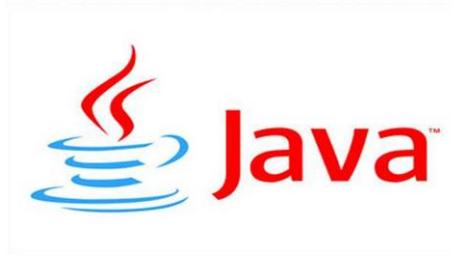


Programmation Orientée Objet

Application avec Java



Problématique du développement logiciel

La construction d'un logiciel est complexe quand elle met en oeuvre de nombreuses ressources

1. humaines
2. matérielles
3. technologiques

Suivre :

1. Un processus bien défini :
 - Prévoir et planifier les travaux
 - Coordonner les activités de conception, de fabrication et de validation
 - Réagir à l'évolution des objectifs
2. Une méthode rigoureuse basée sur des modèles
 - Représenterations sémantiques simplifiées d'un système visant à l'analyser et à le comprendre pour mieux le concevoir

Les étapes du cycle de vie du logiciel

1. L'expression des besoins (client, fournisseur)

- Les fonctionnalités du système étudié
- Comment utiliser ce système ?

2. Les spécifications du système (utilisateur, expert, fournisseur)

- Lever les ambiguïtés, éliminer les redondances du cahier des charges

3. L'analyse (utilisateur, expert, fournisseur)

- Phase indépendante de toute considération technique et informatique visant à définir le système (s'accorder sur le «quoi»)

4. La conception (expert en informatique)

- Prise en compte de l'environnement technique pour déterminer la manière de résoudre le problème posé (s'accorder sur le «comment»)

5. L'implémentation (expert en informatique)

- Traduction de la conception dans un langage de programmation, en une base de données, etc.

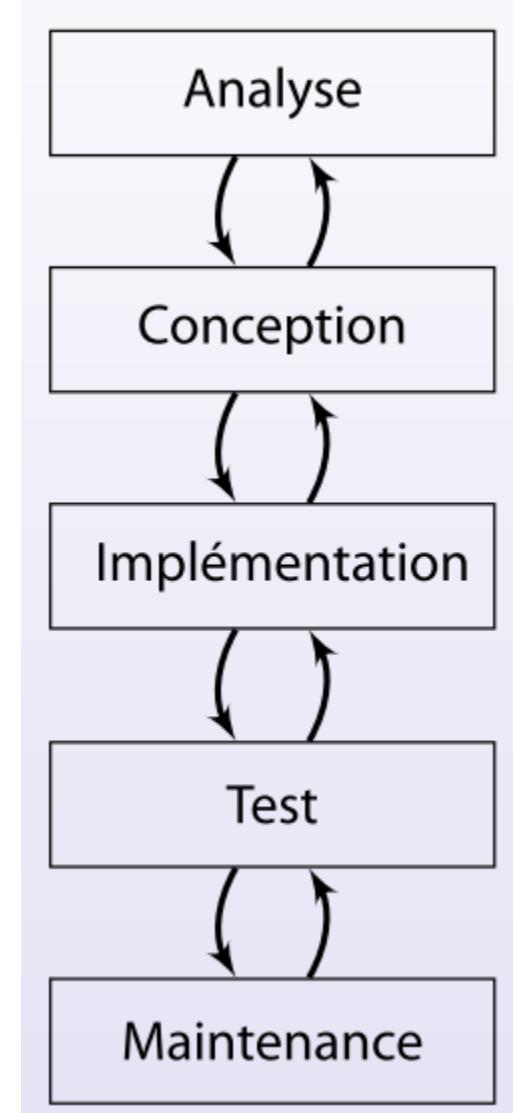
6. Les tests (expert en informatique)

- Vérifier que l'implémentation est correcte

7. La validation (utilisateur, expert, fournisseur)

- Vérification que le système correspond aux besoins

8. La maintenance et l'évolution pendant la phase d'exploitation



La qualité d'un logiciel

On peut mesurer la qualité d'un logiciel par :

1. **L'exactitude** : aptitude d'un programme à fournir le résultat voulu et à répondre ainsi aux spécifications
2. **La robustesse** : aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation
3. **L'extensibilité** : facilité avec laquelle un programme pourra être adapté pour répondre à l'évolution des spécifications
4. **La réutilisabilité** : possibilité d'utiliser certaines parties du logiciel pour résoudre un autre problème
5. **La portabilité** : facilité avec laquelle on peut exploiter un même logiciel dans différentes implémentations
6. **L'efficience** : temps d'exécution, taille mémoire

Grandes lignes du cours

- Introduction et éléments de base du Java
- Classes et Objets
- Encapsulation
- Héritage
- Polymorphisme
- Classe Abstraite
- Interface
- Collections
- Exceptions
- Classes imbriquées
- Interfaces Graphiques
- Gestion des Evénements

Chapitre 1

Introduction & éléments
de base du langage Java

chapitre 1

■ Introduction à la POO

■ Principes de la POO

■ Présentation du langage Java

■ Syntaxe et bases du langage Java

chapitre 1

 **Introduction à la POO**

 **Principes de la POO**

 **Présentation du langage Java**

 **Syntaxe et bases du langage Java**

Que représente pour vous, la programmation orientée objet?

Représentation du monde réel



- **Paradigme** au sein de la programmation informatique.
- Capacité de regrouper des paramètres dans un **ensemble**.
- Ensemble de principe clés qui permettent la mise en place d'un **logiciel**.
- Modèle de langage de programmation qui s'articule autour d'**objets** et de **données**, plutôt que d'**actions** et de **logique**.

Problématique de la programmation

Le cycle de vie d'un système peut être décomposé en deux grandes phases :

- **Une phase de production** qui consiste à réaliser le logiciel.
- **Une phase de maintenance** qui consiste à corriger et à faire évoluer le logiciel.

Lors de la production du système (au sens industriel du terme), le concepteur a deux grandes options :

- soit orienter sa conception sur les **traitements**.
- soit orienter sa conception sur les **données**.

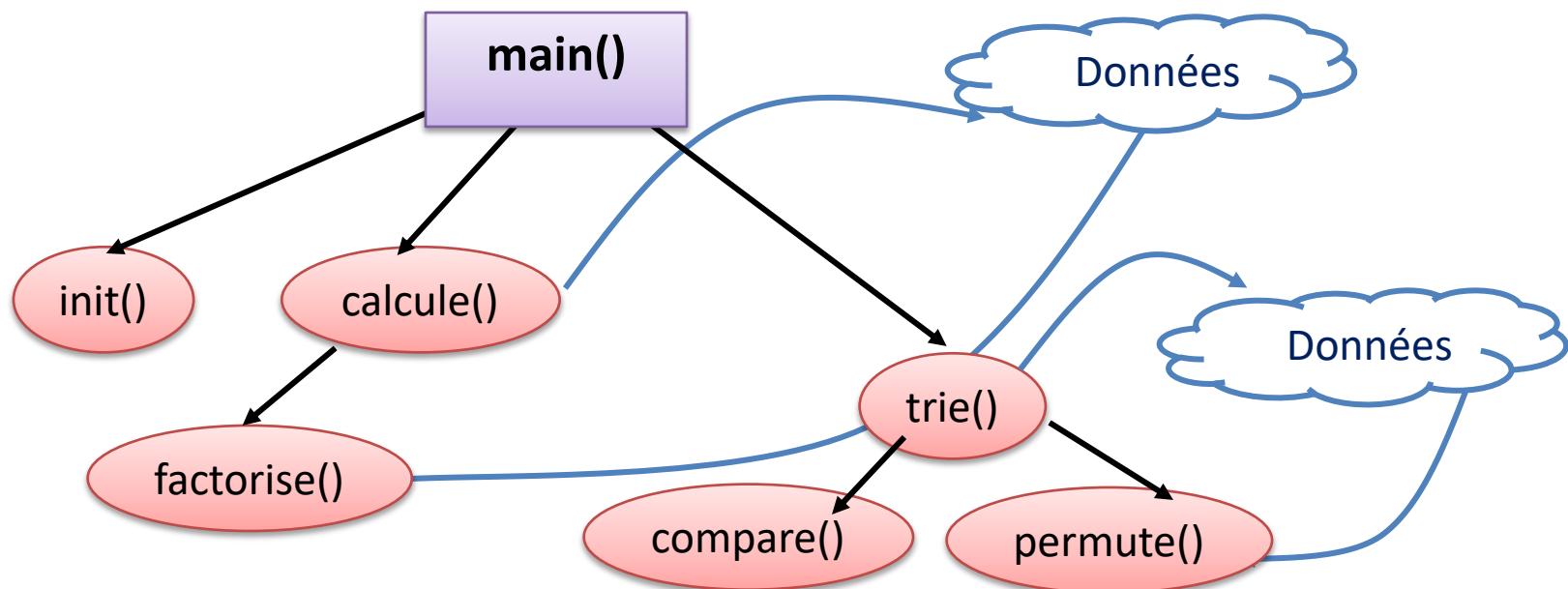
Conception par traitement

- Repose sur l'équation suivante :

$$\text{Programme} = \text{Structures de données} + \text{Treatment}$$

Dans la PP, la conception d'un programme est conduite par traitements :

- Consiste à décomposer le programme en modules simples.
 - effectuent un traitement sur des données (procédure)
 - retournent une valeur après leur invocation (fonction)



Conception par Objet

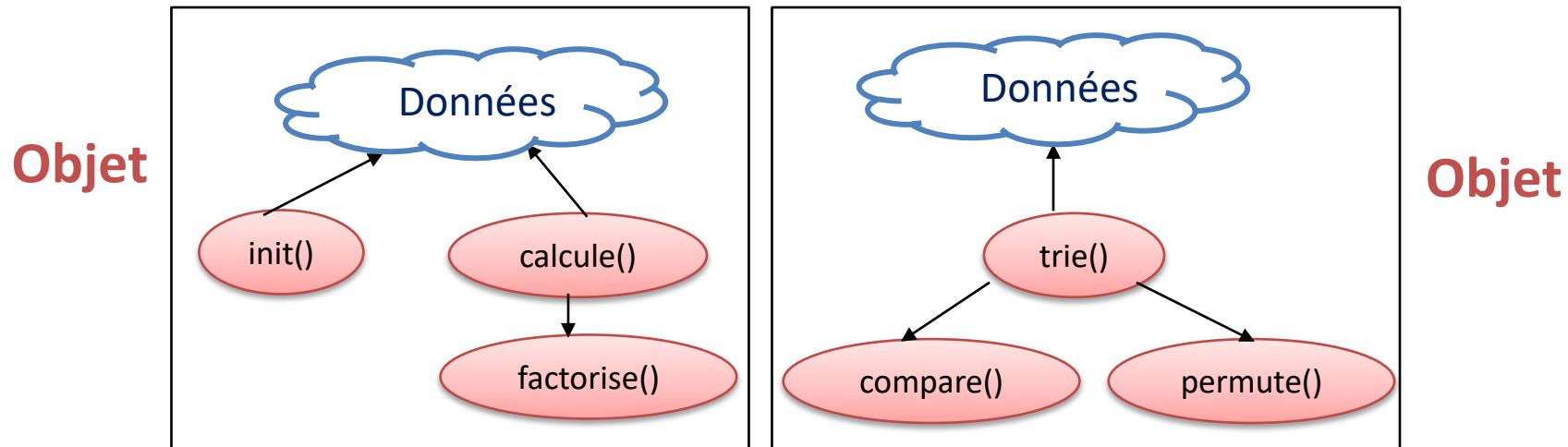
- Repose sur l'équation suivante :

$$\text{OBJET} = \text{Données} + \text{Méthodes}$$

Dans la POO, la conception d'un programme est basée sur les données.

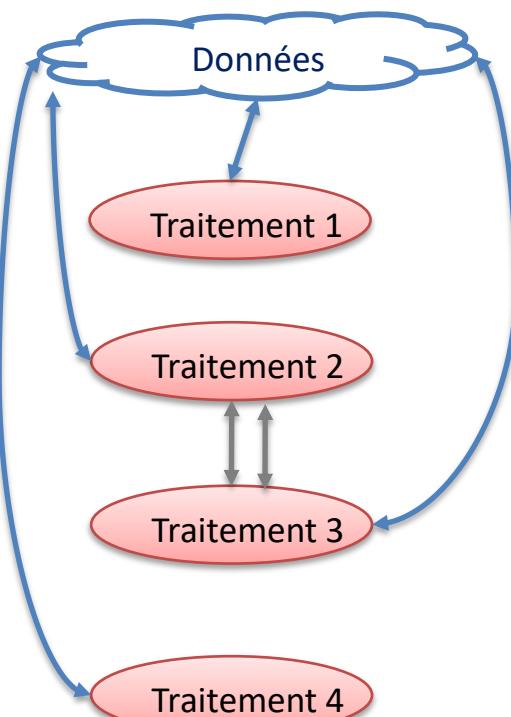
→ Composé de plusieurs objets qui contiennent :

- ▶ des données "internes".
- ▶ des traitements manipulant ces données ou d'autres données.

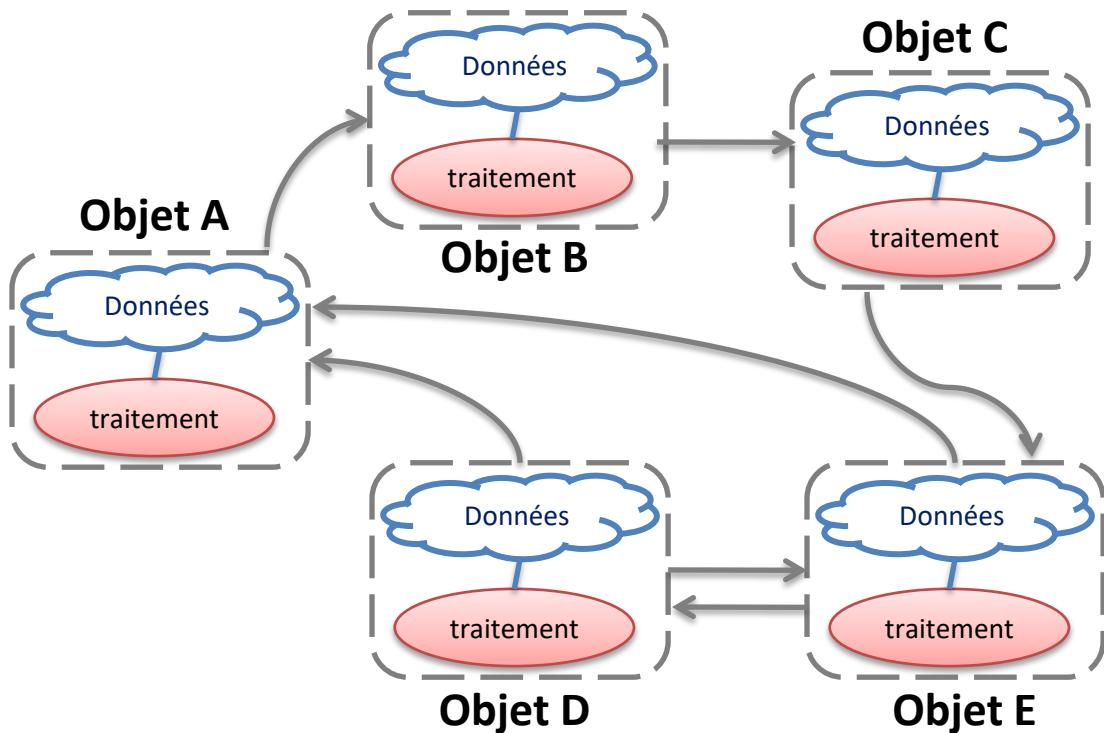


Programmation Orientée Objet vs Programmation Procédurale

Le programme est :
la liste des tâches et des opérations à exécuter



Le programme est
l'ensemble des objets et des interactions entre ces objets



Exemples de langages procéduraux:
Fortran, Lisp, C, Pascal ...

Exemples de langages orientés-objet :
Objective C, SmallTalk, C++, Python, Ruby, Java, ...

Ne commencez pas par vous demander **ce que fait l'application** mais **ce qu'elle manipule**.



Programmation Orientée Objet vs Programmation Procédurale

P.P

- Difficulté de réutilisation du code
- Critères de qualité facilement violés : modularité, lisibilité, ...
- Difficulté de la maintenance de grandes applications
- ...

P.O.O

- Faciliter la réutilisation de code
- Encapsulation et abstraction.
- Facilité de l'évolution du code.
- Améliorer la conception et la maintenance des grands systèmes.
- Programmation par composants

Que doit faire le programme?

De quoi doit être composé le programme?

EXAMPLE: BANQUE

Un main qui doit :

- ▶ Conserver l'état de tous les clients.
- ▶ Possède les méthodes (dépôt, retrait et transfert).

▶ Une banque gère les comptes de ses clients.

▶ Un client peut posséder plusieurs comptes.

▶ La banque effectue des opérations (dépôt, retrait et transfert) sur les comptes.

▶ Chaque compte est responsable de stocker son solde.

Pourquoi utiliser l'orienté objet?

- ▶ Améliore :
 - ▶ Compréhension du logiciel
 - ▶ Séparation des responsabilités
 - ▶ Modularité
 - ▶ Encapsulation
 - ▶ Représentation des échanges entre les composants du programme
 - ▶ Qualité du code
 - ▶ Maintenance
 - ▶ Réutilisation
 - ▶ Partage de propriétés(comportement ou données)

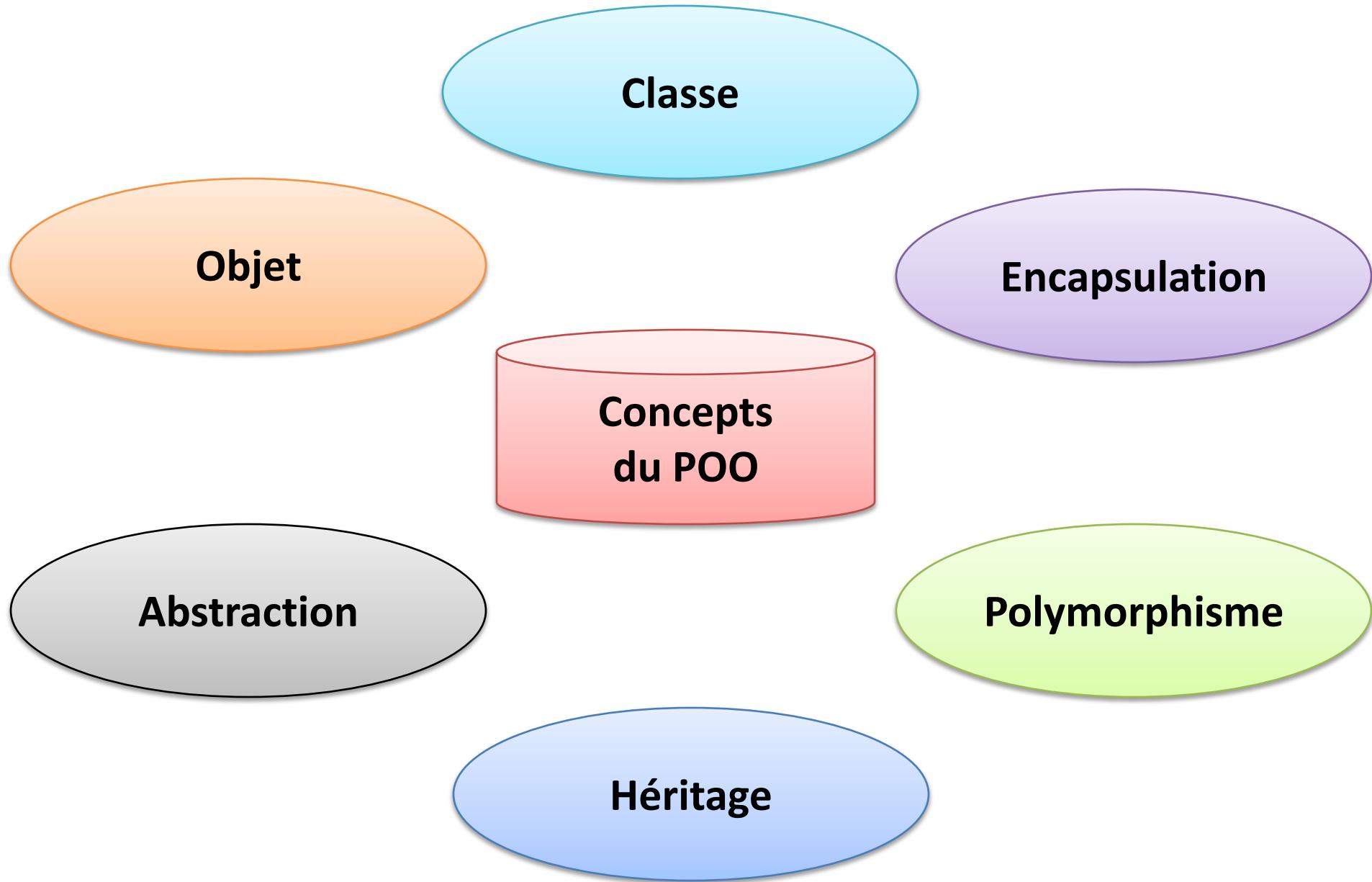
chapitre 1

 Introduction à la POO

 Principes de la POO

 Présentation du langage Java

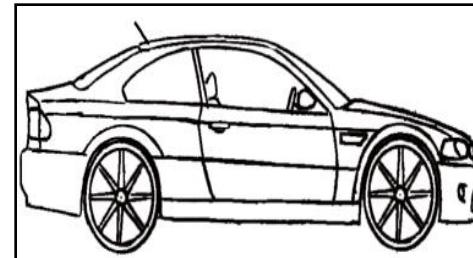
 Syntaxe et bases du langage Java



Classe

Classe

- ▶ Un modèle que vous pouvez utiliser pour créer des objets ayant des propriétés communes.
- ▶ Définit les caractéristiques d'un objet , telles que les données qu'il peut contenir et les opérations qu'il peut exécuter.
- ▶ Généralisation de la notion de type des programmes structurés.
- ▶ Un moule pour la création des instances d'objet.



