

INTRODUCTION AU LANGAGE JAVA



Motivation : Pourquoi étudier Java ?

2

Jan 2020	Jan 2019	Change	Programming Language	Ratings	Change
1	1		Java	16.896%	-0.01%
2	2		C	15.773%	+2.44%
3	3		Python	9.704%	+1.41%
4	4		C++	5.574%	-2.58%
5	7	⬆	C#	5.349%	+2.07%
6	5	⬇	Visual Basic .NET	5.287%	-1.17%
7	6	⬇	JavaScript	2.451%	-0.85%
8	8		PHP	2.405%	-0.28%
9	15	⬆	Swift	1.795%	+0.61%
10	9	⬇	SQL	1.504%	-0.77%
11	18	⬆	Ruby	1.063%	-0.03%
12	17	⬆	Delphi/Object Pascal	0.997%	-0.10%
13	10	⬇	Objective-C	0.929%	-0.85%
14	16	⬆	Go	0.900%	-0.22%
15	14	⬇	Assembly language	0.877%	-0.32%
16	20	⬆	Visual Basic	0.831%	-0.20%
17	25	⬆	D	0.825%	+0.25%
18	12	⬇	R	0.808%	-0.52%
19	13	⬇	Perl	0.746%	-0.48%
20	11	⬇	MATLAB	0.737%	-0.76%

→ Classement basé sur le TIOBE Index.

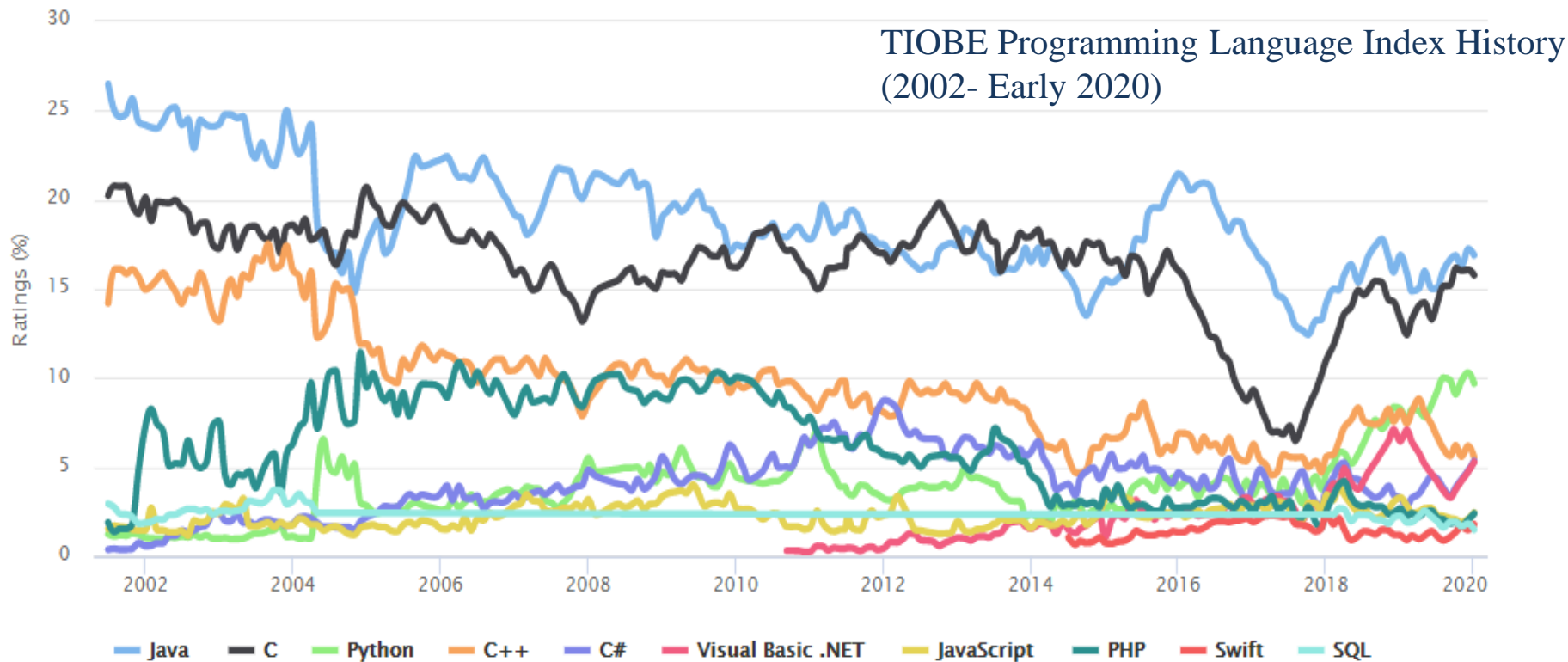
TIOBE Index : Index qui mesure chaque mois la popularité de chaque langage de programmation.

Motivation : Pourquoi étudier Java ?

3

TIOBE Programming Community Index

Source: www.tiobe.com



- **Langage java** : Éternel leader de l'index TIOBE depuis des années.
- Langage C en deuxième place, le plus utilisé par les développeurs pour les objets connectés dits contraints (Internet of things ou IoT en anglais).

Un peu d'histoire

4

- ❑ Le projet Java voit le jour en **1991**, dans le secret d'une équipe de Sun Microsystems.
- ❑ Cette équipe (**green team**) a cherché à concevoir **un langage applicable** à de petits appareils électriques (en code embarqué) : **Green Project**



Un peu d'histoire

5

- **Le Green Project** a donc démarré afin d'étudier l'impact de la convergence entre les **appareils ménagers** contrôlés numériquement et **les ordinateurs**. En utilisant une syntaxe proche de celle de C++
- Finalement, c'est **James Gosling**, l'un des membre du Green Projet, qui est à l'origine de ce langage nouveau, qu'il baptisa **Oak**.
- **Oak** devient alors **FirstPerson**. Pas de succès commercial pour **FirstPerson**.
- **En 1993**, on assiste à l'arrivée **du protocole http** et du **navigateur Mosaic**, ce qui fut un événement crucial pour le projet.
- C'est à cette période que l'équipe comprit qu'Internet serait **le réseau idéal** pour positionner leur produit. En 1995, James Gosling dévoila un navigateur appelé **WebRunner** capable de **montrer du contenu html mélangé à des applets**(petite application qui se télécharge lors de la consultation de certains sites Internet)
- WebRunner devient **HotJava** et **le site java.sun.com** s'ouvre officiellement au grand public.
- **le 23 mai 1995**, la dénomination de cette technologie sera **Java** (*café en argot américain*), en honneur à la boisson préférée des programmeurs, c'est-à-dire **le café**.
- **20 avril 2009** : Sun Microsystems annonce son rachat par **Oracle Corporation**

James Gosling : The founder and lead designer behind the Java programming language

6

James Gosling



Évolution des versions de Java au fil des années

7

- ❑ **1995**: JDK 1.0 en version Béta (expérimentale).
- ❑ **1996** : JDK 1.0, appelé Java 1.0
- ❑ **1997** : JDK 1.1, appelé Java 1.1
- ❑ **1998** : JDK 1.2, appelé J2SE 1.2
- ❑ **2000** : JDK 1.3, appelé J2SE 1.3
- ❑ **2002** : JDK 1.4, , appelé J2SE 1.4
- ❑ **2004** : JDK 1.5, appelé J2SE 1.5
- ❑ **2006** : JDK 1.6, appelé Java SE 6
- ❑ **2011** : JDK 1.7, appelé Java SE 7
- ❑ **2014** : JDK 1.8, appelé Java SE 8
- ❑ **septembre 2017** : JDK 9, appelé Java SE 9
- ❑ **mars 2018** : JDK 10, appelé Java SE 10
- ❑ **septembre 2018** : JDK 11, appelé Java SE 11
- ❑ **Mars 2019** : JDK12, Appelé Java SE 12
- ❑ **Septembre 2019** : JDK 13, appelé Java SE 13

JDK : Java Development Kit

J2SE : Java 2 Standard Edition

SE : Standard Edition

Environnements de développement intégré JAVA (IDE)

8

- ❑ Un IDE ou un Environnement de développement (Integrated Development Environment) est un logiciel qui rassemble des outils permettant de développer d'autres logiciels tels que des applications mobiles, des logiciels pour ordinateur ou consoles de jeux, des sites web, etc... ainsi que de réaliser des librairies ou des frameworks.

Les outils d'un IDE peuvent être :

- ❑ Un éditeur de code intelligent (Coloration, autocomplétions, mise en forme) ;
- ❑ Un simulateur (logiciel permettant de tester l'exécution de son logiciel) ;
- ❑ Un compilateur (qui va transformer le code source rédigé par le développeur en code binaire) ;
- ❑ Un débogueur (fonctionnalité d'aide à la correction de bugs).

