

# **LOG100/GTI100**

## **Programmation en génie logiciel**

### *Automne 2024*

## **Introduction à JAVA**

Chargé de cours : Anes Abdennebi

Les diapositives appartiennent à l'origine au  
Valentin Vervondel, MRes  
et Sègla Kpodjedo, PhD  
Avec modifications

# Langages de programmation

2

- Langage de programmation : langage formel conçu pour communiquer des instructions à une machine
  - ▣ Généralement décrit par une syntaxe (la forme) et une sémantique (le fond)



- Ada Lovelace (1815–1852), fille de Lord Byron
  - ▣ 1842–1843 : sa méthode détaillée pour calculer les nombres de Bernoulli avec la machine analytique de Charles Babbage est considérée comme le premier programme informatique

# Petite histoire de la programmation

3

## History of Popular Programming Languages: A timeline



**Autocode** (1952) 1<sup>er</sup> compilateur

**Flow-Matic** (1955-59) Grace Hopper

**Fortran** (Formula Translating System, 1957) John Backus, IBM. Haute Performance

**LISP** (LISt Processing, 1958) John McCarthy, Intelligence Artificielle  
 $(* 2 (\cos \theta) (+ 4 6)) = ???$

**COBOL** (COmmon Business-Oriented Language, 1959) conçu pour être lisible par des non-ingénieurs donc assez verbeux (beaucoup de mots réservés, sections, paragraphes, etc.)

**ALGOL** (ALGOrithmic Language 58, 60, 68) : efforts pour créer un langage universel, par des pionniers comme McCarty, Backus, Naur etc.

else if, passage par référence, points-virgule, structure de blocs, etc.

Sources : <http://cs.umw.edu/~finlayson/class/spring15/cpsc401/notes/02-evolution.html>

Pascal Bornet on X: "Timeline of the most popular programming languages <https://t.co/EqMMFHV8lf>"

# Petite histoire de la programmation

4

## History of Popular Programming Languages: A timeline



**BASIC** (Beginner's All-purpose Symbolic Instruction Code, 1964) Kemeny & Kurtz. Pour accommoder des non-scientifiques/techniques. Premier langage populaire à être majoritairement interprété et non compilé.

**Simula** (1967) Dahl & Nygaard. Premier langage orienté-objet. Introduit les notions de classe, objet, héritage, fonctions virtuelles, etc.

**Pascal** (1970) Niklaus Wirth

**C** (1972) Dennis Ritchie, AT&T Bell Labs. Langage de programmation système. Conçu pour être un langage assembleur portable.