Objet

Objet

- ▶ Un objet est une instance d'une classe.
- ▶ Si une classe peut s'apparenter à un modèle, un objet représente ce qui est créé à partir de ce modèle.
- ▶ La classe est la définition d'un élément, l'objet est l'élément lui-même.
- ▶ Le terme instance est souvent utilisé à la place du terme objet.



Objet

Objet

Un objet : est une entité autonome regroupant un état et les fonctions permettant de manipuler cet état.

Attribut : est une des variables de l'état d'un objet

Objet = Etat + Comportement [+ Identité]

1. **Etat** : valeurs des attributs (données) d'un objet
2. **Comportement** : opérations possibles sur un objet déclenchées par des stimulations externes (appels de méthodes ou messages envoyées par d'autres objets)
3. **Identité** : chaque objet à une existence propre (il occupe une place en mémoire qui lui est propre). Les objets sont différenciés par leurs noms.

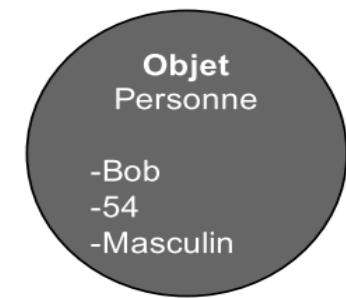
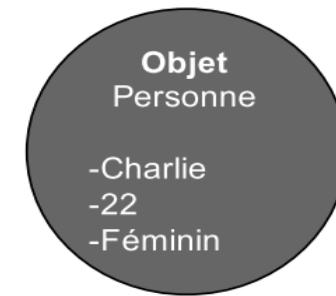
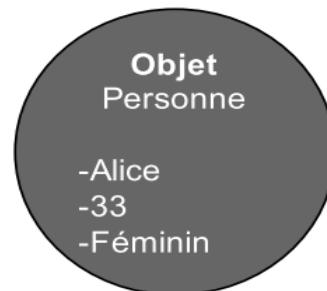
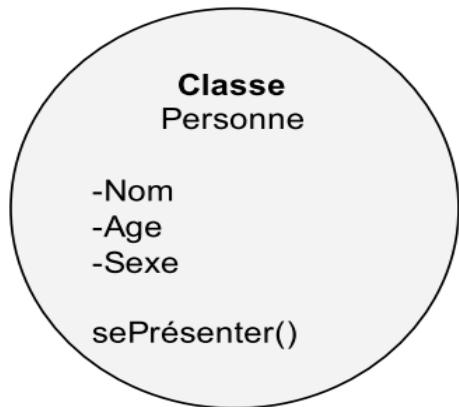
Un objet possède également une durée de vie :

➔ la durée de temps entre sa création et sa destruction.

Classe et Objet

Classe

Une classe est un modèle qui définit les attributs et les opérations d'un objet et qui est créée au moment du design.

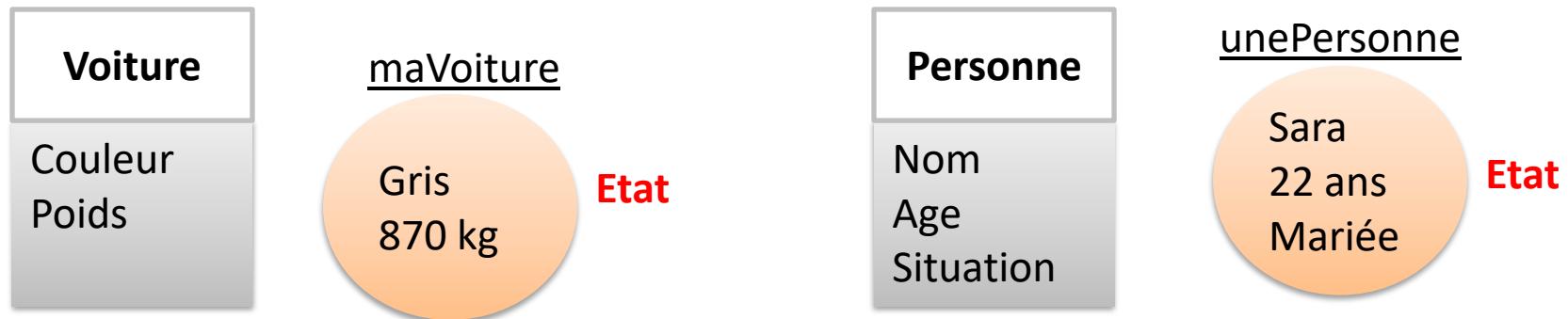


Objet

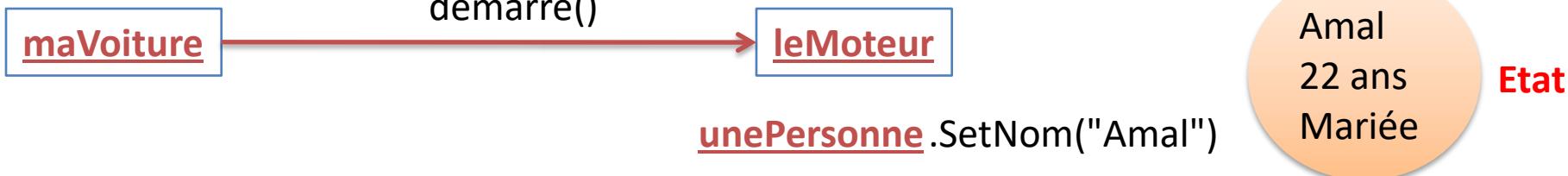
Un objet est une instance en cours d'exécution d'une classe qui utilise la mémoire et possède une durée de vie finie

Classe et Objet

Une classe est un modèle d'objet. A l'inverse, un objet a une réalité matérielle car il possède des champs avec valeurs.



- Un objet peut recevoir un message qui déclenche
 - une méthode qui modifie son état.
 - une méthode qui envoie un message à un autre objet



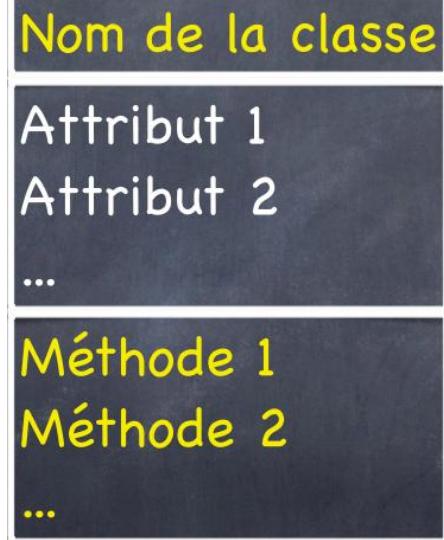
Classe et Objet

Exemples

- ▶ Une **personne** est un **objet** caractérisé par **un nom, un prénom, une date de naissance, une profession ... et pouvant marcher, parler, réfléchir, écrire, ...**
- ▶ Un **compte** en banque est un **objet** caractérisé par **un numéro de compte, un propriétaire, un taux d'intérêt, un solde, un solde minimum, ... et offre des services : consultation du solde, retrait, virement ...**
- ▶ Un **objet graphique** est un **objet** caractérisé par **une taille, par une position, une forme, une couleur ... et peut changer de taille, de forme, de couleur ...**

Classe et Objet

Représentation graphique d'une classe



Exemple

```
class CompteBancaire {  
  
    String proprietaire;  
    double solde;  
  
    double getSolde() {  
        return solde;  
    }  
  
    void credite(double val) {  
        solde = solde + val;  
    }  
}
```

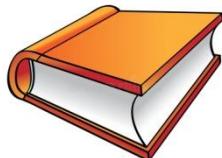
Classe et Objet

Identifier les objets

- A partir d'un cahier des charges
 - 1. liste des mots clefs (tout mot clef peut être une classe)
 - 2. réduire la liste en supprimant :
 - les synonymes
 - les classes non pertinentes
 - 3. Les classes peuvent évoluer : ajout de nouvelles classes lors de l'implémentation pour créer une liste de classes par exemple.
- La difficulté d'identifier les classes :
 - 1. quand l'application contient des objets physique.
 - 2. quand l'application est abstraite.
- Les mots clefs qui n'ont pas été utilisés peuvent être des attributs
 - 1. Parfois difficile de choisir entre classe et attribut
 - Un attribut est caractérisé par sa valeur
 - Une classe est caractérisée par ses attributs et ses méthodes
 - 2. L'attribut d'une classe peut être un objet (agrégation)
- Les services que doivent rendre une classe sont ses opérations

Classe et Objet

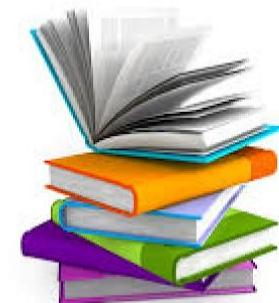
Gestion du bibliothèque



Les misérables
Victor Hugo



Le Monde



Romans policiers
Agatha Christie



Amina Radi
Bibliothécaire



Mohammed Alami
Directeur



Amal Naji
lectrice



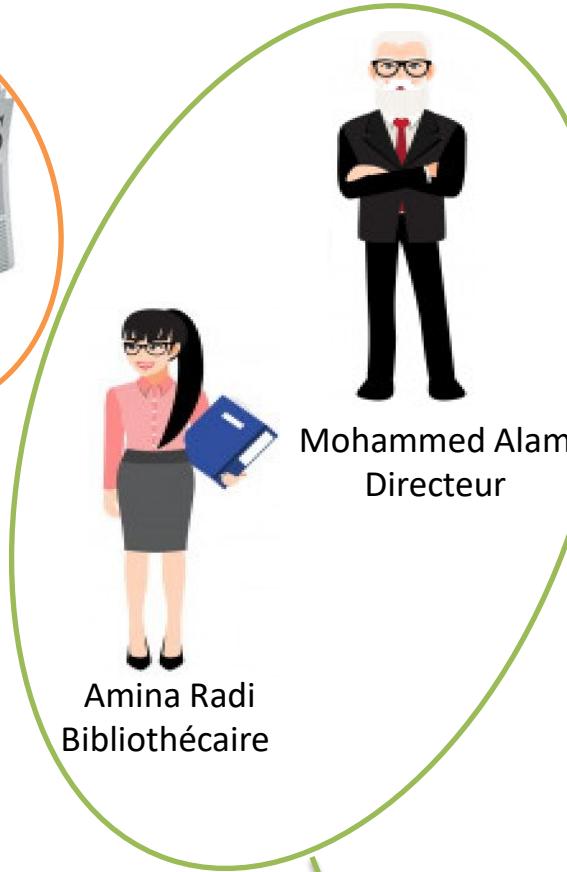
Nabil Salami
Lecteur

Classe et Objet

Gestion du bibliothèque



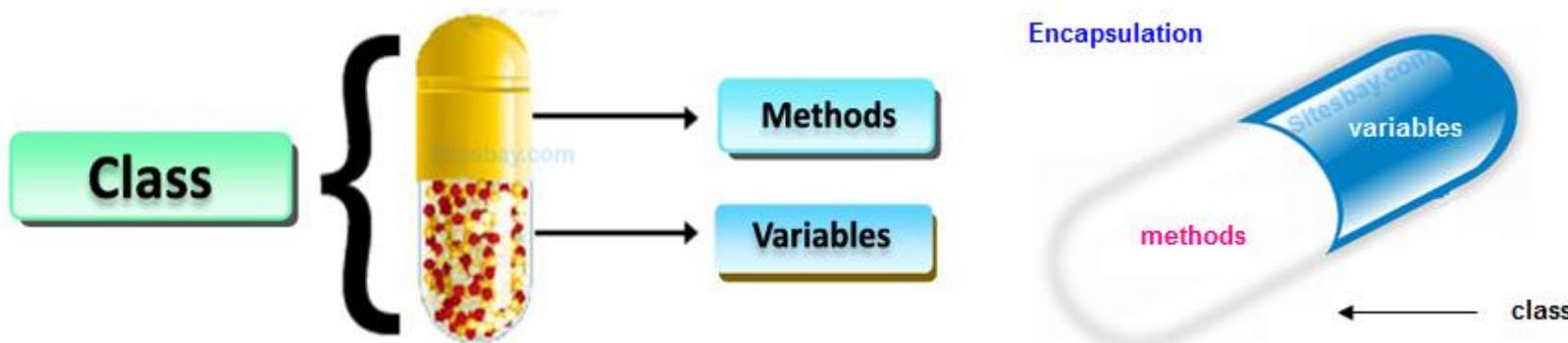
Le Monde



Encapsulation

Encapsulation

Un mécanisme consistant à rassembler les données et les méthodes au sein d'une même classe en cachant l'implémentation de l'objet



Les attributs ne peuvent pas être directement manipulés de **l'extérieur de la classe**, mais seulement indirectement par l'intermédiaire des méthodes.

Avantages:

- ▶ Protéger les données d'un objet.
- ▶ Redéfinir la représentation interne des attributs, sans que cela affecte la manipulation externe d'un objet de cette classe.
- ▶ Facilite la mise à jour des applications.

Encapsulation

Encapsulation

- ▶ On peut en voir l'intérêt dans un projet de développement réalisé par plusieurs développeurs (ce qui est généralement le cas en entreprise).
- ▶ Chaque développeur est responsable d'une partie du projet.
- ▶ Si les classes dont il est le responsable sont proprement encapsulées, il pourra modifier par la suite leurs représentations internes, sans que cela perturbe le travail des autres développeurs du projet susceptibles d'utiliser ces classes.

Example

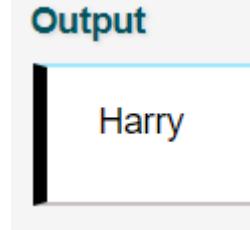
```
class Employee
{
    private String name;

    public String getName()
    {
        return name;
    }

    public void setName(String name){
        this.name=name;
    }
}

class Demo
{
    public static void main(String[] args)
    {
        Employee e=new Employee();
        e.setName("Harry");
        System.out.println(e.getName());
    }
}
```

Output

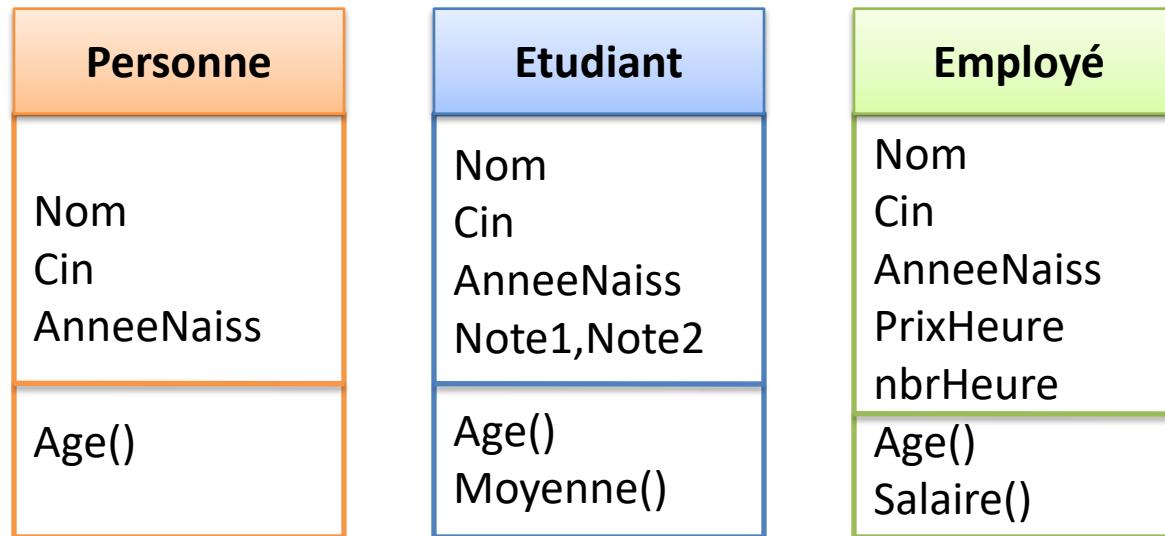


```
Harry
```

Héritage

Héritage

Problème :



Limites:

- ▶ Duplication de code.
- ▶ Une modification faite sur un attribut (ou méthode) commune doit être refaite dans les autres classes.

Héritage

Héritage

Solution:

Super-Classe



Héritent de

Sous-Classes

- ▶ Placer dans la classe mère tous les attributs et toutes les méthodes communs à toutes les classes.
- ▶ Les classes filles ne comportent que les attributs ou méthodes plus spécifiques.
- ▶ Les classes filles héritent automatiquement les attributs et les méthodes qui n'ont pas besoin d'être réécrits.

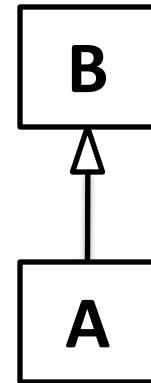
Avantage:

- ▶ Réutilisation du code.
- ▶ une seule modification des attributs (ou méthodes) en commun.