Top 10 Java IDE's 2019

9

- ❑ Eclipse
- ❑ BlueJ
- ❑ IntelliJ IDEA
- ❑ JGRASP
- ❑ Netbeans
- ❑ Android Studio
- ❑ Java Inventor
- ❑ Codenvy
- ❑ JBuilder
- ❑ JCreator



Les plates-formes d'exécution JAVA

10

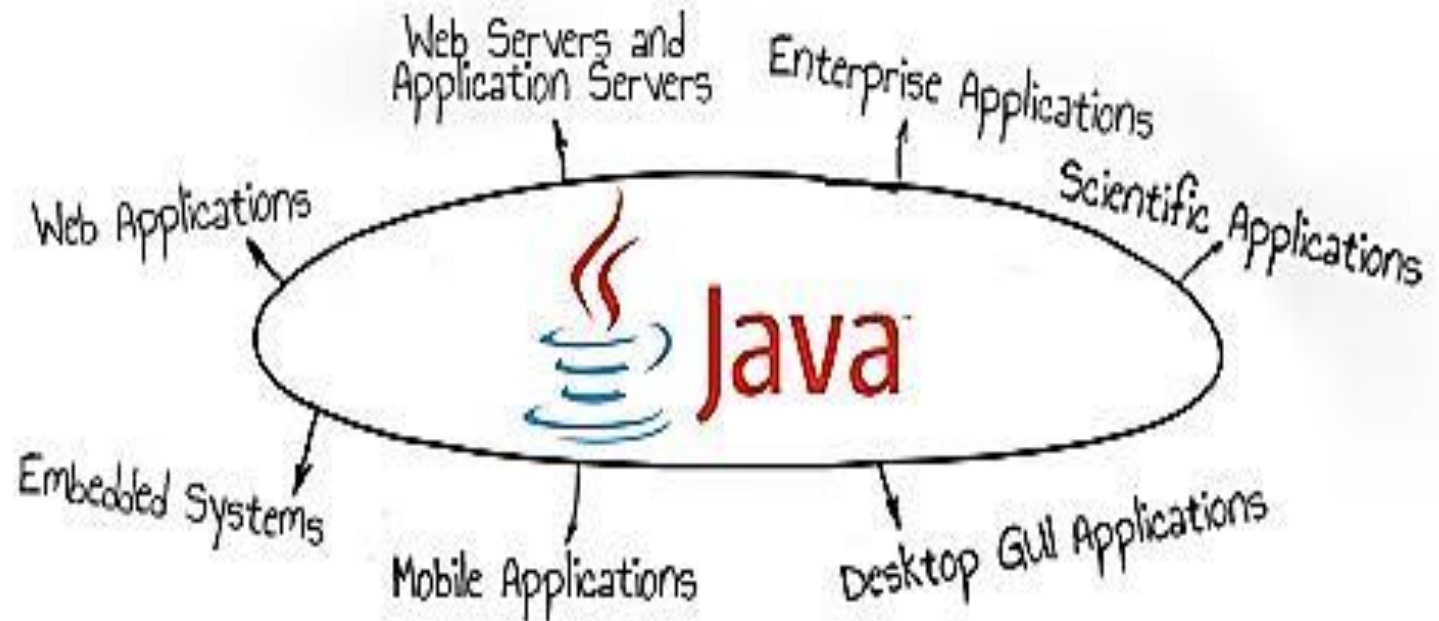
Trois environnements d'exécutions différents :

- ❑ **Java SE** (Java Platform, Standard Edition): Java SE pour applications classiques, desktop.
- ❑ **Java EE** (Java Platform, Enterprise Edition): Java EE pour développer et déployer des applications serveur, Web services, etc.
- ❑ **Java ME** (Java Platform, Micro Edition): JavaME pour les applications embarqués, PDA, téléphones, TV, etc.
- ❑ **Java Card** est utilisée pour les cartes à puce, telles que les cartes bancaires, les cartes SIM en communication mobile, etc.

Qu'est-ce qu'on peut produire avec JAVA ?

11

On peut faire de nombreuses sortes de programmes avec Java :









































































Statistiques d'oracle corporation

12

- ❑ 97% des bureaux d'entreprise exécutent Java.
- ❑ 89% des bureaux (ou ordinateurs) des États-Unis exécutent Java.
- ❑ 9 millions de développeurs Java dans le monde.
- ❑ Choix n°1 des développeurs.
- ❑ Plate-forme de développement n°1.
- ❑ 3 milliards de téléphones mobiles exécutent Java.
- ❑ 100% des lecteurs Blu-ray livrés avec Java.
- ❑ 5 milliards de cartes Java utilisées.
- ❑ 125 millions de périphériques TV exécutent Java.
- ❑ Les 5 fabricants d'équipement d'origine principaux fournissent Java ME

Motivation : Pourquoi JAVA est si Populaire ?

13

Caractéristique	Java	SmallTalk	TCL	Perl	Shells	C	C++
Simple							
Orienté Objet							
Robuste							
Sécurisé							
Interprété							
Dynamique							
Portable							
Architecture Neutre							
Multithread							
Exceptions							
Performance	Haute	Moyenne	Basse	Moyenne	Basse	Haute	Hautes

Points forts de Java?

14

- ❑ Langage orienté objet
- ❑ Langage Modulaire
- ❑ Langage portable
- ❑ Langage rigoureux
- ❑ Langage sécurisé

La programmation procédurale vs orientée objet

15

Programmation procédurale



VS Programmation orientée objet



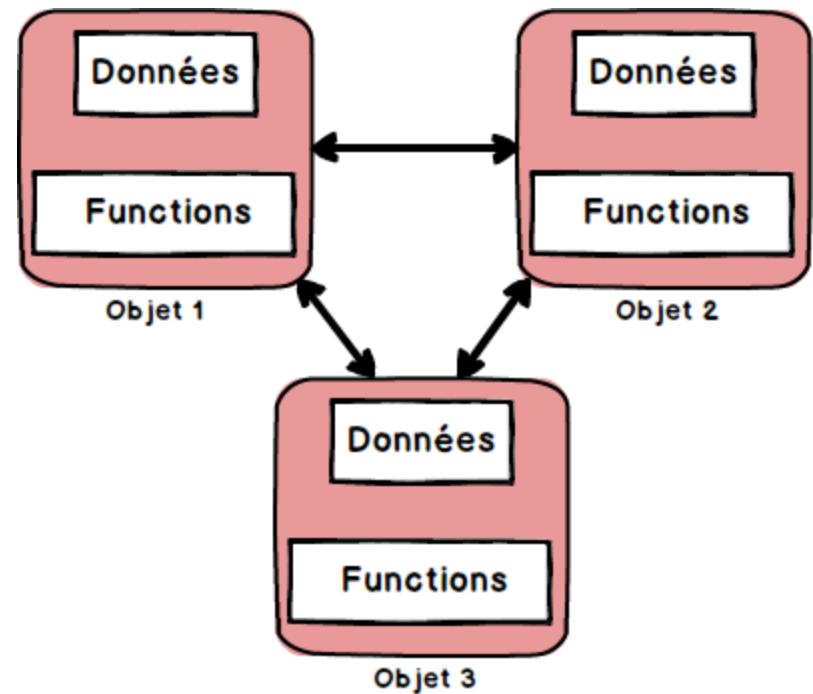
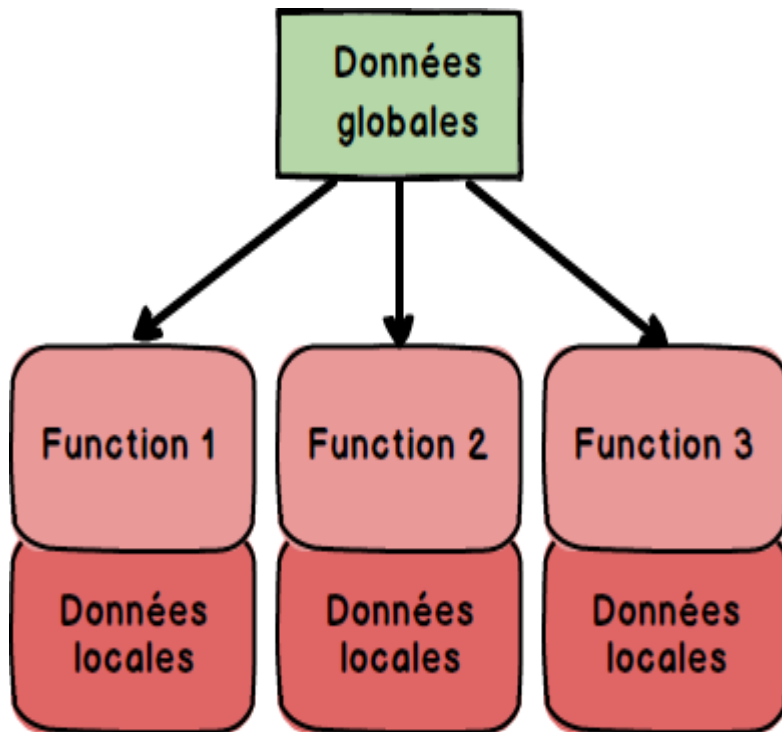
La programmation procédurale vs orientée objet

