Stéphane Derrode

CM1

From Python to Java

- Java compilation model & Maven
- Static typing & primitive types
- Classes, objects & encapsulation
- Object references — == vs equals()
- String class & core syntax

Module Overview

Part 1 — Java Fundamentals	Part 2 — GUI with Swing	Part 3 — Android
CM1 From Python to Java	**CM4** Swing: Components & Layouts	**CM6** Android Architecture
CM2 Collections, Exceptions & I/O	**CM5** Graphics 2D & Events	**CM7** Layouts, Navigation & Intents
CM3 Inheritance, Interfaces		**CM8** Persistence & Threads

Projects: ★ Spider Game (Part 2) ★ Calculator / Currency Converter (Part 3)

Exam 50% + BEs 50%

CM1 — Agenda

01 Java vs Python — key differences

02 Maven project setup

03 Types, variables & control flow

04 Classes & objects — Voiture

05 References, == and equals()

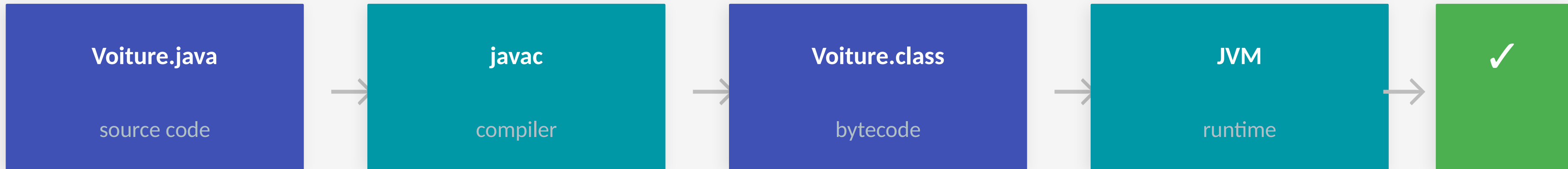
06 String class & StringBuilder

Java vs Python — The Big Picture

Python	Java
Interpreted (CPython)	Compiled → JVM bytecode
Dynamic typing	Static typing
No explicit compilation	javac + java (or Maven)
Indentation-based blocks	Curly-brace blocks
No type declarations	Type declared for every variable
Runs as .py script	Runs as .class on JVM

you know this

The Java Compilation Model



With Maven, you don't call javac manually

`mvn compile` — compiles all sources `mvn exec:java` — compiles & runs `mvn package` — creates a .jar

→ In VSCodium: File ▸ Open Folder... then select your project — Maven is detected automatically

Static Typing — Variables & Primitive Types

Python

```
# No type declaration needed
name = "Alice"
age  = 21
speed = 99.8
started = True

# Type can change at runtime
age = "twenty one"  # OK in Python
```

Java

```
// Type declared explicitly
String  name    = "Alice";
int     age     = 21;
double  speed   = 99.8;
boolean started = true;

// Type is FIXED at compile time
age = "twenty one"; // ERROR ✗
```

Primitive types

Type	Description	Range	Example
int	32-bit integer	-2^{31} to $2^{31}-1$	`int power = 4;`
double	64-bit float	~15 sig. digits	`double speed = 0.0;`
boolean	true/false	true or false	`boolean started = false;`
char	single character	Unicode	`char c = 'A';`
long	64-bit integer	-2^{63} to $2^{63}-1$	`long l = 600851475143L;`

Control Flow — Syntax Side-by-Side

Python

```python
if speed >= 130:
    print("too fast!")
elif speed > 0:
    print("moving")
else:
    print("stopped")

for v in [0, 50, 130]:
    print(v)

i = 0
while i < 5:
    i += 1

month = 2
match month:
    case 1: name = "January"
    case 2: name = "February"
    case _: name = "Other"
```

Java

```java
if (speed >= 130) {
    System.out.println("too fast!");
} else if (speed > 0) {
    System.out.println("moving");
} else {
    System.out.println("stopped");
}

int[] vals = {0, 50, 130};
for (int v : vals) {
    System.out.println(v);
}

int i = 0;
while (i < 5) { i++; }

int month = 2;
switch (month) {
    case 1 -> name = "January";
    case 2 -> name = "February";
    default -> name = "Other";
}
```

Classes & Objects — the Voiture Example

UML class diagram

Voiture
- puissance : int
- estDemarree : boolean
- vitesse : double
+ Voiture(int)
+ deQuellePuissance() : int
+ demarre()
+ accelere(double)
+ toString() : String

```java
public class Voiture {
    private int     puissance;
    private boolean estDemarree;
    private double  vitesse;

    public Voiture(int p) {
        if (p > 0) puissance = p;
        else       puissance = 5;
        estDemarree = false;
        vitesse     = 0.0;
    }

    public int deQuellePuissance() {
        return puissance;
    }
    public void demarre() {
        estDemarree = true;
    }
    public void accelere(double v) {
        if (estDemarree)
            vitesse += v + 25;
    }
    public String toString() {
        return "Voiture[p=" + puissance
            + ", v=" + vitesse + "]";
    }
}
```

Objects, Constructors & Object Arrays

Using objects

```
Voiture maClio = new Voiture(4);
maClio.demarre();
maClio.accelere(99.8);
System.out.println(maClio);
// Voiture[p=4, v=124.8]

// Multiple constructors
public Voiture() { this(5); }  // default
public Voiture(int p, boolean d, double v) {
    puissance   = p;
    estDemarree = d;
    vitesse     = v;
}
```

Array of objects

```
// 1. Declare   2. Allocate   3. Initialise
Voiture[] fleet = new Voiture[3];
fleet[0] = new Voiture();
fleet[1] = new Voiture(6);
fleet[2] = new Voiture(28, true, 0);

// Or all at once:
Voiture[] fleet = {
```

new keyword

Allocates memory on the heap and calls the constructor. Returns a reference to the new object.