Héritage

Héritage

- ▶ L'héritage est mécanisme permettant le partage et la réutilisation de propriétés entre les objets, (à partir d'une classe existante on ajoute de nouvelles données, de nouvelles méthodes).
- ▶ L'héritage exprime une relation de similarité entre les objets.
- ▶ L'héritage définit La relation (Généralisation/Spécialisation)
 - A < est un type de > B
 - Les objets instance de A sont instance de B.
- ▶ La classe parente est la (**super- classe**, classe- mère).
- ▶ La classe qui hérite est la (**sous- classe**, classe- fille) .



JAVA NE PERMET PAS L' HERITAGE MULTIPLE

(héritage de plusieurs super- classes) mais les méthodes d'interface permettent de traiter de façon simple des situations similaires.



Héritage

Héritage

```
class Personne {  
  
    String nom;  
    long cin;  
  
    public void setNom(String nom){  
        this.nom=nom;}  
  
    public String getNom(){  
        return nom;}  
  
    public void set Cin(long cin){  
        this.cin=cin;}  
  
    public long get Cin(){  
        return cin;}}
```

```
class Etudiant extends Personne {  
  
    float moyenne;  
  
    public void setMoyenne(float moyenne){  
        this.moyenne=moyenne;}  
  
    public float getMoyenne (){  
        return moyenne;}  
}
```

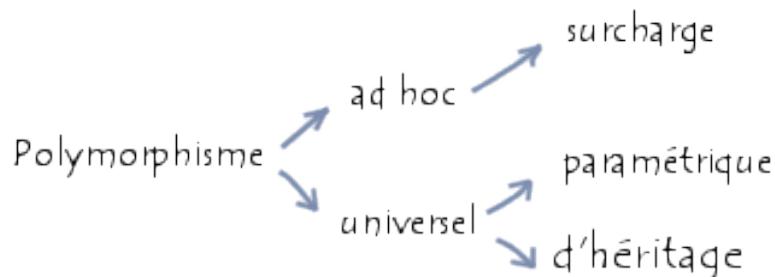
Polymorphisme

Polymorphisme

- ▶ Le polymorphisme est un concept fondamental de la programmation orientée objet.
- ▶ Dans la langue grec, il signifie « **prendre plusieurs formes** ».
- ▶ L'héritage concerne les classes (et leur hiérarchie), le polymorphisme est relatif aux méthodes des objets.
- ▶ Possibilité de définir plusieurs méthodes avec le même nom.

On distingue généralement trois types de polymorphisme :

- ❑ Le polymorphisme ad hoc (**surcharge** , **overloading**)
- ❑ Le polymorphisme paramétrique (**généricité** , **template**)
- ❑ Le polymorphisme d'héritage (**redéfinition** , spécialisation, **overriding**)



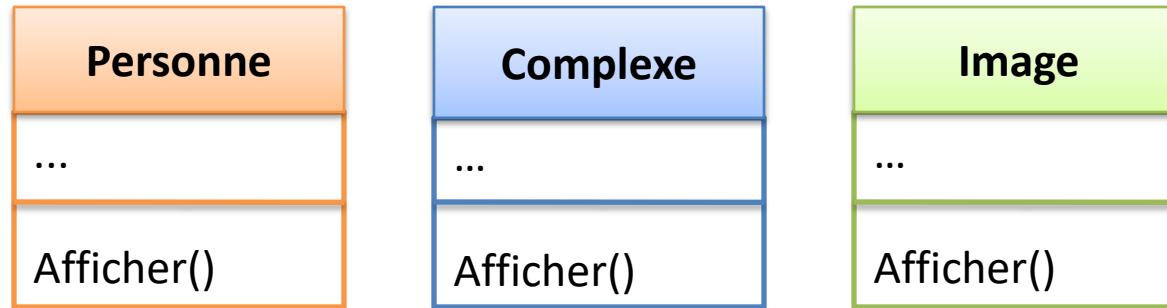
Polymorphisme

Polymorphisme

Le polymorphisme ad hoc (surcharge , overloading)

- ▶ Permet d'avoir des fonction de même nom, de fonctionnalités similaires mais dans **des classes** radicalement **différentes**.

Exemple:



- ▶ Les trois classes peuvent avoir chacune une méthode Afficher().
- ▶ Cela permettra de ne pas avoir à se soucier du type de l'objet que l'on a si on souhaite l'afficher à l'écran.

Polymorphisme

Polymorphisme

Le polymorphisme paramétrique (généricité , template)

- ▶ Appelé **généricité**, représente la possibilité de définir **plusieurs fonctions de même nom mais possédant des paramètres différents** (en nombre et/ou en type).
- ▶ Rend ainsi possible le choix automatique de la bonne méthode à adopter en fonction du type de donnée passée en paramètre.

Exemple:

On peut définir plusieurs méthodes homonymes *addition()* effectuant une somme de valeurs.

- ***int addition(int, int)*** → la somme de deux entiers
- ***float addition(float, float)*** → la somme de deux flottants
- ***String addition(char, char)*** → la concaténation de deux caractères

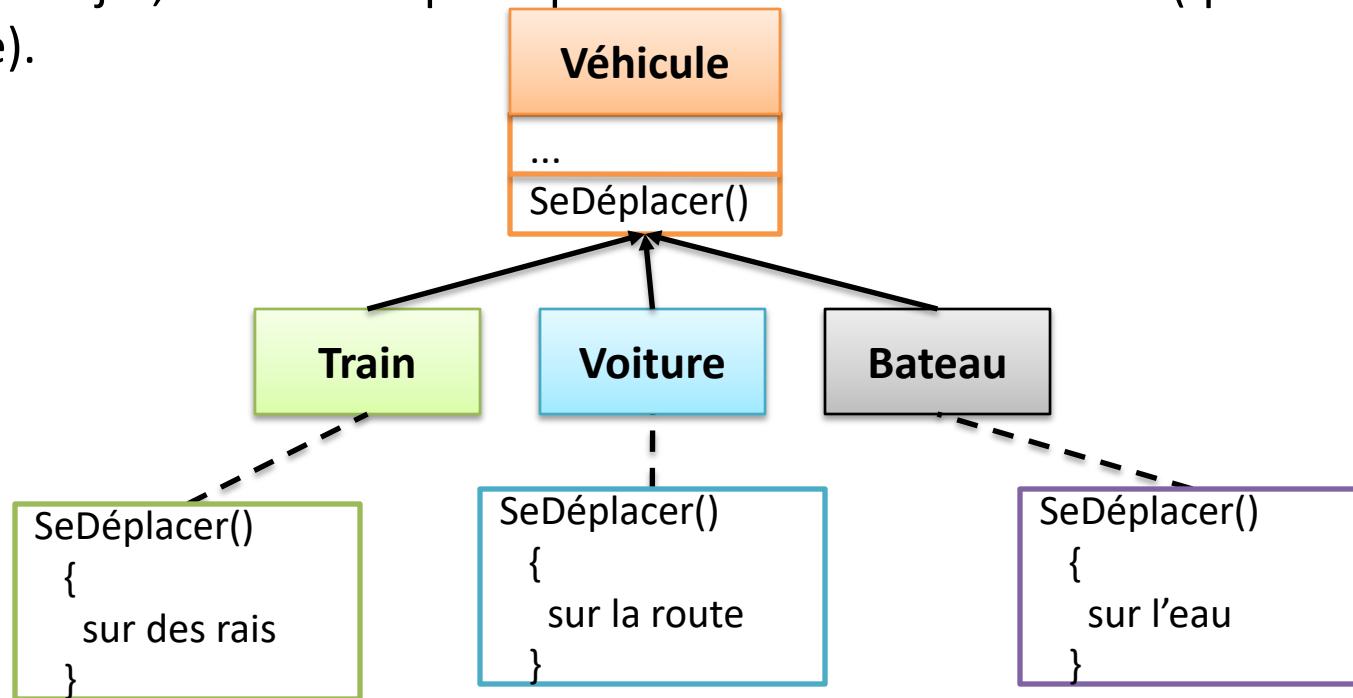
Alors, c'est la ***signature*** d'une méthode qui détermine laquelle sera appelée.

Polymorphisme

Polymorphisme

Le polymorphisme d'héritage (redéfinition, spécialisation, overriding)

- ▶ La possibilité de **redéfinir une méthode dans des classes héritant** d'une classe de base s'appelle la **spécialisation**.
- ▶ Il est alors possible d'appeler la méthode d'un objet sans se soucier de son type intrinsèque : il s'agit du **polymorphisme d'héritage**.
- ▶ Ceci permet de faire **abstraction** des détails des classes spécialisées d'une famille d'objet, en les masquant par une **interface** commune (qui est la classe de base).



Abstraction

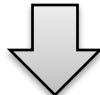
Abstraction

L'**abstraction** est un concept général que vous pouvez trouver dans le monde réel ainsi que dans les langages POO. (Tous les objets du monde réel, comme la voiture, ou la maison, masquant les détails internes fournissent une abstraction).

- ▶ Son objectif principal est de gérer la **complexité** en masquant les détails inutiles à l'utilisateur.
- ▶ Cela permet à l'utilisateur d'implémenter une logique plus complexe sans comprendre ni même penser à toute la complexité cachée.
- ▶ Cela aide à diviser la complexité du logiciel en parties contrôlable.

Les avantages de l'abstraction:

- ▶ Réduire la complexité.
- ▶ Évite la duplication de code et augmente la possibilité de réutilisation.
- ▶ Aide à renforcer la sécurité d'une application ou d'un programme car seuls les détails importants sont fournis à l'utilisateur.



Classe Abstraite & Interface

Abstraction

Classe abstraite

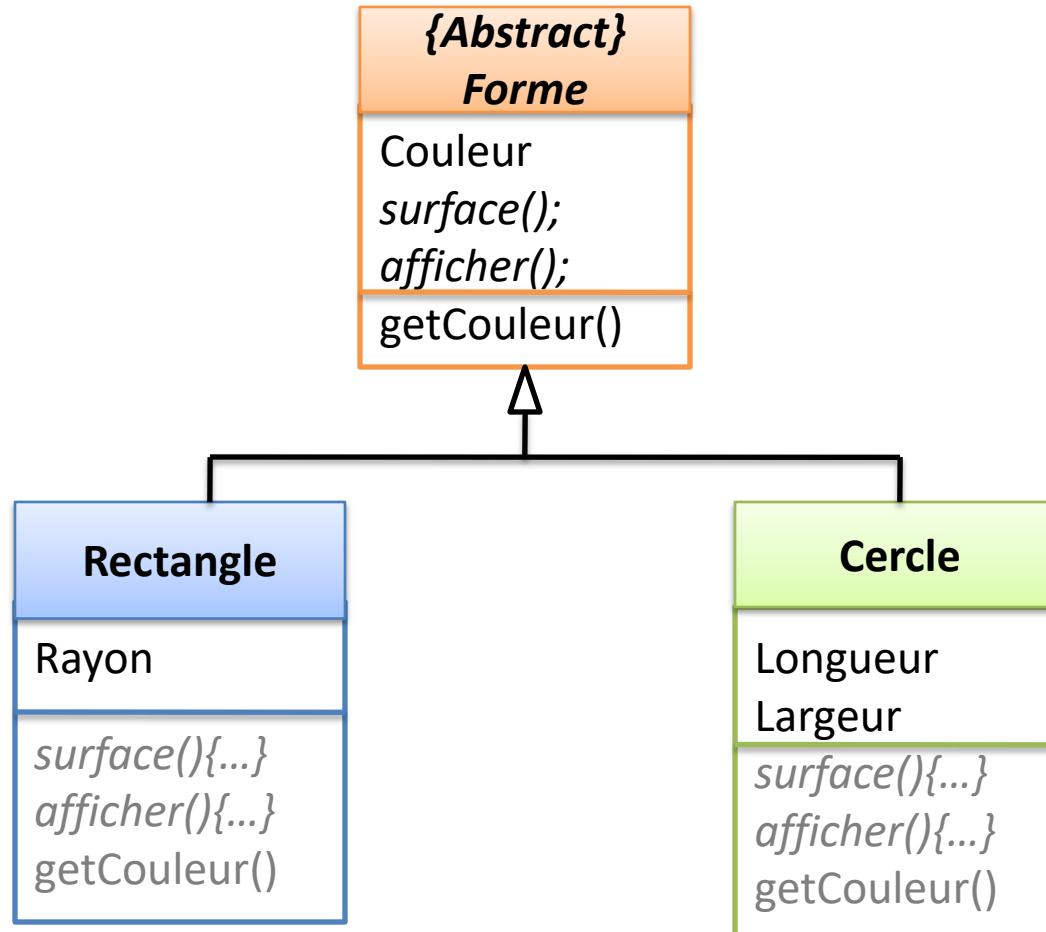
Abstraction



Abstraction

Classe abstraite

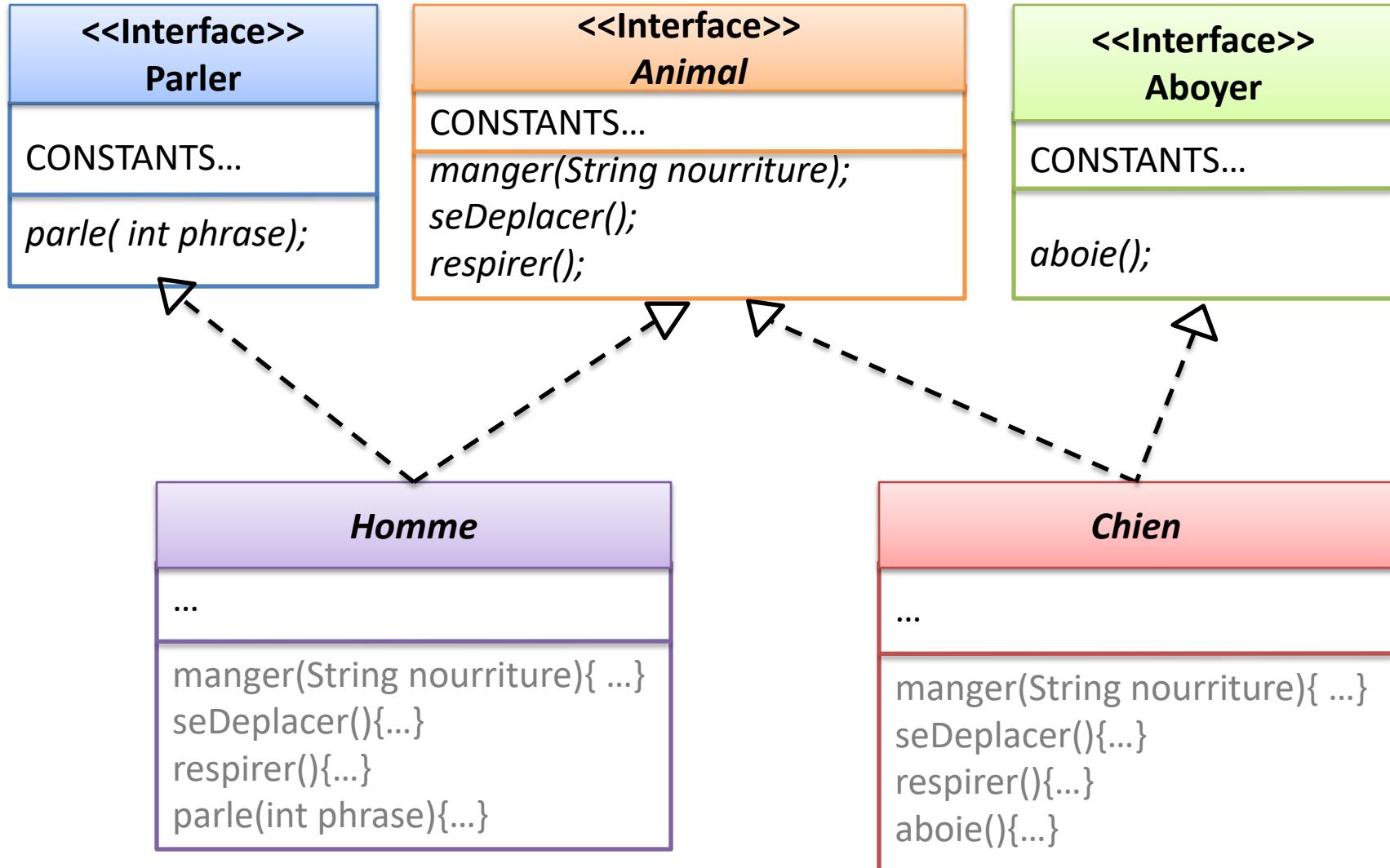
Abstraction



les classes abstraites servent à factoriser du code.

Abstraction Interface

Abstraction



les interfaces servent à définir des contrats de service.

la programmation orientée objet a de nombreux avantages qui expliquent que cette méthode est une des plus utilisées de nos jours :

- ▶ Facilite d'organisation grâce aux objets.
- ▶ Méthode plus intuitive car plus proche de la réalité, principe d'héritage.
- ▶ Gestion de projet plus efficace : la sécurisation via encapsulation permet d'avoir plusieurs développeurs travaillant sur un même projet, chacun ne travaille que sur les implémentations le concernant.
- ▶ Le code est "factorisé" : il est plus facilement lisible et donc corrigible, et est moins lourd.

chapitre 1

 Introduction à la POO

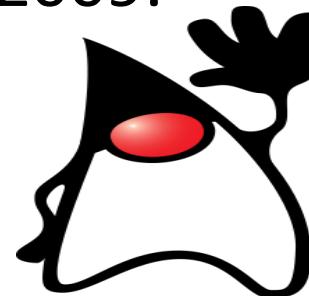
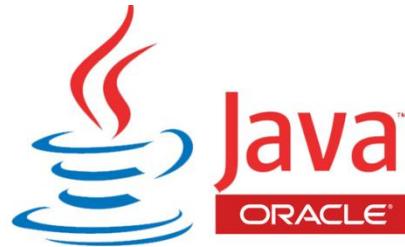
 Principes de la POO

 Présentation du langage Java

 Syntaxe et bases du langage Java

Java c'est quoi?

- La plate-forme et le langage Java sont issus d'un projet de Sun Microsystems datant de 1990.
- Généralement, on attribut sa paternité à trois de ses ingénieurs :
 - James Gosling
 - Patrick Naughton
 - Mike Sheridan
- Une technologie développée par SUN Microsystems™ lancée en 1995 -rachetée par Oracle en 2009.



Java c'est quoi?

- Un langage :
 - Orienté objet
 - fortement typé avec classes
- Syntaxe très proche du C et C++
- Permettant de développer
 - Applications Consoles
 - Applications du bureau
 - Applications web
 - Applications mobiles
 - Web services
 - Jeux...

Attention

Java ≠ JavaScript

Java c'est quoi?

