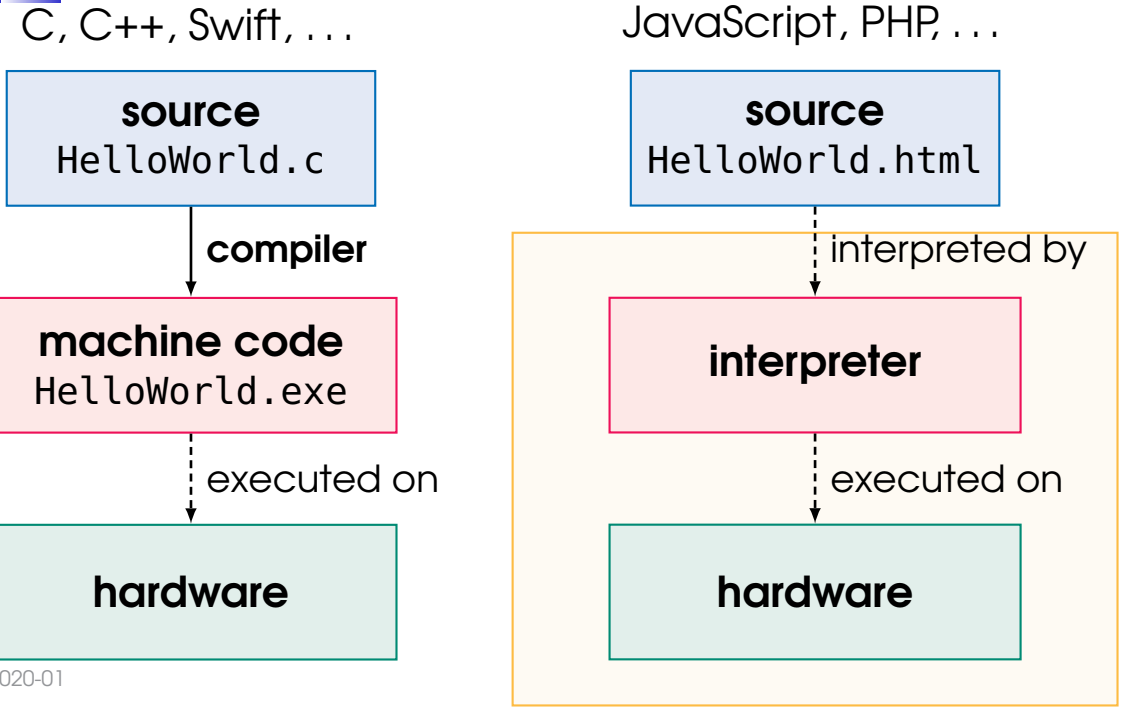
JDK = JRE + tools (compiler, etc.)

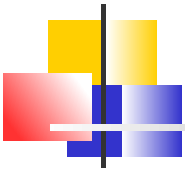
do not confuse the two!

Java Bytecode: Example



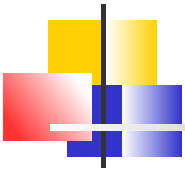
Compilation v Interpretation





JDK: Editing

- source code can be edited in any text editor
  (e.g., Notepad, emacs, . . .)
- *MS Word* caveat:
  by default, Word does not save in ASCII text format
- make sure to save the code before compiling!
- the file name must be the same
  as the name of the class (with the .java extension)
  (**case sensitive!**)



Compiling with JDK

- invoke the command-line compiler:

```
javac <source>.java
```

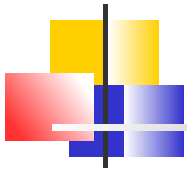
- compiles `<source>` and all classes it depends on into **Java bytecode** files (`<source>.java`, etc.)
- for example:

```
javac HelloWorld.java
```

produces the file `HelloWorld.class`

(provided there are no errors)

- make sure the compiler and JVM are in the command path (PATH)



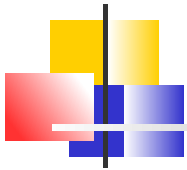
Execution in JDK

- starting the Java Virtual Machine (JVM):

```
java <source>
```

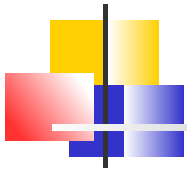
- the named class is loaded and execution is started
  (other classes are loaded as needed)
- only possible if the class has been **compiled**
  into Java bytecode

How does the JVM know where to start the execution?