16

Programmation procédurale

vs

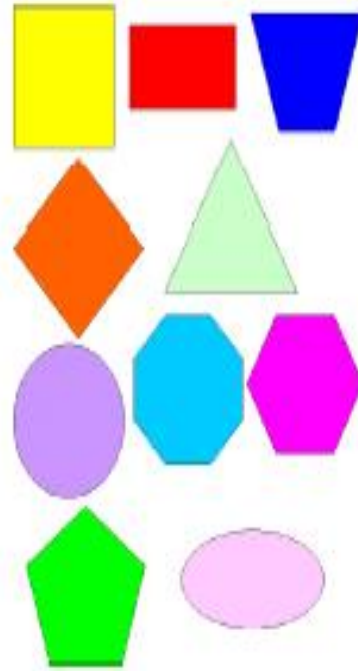
Programmation orientée objet



JAVA : Langage orienté objet

17

Objets



Classe:

Vehicule

FormeGeometrique

Animal

JAVA : Langage orienté objet

18

Notion de classe d'objets

- ❑ Pour un développeur java tout est objet.
- ❑ Un objet est une variable (presque) comme les autres. Il faut notamment qu'il soit déclaré avec son type.
- ❑ Le type d'un objet est un type complexe (par opposition aux types primitifs entier, caractère, ...) qu'on appelle une classe.
- ❑ le programmeur écrit uniquement des classes correspondant aux objets de son système

JAVA : Langage orienté objet (Classe)

19

Une classe regroupe :

- ❑ Un ensemble de données (qui peuvent être des variables primitives ou des objets)
- ❑ un ensemble de méthodes de traitement de ces données et/ou de données extérieures à la classe.
- ❑ On parle d'encapsulation pour désigner le regroupement de données dans une classe.

Exemple d'une classe en code JAVA

20

Le nom de la classe

```
class Rectangle {
```

```
    int longueur;  
    int largeur;  
    int origine_x;  
    int origine_y;
```

Les
attributs de
la classe

```
    Rectangle(int lon, int lar) {  
        this.longueur = lon;  
        this.largeur = lar;  
        this.origine_x = 0;  
        this.origine_y = 0;  
    }
```

Le
constructeur
de la classe

```
    void deplace(int x, int y) {  
        this.origine_x = this.origine_x + x;  
        this.origine_y = this.origine_y + y;  
    }
```

Méthode 1
de la classe

```
    int surface() {  
        return this.longueur * this.largeur;  
    }
```

Méthode 2
de la classe

```
}
```

JAVA : Langage orienté objet (Objet)

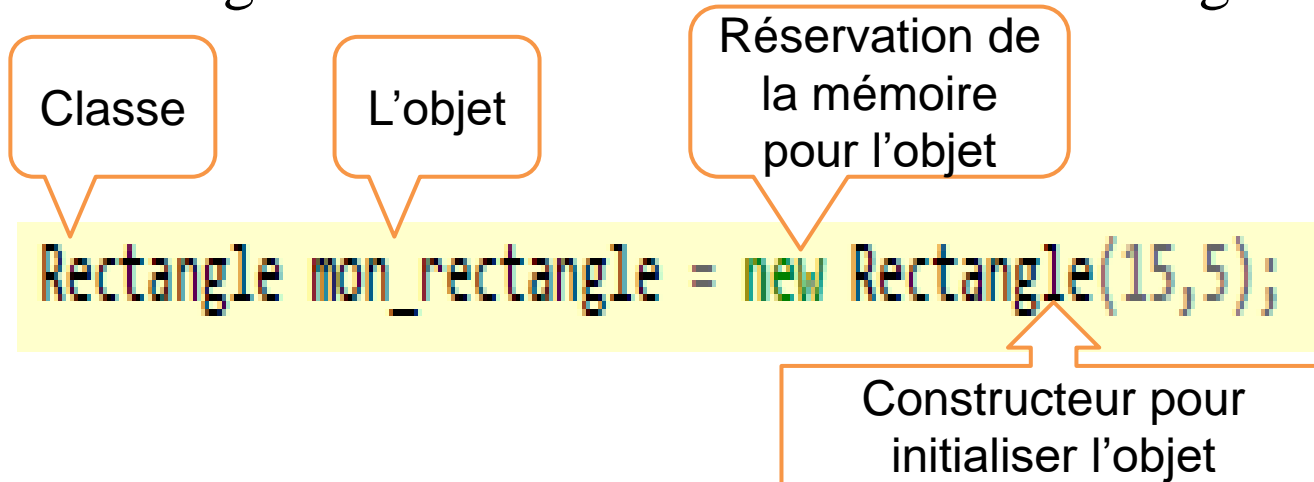
21

- ❑ Un objet est une instance (anglicisme signifiant « cas » ou « exemple ») d'une classe
- ❑ Il est référencé par une variable ayant un état (ou valeur).
- ❑ Pour créer un objet, il est nécessaire de déclarer une variable dont le type est la classe à instancier, puis de faire appel à un constructeur de cette classe.

Exemple d'objets de la classe Rectangle

22

Objet rectangle = Instanciation de la classe Rectangle



Accès aux variables

This diagram shows how to access a variable attribute. A box labeled 'Nom de l'objet' points to 'mon_rectangle' in the code, and a box labeled 'attribut' points to 'longueur'.

Nom de l'objet

attribut

```
int temp = mon_rectangle.longueur;
```

Accès aux méthodes

This diagram shows how to access a method. A box labeled 'Nom de l'objet' points to 'mon_rectangle', and a box labeled 'Nom méthode' points to 'deplace'.

Nom de l'objet

Nom méthode

```
mon_rectangle.deplace(10,-3);
```

Programme JAVA

23

- Tout programme JAVA a au moins une classe.
- Règle: toute classe publique doit être dans un fichier qui a le même nom que la classe
- Règle: tout code doit être à l'intérieur d'une classe

Classe
exécutable

```
public class HelloWorld {  
    /* Un style de commentaire  
       sur plusieurs lignes. */  
    public static void main(String[] args) {  
        // Un commentaire sur une seule ligne  
        System.out.println("Bonjour à vous les IR1!");  
    }  
}
```

La
méthod
e Main

- Ça définit une classe, qui est une unité de compilation
- Comme il y a une méthode main, cette classe est « exécutable »
- Pour exécuter un programme java il faut au moins une fonction main.

Points forts de Java?

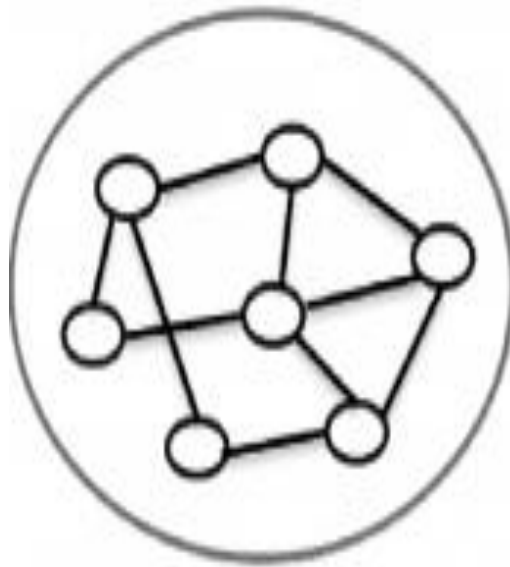
24

- ❑ Langage orienté objet
- ❑ **Langage Modulaire**
- ❑ Langage portable
- ❑ Langage rigoureux
- ❑ Langage sécurisé

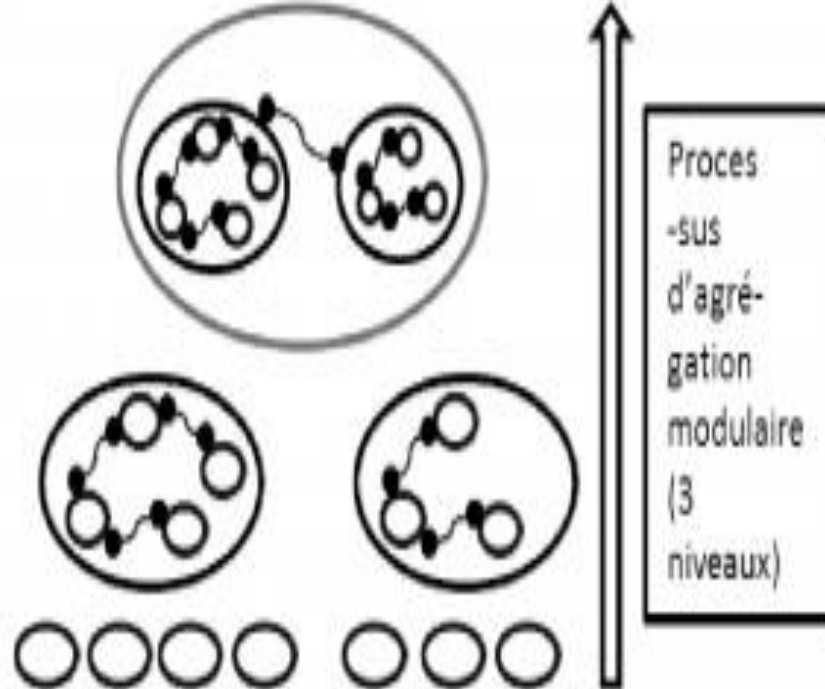
Architecture intégrale vs architecture modulaire