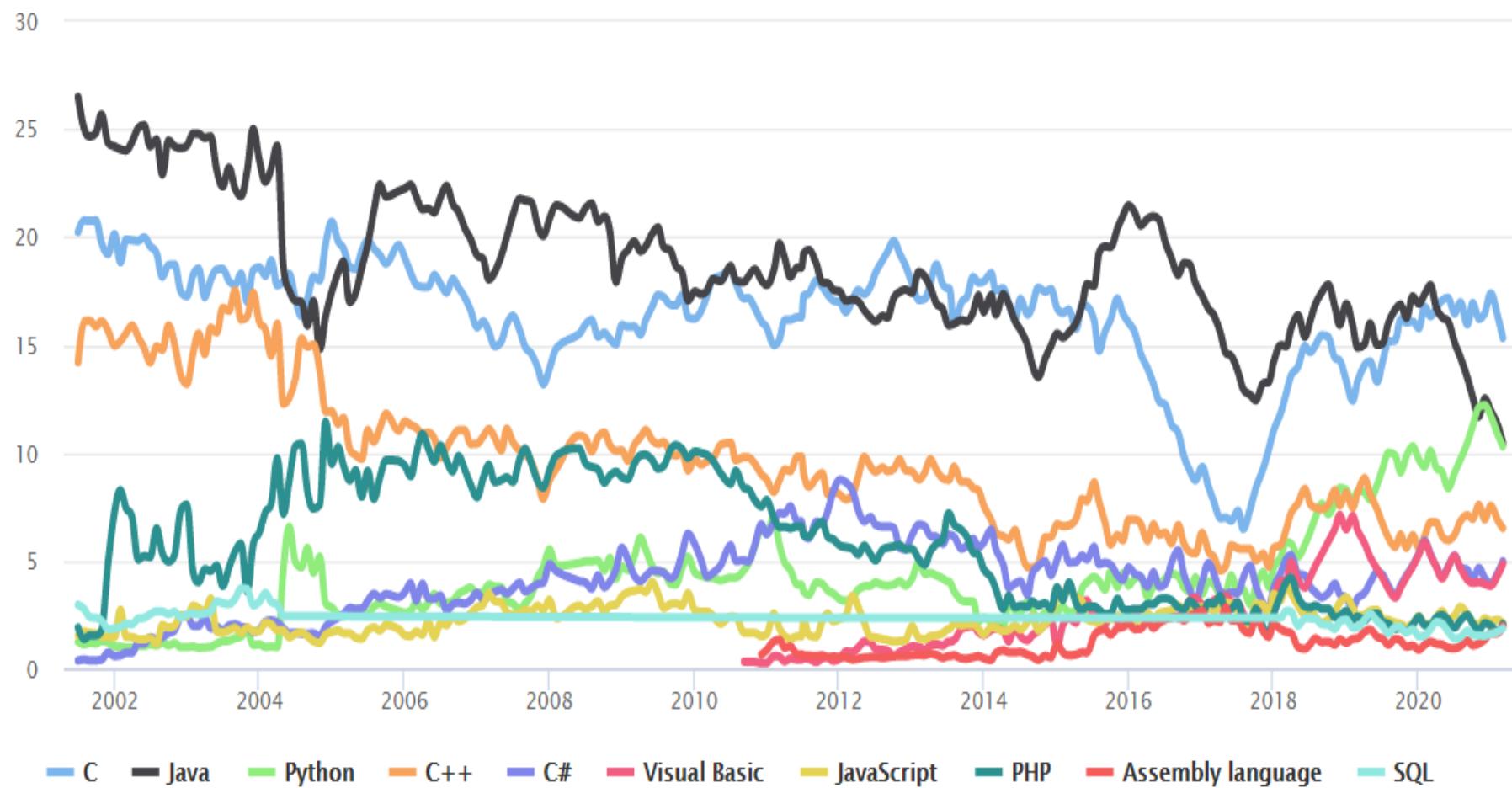
- Langage de haut niveau (pas de gestion de mémoire, pas d'allocation dynamique, pas de pointeur... comme en C et C++)
- Disposant d'une bonne documentation, des supports vidéos, plusieurs exemples sur internet
- Permettant de développer des programmes :
 - robustes
 - sécurisés et fiables
 - bien structurés et maintenables
 - portables: Windows, Mac OS, Linux (Write once, run everywhere)
- Un des langages les plus utilisés dans le monde

Popularité des langages de programmation

Java est devenu aujourd’hui l’un des langages de programmation les plus utilisés.

TIOBE Programming Community Index

Source: www.tiobe.com



Popularité des langages de programmation

Mar 2021	Mar 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	15.33%	-1.00%
2	1	▼	Java	10.45%	-7.33%
3	3		Python	10.31%	+0.20%
4	4		C++	6.52%	-0.27%
5	5		C#	4.97%	-0.35%
6	6		Visual Basic	4.85%	-0.40%
7	7		JavaScript	2.11%	+0.06%
8	8		PHP	2.07%	+0.05%
9	12	▲	Assembly language	1.97%	+0.72%
10	9	▼	SQL	1.87%	+0.03%
11	10	▼	Go	1.31%	+0.03%
12	18	▲	Classic Visual Basic	1.26%	+0.49%
13	11	▼	R	1.25%	-0.01%
14	20	▲	Delphi/Object Pascal	1.20%	+0.48%
15	36	▲	Groovy	1.19%	+0.94%
16	14	▼	Ruby	1.18%	+0.13%
17	17		Perl	1.15%	+0.24%
18	15	▼	MATLAB	1.04%	+0.05%
19	13	▼	Swift	0.95%	-0.28%
20	19	▼	Objective-C	0.91%	+0.17%

Java en quelques mots:

- Dans un des premiers papiers* sur le langage JAVA, SUN le décrit comme suit :

«Java : a simple, object-oriented, distributed, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language»

- *White Paper :The Java Language Environment-James Gosling, Henry McGilton -May 1996
- <http://java.sun.com/docs/white/langenv>

Java en quelques mots:

- **Simple :**
 - Il est toutefois allégé de toutes les sources d'erreurs du C++ (pointeurs, gestion mémoire, etc.) et des fonctionnalités "tordues" et peu utilisées.
- **Objet :**
 - Inspiré du C++, Objective C et Smalltalk , Java est un langage fortement objet.
- **Distribué :**
 - Java contient un ensemble de classes permettant d'utiliser des protocoles TCP/IP tels que HTTP et FTP. L'utilisation des URL se fait aussi "simplement" que l'ouverture de fichiers locaux.
- **Robuste :**
 - Le maximum de vérifications sont faites à la compilation (protection des classes,...).
 - Pas de gestion explicite de la mémoire (GC en thread), pas de pointeur, ..
- **Sécurisé :**
 - Java est très sécurisé (SecurityManager).

Java en quelques mots:

- **Interprété :**
 - Avantage au développement.
 - Mais est pour l'instant un inconvénient à l'exécution (performances).
- **Indépendant de l'architecture :**
 - C'est la machine virtuelle qui assure cette indépendance.
- **Portable :**
 - Contrairement au C/C++ Java n'est pas dépendant de l'implémentation, les types de base ont des tailles fixes.
- **Multithread :**
 - C'est la possibilité d'exécuter plusieurs tâches simultanément.
- **Dynamique :**
 - Java a été conçu pour fonctionner dans un environnement évolutif.
- **Compact :**
 - L'interpréteur Java n'a besoin que de 215Ko.
 - Chargement dynamique des classes.

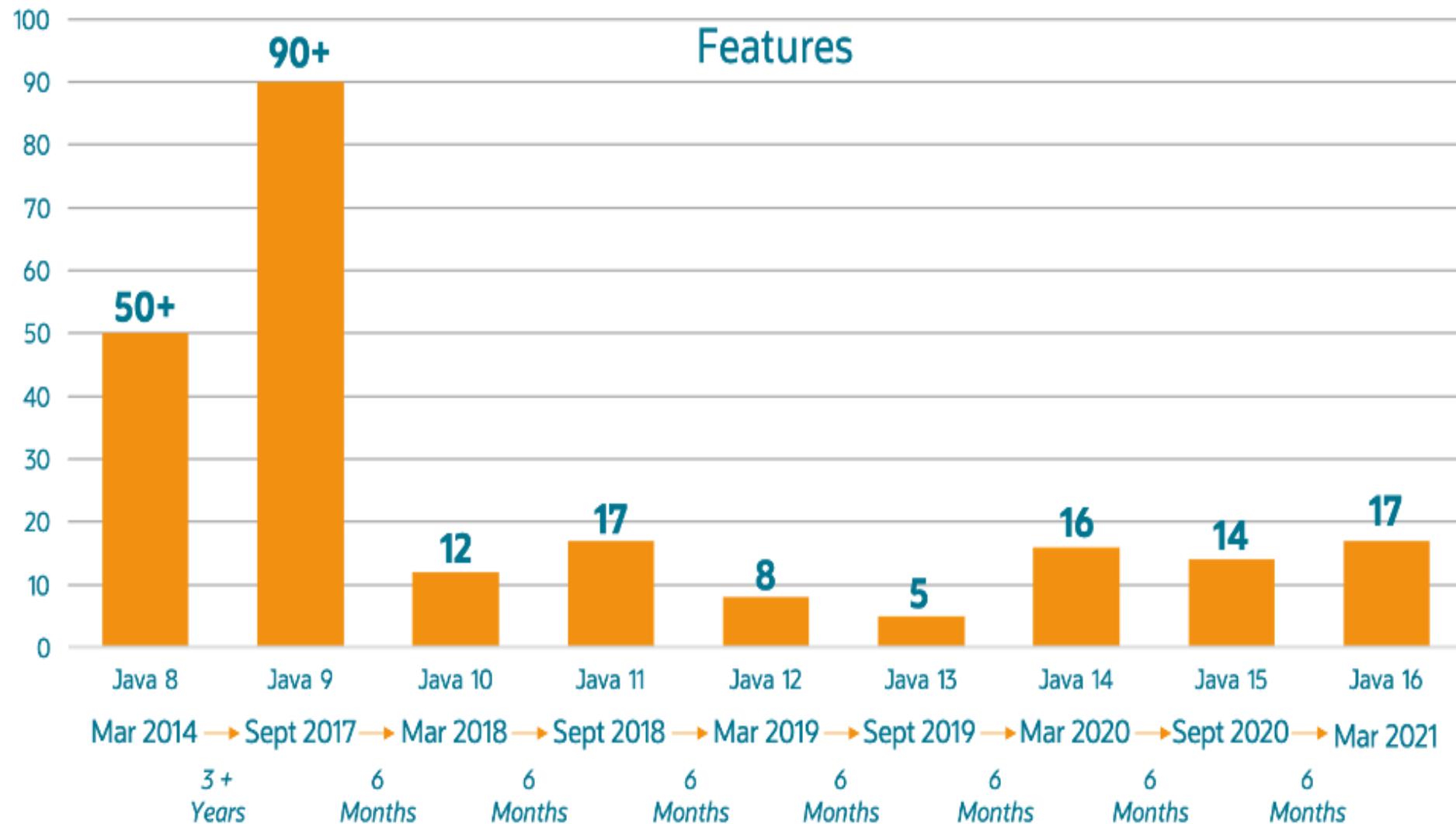
Versions de Java

- Java n'est pas un langage normalisé et il continue d'évoluer. Cette évolution se fait en ajoutant **de nouvelle API**, mais aussi en modifiant **la machine virtuelle**.
- L'ensemble de ces modifications est géré par le **JCP** (Java Community Process ; <http://www.jcp.org/>) dans lequel Sun continue de tenir une place prépondérante.
- Il peut alors être nécessaire d'identifier une version précise du compilateur et/ou de la machine virtuelle.

Versions de Java

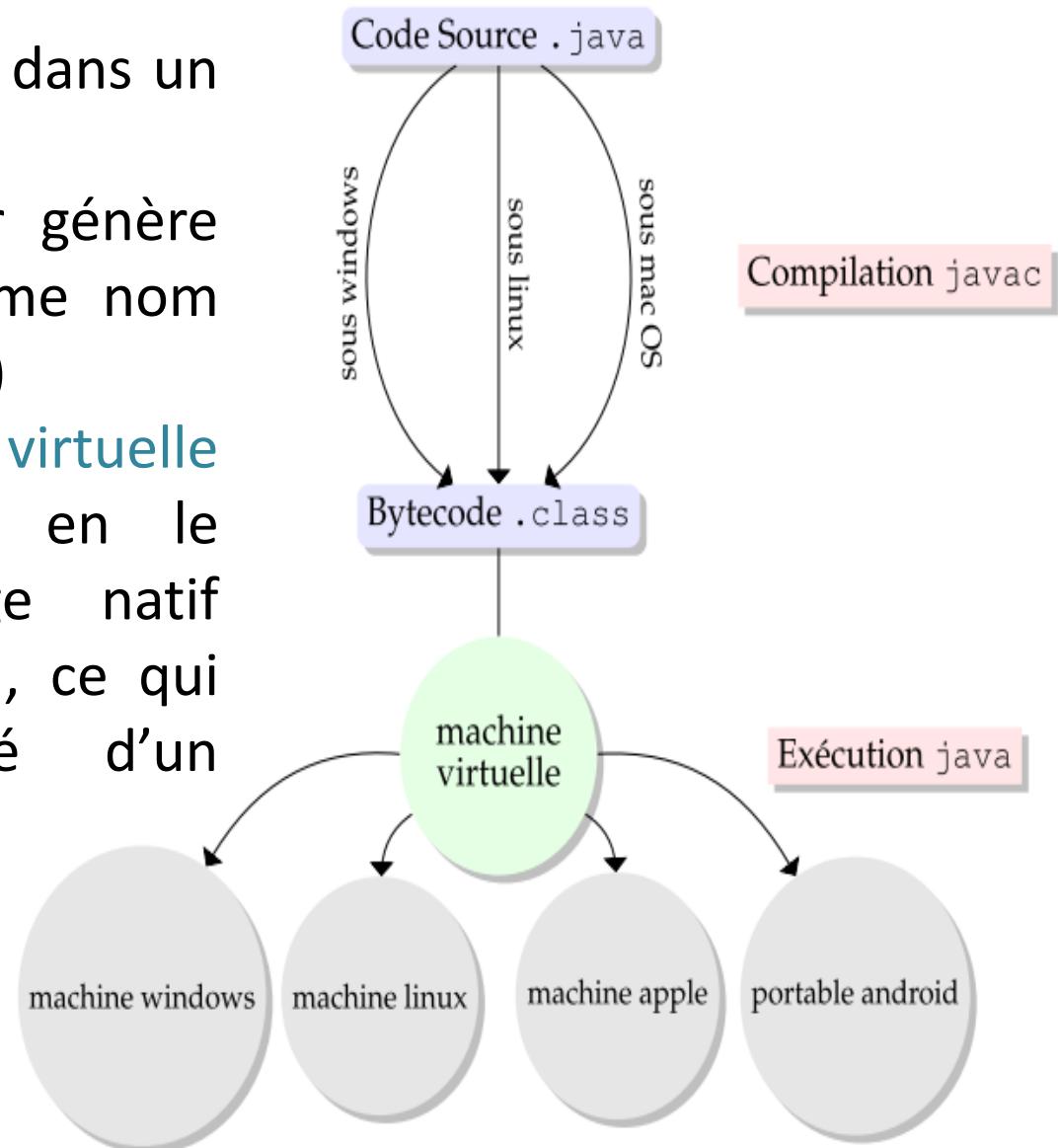
- ◆ **Java 1** (sortie en 1996)
- ◆ **Java 2** (sortie en 1998, nommée Playground) : Swing + collection
- ◆ **Java 4** (sortie en 2002, nommée Merlin) : expressions régulières+parser XML (JAXP)
- ◆ **Java 5** (sortie en 2004, nommée Tiger) : généricité, annotation + énumérations + plus besoin de convertir les types wrappers en primitifs (et inversement)
- ◆ **Java 6** (sortie en 2006, nommée Mustang) : JAX-WS (Web services REST)
- ◆ **Java 7** (sortie en 2011, nommée Dolphin) : String dans switch
- ◆ **Java 8** (sortie en Mars 2014, nommée Spider) : interface fonctionnelle, méthode par défaut, expression Lambda, MapReduce pour les collections
- ◆ **Java 10** (sortie en Mars 2018) : mot-clé var
- ◆ **Java 11** (sortie en Septembre 2018) : simplifier l'exécution d'un programme en ligne de commande
- ◆ **Java 12** (sortie en Mars 2019) : simplification de switch et String multi-lignes
- ◆ **Java 13** (sortie en Septembre 2019) : les blocs de texte, mot clé « **yield** » sur les expressions switch
- ◆ **Java 14** (sortie en Mars 2020) : Switch expressions, Pattern matching, text blocks, records, NPEs plus claires, l'outil de packaging multiplateforme...
- ◆ **Java 15** (sortie en Septembre 2020) : hidden classes, ZGC, Pattern Matching for instanceof, record, Sealed Classes...
- ◆ **Java 16** (sortie en Mars 2021) : Une API vectorielle améliorée, Méta-espace élastique ...

Versions de Java



Comment ça fonctionne?

1. On écrit un programme dans un **fichier .java**
2. Ensuite, le compilateur génère un **fichier.class** du même nom (contenant du **bytecode**)
3. Puis, la machine virtuelle exécute le bytecode en le traduisant en langage natif (langage de bas niveau, ce qui assure la portabilité d'un programme java)



Comment ça fonctionne? Bytecode

```
...
0      new #46 <Class javax/realtime/PriorityParameters>
3      dup
4      bipush   8
6      invokespecial #49
          <Method javax/realtime/PriorityParameters.<init> (I)V>
9      astore_1
...
118     new #68 <Class javax/realtime/PeriodicParameters>
121     dup
122     aload    4
124     aload    5
126     aload    8
128     aload    5
130     aconst_null
131     aconst_null
132     invokespecial #71
          <Method javax/realtime/PeriodicParameters.<init>
          (Ljavax/realtime/HighResolutionTime;
           Ljavax/realtime/RelativeTime;
           Ljavax/realtime/RelativeTime;
           Ljavax/realtime/RelativeTime;
           Ljavax/realtime/AsyncEventHandler;
           Ljavax/realtime/AsyncEventHandler;)V>
135     astore    11
...
175     new #73 <Class SimpleIOSControllerThread>
178     dup
179     aload_1
180     aload    11
182     invokespecial #76 <Method SimpleIOSControllerThread.<init>
          (Ljavax/realtime/SchedulingParameters;
           Ljavax/realtime/ReleaseParameters;)V>
185     astore    14
...
219     aload    14
221     invokevirtual #84 <Method java/lang/Thread.start ()V>
...
...
```

byte-code assure la portabilité des programmes Java

- langage d'une Machine Virtuelle
- à l'exécution un interpréteur simule cette machine virtuelle

Comment ça fonctionne?

Machine Virtuelle (JVM)

- Un ordinateur fictif s'exécutant sur un ordinateur réel
- Environnement d'exécution pour applications Java
- Un des éléments les plus importants de la plate-forme Java.
- Assure l'indépendance du matériel et du système d'exploitation lors de l'exécution des applications Java.