Coding in BlueJ

- BlueJ organises files into **projects**,
  stored in project-specific directories on disk
  do not forget to backup!
- types of BlueJ files:
  - bluej.pkg: contains information about classes
    in the package (one per package)
  - bluej.pkh: backup of the package file
  - *.java: Java source code (text files, one per class)
  - *.class: Java bytecode (binary, one per class)
  - *.ctxt: BlueJ context file with extra information
    about the class (one per class)



Software is Free

- available on BBK's network
  - JDK (allows one to compile and execute programs)
  - BlueJ (preferred Java IDE)
- installing BlueJ for home use
  - download JDK from

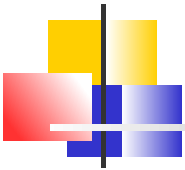
http://www.oracle.com/technetwork/java/javase/downloads

  - download BlueJ from

http://www.bluej.org

  - BlueJ tutorial

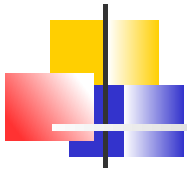
http://www.bluej.org/tutorial/tutorial-v4.pdf



Comments

```
1  /* this is a block comment
2     comments provide additional information
3     that is not readily available in the code itself
4     comments are ignored by the Java compiler */
5  public class HelloWorld {
6        // this is a single-line comment
7        public static void main(String[] args) {
8            String n = "World";
9            System.out.println("Hello, " + n + "!");
10       }
11 }
```

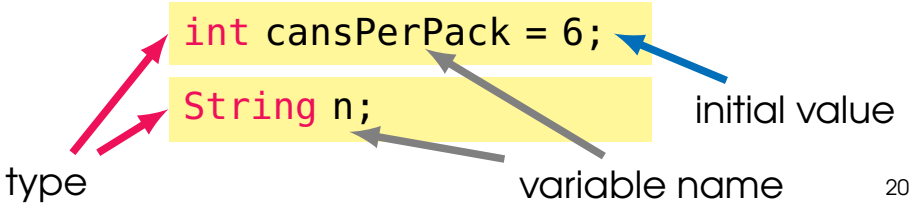
NB: Python uses # for single-line comments

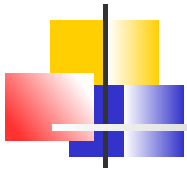


Variables

- A **variable** is a storage location with a name

- Every variable must be **declared** before its first use
  otherwise, a **compile-time** error
  (unlike in Python)
- When declaring a variable, you specify
  - the **type** of its values
  - and optionally its **initial value**





Variable Names (aka Identifiers)

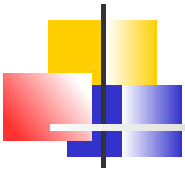
- variable names must start with a letter (or _ or $), the remaining characters must be letters, _, $ or digits
  (but cannot be a reserved word)
- identifiers are **case-sensitive**

by convention, variable names start with a lower-case letter

Q:

```
cansPerPack      ✓
cans_per_pack    ✓
cans per pack    ✗
__cansperpack    ✓
CaNsPeRpAcK      ✓
```

```
digit      ✓
0digit     ✗
digit0     ✓
int        ✗
INT        ✓
```

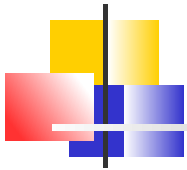


Primitive Data Types

- `int` 32-bit two's complement integer

  (-2,147,483,648 to 2,147,483,647)