25

Architecture intégrale



Architecture modulaire



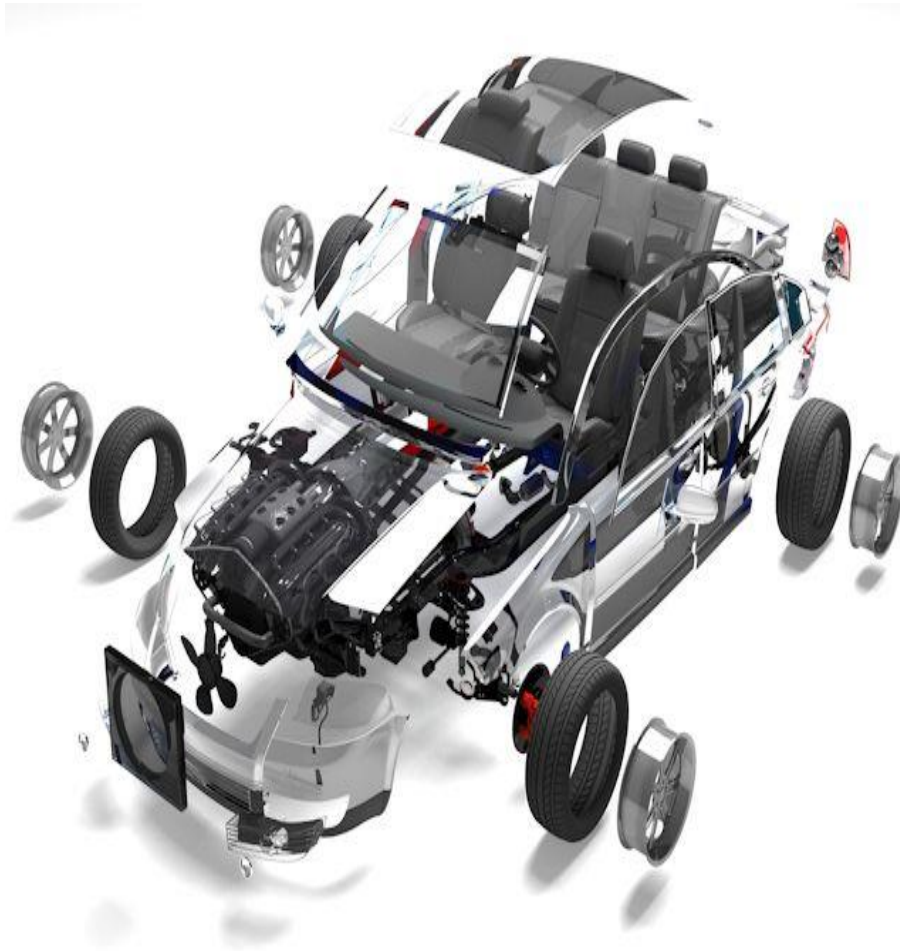
Légende :

○ Composant élémentaire ; — Connexion physique entre élément ; — Interface modulaire

→ La modularité vise à réduire le degré de complexité qui affecte tout système technique complexe.

JAVA : Langage modulaire !!

26

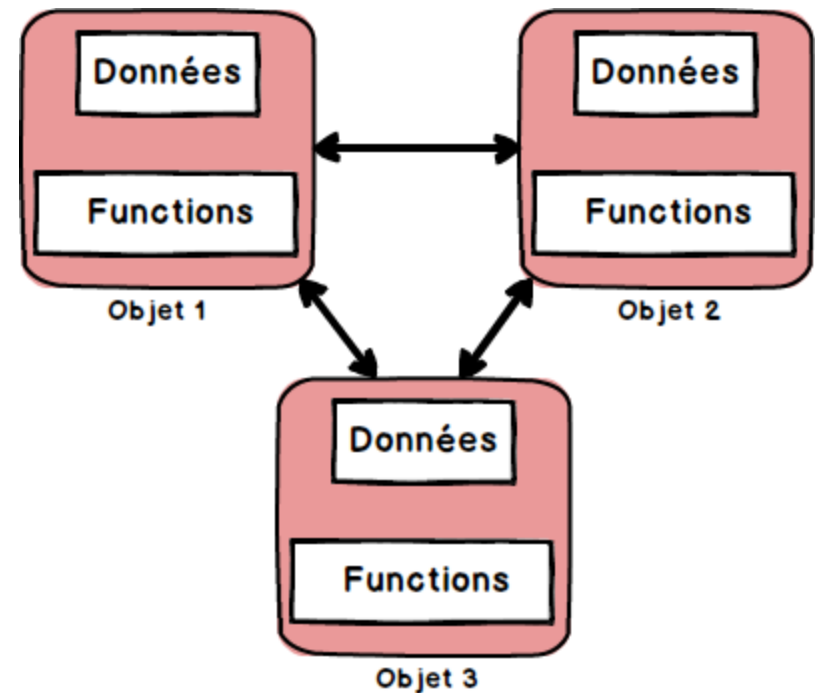
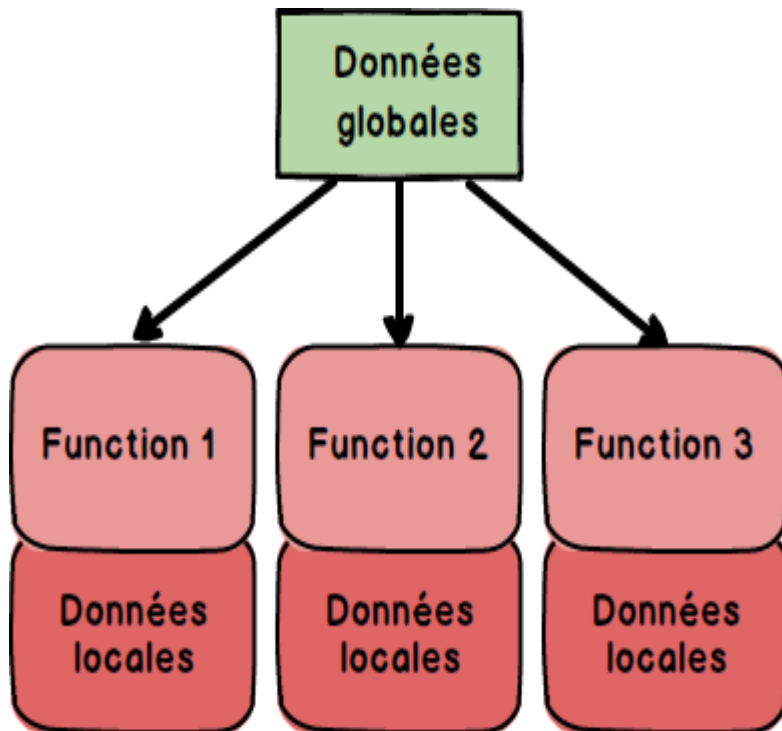


La programmation procédurale vs orientée objet

27

Programmation intégrale

vs Programmation modulaire



JAVA : Langage Modulaire

28

- ❑ La conception par classes, représentant à la fois les données, les actions et les **responsabilités** des objets de cette classe, permet de bien **distinguer et séparer les concepts**.
- ❑ Le fait de définir des « **interfaces** », au sens « moyens et modalités de communication avec l'extérieur » permet de **cacher** les détails d'**implémentation** et d'éviter les dépendances trop fortes.
- ❑ Tout ça favorise la réutilisabilité et la **composition / délégation** : l'assemblage des composants en respectant leurs responsabilités

JAVA : Langage Modulaire

29

L'objectif de la modularité est de produire du code :

- ❑ Facile à développer, à maintenir et à faire évoluer.
- ❑ Réutilisable, tout ou en partie, sans avoir besoin de le dupliquer.
- ❑ Générique, et dont les spécialisations sont transparentes.

Points forts de Java?

30

- ❑ Langage orienté objet
- ❑ Langage Modulaire
- ❑ **Langage portable**
- ❑ Langage rigoureux
- ❑ Langage sécurisé

