



# Programmation Orientée Objet Java

---



Enseignant : Ramzi Mahmoudi

**mahmoudr@esiee.fr**

Année Universitaire: 2012/2013

# Plan des chapitres

---

1. Introduction
2. Les bases du langage
3. La Programmation Objet avec Java
4. Des concepts avancés de la programmation orientée objet Java
5. Les classes de base

# Chapitre I

## Introduction

---

Le développement d'applications  
aujourd'hui

# Développement de logiciels

---

- Une application peut comprendre des milliers de lignes de code
  - Systèmes bancaires ou systèmes de contrôle (l'électricité, les téléphones, l'aéroport),
  - Systèmes de simulation ou de modélisation (les domaines de la biologie, de l'économie)
- 👉 Un système implique alors la participation de plusieurs programmeurs

# Programmation proche de la machine

---

- Nécessité de faciliter cette analyse et surtout sa transcription la plus directe possible en un programme
  - L'algorithme est une solution **conceptuelle**, le programme est la solution technique mettant en œuvre l'algorithme.
- Au début de l'informatique, les langages étaient proches de la machine
  - une solution conceptuelle était difficile à mettre en œuvre sans de bonnes connaissances techniques.

# La « Crise du Logiciel » (1960, 1970)

---

- Coût de production de logiciel qui marche est trop élevé
  - Idée: Diviser pour régner

Il faut structurer les systèmes en composants pour faciliter la coopération des programmeurs, favoriser les tests, la réutilisation des codes, ...

# Programmation orientée traitement

---

- Principe : « diviser pour régner » :
  - on conçoit un **ensemble de procédures** pour résoudre le problème
  - **on décide d'abord de la manière dont on va manipuler les données puis on conçoit les structures de données pour faciliter cette manipulation.**
  - une procédure est un regroupement d'instructions dans un bloc que l'on pourra appeler par son nom

```
void affichage(int i)
{
int j;
Ecrire ( "entrez une valeur pour j"); Lire (j);
Ecrire ("la variable passée en paramètre vaut : " ,i);
Ecrire ("la variable déclaré dans la procédure vaut: ", j);
}
```

# Programmation orientée traitement

---

- Limitation de cette approche (exemple) :
  - **Un changement dans la structure des données peut entraîner de profondes modifications dans l'organisation des procédures.**
  - il y a la difficulté de :
    - Chercher les fonctions concernées par ce changement
    - Pour les fonctions concernées :
      - Ajout ou suppression d'arguments et de variables locales
      - Ajout ou modification ou suppression d'instructions

# Développement de logiciels

---

## ■ Problème

- Développer des logiciels de **qualité** à un coût acceptable

## ■ Solution

- Méthode de construction de logiciel *modulaire*



## Conception et programmation par objets:

- méthodologie de conception de logiciel qui répond bien aux exigences de qualité

# Les langages Objets

---

- *Langages objets : 1969*
  - *Simula*
  - *Smalltalk*
  - *C → Objective C, C++*
  - *Pascal : Pascal Objet, Delphi (Borland)*
  - *Basic : VBA (Microsoft)*
  - ***Java***
  - *C# (Microsoft)*
  - *Python (script)*

I.1.

# Paradigmes de la programmation par objets

---

SmallTalk, Eiffel, C++, Java, C#, ...

# Développement...

---

- Approche procédurale :  
"Que doit faire mon programme ?"

## Exemple: Gestion d'une bibliothèque



Germinal  
E. Zola



Le Monde



Le seigneur des anneaux  
J.R.R. Tolkien



Michel Martin  
Bibliothécaire



Alice Dupont  
Directrice



Arsène Deschamps  
Lecteur



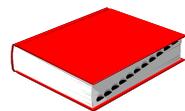
Anne Durand  
Lectrice

# Développement...

- Approche procédurale :  
"Que doit faire mon programme ?"