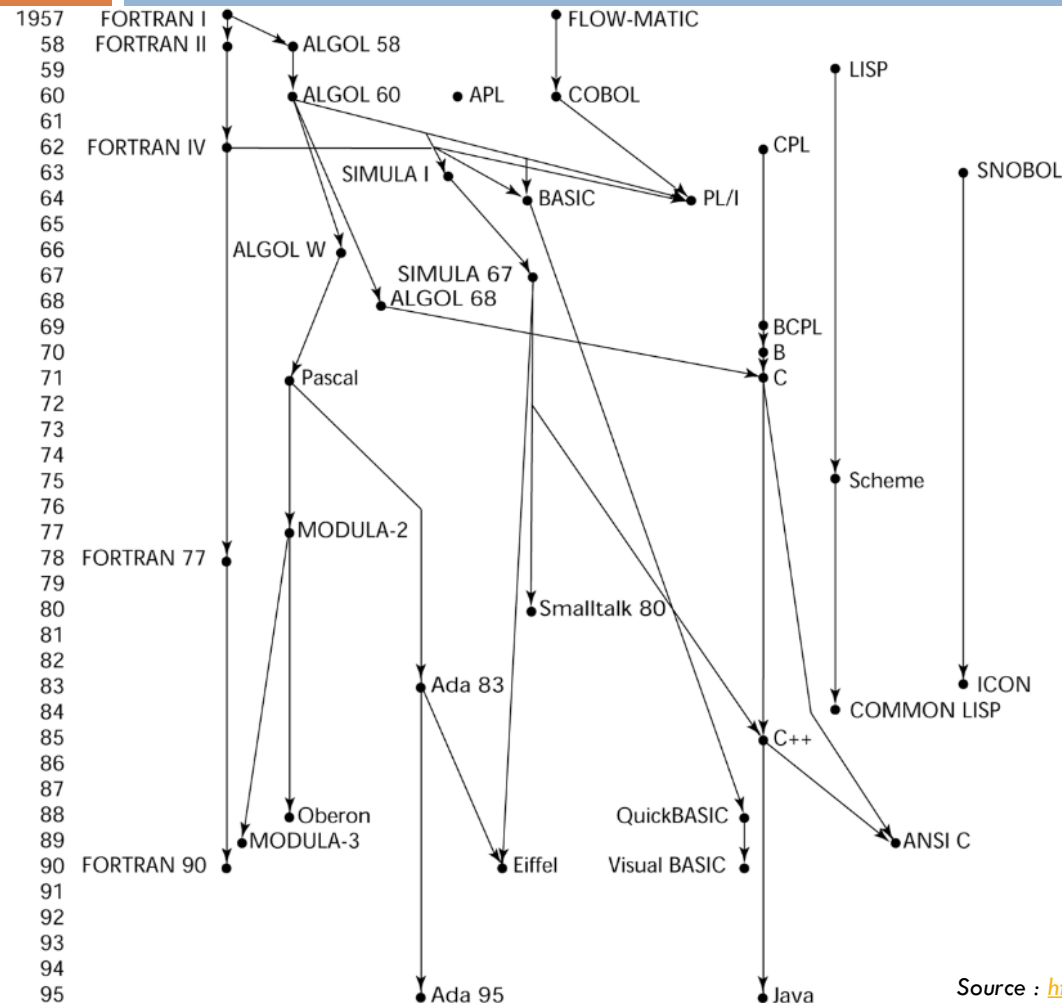
**Prolog** (1972) Langage de programmation logique (un pionnier et le plus populaire). Langage déclaratif.

Sources : <http://cs.umw.edu/~finlayson/class/spring15/cpsc401/notes/02-evolution.html>

Pascal Bornet on X: "Timeline of the most popular programming languages <https://t.co/EqMMFHV8lf>"

# Petite histoire de la programmation

5



**Ada** (1983) Département de la défense US, typage extrêmement fort. En hommage à Ada Lovelace

**C++** (1983) Bjarne Stroustrup. D'abord "new C", "C with classes". Performance, flexibilité

**MATLAB** (Matrix Lab), 1984. Manipulation de matrices, ingénieurs

**Eiffel** (Bertrand Meyer), 1985. Programmation par contrats, pre-/post-conditions, invariants

**Objective-C**, 1986 — **Perl**, 1987

**Python**, 1991 — **Visual Basic**, 1991

**Ruby**, 1993 — **Lua**, 1993. Langage de scripts, populaire dans le jeu vidéo

1995 : **Ada95**, **Delphi** (Object Pascal), **PHP**, **JavaScript**, **Java**

2000 : **ActionScript**

2001 : **C#**, **Visual Basic .NET**

# Paradigmes de programmation

6

- ***Spécifique au domaine vs. Usage général***
  - ▣ Spécifique au domaine: SQL, HTML, XML, UML, ...
  - ▣ Usage général: Pascal, Java, Python, C, C++, ...
- Exemples dans la diapositive suivante.