Multiple constructors

Same class name, different parameters (overloading). Use this(...) to delegate to another constructor — avoids code duplication.

Object array

Voiture[] fleet = new Voiture[3] creates an array of 3 null references. Each element must be initialised separately with new.

Access modifiers reminder

maClio.puissance → compile error (private).
maClio.deQuellePuissance() → OK (public). Encapsulation is enforced at compile time.

Object References — == vs equals()

== compares references

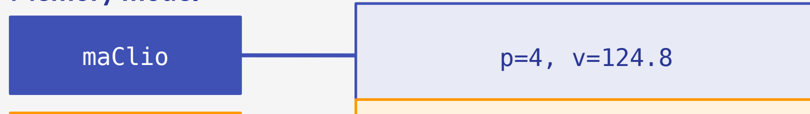
```
Voiture maClio  = new Voiture(4);
maClio.demarre();
maClio.accelere(99.8);

Voiture saPolo  = new Voiture(4);
saPolo.demarre();
saPolo.accelere(99.8);

// Same content, different objects!
if (maClio == saPolo)
    System.out.println("true");
else
    System.out.println("false");  // ← PRINTED
                                   //   (two
distinct objects)

// Assignment copies the REFERENCE
Voiture saBerline = new Voiture(6);
saBerline = maClio;  // both now point to same
object
if (maClio == saBerline)
    System.out.println("true");  // ← PRINTED
```

Memory model



≠ different objects → == false

Override equals() for content comparison

```
// In class Voiture:
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (!(obj instanceof Voiture)) return false;
    Voiture other = (Voiture) obj;
    return this.puissance == other.puissance;
}

// Now:
Voiture maClio = new Voiture(4);
Voiture saPolo = new Voiture(4);

maClio == saPolo           // false (references)
maClio.equals(saPolo)      // true  (content)
```

Rule — always use .equals() for objects

== on objects compares memory addresses (references). Always use .equals() to compare content. This applies to String too: never use == to compare strings.

Access Modifiers

`public`

Accessible from anywhere

`private`

Accessible only within the same class

`protected`

Same class + subclasses + same package (seen in CM3)

`(default)`

Same package only — no keyword

Best practice:

Make all fields private. Make methods public only if they are part of the class contract. Start restrictive — you can always open access later, never the reverse.

The String Class

Key methods

Method	Example	Result
`.length()`	`s.length()`	`// 7`
`.charAt(i)`	`s.charAt(0)`	`// 'V'`
`.substring()`	`s.substring(1,4)`	`// "oit"`
`.toUpperCase()`	`s.toUpperCase()`	`// "VOITURE"`
`.contains()`	`s.contains("oit")`	`// true`
`.equals()`	`s.equals("Voiture")`	`// true — not ==`
`.split()`	`s.split(",")`	`// String[]`

Strings are immutable

Every operation creates a NEW String object.

```
String s = "Voi";
s = s + "ture"; // new object
// original "Voi" still exists in memory
```

Use StringBuilder for repeated concatenation:

```
StringBuilder sb = new StringBuilder();
sb.append("Voiture[p=");
sb.append(puissance);
sb.append("]");
String result = sb.toString();
```

String: always use .equals(), never ==

Two distinct String objects with identical content will fail the == test. Use s1.equals(s2) — same rule as for Voiture.

Maven Project Structure

```
my-project/
├── pom.xml                  ← project config
└── src/
    ├── main/
    │   └── java/
    │       └── com/example/
    │           ├── Voiture.java   ← our
    │ class
    │           └── Main.java      ← main()
    └── test/
        └── java/
            └── com/example/
                └── VoitureTest.java
```

pom.xml — Java 17 configuration

```xml
<project>
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.example</groupId>
 <artifactId>my-project</artifactId>
 <version>1.0</version>
 <properties>
   <maven.compiler.source>17
   </maven.compiler.source>
   <maven.compiler.target>17
   </maven.compiler.target>
 </properties>
</project>
```

In VSCodium: File ▸ Open Folder... → select project root | Ctrl+Shift+P → Maven: Create Maven Project | ▶ Run above main()

CM1 — Key Takeaways

Compilation	Java → bytecode; Maven automates the build; VSCodium detects Maven projects automatically
Static typing	Every variable has a declared type — errors caught at compile time, not runtime
Classes	Fields private; public constructor; getters/setters; toString() for display
References	Objects are manipulated by reference; == compares addresses; use .equals() for content
Arrays	Voiture[] fleet = new Voiture[3] — creates null slots; each must be initialised with new
String	Immutable — use .equals() not ==; StringBuilder for repeated concatenation

Up next → TD1: Maven setup in VSCodium & first Voiture-based exercises