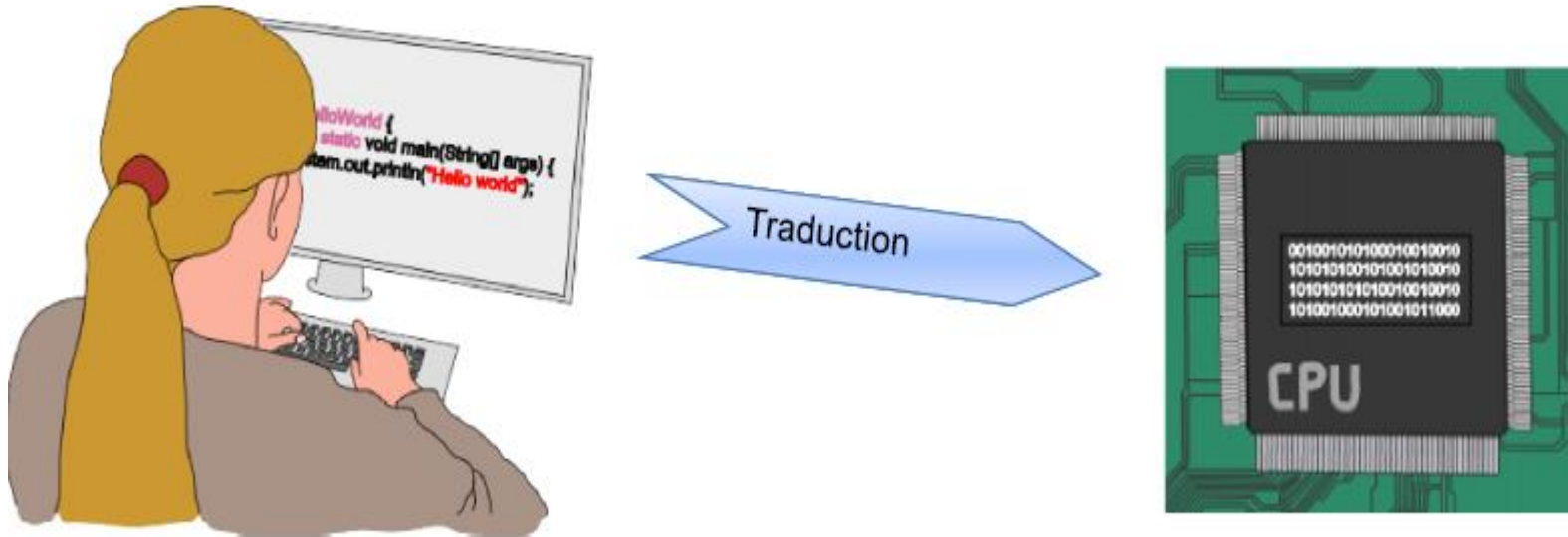
Rappel : Compilateur vs Interpréteur

31

- ❑ le **compilateur** traduit les programmes dans leur ensemble : tout le programme doit être fourni en bloc au compilateur pour la traduction.
- ❑ l'**interpréteur** traduit les programmes instruction par instruction dans le cadre d'une interaction continue avec l'utilisateur.

JAVA : Langage Portable

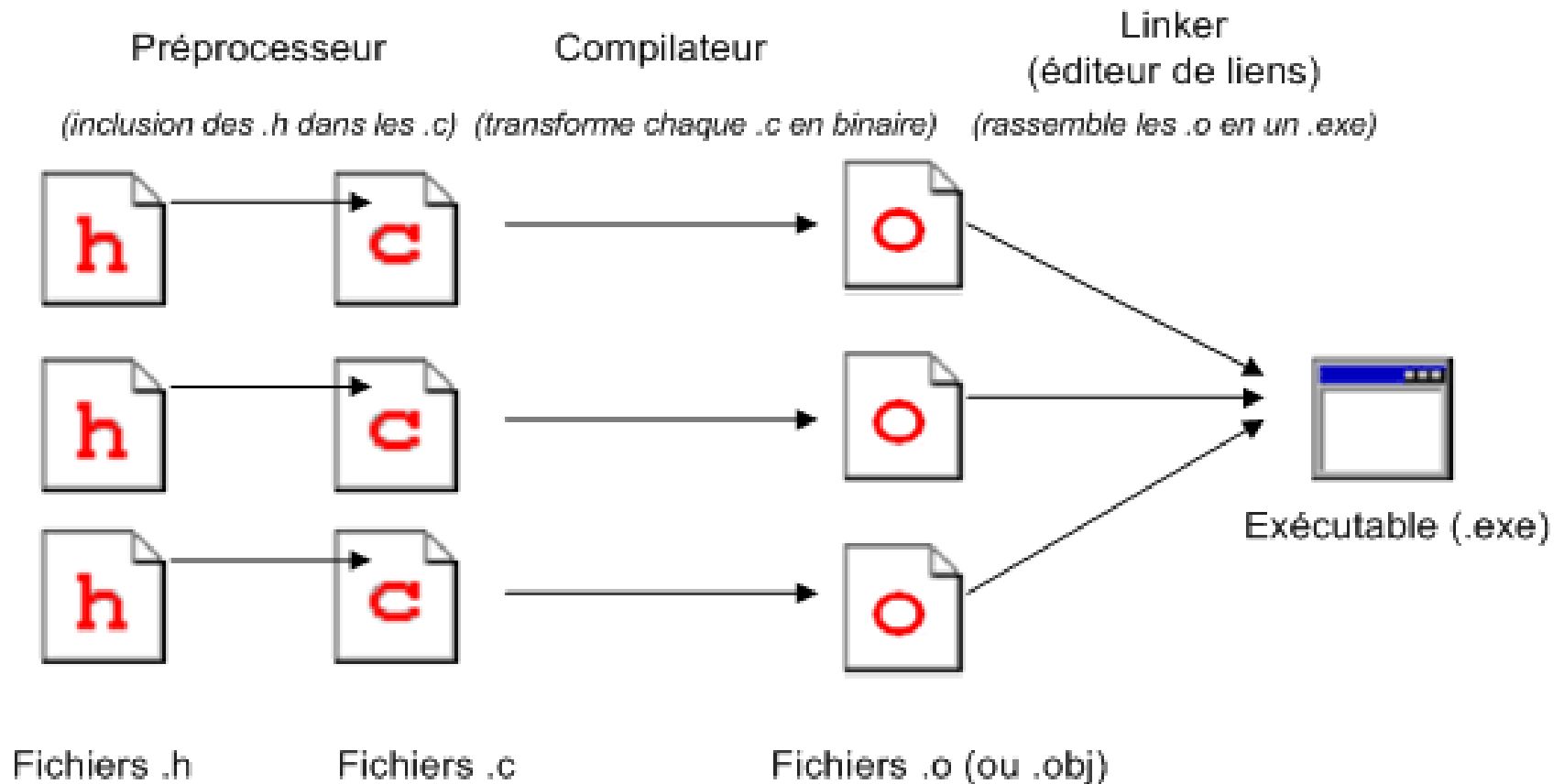
32



N.B : JAVA utilisent à la fois un compilateur et un interpréteur.
(Pas comme C/C++ qui utilisent juste un compilateur)

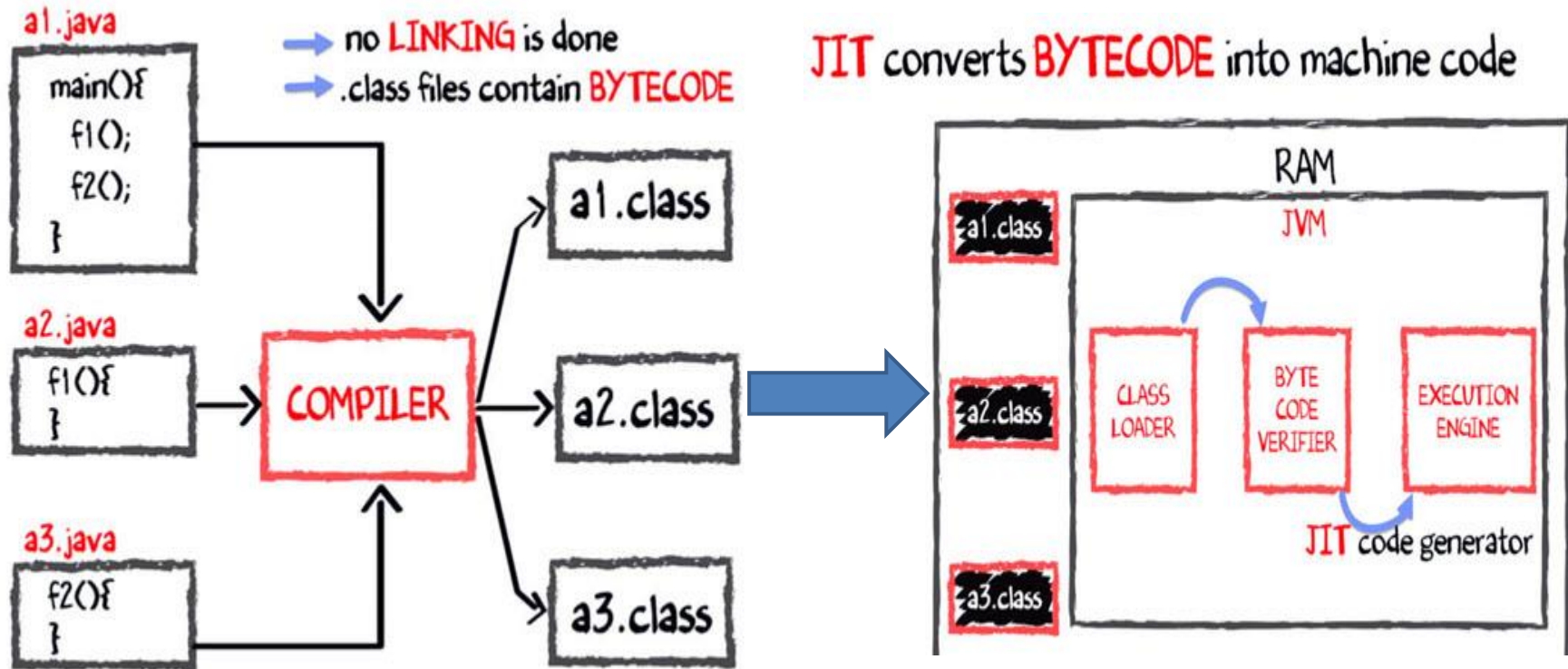
Rappel : compilation d'un programme en C/C++