# Paradigmes de programmation

7

## □ Spécifique au domaine (e.g., SQL)

- Vous trouverez ci-dessous une sélection de la table « Clients » (EN: Customers) de l'exemple de base de données Northwind :

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

□ **SELECT** CustomerName, Country **FROM** Customers

Source: w3schools.com

# Paradigmes de programmation

8

## □ Usage Général (e.g., C++)

```
int num = 10;  
if (num < 10){  
    cout << 'Rien à faire!!!' << endl;  
}else{  
    int sqr_num = 10 * 10;  
    cout << 'Le carré de ' << num << ' est ' << sqr_num << endl;  
}
```



# Paradigmes de programmation

9

## □ Programmation **impérative** vs. **déclarative**

### ▣ Impérative (Comment le faire?)

- Procédurale (Pascal original)
- Orientée objet (C++, Java, C#)

### ▣ Déclarative (Quoi faire?)

- Fonctionnelle (LISP)
- Logique (Prolog)
- Programmation par contraintes (CSP)

# Paradigmes de programmation

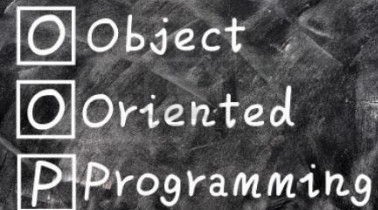
10

- Quelques catégories de langage :
  - ▣ Programmation système (ex: Assembly, FORTRAN, C)
  - ▣ Langage de script (ex: Python, Ruby, HTML)
  - ▣ Langage concurrent, distribué, parallèle (ex: C++, C, CUDA)
- Programmation générique
  - ▣ Les types peuvent être spécifiés plus tard
    - Mécanisme des *templates*
- Méta-programmation
  - ▣ Habilité du programme à traiter des programmes comme des données
    - Réflexion : habilité d'un programme à se traiter comme donnée

# Programmation Orientée Objet (POO)

11

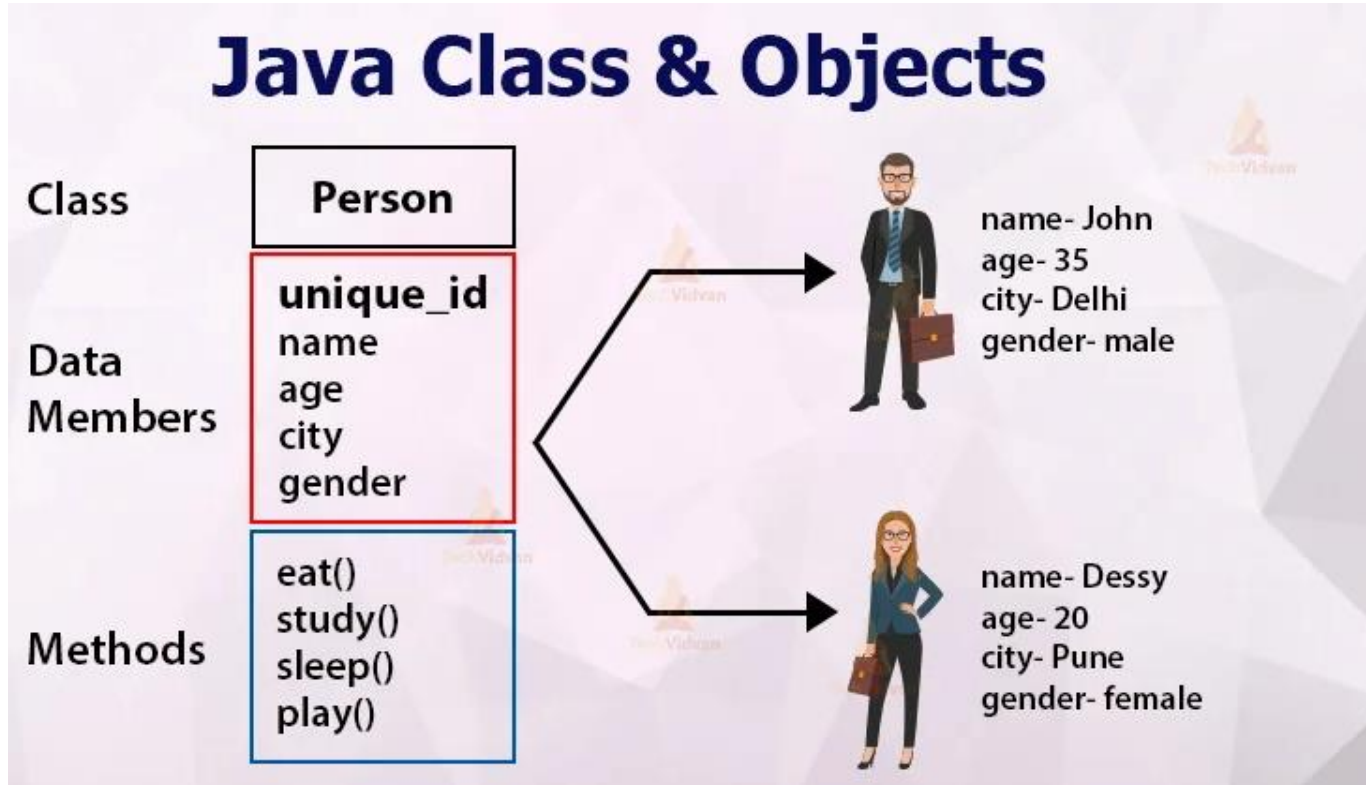
- Offre une **modélisation** du problème en se basant sur des **objets**
- Le programme est un ensemble d'**objets** et leurs **interactions**
- Les objets ont différentes **responsabilités**