Avantages :

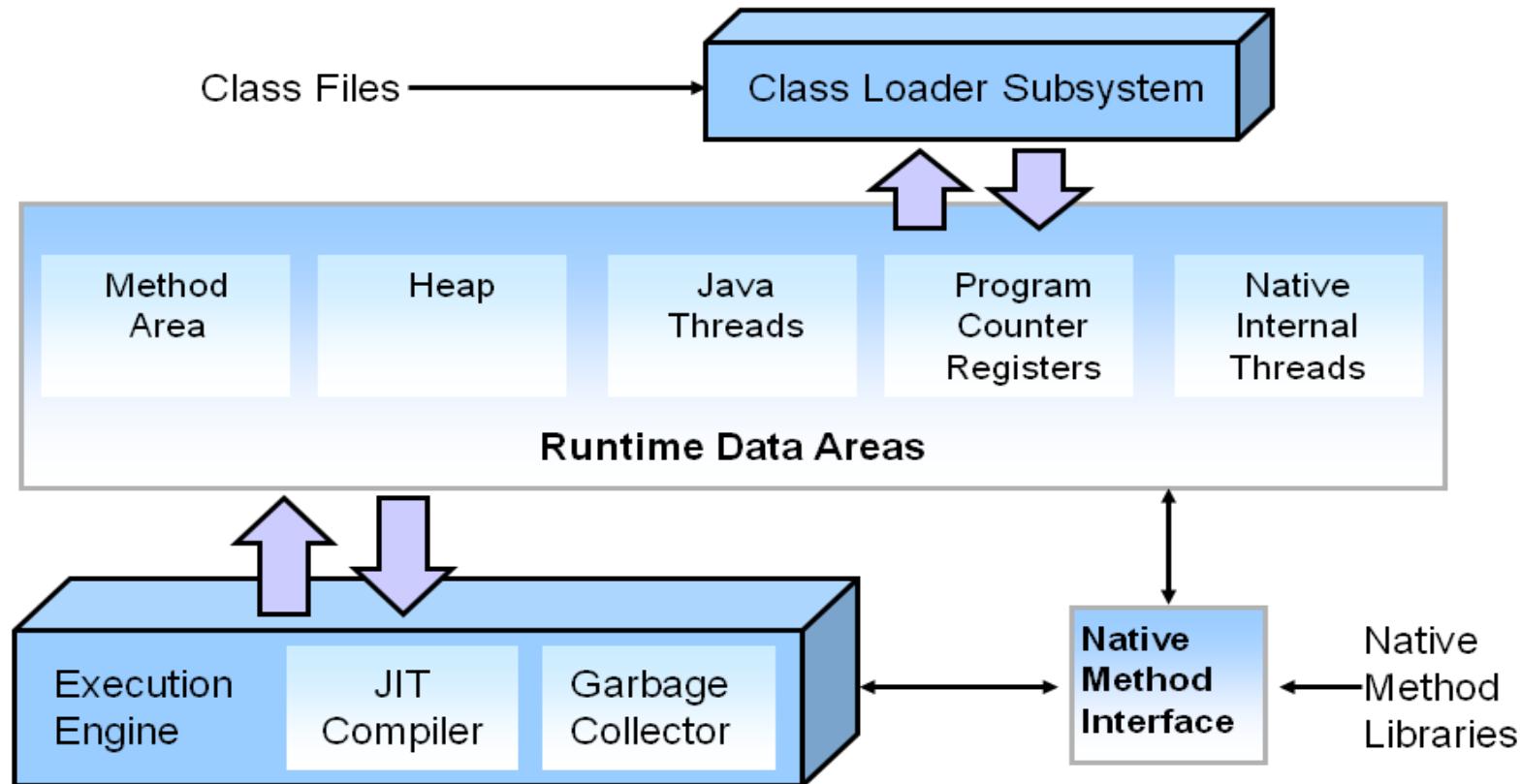
- Interprétation du bytecode
- Interaction avec le système d'exploitation
- Gestion de sa mémoire grâce au ramasse-miettes

Inconvénients :

- coût en ressources de la machine virtuelle.

Comment ça fonctionne? Machine Virtuelle (JVM)

HotSpot JVM: Architecture

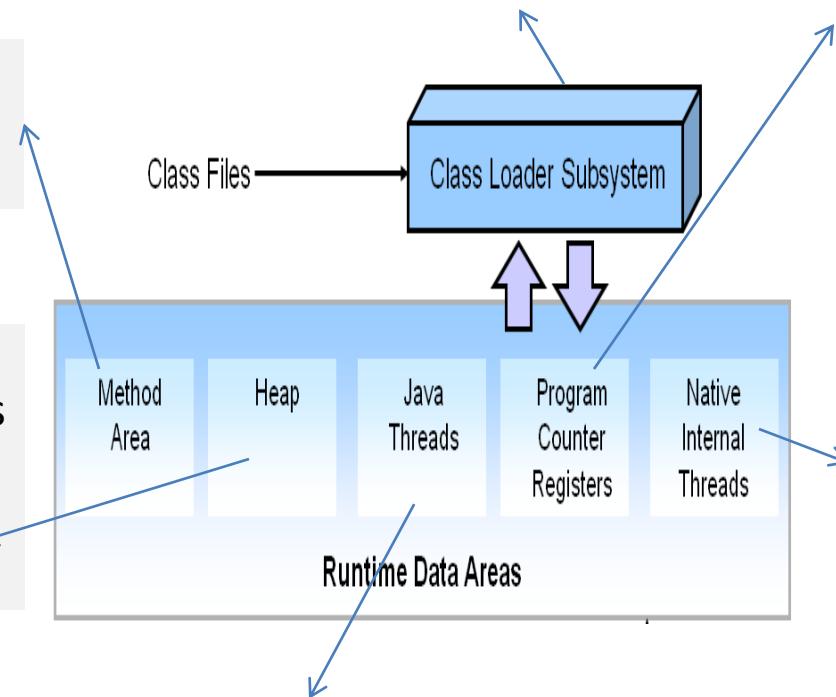


Comment ça fonctionne? Machine Virtuelle (JVM)

stocke les données pour chaque classe (champs, constantes ,données ...)

Le **tas** est l'endroit où tous les objets sont stockés dans JVM. Le tas contient même des tableaux car les tableaux sont des objets.

charger les fichiers de classe.
vérifie les fichiers de classe à l'aide d'un vérificateur de bytecode

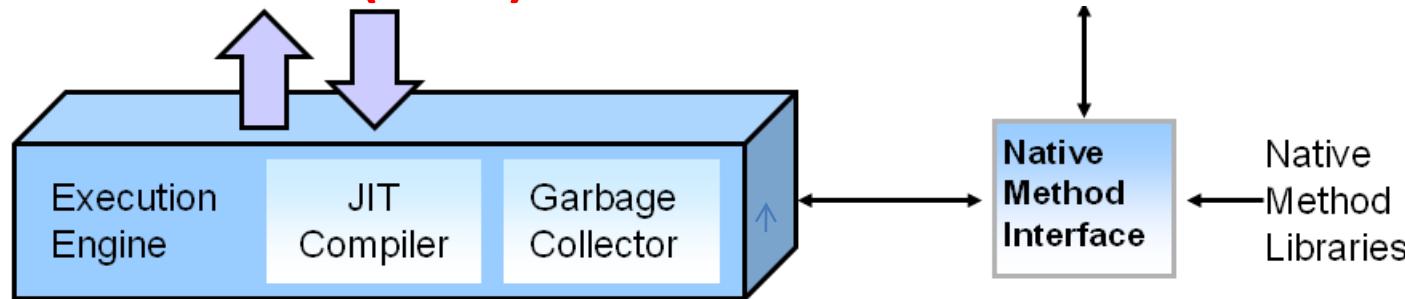


Piles de threads :chaque thread a sa propre pile.(variables locales, paramètres, adresses de retour.) .Elles stockent des références d'objet.

Les registres de compteur de programme (l'adresse des instructions en cours d'exécution et l'adresse de l'instruction suivante).

Piles de méthodes natives: pour chaque thread, une pile native distincte est créée. Il stocke les informations de méthode native.

Comment ça fonctionne? Machine Virtuelle (JVM)



Le moteur d'exécution exécute le «.class» (bytecode). Il lit l'octet-code ligne par ligne, utilise les données et informations présentes dans diverses zones de mémoire et exécute des instructions.

Il peut être classé en trois parties:

Le compilateur JIT (Just In Time) : compile le bytecode en code machine au moment de l'exécution, Il est utilisé pour augmenter l'efficacité d'un interpréteur.

Le nettoyage de la mémoire : le processus par lequel la machine virtuelle Java supprime les objets (objets inutilisés) du tas pour récupérer de l'espace sur le tas.

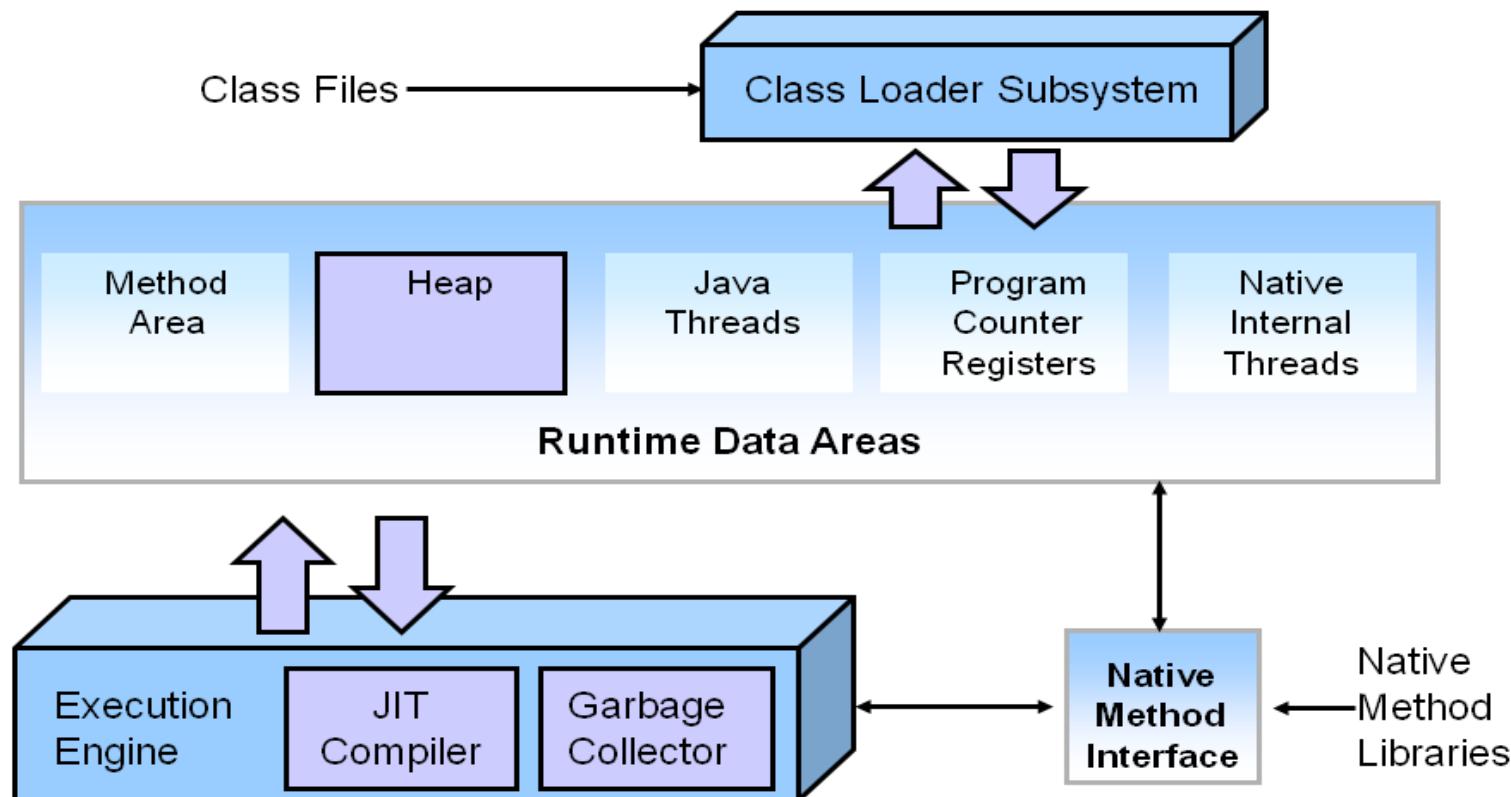
Interpréteur: responsable de la lecture du bytecode, puis de l'exécution des instructions.

L'interface de méthode native est une interface qui connecte la JVM aux bibliothèques de méthodes natives pour exécuter des méthodes natives.

Si nous exécutons la JVM sous Windows, alors l'interface de méthode native connectera la JVM avec les bibliothèques de méthodes Windows pour exécuter les méthodes Windows

Comment ça fonctionne? Machine Virtuelle (JVM)

Key HotSpot JVM Components

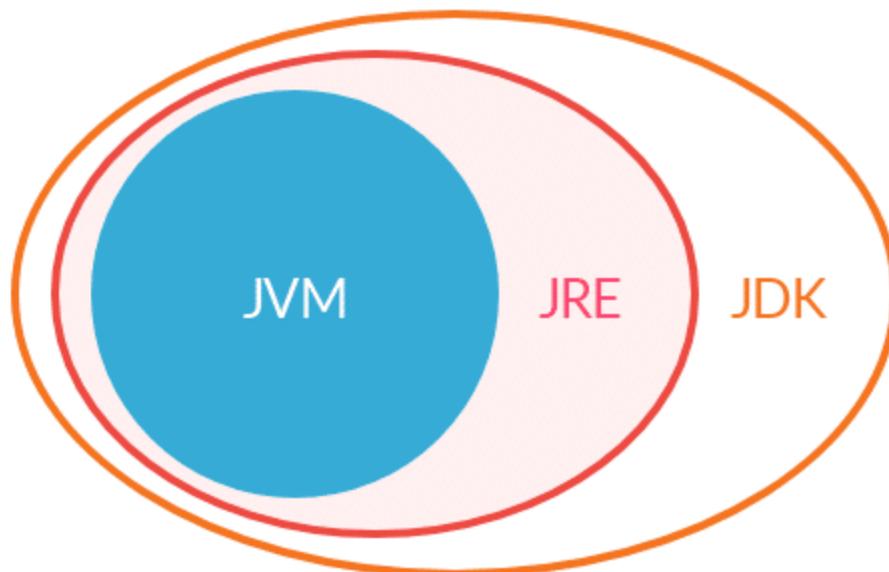


Comment ça fonctionne? Machine Virtuelle (JVM)

La JVM est généralement installée sur un ordinateur au travers du **Java Runtime Environment** (JRE), qui comporte la JVM et des librairies permettant d'exécuter n'importe quel programme Java.

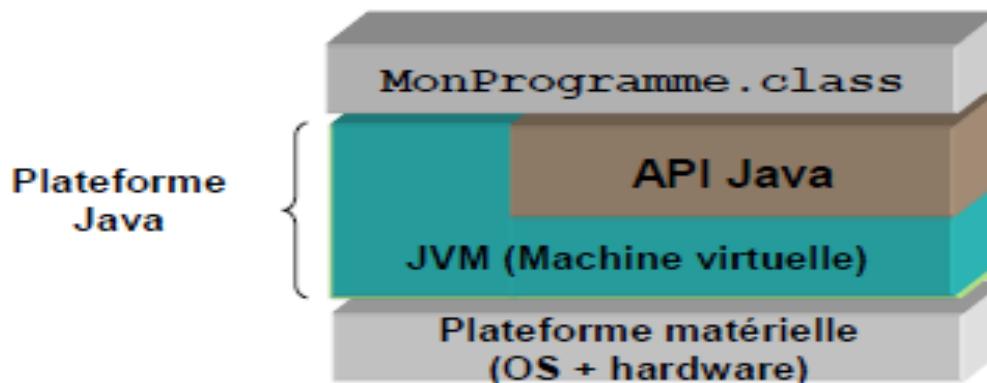
Elle peut également être installée au travers du **Java Development Kit**, qui comporte en plus des librairies permettant de créer des programmes java, comme le compilateur « javac ».

Il en existe deux principaux: celui maintenu par Oracle (**Oracle JDK**), et l'implémentation libre, **OpenJDK**.



Comment ça fonctionne? La plateforme JAVA

- La plateforme est un environnement matériel et/ou logiciel dans lequel un programme s'exécute.
- La plateforme Java est entièrement logicielle et s'exécute au dessus des plateformes matérielles.
- C'est le duo composé par la **JVM** et l'**API Java**



L'**API Java** est un ensemble de librairies logicielles proposant un large choix de fonctionnalités, regroupées en **packages**.

- Accès aux bases de données, la gestion de la sécurité, la création d'interfaces graphiques, ou encore les appels réseau.
- Comporte également des outils pour débugger, monitorer et documenter une application.

Comment ça fonctionne?

Les différentes éditions de Java

Java Standard Edition (J2SE / Java SE)

Environnement d'exécution et ensemble complet d'API pour des applications de type desktop. Cette plate-forme sert de base en tout ou partie aux autres plate-formes

Java Enterprise Edition (J2EE / Java EE)

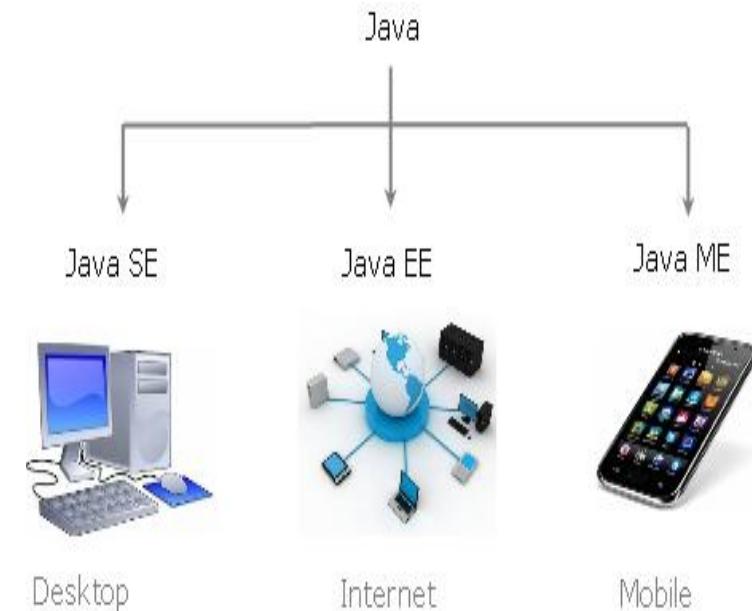
Environnement d'exécution reposant intégralement sur Java SE pour le développement d'applications d'entreprises

Java Micro Edition (J2ME / Java ME)

Environnement d'exécution et API pour le développement d'applications sur appareils mobiles et embarqués dont les capacités ne permettent pas la mise en oeuvre de Java SE

Les types d'applications qui peuvent être développées en Java sont nombreux et variés :

- Applications desktop
- Applications web : servlets/JSP, portlets, applets
- Applications pour appareil mobile (MIDP) : midlets
- Applications pour appareil embarqué (CDC) : Xlets
- Applications pour carte à puce (Javacard) : applets Javacard
- Applications temps réel



Comment ça fonctionne? Écosystème JAVA

Comme pour toute technologie, il existe un ensemble de programmes qui sont particulièrement complémentaires avec un programme écrit en Java.

- Les **serveurs web**: pour afficher une page web, ou présenter une API . On peut citer **Tomcat**, ou encore des serveurs embarquant les librairies du standard JEE, comme **Wildfly**, ou **Glassfish**.
- Des **environnements de développement**, ou **IDE**, permettant de coder le java, comme **IntelliJ** ou **Eclipse**.
- Des **frameworks**, facilitant l'écriture de programmes java, comme **Struts**, ou **Spring**, des gestionnaires de dépendances (**Maven**), des outils d'intégration continue (**Jenkins**)...

De quoi on a besoin (le minimum)?

- Un éditeur de texte (bloc notes, notepad++)
- Le kit de développement Java ([JDK](#)) contenant :
 - Java Runtime Environment ([JRE](#))
 - La machine virtuelle de Java ([JVM](#))
 - Des librairies (JSE : Java Standard Edition, JEE : Java Enterprise Edition, JME Java Micro Edition...) conformes aux JSR (Java Specification Requests)
 - Les commandes permettant la création, la compilation et l'exécution d'un programme Java
 - **javac** : pour compiler
 - **java** : pour exécuter
 - **javadoc** : pour générer une documentation
 - **jar** : pour archiver

De quoi on a besoin (le minimum)?

JDK: téléchargement

<https://www.oracle.com/java/technologies/javase-downloads.html>

The screenshot shows the Oracle Java SE Downloads page. At the top, there's a navigation bar with links for Products, Resources, Support, Events, Developer, View Accounts, Contact Sales, and Java SE Subscriptions. Below the navigation, the page title is "Java SE Downloads" and "Java Platform, Standard Edition".

Java SE 16
Java SE 16 is the latest release for the Java SE Platform.

Links include:

- Documentation
- Installation Instructions
- Release Notes
- Oracle License
 - Binary License
 - Documentation License
- Java SE Licensing Information User Manual
 - Includes Third Party Licenses
- Certified System Configurations
- Readme

Java SE 15
Java SE 15.0.2 is the latest release for the Java SE 15 Platform.