33



JAVA : Langage Portable

34



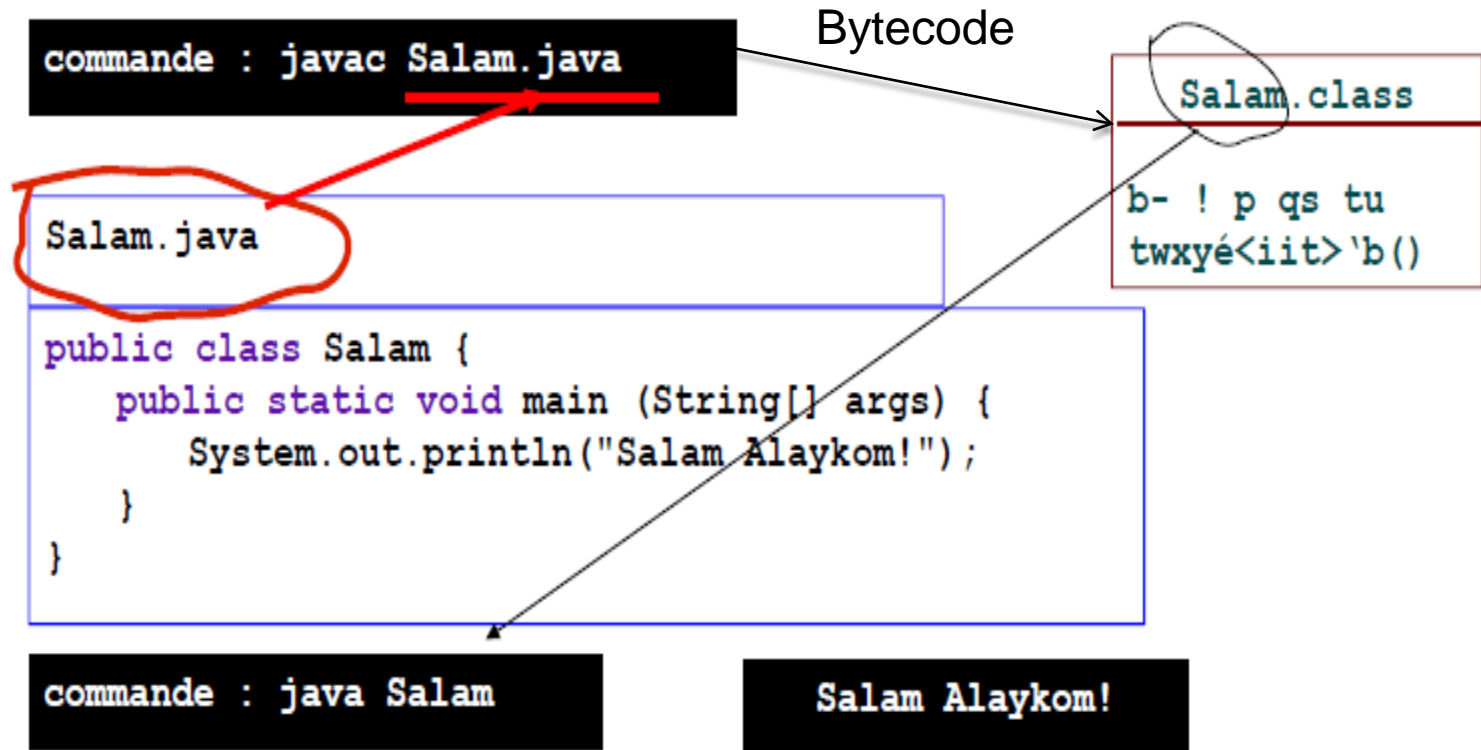
JAVA : Langage Portable

35

- ❑ Un programmeur Java écrit son code source, sous la forme de classes, dans des fichiers dont l'extension est **.java**
- ❑ Ce code source est alors compilé par le compilateur **javac** en un langage appelé **bytecode** et enregistre le résultat dans un fichier dont l'extension est **.class**
- ❑ Il doit être interprété par la **machine virtuelle de Java** qui transforme alors le code compilé en **code machine** compréhensible par la machine.
- ❑ **C'est la raison pour laquelle Java est un langage portable** : le bytecode reste le même quelque soit l'environnement d'exécution.

JAVA : Langage Portable

36



L'interpréteur Java s'appelle la machine virtuelle Java
(Java Virtual Machine (JVM))

JAVA : Langage Portable

37

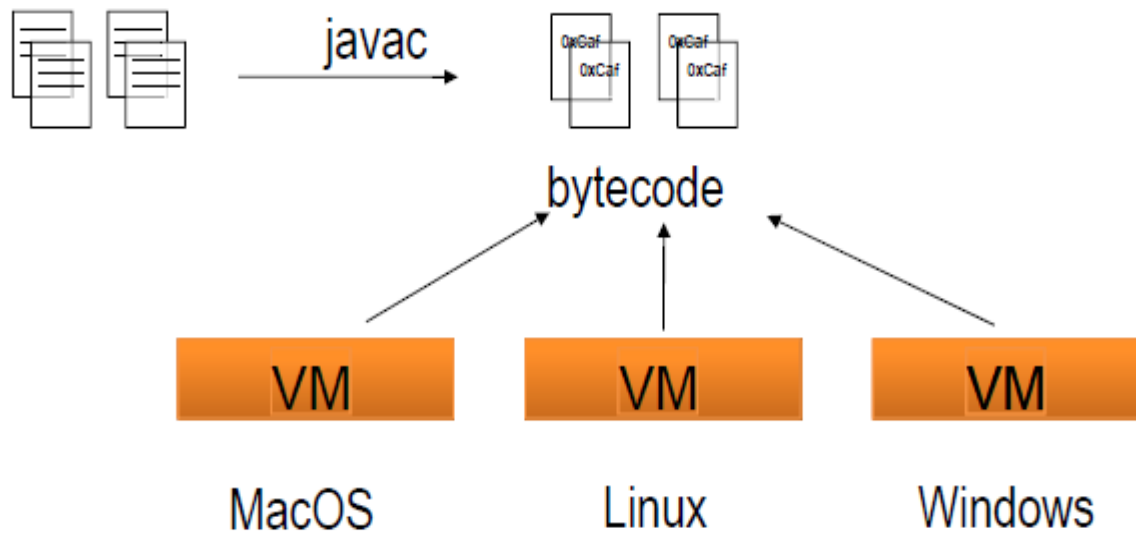
Bytecode

- ❑ Bytecode Java est un langage intermédiaire entre compilation et exécution.
- ❑ Le terme Bytecode vient de Byte car chaque instruction de la JVM est codée à l'aide d'un Byte (à chaque instruction correspond un opcode ou code opération de 8 bits).
- ❑ Exemple : l'addition de deux entiers est représenté par l'opcode **IADD** et correspond à **60** en **hexadécimal**.

JAVA : Langage Portable

38

- Le byte-code assure la portabilité entre différents environnements (machine/OS)



Et Solaris, HP-UX, BSD etc...

Points forts de Java?

39

- ❑ Langage orienté objet
- ❑ Langage Modulaire
- ❑ Langage portable
- ❑ Langage rigoureux
- ❑ Langage sécurisé

JAVA : Langage rigoureux et sécurisé

40

- ❑ Java est un langage **rigoureux** car la plupart des erreurs se produisent à la compilation et non à l'exécution.
- ❑ Java est un langage très rigoureux sur le typage de données.
- ❑ Java est un langage **sécurisé** car la JVM contrôle le champ d'exécution d'un programme pour empêcher d'aller faire des choses non autorisé :
 - ✓ accéder directement à certaines fonctions;
 - ✓ Injecter du code;
 - ✓ exécuter du code non prévus en injectant des paramètres;
 - ✓ accéder directement à la mémoire;
 - ✓ etc...
- ❑ Il améliore la **sécurité des données** grâce à l'encapsulation de données.

Autres Avantages : Ramasse miette (Garbage collector)

41

- ❑ Ramasse miette → Gestion automatique de la mémoire. (En C/C++ gestion manuelle **malloc and free**).
- ❑ Le ramasse-miettes est une fonctionnalité de la JVM qui a pour rôle de gérer la mémoire notamment en libérant celle des objets qui ne sont plus utilisés.

Mais comment ??

Le ramasse miette a plusieurs rôles :

- s'assurer que tout objet dont il existe encore une référence n'est pas supprimé.
- récupérer la mémoire des objets inutilisés (dont il n'existe plus aucune référence)
- éventuellement défragmenter (compacter) la mémoire de la JVM selon l'algorithme utilisé.
- intervenir dans l'allocation de la mémoire pour les nouveaux objets à cause du point précédent.



Autres Avantages

42

- ❑ Pas de la notion de pointeurs.
- ❑ Les **Api Java** dispose d'une **bonne documentation** et d'**exemples** trouvables sur internet.
- ❑ La **communauté Java** est **énorme**. Une question, un problème, il y aura toujours quelqu'un pour vous répondre.
- ❑ La **rapidité** de conception d'un programme, console, IHM, bref c'est simple et rapide. Le fait que des tas de classes, api existent déjà vous font gagner un temps fou.
- ❑ Java est en **constante évolution** du fait de sa masse importante d'utilisateur.