□ Object  
□ Oriented  
□ Programming

# Programmation Orientée Objet (POO)

12



# Programmation Orientée Objet (POO)

13

- Plus simple à **concevoir** : on *modélise* le monde par des *entités*
- Plus simple à **modifier** et **faire évoluer** : favorise la **réutilisation** du code
- Améliore la **qualité** du logiciel : facilite la **factorisation** et la **revue** du code

CODE REVIEW



# Programmation Orientée Objet (POO)

14

- Analyse & Conception
  - ▣ On détermine les **objets**
  - ▣ On **décrit** les objets (*données*) et leurs **responsabilités**
  - ▣ On identifie les **opérations** (*moyen de les traiter*)
  - ▣ On identifie les **liens** entre les objets, et comment ils **interagissent**

# Programmation Orientée Objet (POO)

15

- **Classe** : *usine* qui permet de créer des *objets*
  - ▣ Décrit les *attributs* représentant l'**état**
  - ▣ Définit les *méthodes* pour le **comportement**
  
- Un **objet** créé à partir d'une **classe** est une *instance* de cette **classe**
  - ▣ `Student bob = new Student();`

# Programmation Orientée Objet (POO)

16

- ❑ **Encapsulation** : restreint l'accès aux **membres** (attributs) et **méthodes** (fonctions) d'un objet pour la **protection des données**
- ❑ Rendre visible *uniquement* ce qui est **nécessaire**
- ❑ Favorise la *sécurité* et la *robustesse*







DE RETOUR À JAVA...



# Caractéristiques du langage Java

18

- Portable, simple, robuste, larges fonctionnalités
- Semi-compilé (compilé puis interprété)
  - ▣ Code source (\*.java) traduit en code binaire (\*.class)
  - ▣ Code binaire interprété par la *machine virtuelle*

The logo for the Java Virtual Machine (JVM), consisting of the letters 'JVM' in a bold, purple, sans-serif font.