  Integer.MIN_VALUE Integer.MAX_VALUE
- `long` 64-bit two's complement integer
- `short` 16-bit two's complement integer
- `byte` 8-bit two's complement integer
- `double` double-precision 64-bit IEEE 754 floating point
- `float` single-precision 32-bit IEEE 754 floating point
- `boolean` Boolean value (`true` or `false`)
- `char` 16-bit Unicode character



Variable Assignment

An assignment statement stores a new value in a variable, **replacing** the previously stored value

(so, the previous value is lost)

```
cansPerPack = 8;
```

variable name

expression

cansPerPack | 8

6 is lost

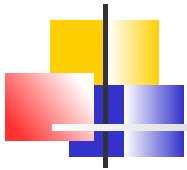
Q: how do you swap the contents of two variables, a and b?

```
int a;
int b;
```

a | 4

b | 5

(the answer is at the end)



Assignment v Equality

The assignment operator = **does not** denote mathematical equality

Q: what is the meaning of

```
x = x + 1;
```

1. take the current value of x
2. evaluate x + 1
3. assign the resulting value to the variable x

Use **==** to compare numbers — more in week 2

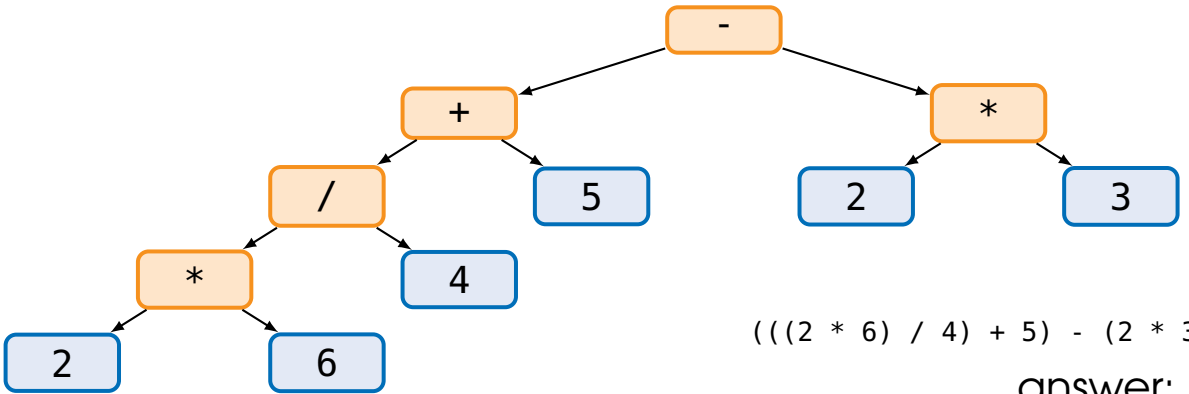
- Pascal uses := for assignment and = for equality

"Software is getting slower more rapidly than hardware becomes faster"
(Niklaus Wirth, 1995)

Arithmetic Expressions

Java uses the **natural precedence** of arithmetic operations: `*`, `%` (remainder), `/` before `+`, `-`

Q: what is the value of `2 * 6 / 4 + 5 - 2 * 3` ?