Cycle de développement d'un programme java

43

1. Les phases **d'écriture** et de **correction** se fait à l'aide d'un éditeur de texte (emacs, geany, gedit, Notepad++, . . .) ou d'une IDE (Netbeans, Eclipse, ...).

Exemple: **Notepad++ Salut.java**

2. La phase de **compilation** se fait à l'aide de la commande javac.

Exemple: **javac Salut.java**

3. La compilation produit un fichier **.class**.

Exemple: le fichier **Salut.class** est généré par le compilateur

4. La phase **d'interprétation** se fait à l'aide de la commande **java**.

Exemple: **java Salut**

Conventions de nommage en Java

44

Nom de classe :

- ❑ Commencez chaque mot par une majuscule :
 - HelloWorld
 - MonPremierProgramme

Nom de variable ou méthode :

- ❑ Commencez chaque mot par une majuscule, sauf le premier :
 - maPremiereVariable
 - maPremiereMethode
 - nb1
 - main
- ❑ Usage non recommandé :
 - mapremierevariable
 - ma_premiere_variable

N.B : Il faut évidemment que l'identificateur ne soit pas un mot réservé du langage (comme int out et for).

La nature des variables en Java

45

Les **variables locales** comme les **champs** des classes et des objets ne peuvent être que de deux natures :

- ❑ De **type « primitif »**
- ❑ De **type évolué** : tableau ou chaîne de caractères (String) ou objet (Classe).

Types primitifs

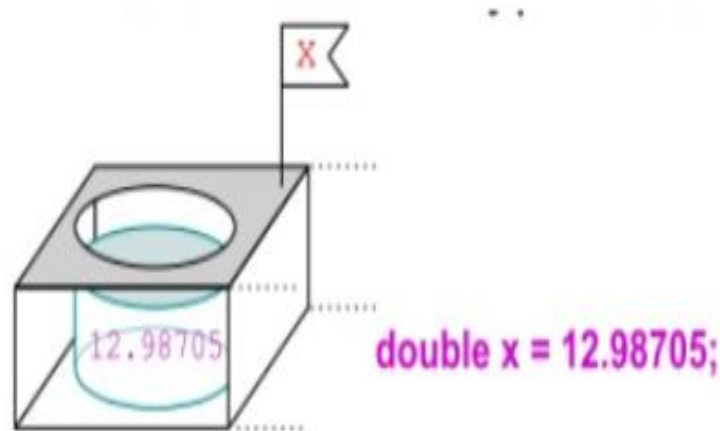
46

Type primitif	Signification	Place occupée en mémoire
byte	Entier très court allant de -128 à +127	1 octet
short	Entier court allant de -32768 à +32767	2 octets
int	Entier allant de -2 147 483 648 à +2 147 483 647	4 octets
long	Entier long allant de -2^{63} à $+2^{63} - 1$	8 octets
float	Nombre réel allant de $-1.4 * 10^{-45}$ à $+3.4 * 10^{38}$	4 octets
double	Nombre réel double précision allant de $4.9 * 10^{-324}$ à $+1.7 * 10^{308}$	8 octets
char	Caractère unicode (65536 caractères possibles)	2 octets
boolean	variable booléenne (valeurs : vrai ou faux)	1 octet

Type primitif

47

Dans ce cas de type, la déclaration de la variable réserve la place mémoire pour stocker sa valeur (qui dépend de son type)



Type évolué

48

La déclaration d'une variable de type évolué ne fait que réserver la place d'une **référence** (une sorte de pointeur) qui permettra d'accéder à l'endroit en mémoire où est effectivement stocké un :

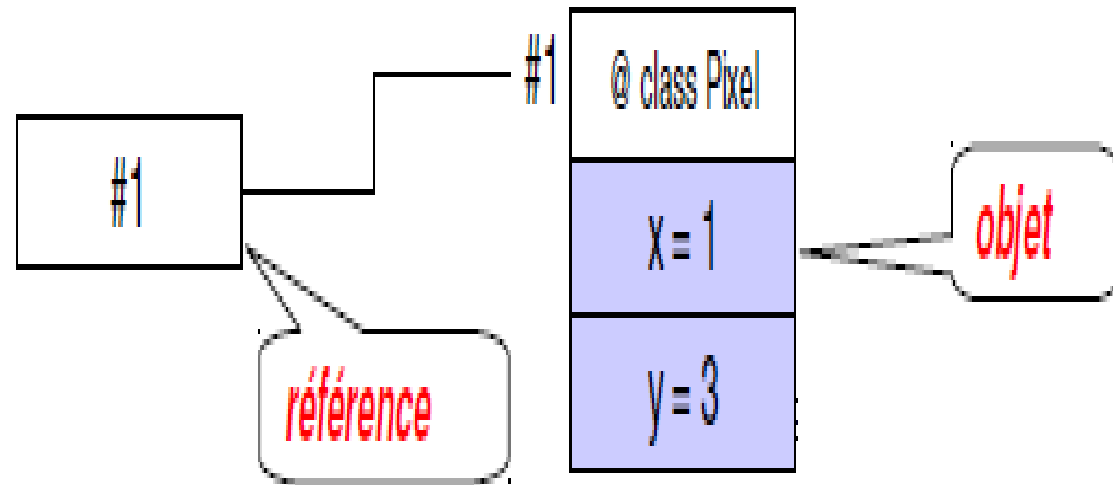
- ❑ objet
- ❑ ou une chaîne de caractère (String)
- ❑ ou un tableau

null si la référence est inconnue.

Type évolué : objet (Classe)

49

```
Pixel p1;  
p1=new Pixel(1,3);
```



Type évolué : Les tableaux

50

Deux syntaxes pour l'allocation :

`int[] monTableau = new int[10];`

`int monTableau[] = new int[10];`

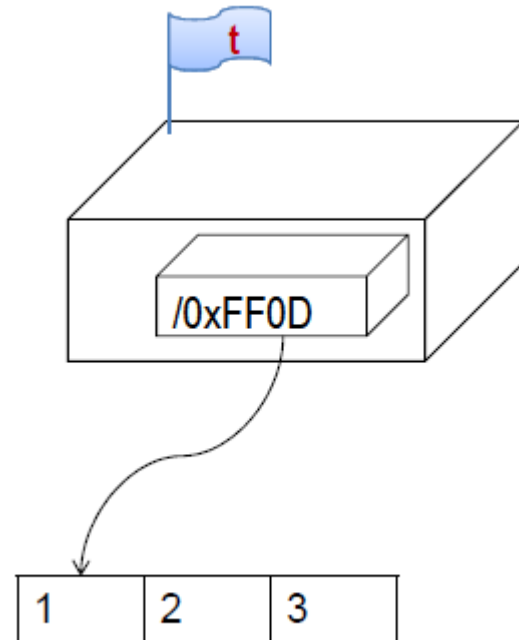
```
public class TabPoint {
    public static void main(String[] arg) {
        Point[] tp = new Point[3];
        tp[0] = new Point(1, 2);
        tp[1] = new Point(4, 5);
        tp[2] = new Point(8, 9);
        for (int i=0; i<tp.length; i++) tp[i].affiche();
    }
}

class Point {
    private int x, y;
    public Point(int x, int y) { this.x = x; this.y = y; }
    public void affiche() {
        System.out.println("Point : " + x + ", " + y);
    }
}
```

Type évolué : Les tableaux

51

```
int [] t = {1, 2, 3};
```



Type évolué : chaîne de caractères (String)

52

- ❑ Une chaîne de caractères est déclarée avec le mot-clé « **String** ».
- ❑ Les chaînes de caractères ne sont pas considérées en Java comme un type primitif ou comme un tableau de caractère. On utilise une classe particulière, nommée **String**, fournie dans le package **java.lang**.

```
String s = new String(); //pour une chaine vide  
String s2 = new String("hello world");  
// pour une chaîne de valeur "hello world"
```

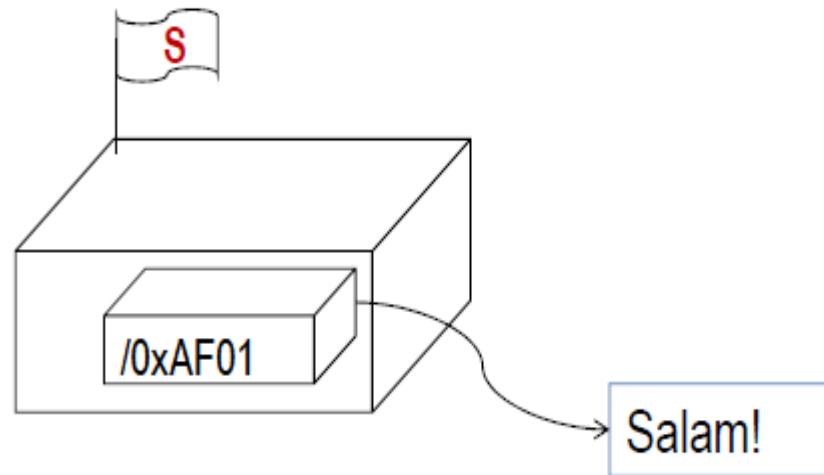
- ❑ Il se manipule comme un type de base :
String s = "hello world"; ⇔ **String s = new String ("hello world");**
- ❑ L'opérateur «+» est utilisé pour concaténer les chaînes de caractères.
- ❑ Exemple :

```
String s1 = "hello";  
String s2 = "world";  
String s3 = s1 + " " + s2;  
//Après ces instructions s3 vaut "hello world"
```

Type évolué : chaîne de caractères (String)

53

```
String s = "Salam!";
```



Java est un langage à typage fort :

- Chaque variable a un type.
- La valeur doit être du même type que la variable.
- On ne peut (normalement) pas mélanger des variables de type différent.