# Historique des *releases*

19

- JAVA (**orienté-objet**, structuré, impératif, fonctionnel, générique, réflexif, “concurrent”)
  - ▣ JDK 1.0 (January 1996) **Oak**
  - ▣ JDK 1.1 (February 1997) : classes internes, JavaBeans, JDBC, introspection
  - ▣ J2SE 1.2 (December 1998) : **PlayGround**. Swing
  - ▣ J2SE 1.3 (May 2000) : **Kestrel**. Hotspot JVM
  - ▣ J2SE 1.4 (February 2002) : **Merlin**. assert, support IPV6, parseur XML, etc.
  - ▣ J2SE 5.0 (September 2004) : **Tiger**
    - Programmation générique, annotations (@Override, @Deprecated, etc.), énumérations, autoboxing/unboxing (conversions automatiques entre types primitifs et objets), ...
  - ▣ Java SE 6 (December 2006) : **Mustang**. améliorations diverses (sécurité, JVM etc.)
  - ▣ Java SE 7 (July 2011) : **Dolphin**
    - Strings dans switch, inclusion de JavaFX dans le JRE et JDK
  - ▣ Java SE 8 (March 2014)

# Syntaxe Java : les bases

20

## □ Identifiants :

- Alphanumériques, \$, \_ (ne commence pas par un chiffre)

## □ Blocs de code délimités par { } : portée locale

## □ Mots-clés réservés :

abstract	continue	for	new	switch
assert	default	goto <sup>[*]</sup>	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const <sup>[*]</sup>	float	native	super	while

<sup>[\*]</sup> réservé mais non-utilisé

# Syntaxe Java : les bases

21

## □ Variable : valeur d'un type

- ▣ `int i = 3;`
- ▣ `float f = 5.9f;`
- ▣ `char c = 'A';`
- ▣ Sensible à la casse (majuscule / minuscule)

## □ Afficher du texte sur la sortie standard

- ▣ `System.out.print("message");` // **sans** nouvelle ligne
- ▣ `System.out.println("message");` // **avec** nouvelle ligne
- ▣ Généralement dans la console

# Syntaxe Java : les bases

22

## Commentaires :

□ //

□ /\* \*/

□ /\*\* \*/

(Javadoc)

C: Class, I: Interface,  
E: Enum, F: Field  
M: Method,  
OM: Overriding Method

Tag & Parameter	Usage	Applies to
<b>@author</b> <i>John Smith</i>	Describes an author.	C, I, E
<b>@version</b> <i>version</i>	Provides software version entry. Max one per Class or Interface.	C, I, E
<b>@since</b> <i>since-text</i>	Describes when this functionality has first existed.	C, I, E, F, M
<b>@see</b> <i>reference</i>	Provides a link to other element of documentation.	C, I, E, F, M
<b>@param</b> <i>name</i> <i>description</i>	Describes a method parameter.	M
<b>@return</b> <i>description</i>	Describes the return value.	M
<b>@exception</b> <i>classname</i> <i>description</i> <b>@throws</b> <i>classname</i> <i>description</i>	Describes an exception that may be thrown from this method.	M
<b>@deprecated</b> <i>description</i>	Describes an outdated method.	C, I, E, F, M
<b>{@inheritDoc}</b>	Copies the description from the overridden method.	OM
<b>{@link}</b> <i>reference</i>	Link to other symbol.	C, I, E, F, M

# Syntaxe Java : structure de programme

23

- Point d'entrée du programme (dans une classe) :

```
class MyApp {  
    public static void main(String[] args) { }  
}
```

- Packages et classes

```
package myapplication.mylibrary;  
public class MyClass { }  
  
myapplication/mylibrary/MyClass.java
```

- Importation de packages et d'attributs statiques

# Syntaxe Java : méthodes

24

## □ Déclaration de méthode :

```
public static type_retour nom_méthode(params) {  
    // ...  
}
```

## ▣ Zéro ou plus paramètre(s) séparés par une virgule

- type\_param nom\_param

- Copiés lors de l'appel : attention avec les objets (références)