Links include:

- Documentation
- Installation Instructions
- Release Notes
- Oracle License
 - Binary License
 - Documentation License
- Java SE Licensing Information User Manual
 - Includes Third Party Licenses
- Certified System Configurations
- Readme

Looking for Oracle OpenJDK builds?

Links include:

- Oracle Customers and ISVs targeting Oracle LTS releases: Oracle JDK is Oracle's supported Java SE version for customers and for developing, testing, prototyping or demonstrating your Java applications.
- End users and developers looking for free JDK versions: Oracle OpenJDK offers the same features and performance as Oracle JDK under the GPL license..

To Learn more about these options visit Oracle JDK Releases for Java 11 and Later.

Java SE 11 (LTS)
Java SE 11.0.10 is the latest release for the Java SE 11 Platform.

Links include:

- Documentation
- Installation Instructions
- Release Notes
- Oracle License
 - Binary License
 - Documentation License
- Java SE Licensing Information User Manual
 - Includes Third Party Licenses
- Certified System Configurations
- Readme

Download links for Oracle JDK are provided for each section, with options for "JDK Download" and "Documentation Download".

De quoi on a besoin (le minimum)?

Remarque

Pour lancer un programme en ligne de commande, il faut :

- aller dans **Panneau de configuration**, chercher **Système** et cliquer sur **Paramètres systèmes avancés**
- choisir **Variables d'environnement** puis dans la zone **Variables utilisateur** sélectionner **Path** et cliquer sur **Modifier**
- cliquer sur **Nouveau** puis saisir le chemin vers la **JDK** dans la **zone de saisie qui a apparu**
- valider

Compilation

Pour compiler

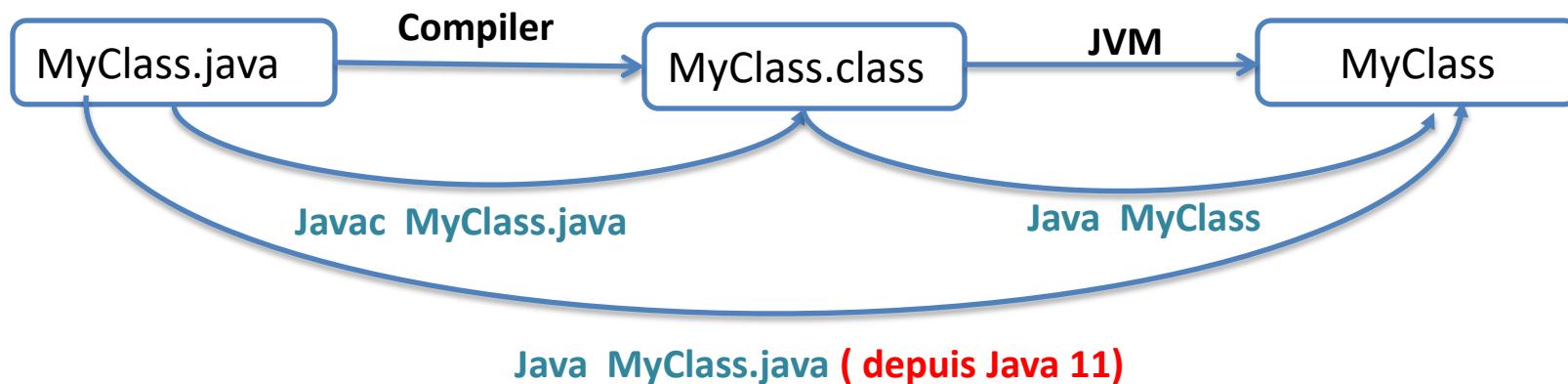
```
javac MyClass.java
```

S'il existe plusieurs versions de JDK sur la machine

```
javac -target 8 -version 8 MyClass.java
```

Pour exécuter (ça affiche Hello world from console)

```
java MyClass
```



IDE Java

On peut aussi utiliser un IDE (Environnement de développement intégré)

- Eviter d'utiliser la console et les commandes
- IDE intègre un compilateur lancé même pendant l'écriture du code
- Profiter de la coloration syntaxique, l'auto-complétion, l'indentation automatique...
- Avoir une bonne structuration du projet
- **Nombreux IDE (Integrated Development Environment) pour java**
 - Editeur syntaxique, débogueur, compilateur, exécution
- **Commerciaux**



JCreator
Xinox



WebSphere Studio
Site Developer for Java
IBM



JBuilder
Codegear



JDeveloper
Oracle



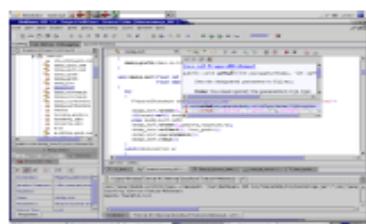
Visual J++
Microsoft

C# .net

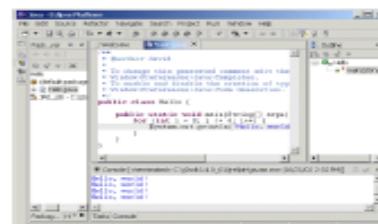
IntelliJIDEA
JetBrains

...

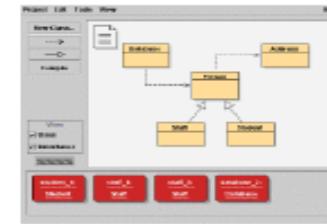
- **Open-source et/ou freeware**



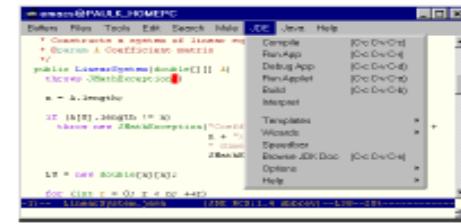
NetBeans
www.netbeans.org



Eclipse
www.eclipse.org



BlueJ
www.bluej.org



Emacs + JDE
<http://sunsite.auc.dk/jde>

Eclipse

- Open-source
- Ecrit en Java
- Multi-langage : Java, C++, PHP, Cobol, C#, JavaScript..

The screenshot shows the Eclipse Installer interface, which lists five different Eclipse IDE configurations:

- Eclipse IDE for Java Developers**: The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.
- Eclipse IDE for Enterprise Java Developers**: Tools for Java developers creating Enterprise Java and Web applications, including a Java IDE, tools for Enterprise Java, JPA, JSF, Mylyn, Maven, Git and...
- Eclipse IDE for C/C++ Developers**: An IDE for C/C++ developers with Mylyn integration.
- Eclipse IDE for JavaScript and Web Developers**: The essential tools for any JavaScript developer, including JavaScript, HTML, CSS, XML languages support, Git client, and Mylyn.
- Eclipse IDE for PHP Developers**: The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and XML.

Avant de commencer

Les règles de nommage en Java:

- Pour les classes et les fichiers : **Le Pascal case**
Convention : les mots sont liés sans espace. Chaque mot commence par une Majuscule.
Exemples : **MyVariableName, MyUrl**
- Pour les variables, les objets et les méthodes : **Le Camel case**
Convention : les mots sont liés sans espace. Chaque mot commence par une majuscule à l'exception du premier.
Exemples : **myVariableName, myUrl**
- Pour les noms de projets : **Le Kebab case**
Convention : les mots sont en minuscule et sont liés par des tirets (-).
Exemples : **my-variable-name, my-url**

Avant de commencer

Les instructions

- Chaque instruction se termine par ;
- Il est possible d'écrire plusieurs instructions sur une même ligne (mais ce n'est pas une bonne pratique)
- Eclipse nous facilite le formatage et l'indentation du code avec le raccourci CTRL + Shift + F

Comment organiser un projet Java?

- Une classe par fichier
- Organiser les classes par package selon la sémantique
- Une classe ne peut être définie dans plusieurs fichiers (pas de classe partielle en Java)
- Il est possible de créer deux classes avec le même nom dans deux packages différents

chapitre 1

 Introduction à la POO

 Principes de la POO

 Présentation du langage Java

 Syntaxe et bases du langage Java

Un premier projet Java

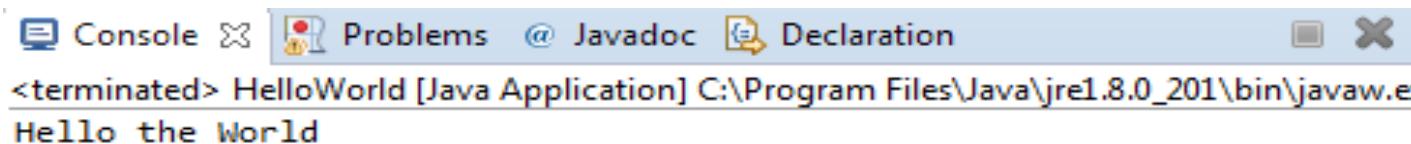
The screenshot shows an IDE window with a Java file named `HelloWorld.java`. The code contains a single class definition:public class HelloWorld{ public static void main(String[] args){ System.out.println("Hello the World"); }}

```
Annotations on the left side of the code explain the code structure:
```

- Nom de la classe (Name of the class) points to the word `HelloWorld`.
- La fonction principale (Main function) points to the `main` method.
- Méthode d'affichage dans la fenêtre console (Display method in the console window) points to the `System.out.println` statement.

An orange callout box on the right side provides additional information about the `args` parameter:

Permet de récupérer des arguments transmis au programme au moment de son lancement



Les commentaires

Trois types de commentaires

Commentaire mono-ligne

```
// ceci est un commentaire sur une seule ligne
```

Commentaire sur plusieurs lignes

```
/* ceci est un  
commentaire  
sur trois lignes */
```

Commentaire pour documentation

```
/** Commentaires destinés à la documentation  
@author Nom Auteur  
*/
```

Les mots clés de Java

L'identificateur ne peut pas être un mot réservé Java :

abstract	continue	for	new	switch
assert	default	goto	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	float	native	super	while

Les variables

Une variable

- Un pointeur vers une zone mémoire
- Permettant de stocker une ou plusieurs données
- Pouvant avoir plusieurs valeurs différentes dans un programme

Java est un langage de programmation fortement typé

- Il faut préciser le type de chaque variable
- Une variable peut avoir de valeurs différentes mais ne peut changer de type.

Deux modes de typage en Java

- **Statique** : le type est précisé à la déclaration de la variable.
- **Dynamique** : la variable est déclarée avec le mot-clé **var** et son type est dynamiquement déterminé à partir de la valeur initiale.

[disponible depuis Java 11]

Les variables : typage statique

Déclarer une variable

type nomVariable;

Deux choix possibles pour typer les variables

Types simples (primitifs)

commencent par une lettre en minuscule.

-Exemple :

`int x;`

-Déclarer et initialiser une variable :

`int x = 5;`

Types objets (référence)

commencent par une lettre en majuscule.

-Exemple :

`Point x;`

-Déclarer et initialiser une variable :

`Point x =new Point(15,4);`

Les variables : typage statique

Les types primitifs

Type	Signification	Taille (en octets)	Plage de valeurs acceptées
char	Caractère Unicode	2	0 à 65535
byte	Entier très court	1	-128 à 127
short	Entier court	2	-32 768 à 32 767
int	Entier	4	-2 147 483 648 à + 2 147 483 647
long	Entier long	8	$-9,223 \times 10^{18}$ à $9,223 \times 10^{18}$
float	Nombre réel simple	4	1.4×10^{-45} à 3.4×10^{38}
double	Nombre réel double	8	$4,9 \times 10^{-324}$ à $1,8 \times 10^{308}$
boolean	Valeur logique (booléen)	1	true (vrai), ou false (faux)

Pas de type primitif pour les chaînes de caractère.

Les variables : typage statique

Casting des primitives

Java est un langage fortement typé

- *le type de donnée est associé au nom de la variable, plutôt qu'à sa valeur. (Avant de pouvoir être utilisée une variable doit être déclarée en associant un type à son identificateur).*
- *la compilation ou l'exécution peuvent détecter des erreurs de typage*

Dans certains cas, il est nécessaire de forcer le programme à considérer une expression comme étant d'un type qui n'est pas son type réel ou déclaré

On utilise pour cela le **casting** (transtypage) :
(type-forcé) expression

Les variables : typage statique

Casting des primitives

Type1 nom1 = valeur1

Type2 nom2 = valeur2

Sur-Casting : la **conversion** peut rester implicite si le type1 est moins fort que le type2.

```
int i =10;  
double x =i;
```

Sous-Casting : la **conversion** doit devenir explicite si le type1 est strictement plus fort que le type2. (il faut indiquer au compilateur d'effectuer le cast)

```
double x=4.1532;  
int i =(int) x;
```

Les variables : typage statique

Casting des primitives

Source data type

	byte	short	int	long	float	double	char	boolean
byte	-	✗	✗	✗	✗	✗	✗	✗
short	✓	-	✗	✗	✗	✗	✗	✗
int	✓	✓	-	✗	✗	✗	✗	✗
long	✓	✓	✓	-	✗	✗	✗	✗
float	✓	✓	✓	✓	-	✗	✗	✗
double	✓	✓	✓	✓	✓	-	✗	✗
char	✗	✗	✗	✗	✗	✗	-	✗
boolean	✗	✗	✗	✗	✗	✗	✗	-

— Data assignment possible but no automatic conversion happens
✗ not compatible because source data type is higher than destination data type
✓ Compatible because source data type is lower than destination data type

le cast pour les types compatibles

```
int x = 100;
byte z = (byte) x;
System.out.println(z);
```

Attention aux valeurs qui dépassent l'intervalle

- int x = 200;
- byte z = (byte) x;
- System.out.println(z);

Les variables : typage statique

Wrapper des types primitifs

- Les classes enveloppes sont immuables.

Type primitif	Classe enveloppe	
boolean	Boolean	
char	Character	
byte	Byte	
short	Short	
int	Integer	Sous-classes de la classe Number
long	Long	
float	Float	
double	Double	

Il existe aussi plusieurs autres types objets : String, Date...

Les variables : typage statique

Wrapper des types primitifs

- Les classe enveloppes (*Wrappers*) sont des classes qui encapsulent les données de type primitif, afin de pouvoir les utiliser comme des objets ordinaires.
- Depuis Java 5, les opérations de conversion d'un type primitif vers un objet d'une classe enveloppe, et la conversion inverse sont devenues automatique : *boxing* et *unboxing*.
- Toutes les classes wrappers contiennent une méthode (*TypeObjet.parseTypeSimple(string)*) qui permet de convertir une chaîne de caractère en type primitif de cette classe.

Les variables : typage statique

Quelques méthodes de la classe String

- **length()** : retourne le nombre de caractère de la chaîne.
- **indexOf(x)** : retourne l'indice de la première occurrence de la valeur de x dans la chaîne, -1 sinon.
- **contains()** : retourne true si la chaîne contient x, false sinon.
- **charAt(i)** : retourne le caractère d'indice i dans la chaîne.
- **substring(i,j)** : permet d'extraire une sous- chaîne de la chaîne à partir de l'indice i jusqu'au l'indice j - 1
- **equals(str)** : permet de comparer la chaîne à str et retourne true en cas d'égalité, false sinon.
- **replace(old,new)** : permet de remplacer toute occurrence de la chaîne old dans la chaîne courante par new et retourne la nouvelle chaîne
- ...

Les variables : typage statique

Quelques méthodes de la classe String

Exemple : replace(old,new)

```
String string = "bonjour les bons jours";
String string2 = string.replace("jour", "soir");
System.out.println(string2);
```

Exemple : lenght()

```
String string = "bonjour les bons jours";
System.out.println(string.lenght());
```

Exemple de conversion d'une chaîne de caractères

```
String string = "2";
byte z = Byte.parseByte(string);
System.out.println(z); // affiche 2
```

Exemple de conversion d'un Integer en chaîne de caractère

```
Integer x = 2;
String string = x.toString();
System.out.println(string); // affiche 2
```

Les variables : typage statique

Quelques méthodes de la classe Math

On peut aussi utiliser les méthodes statiques de la classe Math

- **Math.abs(x)** : retourne la valeur absolue de x
- **Math.pow(x,y)** : retourne la x puissance y
- **Math.max(x,y)** : retourne le max de x et y
- **Math.min(x,y)** : retourne le min de x et y
- **Math.sqrt(x)** : retourne la racine carré de x
- **Math.floor(x), Math.ceil(x) et Math.round(x)** : retournent l'arrondi de x
- **Math.random()** : retourne une valeur aléatoire entre 0 et 1
- ...

Les variables : typage statique

Pourquoi les types primitifs et les types objets?

- **Les types primitifs** sont moins coûteux en mémoire
- **Les types objets** offrent plusieurs méthodes à appliquer sur les valeurs : conversion, nombre de caractère...