`(((2 * 6) / 4) + 5) - (2 * 3)`

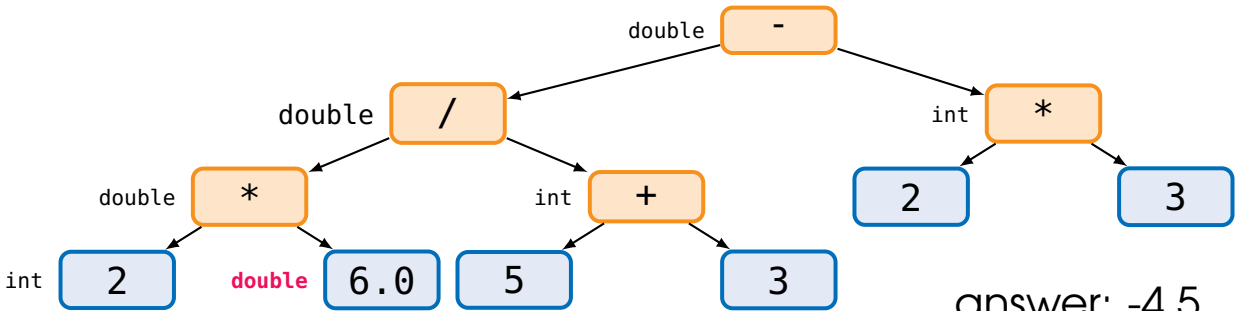
answer: 2

Expressions are Typed

Java uses the natural precedence
of arithmetic operations: *, %, / before +, -

if in doubt, use brackets

Q: what is the value of `2 * 6.0 / (5 + 3) - 2 * 3` ?

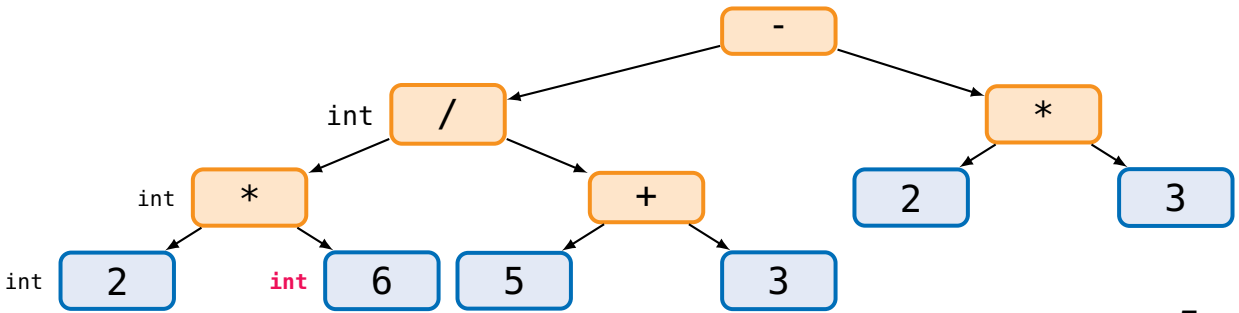


answer: -4.5

Integer Arithmetic Operations

if one argument is `double` then the result is `double`
if both arguments are `int` then the result is `int` (rounded toward 0)

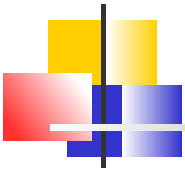
Q: what is the value of `2 * 6 / (5 + 3) - 2 * 3` ?



answer: -5

NB: beware of the **unintended integer division**

(unlike Python, Java uses / for both floating-point and integer division)



Strings

- strings are sequences of characters:

```
String name = "index.html";
```

literal

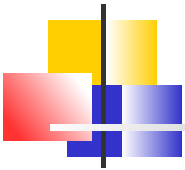
- the `length` method yields the number of characters in the string:

```
int len = name.length();
```

the empty string `""` is of **length** 0

- `\"`, `\n`, `\\`, `\'` are examples of **escape sequences**

(double quotes, new line, backslash, quotes) (like in Python)



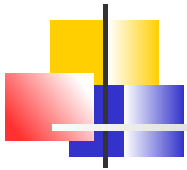
Strings Concatenation

- use the + operator to **concatenate** strings

```
String name = "Harry";
String lastname = "Morgan";
String fullname = name + " " + lastname;
```

NB: whenever one of the arguments of + is a string,
the other argument is **converted** to a string

(in Python, conversion function `str()` is needed)



Substrings (1)

- string **positions** are counted starting with **0**

```
String name = "index.html";
```

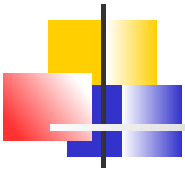
	i	n	d	e	x	.	h	t	m	l
position	0	1	2	3	4	5	6	7	8	9

- `substring(`i`,`j`)`: string made up of the characters starting at position i and containing all the characters up to, *but not including*, the position j:

```
String filename = name.substring(0,5);
```