## ▣ Toujours dans une classe

## ▣ Paramètres et variables sont **locaux** (portée entre {})

## ▣ return pour retourner une valeur (sauf si void)



# Syntaxe Java : structures de contrôle

25

## Conditions

```
    if ... else, else if
if (i == 3) { doSomething(); }
else if (i == 2) { doSomethingElse(); }
else { doSomethingDifferent(); }
```

```
    switch
switch (ch) {
    case 'A':
        doSomething();
        break;
    case 'B':
    case 'C':
        doSomethingElse();
        break;
    default:
        doSomethingDifferent();
        break;
}
```

```
    ? :
int a = 1;
int b = 2;
int minVal = (a < b) ? a : b;
```

a	b	a && b	a    b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

# Syntaxe Java : structures de contrôle

26

## Conditions



# Syntaxe Java : structures de contrôle

27

## Boucles

- `while (i < 10) { doSomething(); }`
- Appeler `doSomething()` au moins une fois :
  - ▣ `do { doSomething(); } while (i < 10);`
  - ▣ `break` : quitter la boucle
  - ▣ `continue` : sauter une itération
- `for (int i = 0; i < 10; i++) { doSomething(); }`
  - ▣ Les trois parties sont optionnelles
- `for (int i = 0, j = 9; i < 10; i++, j -= 3) { doSomething(); }`
- `for (int i : intArray) { doSomething(i); }`

# Syntaxe Java : structures de contrôle

28

## Boucles



## Transition vers les Boucles

- ❑ De « multiples instructions » vers 'While' ou 'For' (exemple de la somme des 100 premiers nombres)
- ❑ Parcourir l'intArray
- ❑ Trouver la position d'un élément du tableau

# Syntaxe Java : structures de contrôle

30

## Sauts

C'est du code mort  
car il ne s'exécute  
qu'une seule fois  
(l'étiquette `outerloop`)

```
outerloop:
for (int i = 0; i < 10; i++) {
    while (true) {
        break outerloop;
    }
}
```

```
char ch;
while (ch = getChar()) {
    if (ch == ' ') {
        continue;
    }
    doSomething();
}
```

```
static int sum(int n) {
    if (n < 0) return;
    int sum = 0;
    for (int i = 0; i < n; i++)
        sum = sum + i;
    return sum;
}
```

```
void doSomething(boolean streamClosed) {
    if (streamClosed) {
        return;
    }
    readFromStream();
}

int calculateSum(int a, int b) {
    int result = a + b;
    return result;
}
```

```
void doSomething(boolean streamClosed) {
    try {
        if (streamClosed) {
            return;
        }
        readFromStream();
    } finally {
        freeResources();
    }
}
```

ex: le fichier ne doit  
pas rester ouvert

# Syntaxe Java : primitives et tableaux

31

Primitive Types

Type Name	Wrapper class	Value	Size	Default Value
<b>byte</b>	java.lang.Byte	integer	8-bit	0
<b>short</b>	java.lang.Short	integer	16-bit	0
<b>int</b>	java.lang.Integer	integer	32-bit	0
<b>long</b>	java.lang.Long	integer	64-bit	0
<b>float</b>	java.lang.Float	Floating-point number	32-bit	0.0f
<b>double</b>	java.lang.Double	Floating-point number	64-bit	0.0
<b>boolean</b>	java.lang.Boolean	boolean	undefined	false
<b>char</b>	java.lang.Character	UTF-16 code unit	16-bit	'\u0000'

```
Integer bar;  
int foo = 42;  
bar = foo;  
int foo2 = bar;
```

```
int[] numbers = new int[5];  
numbers[0] = 2;  
numbers[1] = 5;  
int x = numbers[0];  
int size = numbers.length;
```

```
int[] numbers1 = new int[] { 20, 1, 42, 15, 34 };  
int[] numbers2 = { 20, 1, 42, 15, 34 };
```

```
int[][] numbers2d1 = new int[3][3];  
numbers2d1[1][2] = 2;  
int[][] numbers2d2 = {  
    {2, 3, 2},  
    {1, 2, 6},  
    {2, 4, 5}};
```

```
int[][] numbers = new int[2][];  
numbers[0] = new int[3];  
numbers[1] = new int[2];
```

# Syntaxe Java

32