Gérer les emprunts  
de livres

## Exemple: Gestion d'une bibliothèque



Germinal  
E. Zola



Le Monde



Le seigneur des anneaux  
J.R.R. Tolkien



Michel Martin  
Bibliothécaire



Alice Dupont  
Directrice



Arsène Deschamps  
Lecteur



Anne Durand  
Lectrice

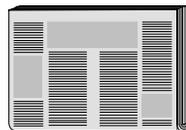
# Objet

---

- Approche procédurale :  
"Que doit faire mon programme ?"
- Approche orientée-objet :  
"De quoi doit être composé mon programme ?"
- Cette composition est conséquence d'un choix de modélisation fait pendant la conception



Germinal  
E. Zola



Le Monde



Le seigneur des anneaux  
J.R.R. Tolkien



Michel Martin  
Bibliothécaire



Alice Dupont  
Directrice



Arsène Deschamps  
Lecteur



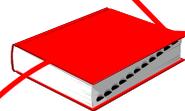
Anne Durand  
Lectrice

# Classe

Des objets similaires peuvent être informatiquement décrits par une même abstraction : une **classe**

- même **structure de données** et **méthodes de traitement**
- valeurs différentes pour chaque objet

Classe **Livre**



Germinal  
E. Zola



Le seigneur des anneaux  
J.R.R. Tolkien

Classe **Journal**



Le Monde

Classe **Employé**



Michel Martin  
Bibliothécaire



Alice Dupont  
Directrice

Classe **Lecteur**



Arsène Deschamps  
Lecteur



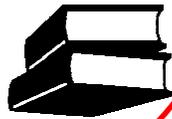
Anne Durand  
Lectrice

# Classe

Classe **Livre**  
-titre, auteur



Germinal  
E. Zola



Le seigneur des anneaux  
J.R.R. Tolkien

Classe **Journal**  
-nom, date



Le Monde

Classe **Employé**  
-nom, prénom, statut



Michel Martin  
Bibliothécaire



Alice Dupont  
Directrice

Classe **Lecteur**  
-nom, prénom

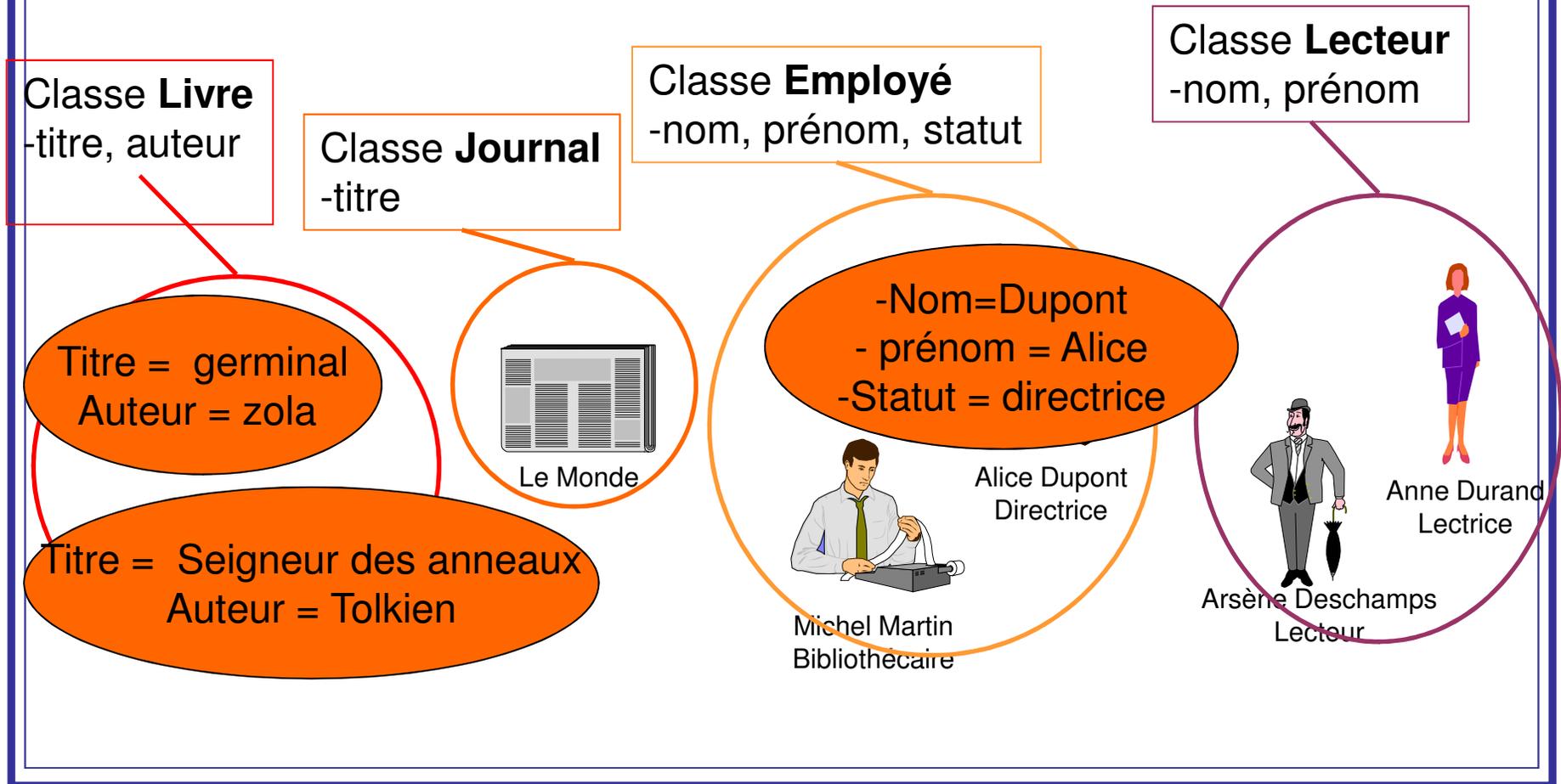


Arsène Deschamps  
Lecteur



Anne Durand  
Lectrice

# Classe



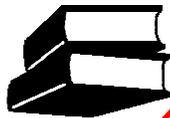
# Classe

Classe **Livre**  
-titre, auteur  
+ **Date** emprunter(Lecteur)

Classe **Journal**  
-titre



Germinal  
E. Zola



Le seigneur des anneaux  
J.R.R. Tolkien



Le Monde

Classe **Employé**  
-nom, prénom, statut



Michel Martin  
Bibliothécaire



Alice Dupont  