- **NB: no negative** positions! (unlike in Python)

```
String last = name.substring(name.length() - 1,
                             name.length());
```



Substrings (2)

- string **positions** are counted starting with **0**

```
String name = "index.html";
```

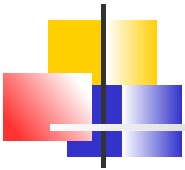
	i	n	d	e	x	.	h	t	m	l
position	0	1	2	3	4	5	6	7	8	9

- `substring(`i`)`: all characters from the position i to the end of the string:

```
String ext = name.substring(name.length() - 4);
```

- these are examples of instance methods of the class `String` (week 6): use `variable.method-name`, not `function(variable)` like in Python
- there is no string format operation, like `%` in Python; use `String.format` instead

http://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/String.html#format(java.lang.String,java.lang.Object...)



Methods

Python:
```
def sq(x):
    Python code
```

- A **method** is a named sequence of instructions

method name

```
public static int sq(int x) {
    Java code (sequence of instructions)
}
```

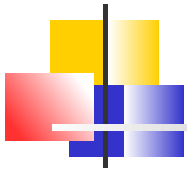
parameter
(type and name)

type of **return value**

`void` means the method does not return any value

- **Parameter values** are supplied when a method is called
- The **return value** is the result that the method computes
- Method ≈ algorithm ≈ function (in Python)

NB: until week 6, all methods will be `public static`



Some Remarks on Code Structure

- every **method** must be declared in a class

  that is, inside the curly brackets in `public class` *`class-name`* `{ ... }`

- all **code** should occur in one of the methods

  (unlike in Python, where code can also be at "top level")
  until week 6: inside the curly brackets in

  `public static` *`return-type method-name`*`(`*`parameters`*`) { ... }`

  one exception: initialisation blocks are not in the scope of SP1

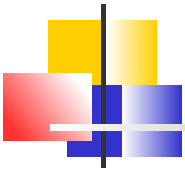
- the order of method declarations is **not important**

  (unlike in Python, where a function has to be declared before use)

- only `System.out.println(...)`, `.print`, `.printf`, etc. actually print something on the screen (standard output)

  `return` does not print the value

  (and `System.out.println(...)` does not return any value to the program)

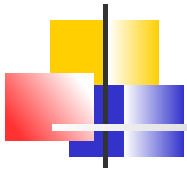


Example: $y = x^2$ as a Method

```
1  public class PrintSquares {
2      // compute x^2
3      public static int sq(int x) {
4          // x is a parameter variable
5          int y = x * x; // compute the value
6          return y; // return the value
7      }
8      public static void main(String[] args) {
9          System.out.println(7 + "^2=" + sq(7));
10         System.out.println(9 + "^2=" + sq(9));
11     }
12 }
```

the output is:

```
7^2=49
9^2=81
```



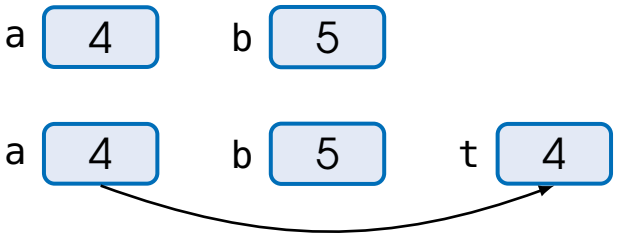
Swapping Values

suppose there are two variables

```
1 int a, b;   // the same as int a; int b;
```

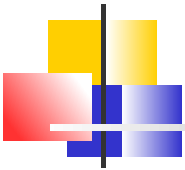
how do you swap the contents of a and b?

```
2 int t = a;
```




```
3 a = b;
```


```
4 b = t;
```



Overview

- compiler, bytecode and JVM
- interpretation v compilation
- JDK and BlueJ
- variables: declaration and initialisation
- primitive data types
- arithmetic operations
- strings
- methods