Les variables : typage statique

Java manipule différemment les types primitifs et les types objets

Code

```
int a= 5;  
char c= 'a';  
String s=" Bonjour";  
int [ ] tab ={1, 4, 10};
```

Pile

tab	845	#52
s	1040	#100
c	a	#10
a	5	# 109

Tas

Bonjour	1040
1	4

1	4	10	845
---	---	----	-----

Les variables : typage statique

Java manipule différemment les types primitifs et les types objets

Code

```
int a= 5;  
char c= 'a';  
String s=" Bonjour";  
int [ ] tab ={1, 4, 10};  
int b= a;
```

Pile

tab	845	#52
s	1040	#100
b	5	#250
c	a	#10
a	5	# 109

Tas

Bonjour	1040
1	4

1	4	10	845
---	---	----	-----

Les variables : typage statique

Java manipule différemment les types primitifs et les types objets

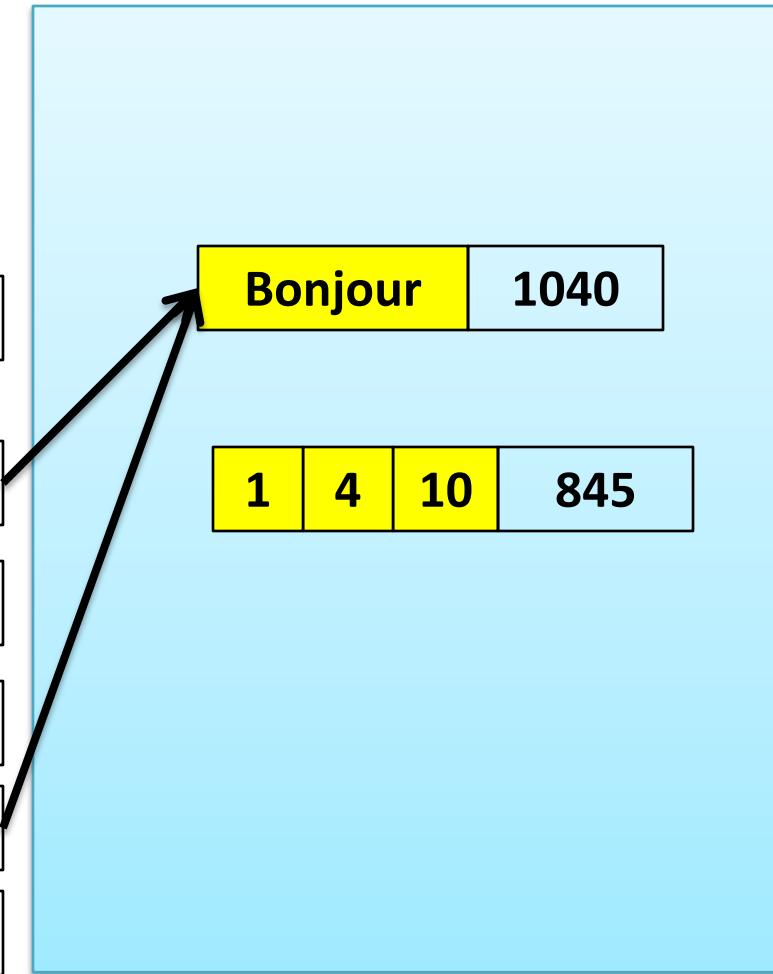
Code

```
int a= 5;  
char c= 'a';  
String s=" Bonjour";  
int [ ] tab ={1, 4, 10};  
int b= a;  
String str= s;
```

Pile

tab	845	#52
s	1040	#100
b	5	#250
c	a	#10
str	1040	#200
a	5	# 109

Tas



Les variables : typage statique

Java manipule différemment les types primitifs et les types objets

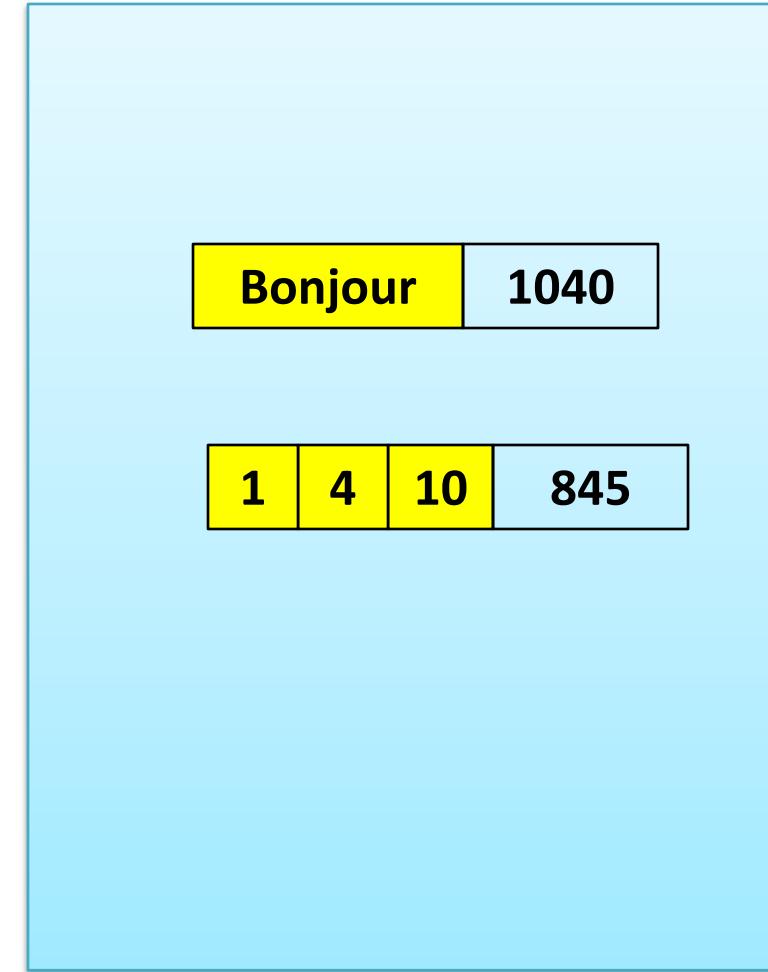
Code

```
int a= 5;  
char c= 'a';  
String s=" Bonjour";  
int [ ] tab ={1, 4, 10};  
  
int b= a;  
  
String str= s;  
str= "Bonsoir";
```

Pile

tab	845	#52
s	1040	#100
b	5	#250
c	a	#10
str	1040	#200
a	5	# 109

Tas



Les variables : typage statique

Java manipule différemment les types primitifs et les types objets

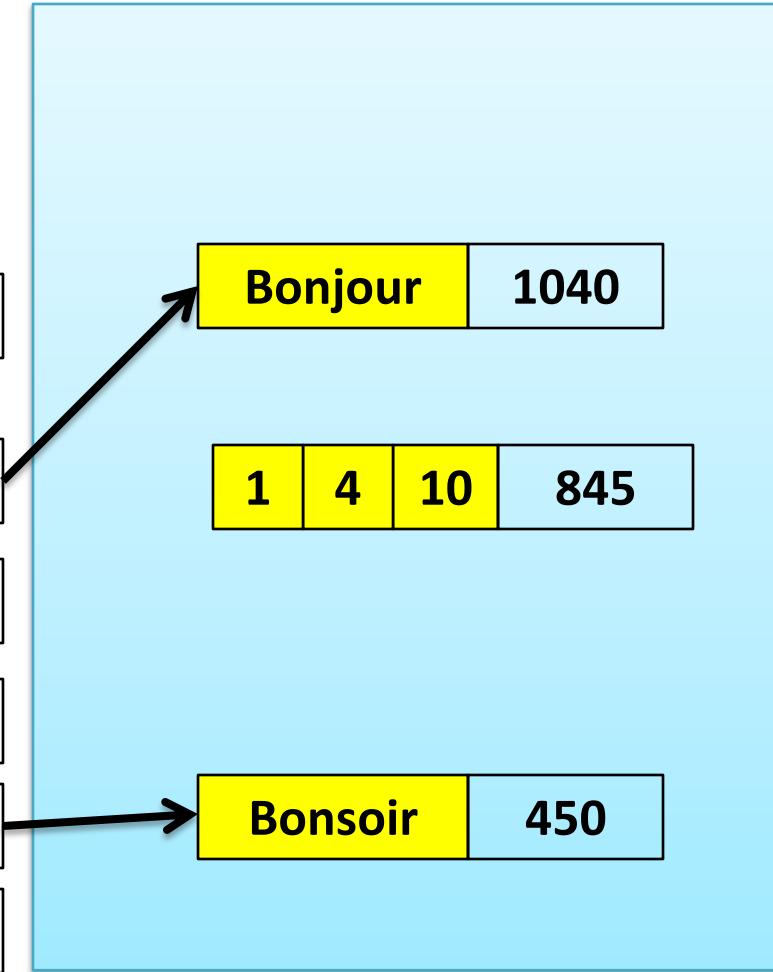
Code

```
int a= 5;
char c= 'a';
String s=" Bonjour";
int [ ] tab ={1, 4, 10};
int b= a;
String str= s;
str= "Bonsoir";
```

Pile

tab	845	#52
s	1040	#100
b	5	#250
c	a	#10
str	450	#200
a	5	# 109

Tas



Les variables : typage statique

Java manipule différemment les types primitifs et les types objets

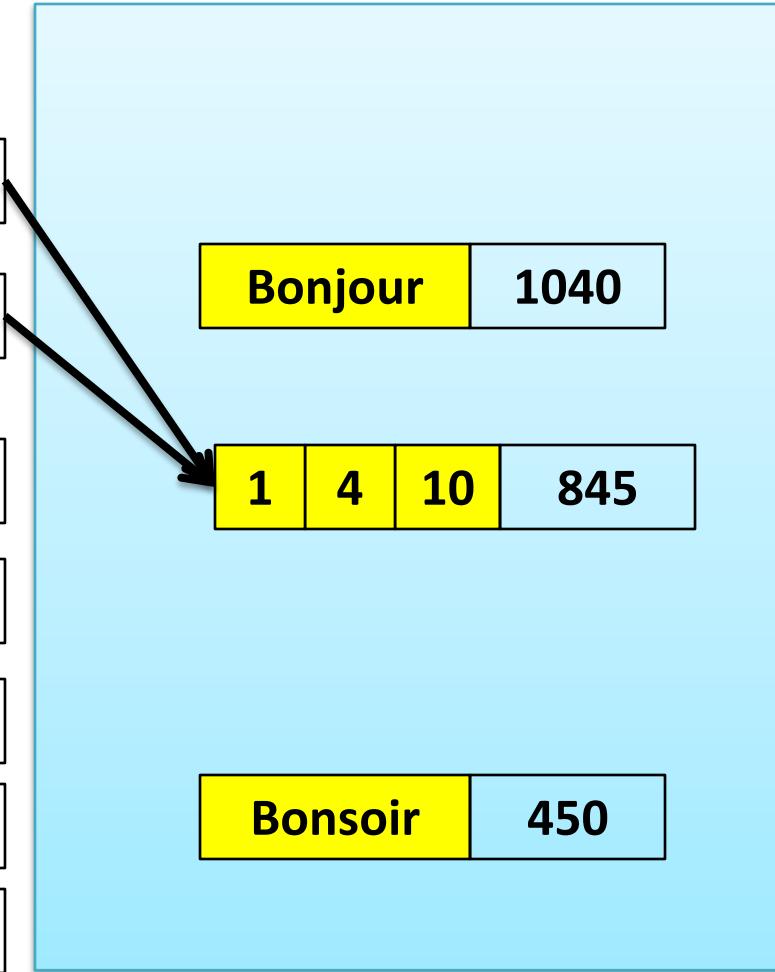
Code

```
int a= 5;
char c= 'a';
String s=" Bonjour";
int [ ] tab ={1, 4, 10};
int b= a;
String str= s;
str= "Bonsoir";
int [ ] t =tab;
```

Pile

t	845	#24
tab	845	#52
s	1040	#100
b	5	#250
c	a	#10
str	450	#200
a	5	# 109

Tas



Les variables : typage statique

Java manipule différemment les types primitifs et les types objets

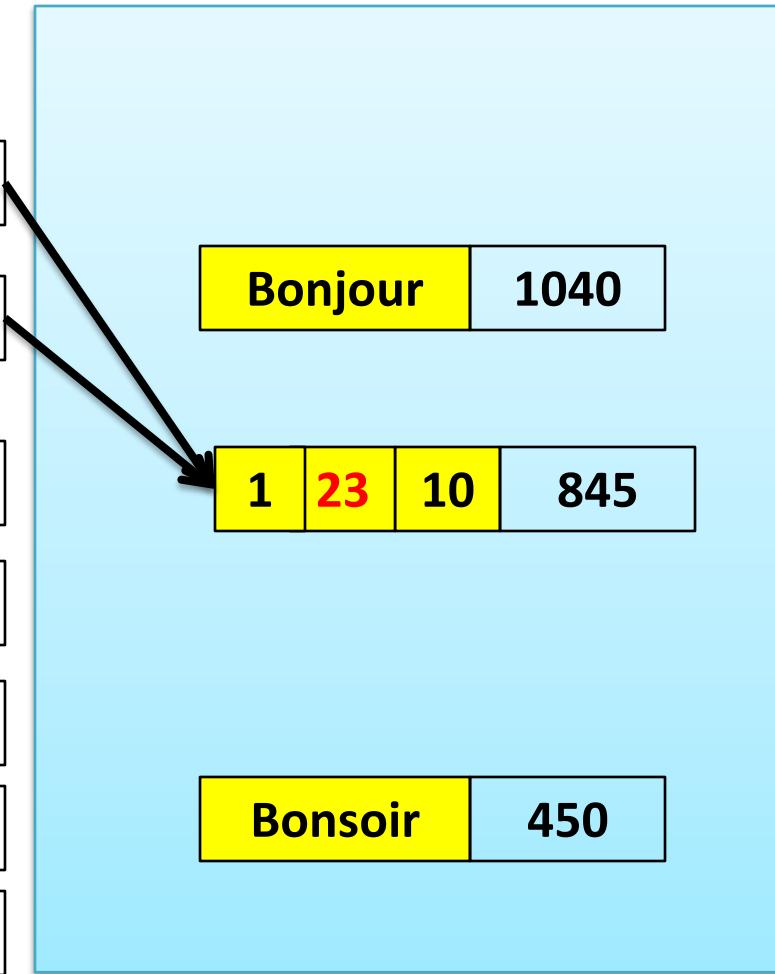
Code

```
int a= 5;
char c= 'a';
String s=" Bonjour";
int [ ] tab ={1, 4, 10};
int b= a;
String str= s;
str= "Bonsoir";
int [ ] t =tab;
t[1]=23;
```

Pile

t	845	#24
tab	845	#52
s	1040	#100
b	5	#250
c	a	#10
str	450	#200
a	5	# 109

Tas



Les variables : typage statique

Java manipule différemment les types primitifs et les types objets

Code

```
int a= 5;
char c= 'a';
String s=" Bonjour";
int [ ] tab ={1, 4, 10};
int b= a;
String str= s;
str= "Bonsoir";
int [ ] t =tab;
t[1]=23;
int d =new int [4];
```

Pile

d	90	#104
t	845	#24
tab	845	#52
s	1040	#100
b	5	#250
c	a	#10
str	450	#200
a	5	# 109

Tas



Les variables : typage statique

- Le ramasse-miettes (garbage collector) est une tâche qui
 - libère la place occupée par les instances non référencées
 - compacte la mémoire occupée

Code

```
int a= 5;
char c= 'a';
String s=" Bonjour";
int [ ] tab ={1, 4, 10};

int b= a;

String str= s;
str= "Bonsoir";

int [ ] t =tab;
t[1]=23;
int d =new int [4];
```

Pile

d	90	#104
t	845	#24

tab	845	#52
-----	-----	-----

str	450	#200
-----	-----	------

Tas

0	0	0	0	104
---	---	---	---	-----

1	23	10	845
---	----	----	-----

Bonsoir	450
---------	-----

Les variables : typage dynamique

Déclarer une variable avec le mot-clé **var**

```
var nomVariable = valeurInitiale;
```

Remarque

Une variable déclarée avec le mot-clé **var**

- doit être initialisée
- a un type déterminé en fonction de sa valeur initiale
- ne peut changer de type

Exemple

```
var x = 10;
```

Ceci génère une erreur

```
x = "bonjour";
```

Les constantes

Une constante un élément qui ne peut changer de valeur

Pour déclarer une constante il faut utiliser le mot-clé **final**

Déclaration d'une constante

final double PI = 3.1415;

L'instruction suivante ne peut être acceptée

PI = 5;

Le nom d'une constante doit être en majuscule.

Les mots composant le nom d'une constante doivent être séparés par un caractère de soulignement.

Les opérations sur les variables

Pour les variables numériques (int, float...)

- = : affectation
- + : addition
- - : soustraction
- * : multiplication
- / : division
- % : reste de la division

Quelques raccourcis

- i = i + 1 ⇒ i++;
- i = i - 1 ⇒ i--;
- i = i + 2 ⇒ i += 2;
- i = i - 2 ⇒ i -= 2;
- ...

L'opérateur + pour concaténer deux chaînes de caractère

```
String string = "bon";
String string2 = "jour";
System.out.println(string + string2);
// affiche bonjour
```

Les opérations sur les variables

Opérateurs de comparaison

- `==` : pour tester l'égalité
- `!=` : pour tester l'inégalité
- `>` : supérieur à
- `<` : inférieur à
- `>=` : supérieur ou égal à
- `<=` : inférieur ou égal à

En java, on ne peut comparer deux valeurs de type incompatible

Opérateurs logiques

- `&&` : et
- `||` : ou
- `!` : non

Les opérations sur les variables

Opérateur **instanceof**

- La syntaxe est :

Objet instanceof nomClasse

- Exemple : **if (x instanceof Livre)**

- Le résultat est un booléen :

- **true** si x est de la classe **Livre**
 - **false** sinon

La lecture d'une saisie

Pour lire une valeur saisie par l'utilisateur dans la console

- il faut utiliser la classe **Scanner**
- il faut préciser le type de la valeur à récupérer

Pour utiliser la classe Scanner, il faut d'abord l'importer :

import java.util.Scanner;

Ensuite il faut créer un objet de la classe Scanner :

Scanner clavier= new Scanner(System.in);

Exemple de lecture d'un entier

int i = clavier.nextInt();

Fermer le scanner

clavier.close();

La lecture d'une saisie

- Pour récupérer les données, il faut faire appel sur l'objet **clavier** aux méthodes décrites ci-dessous. Ces méthodes parcouruent la donnée suivante lue sur l'entrée et la retourne :
 - `String next()` : donnée de la classe `String` qui forme un mot,
 - `String nextLine()` : donnée de la classe `String` qui forme une ligne,
 - `boolean nextBoolean()` : donnée booléenne,
 - `int nextInt()` : donnée entière de type `int`,
 - `double nextDouble()` : donnée réelle de type `double`
- Il peut être utile de vérifier le type d'une donnée avant de la lire :
 - `boolean hasNext()` : renvoie `true` s'il y a une donnée à lire
 - `boolean hasNextLine()` : renvoie `true` s'il y a une ligne à lire
 - `boolean hasNextBoolean()` : renvoie `true` s'il y a un booléen à lire
 - `boolean hasNextInt()` : renvoie `true` s'il y a un entier à lire
 - `boolean hasNextDouble()` : renvoie `true` s'il y a un double à lire.

La lecture d'une saisie

```
Scanner scanner = new Scanner(System.in);
// si on saisit "bonjour tout le monde"
String string = scanner.next();
System.out.println(string);
// affiche bonjour
```

```
Scanner scanner = new Scanner(System.in);
// si on saisit "bonjour tout le monde"
String string = scanner.nextLine();
System.out.println(string);
// affiche bonjour tout le monde
```

À chaque appel à `next()`, on récupère le token suivant

```
Scanner scanner = new Scanner(System.in);
// si on saisit "bonjour tout le monde"
String string = scanner.next();
String s = scanner.next();
System.out.println(s);
// affiche tout
```

La lecture d'une saisie

Le code suivant déclenche une exception car on ne peut affecter la chaîne "tout" à un entier

```
Scanner scanner = new Scanner(System.in);
// si on saisit "bonjour tout le monde"
String string = scanner.next();
int v = scanner.nextInt();
System.out.println(v);
```

Il n'existe pas une méthode spécifique aux caractères, mais on peut faire

```
Scanner scanner = new Scanner(System.in);
String string = scanner.next();
char c = scanner.next().charAt(0);
System.out.println(c);
```

Les conditions et les boucles

Les conditions :

On appelle traitement conditionnel une portion de code qui n'est pas exécutée systématiquement.