Pourquoi?

- Utilisation efficace de la mémoire.
- Vérification de la logique du programme.

Typage fort

55

Un **int** à droite peut devenir un double à gauche mais pas vice versa :

```
int i = 1;
```

```
double d = 1.7;
```

`d = i;` Oui, .0 ajouté (la valeur de d est 1.0) :

→ **Transtypage automatique** `int` → `double`

`i = d;` Non !!!

Mais on peut forcer une conversion `double` → `int` :

`i = (int) d;` Oui, valeur tronquée, i vaut 1

→ **Transtypage explicite (explicit cast)** : `double` → `int`

Attention : Valeur tronquée ne veut pas dire valeur arrondie !

Entrées-Sorties Standard

56

Un minimum d'interactivité...

Affichage :

`System.out.print("bonjour")` / `System.out.println("bonjour")` : prédéfinis dans Java.

Lecture (d'un entier) :

```
import java.util.Scanner ;  
Scanner keyb = new Scanner(System.in) ;  
int i = keyb.nextInt() ;
```

Lecture d'une chaîne de caractère

```
import java.util.Scanner ;  
Scanner sc = new Scanner(System.in);  
System.out.println("Veuillez saisir un mot :");  
String str = sc.nextLine();  
System.out.println("Vous avez saisi : " + str);
```


Opérateurs dans JAVA

57

Opérateurs

Opérateurs arithmétiques

*	multiplication	
/	division	
%	modulo	
+	addition	
-	soustraction	
++	incrémentation	(1 opérande)
--	décrémentation	(1 opérande)

Opérateurs de comparaison

==	teste l'égalité logique
!=	non égalité
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal

Opérateurs logiques

&&	"et" logique	
	ou	
^	ou exclusif	
!	négation	(1 opérande)

Priorités (par ordre décroissant, tous les opérateurs d'un même groupe sont de priorité égale) :

! ++ --, * / %, + -, < <= > >=, == !=, ^ &&, ||

Notation abrégée : $x = x <op> y$, où $<op>$ est un opérateur arithmétique, peut aussi s'écrire : $x <op>= y$

(exemple : $x += y$)

Commentaire

58

(non traités par le compilateur)

/ commentaire sur une ou plusieurs lignes */*

```
/*Ce commentaire nécessite  
2 lignes*/
```

```
int a;
```

// commentaire de fin de ligne

```
int a; // ce commentaire tient sur une ligne
```

```
int b;
```

*/** commentaire d'explication */*

```
/**  
    Une classe pour donner un <b>exemple</b> de documentation HTML.  
*/  
public class Exemple {  
    /** ...Documentation du membre de type entier nommé exemple... */  
    public int exemple;  
}
```

- ❑ Ils sont récupérés par l'utilitaire **javadoc** et inclus dans la documentation ainsi générée.

Affectation

59

le signe **=** est l'opérateur d'affectation et s'utilise avec une expression de la forme:

variable = expression

ou

Variable = valeur;

L'opération d'affectation est associative de droite à gauche : il renvoie la valeur affectée ce qui permet d'écrire :

x = y = z = 0;

Structures de contrôle conditionnelles

60

L'instruction conditionnelle if : La syntaxe de l'instruction if

```
if (expression) instruction;
```

Ou :

```
if (expression) {  
    instruction1;  
    instruction2;  
}
```

Structures de contrôle conditionnelles

61

```
if (expression) {  
    instruction1;  
} else {  
    instruction2;  
}
```

Exemple :

```
if (a<b) {  
    min=a;  
} else {  
    min=b;  
}
```

Structures de contrôle conditionnelles imbriquées

62

```
if (expression1) {  
    bloc1;  
}  
else if (expression2) {  
    bloc2;  
}  
    else if (expression3) {  
        bloc3;  
    } else {  
        bloc5;  
    }  
}
```

Structures de contrôle à choix multiple

63

L'instruction Switch: la syntaxe est la suivante:

```
Switch ( variable ) {  
Case valeur1 : instructions1 ; break;  
Case valeur2 : instructions2 ; break;  
Case valeur3 : instructions2 ; break;  
.  
.  
Case valeurN : instructionsN ; break;  
Default : instructions ; break;  
}
```

Structures de contrôle répétitives

64

```
for (int i=0 ; i<limite ; i=i+increment) {  
    instructions;  
}
```

Exemple :

```
for (inti = 2; i < 10;i++) {  
    System.out.println("Vive Java !");  
}
```


Structures de contrôle répétitives

65

```
while (condition){  
Instructions;  
}
```

Exercice : qu'affiche le code suivant ?

```
int i=5;  
while (i > 1) {  
    System.out.print(i + " ");  
    i = i / 2 ;  
}
```

Structures de contrôle répétitives

66

L'instruction `do .. while`

```
do{  
    Instructions;  
} while (condition)
```

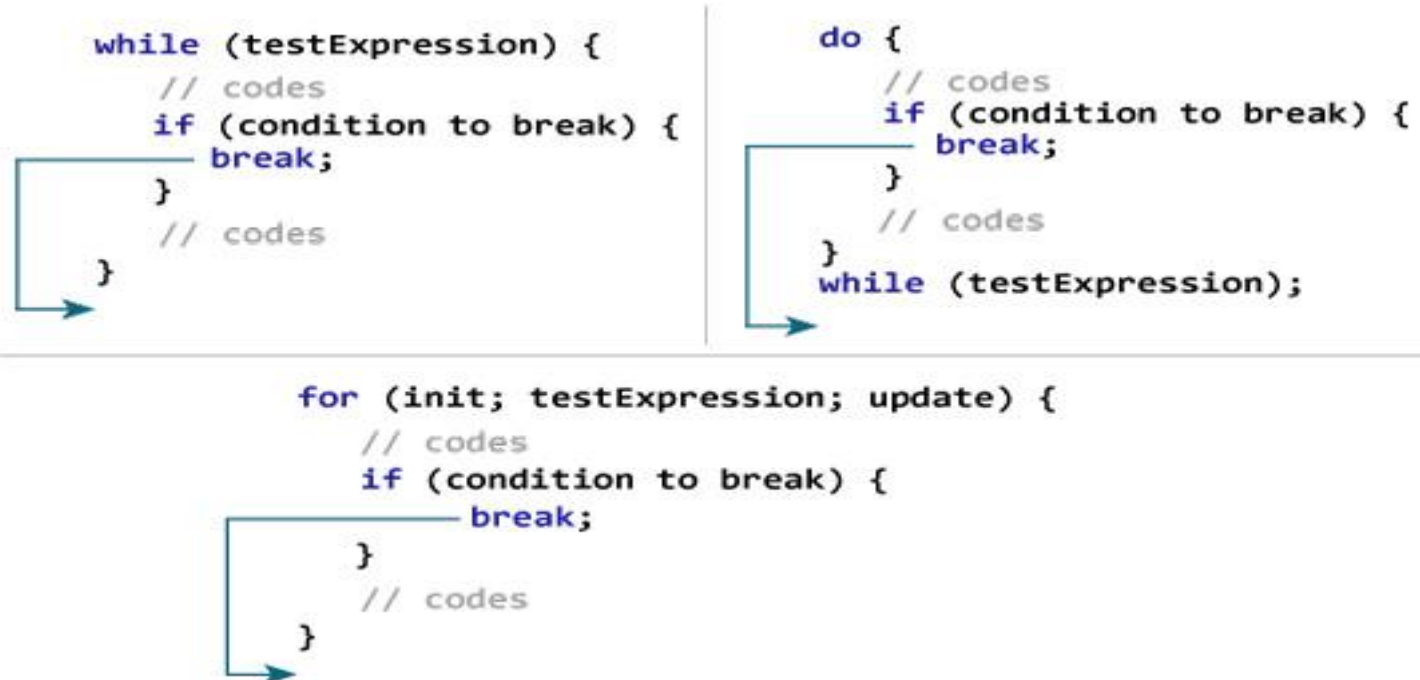
Exercice : qu'affiche le code suivant ?

```
int i=10;  
do{  
    System.out.print(i + " ");  
    i = i / 2 ; }  
while (i > 1)
```

Instruction break

67

break : l'exécution se poursuit après la boucle (comme si la condition d'arrêt devenait vraie) ;



Instruction continue

68

continue : l'exécution du bloc est arrêtée mais pas celle de la boucle.

```
→ while (testExpression) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```

```
do {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
} → while (testExpression);
```

```
→ for (init; testExpression; update) {  
    // codes  
    if (testExpression) {  
        continue;  
    }  
    // codes  
}
```



Merci de votre attention