Les conditions et les boucles

Les conditions :

if

Exécuter si une condition est vraie

```
if (condition) {  
    ...  
}
```

Exemple

```
var x = 3;  
if (x >= 0) {  
    System.out.println(x + " est positif");  
}
```

Les conditions et les boucles

Les conditions :

if ... else

Exécuter un premier bloc si une condition est vraie, un deuxième sinon (le bloc else)

```
if (condition1) {  
    ...  
}  
else{  
    ...  
}
```

Exemple

```
var x = 3;  
if (x > 0)  
{  
    System.out.println(x + " est strictement positif");  
}  
else  
{  
    System.out.println(x + " est négatif ou nul");  
}
```

Les conditions et les boucles

Les conditions :

if ... else if ... else

On peut enchaîner les conditions avec else if (et avoir un troisième bloc voire ... un nième)

```
if (condition1) {  
    ...  
}  
else if (condition2) {  
    ...  
}  
...  
else {  
    ...  
}
```

Tester plusieurs conditions (on utilisant des opérateurs logiques)

```
if (condition1 && !condition2 || condition3){  
    ...  
}  
[else ...]
```

Les conditions et les boucles

Les conditions :

if ... else if ... else

Exemple

```
var x = -3;
if (x > 0)
{
    System.out.println(x + " est strictement positif");
}
else if (x < 0)
{
    System.out.println(x + " est strictement négatif");
}
else
{
    System.out.println(x + " est nul");
}
```

Les conditions et les boucles

Les conditions : Structure conditionnelle switch

```
switch (nomVariable) {  
    case constante-1:  
        groupe-instructions-1;  
        break;  
    case constante-2:  
        groupe-instructions-2;  
        break;  
    ...  
    case constante-N:  
        groupe-instructions-N;  
        break;  
    default:  
        groupe-instructions-par-défaut;  
}
```

- Le **switch** permet seulement de tester l'égalité
- Le **break** permet de quitter le switch une fois un bloc case exécuté
- Il est possible de regrouper plusieurs case
- Le bloc **default** est facultatif, il sera exécuté si la valeur de la variable ne correspond à aucune constante de case

Les conditions et les boucles

Les conditions : **Structure conditionnelle switch**

Exemple

```
int x = 5;
switch (x) {
    case 1:
        System.out.println("un");
        break;
    case 2:
        System.out.println("deux");
        break;
    case 3:
        System.out.println("trois");
        break;
    default:
        System.out.println("autre");
}
```

Les conditions et les boucles

Les conditions : Structure conditionnelle switch

Exemple: Un multi-case pour un seul traitement

```
var x = 5;
switch (x) {
    case 1:
    case 2:
        System.out.println("un ou deux");
        break;
    case 3:
        System.out.println("trois");
        break;
    case 4:
    case 5:
        System.out.println("quatre ou cinq");
        break;
    default:
        System.out.println("autre");
}
```

Les conditions et les boucles

Les conditions : **Structure conditionnelle switch**

Exemple: Si on supprime un break

```
String x = "2";
switch (x) {
    case "1":
        System.out.println("un");
        break;
    case "2":
        System.out.println("deux");
    case "3":
        System.out.println("trois");
        break;
    default:
        System.out.println("autre");
}
// affiche deux trois
```

Les conditions et les boucles

Les conditions :

Le nouveau switch

La variable dans switch peut être

- de types simples : int, short, byte et char
- de type wrappers correspondants : Integer, Short, Byte et Character
- de type énumération depuis Java 6
- de type chaîne de caractère depuis Java 7

depuis Java 12,13,14:

- Expression switch (switch n'est donc plus juste une structure de contrôle (comme les if/else), mais peut renvoyer une valeur).
- Utilisation de l'opérateur arrow (->) qui supprime le besoin d'instructions break. Précisons que seul le code à droite de -> est exécuté.
- plusieurs lignes de code pour calculer la valeur finale du switch. Ces différentes lignes de code devront être placées entre une paire d'accolades et pour indiquer que le calcul est terminé, vous devrez utiliser le mot clé **yield**.

Les conditions et les boucles

Les conditions :

Le nouveau switch

```
1 public class Sample {  
2  
3     public static void main( String [] args ) {  
4  
5         int value = (int) (Math.random() * 11);  
6  
7         switch( value ) {  
8             case 0:  
9             case 1:  
10            case 2:  
11            case 3:  
12            case 4:  
13                System.out.println( "Petit chiffre" );  
14                break;  
15            case 5:  
16            case 6:  
17            case 7:  
18            case 8:  
19            case 9:  
20                System.out.println( "Grand chiffre" );  
21                break;  
22            default:  
23                System.out.println( "Ce n'est plus un chiffre, mais un nombre" );  
24        }  
25    }  
26}  
27}  
28 }
```

Les conditions et les boucles

Les conditions :

Le nouveau switch

```
1 public class Sample {  
2  
3     public static void main( String [] args ) {  
4  
5         int value = (int) (Math.random() * 11);  
6  
7         switch( value ) {  
8             case 0, 1, 2, 3, 4 -> System.out.println( "Petit chiffre" );  
9             case 5, 6, 7, 8, 9 -> System.out.println( "Grand chiffre" );  
10            default -> System.out.println( "Ce n'est plus un chiffre, mais un nombre" );  
11        }  
12    }  
13}  
14}  
15 }
```

Rappels sur la syntaxe traditionnelle du switch

Les conditions et les boucles

Les conditions :

Le nouveau switch

```
1 public class Sample {  
2  
3     public static void main( String [] args ) {  
4  
5         int value = (int) (Math.random() * 11);  
6  
7         String result = switch( value ) {  
8             case 0, 1, 2, 3, 4 -> "Petit chiffre";  
9             case 5, 6, 7, 8, 9 -> "Grand chiffre";  
10            default -> "Ce n'est plus un chiffre, mais un nombre";  
11        };  
12  
13         System.out.println( result );  
14     }  
15 }  
16  
17 }
```

Exemple d'utilisation d'un switch sous forme d'expression

Les conditions et les boucles

Les conditions :

Le nouveau switch

```
1 public class Sample {  
2  
3     public static void main( String [] args ) {  
4  
5         int value = (int) (Math.random() * 11);  
6  
7         System.out.println( switch( value ) {  
8             case 0, 1, 2, 3, 4 -> "Petit chiffre";  
9             case 5, 6, 7, 8, 9 -> "Grand chiffre";  
10            default -> "Ce n'est plus un chiffre, mais un nombre";  
11        } );  
12    }  
13}  
14}  
15 }
```

Utilisation d'un switch dans un appel de méthode

Les conditions et les boucles

Les conditions :

Le nouveau switch

```
1 public class Sample {  
2  
3     public static void main( String [] args ) {  
4  
5         int value = (int) (Math.random() * 11);  
6  
7         String result = switch( value ) {  
8             case 0, 1, 2, 3, 4 -> {  
9                 double sqrt = Math.sqrt( value );  
10                yield "Petit chiffre dont la racine carré vaut " + sqrt;  
11            }  
12            case 5, 6, 7, 8, 9 -> {  
13                double square = value * value;  
14                yield "Grand chiffre dont le carré vaut " + square;  
15            }  
16            default -> "Ce n'est plus un chiffre, mais un nombre";  
17        };  
18  
19        System.out.println( result );  
20    }  
21}  
22}  
23 }
```

Exemple d'utilisation du mot clé yield

Les conditions et les boucles

Les conditions :

Le nouveau switch

Exercice:

Ecrire un programme permettant d'appliquer une opération sur deux valeurs numériques.

L'opérateur ainsi que les valeurs doivent être saisis à partir de la console en suivant le format suivant :

java Demo operateur valeur1 valeur2.

Un minimum de quatre opérateurs doivent être fournis :

add , sous, mul et div.

Voici un petit exemple d'utilisation du programme à développer.

```
C:\Users\lenovo\Desktop>java Exemple demo
Usage: java Demo operateur valeur1 valeur2
C:\Users\lenovo\Desktop>java Exemple add 14 2
16.0
C:\Users\lenovo\Desktop>java Exemple sous 14 2
12.0
C:\Users\lenovo\Desktop>java Exemple mul 14 2
28.0
C:\Users\lenovo\Desktop>java Exemple div 14 2
7.0
C:\Users\lenovo\Desktop>java Exemple dic 14 2
operateur indefini dic
```

Les conditions et les boucles

Les conditions :

Solution:

```
public class Exemple{  
  
    public static void main( String [] args ) {  
  
        if (args.length != 3 ) {  
            System.out.println( "Usage: java Demo operateur valeur1 valeur2" );  
            System.exit( 0 );  
        }  
        String nomOperateur = args[0];  
        double val1 = Double.parseDouble( args[1] );  
        double val2 = Double.parseDouble( args[2] );  
  
        System.out.println ( switch( nomOperateur ) {  
            case "add" -> val1 + val2 ;  
            case "sous" -> val1 - val2;  
            case "mul" -> val1 * val2 ;  
            case "div" -> val1 / val2 ;  
            default -> "operateur indéfini " + nomOperateur;  
        });  
    }  
}
```

Le nouveau switch

Les conditions et les boucles

Les conditions :

Expression ternaire

Simplifier l'écriture avec l'expression ternaire

Exemple

```
int x = 2;  
String type = (x % 2 == 0) ? "pair" : "impair";  
System.out.println(type); // affiche pair
```

Les conditions et les boucles

Les boucles :

Une boucle permet d'exécuter plusieurs fois de suite une même séquence d'instructions.

Les conditions et les boucles

Les boucles : **while**

à chaque itération on teste si la condition est vraie avant d'accéder aux traitements

```
while (condition[s]) {  
    ...  
}
```

N'oubliez pas lorsqu'une boucle fonctionne avec un compteur :

- D'initialiser le compteur avant d'entrer dans la boucle
- D'incrémenter le compteur à la fin du corps
- De contrôler la valeur du compteur dans la condition de boucle

Attention aux boucles infinies, vérifier que la condition d'arrêt sera bien atteinte après un certain nombre d'itérations.

Les conditions et les boucles

Les boucles : **while**

Exemple

```
var i = 0;  
while (i < 5) {  
    System.out.println(i);  
    i++;  
}
```

Résultat

```
0  
1  
2  
3  
4
```

Les conditions et les boucles

Les boucles :

do ... while

La Boucle **do ... while** exécute le bloc au moins une fois ensuite elle vérifie la condition

```
do {  
    ...  
} while (condition[s])
```

Exemple

```
var i = 0;  
do {  
    System.out.println(i);  
    i++;  
} while (i < 5)
```

Résultat

```
0  
1  
2  
3  
4
```

Les conditions et les boucles

Les boucles :

for

```
for (initialisation; condition[s]; incrémentation) {  
    ...  
}
```

La gestion du compteur est automatique (initialisation, incrémantation, sortie de boucle). Il faut connaître à l'avance le nombre d'itérations.

Exemple

```
for (var i = 0; i < 5; i++) {  
    System.out.println(i);  
}
```

Résultat

0
1
2
3
4

Attention aux boucles infinies si vous modifiez la valeur du compteur à l'intérieur de la boucle.

Les conditions et les boucles

Les boucles : **for**

Exercice:

Ecrire un code Java qui permet d'afficher les nombres pairs compris entre 0 et 10.

```
for (var i = 0; i < 10; i++) {  
    if (i % 2 == 0)  
        System.out.println(i);  
}
```

```
for (var i = 0; i < 10; i+=2) {  
    System.out.println(i);  
}
```

Les tableaux

En java, les tableaux sont statiques

- Tous les éléments doivent avoir le même type
- Il faut préciser une taille qu'on ne peut dépasser

Déclaration d'un tableau : syntaxe

```
type [] nomTableau = new type[taille];
```

Exemple: déclaration d'un tableau d'entier de taille 2 :

```
int [] tab = new int[2];
```

Une déclaration de tableau
ne doit pas préciser de
dimensions

```
int monTableau[5];  
// Erreur
```

Affectation de valeurs aux cases de tableau

```
tab[0] = 5;
```

```
tab[1] = 3;
```

Ceci déclenche une exception

```
tab[2] = 10;
```

```
Exception in thread "main" java.lang.  
ArrayIndexOutOfBoundsException: 2  
at org.eclipse.classes.FirstClass.main(  
FirstClass.java:11)
```

Les tableaux

On peut faire une déclaration + une initialisation

```
int [] tab = { 5, 3 };
```

Comme si la taille du tableau a été fixée à deux

Donc, ceci déclenche aussi une exception

```
tab[2] = 10;
```

```
Exception in thread "main" java.lang.  
ArrayIndexOutOfBoundsException: 2  
at org.eclipse.classes.FirstClass.main(  
FirstClass.java:11)
```

Les tableaux

Pour parcourir le tableau

```
for(int i = 0; i < tab.length; i++){
    System.out.println(tab[i]);
}
```

Où encore la version simplifiée (**foreach**)

```
for(int elt : tab){
    System.out.println(elt);
}
```

Lire pour chaque elt de tab

Les tableaux

- Le package **java.util** définit une classe **Arrays**.
- Cette classe expose des méthodes utilitaires qui traitent des tableaux.
- On peut trier ces méthodes dans les catégories suivantes.
- Il existe une version de chacune de ces méthodes pour tous les types primitifs Java, ainsi que pour la classe Object.

- Des méthodes de recherche
 - `int binarySearch(char[] a)` , `int binarySearch(int[] a)`
... `int binarySearch(Object[] a)`
- Des méthodes de tris
 - `sort(char[] a)` , `sort(int[] a)` `sort(Object[] a)`
 - `sort(char[] a, int fromIndex, int toIndex)` , ...
- Des méthodes pour remplissage avec une valeur
 - `fill(char[] a, char val)` , `fill(int[] a, Long val)`
 - `fill(char[] a, char val, int fromIndex, int toIndex)` , ...
- Des méthodes de test d'égalité
 - `boolean equals(char[] a1, char[] a2)`,
 - `boolean equals(int[] a1, int[] a2)` , ...

Les tableaux

Exemple : tri d'un tableau

```
/// tableau de 1000 réels tirés au hasard
// dans l'intervalle [0..1000[
double[] tab= new double[5];
    for (int i = 0; i < tab.length; i++) {
        tab[i] = Math.random()*10;
    }
// tri du tableau
Arrays.sort(tab);
// affiche le tableau trié
    for (int i : tab) {
        System.out.print(i + " ");
    }
```

Les tableaux

- Les tableaux sont des structures de données élémentaires
- Le package `java.util` contient plein de classes pour la gestion de structures de donnée plus évoluées (**collections**):
 - listes
 - Ensemble
 - Arbres
 - ...

Les méthodes

- Méthodes \Leftrightarrow fonctions / procédures
 - Pour factoriser du code
 - Pour structurer le code
 - Pour servir de «sous programmes utilitaires» aux autres méthodes de la classe
 - ...
- En java plusieurs types de méthodes
 - Opérations sur les objets (cf. envois de messages)
 - Opérations statiques (méthodes de classe)
- Pour le moment nous ne nous intéresserons qu'au second type

Les méthodes statiques

Caractérisée par :

visibilité [+ static] + type de retour +
nomMéthode([les paramètres]) + son implémentation

Exemple de déclaration d'une méthode

```
public static int somme (int a, int b) {  
    return a + b;  
}
```

Exemple d'appel d'une méthode

```
System.out.println(somme(2,3));  
// affiche 5
```

Signature de la méthode

```
public static int somme (int a, int b)
```

Les méthodes statiques

Déclaration de méthode avec un nombre indéterminé de paramètres (**varargs**)

```
public static int somme(int... tab) {  
    int res = 0;  
    for (int elt : tab)  
        res += elt;  
    return res;  
}
```

Exemple d'appel

```
System.out.println(somme(2));  
// affiche 2  
System.out.println(somme(2, 3));  
// affiche 5  
System.out.println(somme(2, 3, 5));  
// affiche 10  
System.out.println(somme(2, 3, 5, 9));  
// affiche 19
```

Remarques

- varargs est disponible depuis Java 5.
- On ne peut avoir qu'un seul varargs par méthode
- Le paramètre varargs doit être le dernier dans la liste de paramètres d'une méthode

Le passage de paramètre

```
Public class Test{  
public static void main(String[] args)  
{  
    int entier=5;  
    System.out.println("La valeur avant : "+entier);  
    m1(entier);  
    System.out.println("La valeur après : "+entier);  
}  
public static void m1(int i) {  
    i++;  
}  
}
```

Le passage de paramètre

```
public class Test{
public static void main(String[] args)
{
    String s = "Bonjour ";
    System.out.println(s);
    concat(s, " SMI");
    System.out.println(s);

}
public static void concat(String s, String s2)
{
    s +=s2;
}
}
```

Le passage de paramètre

- En java, **tous les paramètres sont passés par valeur.**
- Lorsque une méthode est appelée de nouvelles variables locales sont créées et toute modification ne concerne que ces dernières.
- Certains programmeurs pensent à tort que les variables de types primitifs sont passées par valeur alors que les objets sont passés par référence.

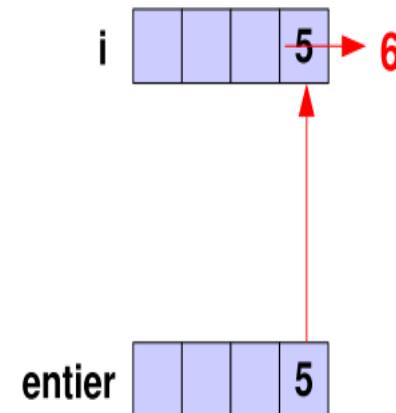
Le passage de paramètre

- Lors des appels de méthode, les **arguments** sont toujours **passés par valeur**
- Dans le cas **des types primitifs**, c'est la valeur de l'argument qui est recopiée dans le paramètre de la méthode.
 - Les modifications sur le paramètre de la méthode sont sans effet sur l'argument
- Dans le cas **des types « objet »**, c'est la valeur de la variable, (**la référence** à l'objet) qui est transmise à la méthode.
 - Les modifications effectuées en suivant cette référence (e.g. modification d'un champ de l'objet) sont répercutés dans la mémoire et sont donc visibles sur l'argument
 - En revanche, la modification de la référence elle-même est sans effet sur l'argument (c'en est une copie)

Le passage de paramètre

- Lors des appels de méthode, les **arguments** sont toujours **passés par valeur**
- Dans le cas **des types primitifs**, c'est la valeur de l'argument qui est recopiée dans le paramètre de la méthode.
 - Les modifications sur le paramètre de la méthode sont sans effet sur l'argument

```
public static void m1(int i) {  
    i++;  
}  
  
public static void main(String[] args) {  
    int entier = 5;  
    m1(entier);  
    System.out.println(entier); // 5  
}
```



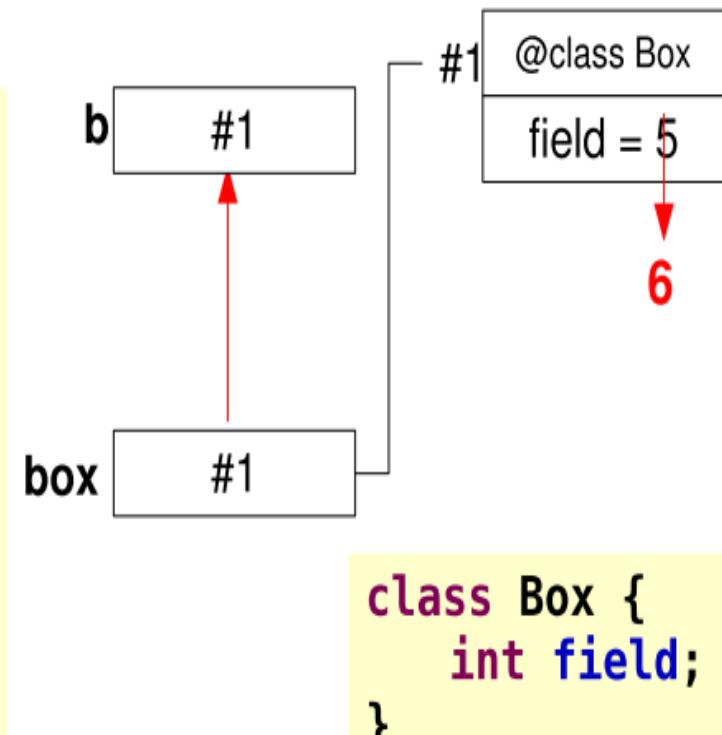
Le passage de paramètre

- Dans le cas **des types « objet »**, c'est la valeur de la variable, (**la référence** à l'objet) qui est transmise à la méthode.

- Les modifications effectuées en suivant cette référence (e.g. modification d'un champ de l'objet) sont répercutés dans la mémoire et sont donc visibles sur l'argument

```
public static void m2(Box b) {
    b.field++;
}

public static void main(String[] args) {
    Box box = new Box();
    box.field = 5;
    m2(box);
    System.out.println(box.field); // 6
}
```



Le passage de paramètre

- Dans le cas **des types « objet »**, c'est la valeur de la variable, (**la référence** à l'objet) qui est transmise à la méthode.

- En revanche, **la modification de la référence elle-même est sans effet sur l'argument** (cela en est fait qu'une copie)

```
public static void m3(Box b) {
    Box tmp = new Box();
    tmp.field = b.field+1;
    b = tmp;
}
public static void main(String[] args) {
    Box box = new Box();
    box.field = 5;
    m3(box);
    System.out.println(box.field); // 5
}
```