Directrice



Arsène Deschamps  
Lecteur



Anne Durand  
Lectrice

Classe **Lecteur**  
-nom, prénom  
+ **ListeDeLivres** livresEmpruntés()

# Classe

Classe **Livre**  
-titre, auteur  
+ **Date** emprunter(Lecteur)

Classe **Lecteur**  
-nom, prénom  
+ **ListeDeLivres** livresEmpruntés()

Classe **Journal**  
-titre

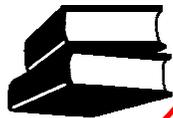
Classe **Employé**  
-nom, prénom, statut

Classe **ListeDeLivres**  
+ getLivre(numéro)  
+ nombres()

Classe **Date**  
-jour, mois, ...



Germinal  
E. Zola



Le seigneur des anneaux  
J.R.R. Tolkien



Le Monde



Michel Martin  
Bibliothécaire



Alice Dupont  
Directrice



Arsène Deschamps  
Lecteur



Anne Durand  
Lectrice

# Contenu d'une classe

---

Une classe est composée de plusieurs **membres** dont chacun est soit :

- un **attribut** : variable typée
- une **méthode** (ou **opération**) : ensemble d'instructions de traitement

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

**Attributs** {

**Méthodes** {

# Programmation orientée objet

---

- Orienté objet est une technique pour modéliser des systèmes réels
  - Comprendre un problème (ou un système) réel
    - complexe, beaucoup d'information, de détails
  - Modèle = vue abstraite du problème
    - passage du monde réel au monde informatique
    - retenir les propriétés essentielles:  
données + opérations

# Programmation orientée objet

---

- Un programme est défini en terme d'*objets*
  - Objet est une entité avec un *état* et un *comportement*
  - Plusieurs instances de la même classe (objets) avec chacune son état propre
  - Les objets communiquent entre eux à l'aide de messages

# Première application

---

- Créer un fichier texte : HelloWorld.java
- Règle de bonne pratique : 1 classe par fichier et 1 fichier par classe (même nom)
- Nom de la classe commence par Majuscule, nom de la fonction en minuscule

```
public class HelloWorld
{
    public static void main (String[]args)
    {
        System.out.println("Hello the world");
    }
}
```

La première ligne du programme doit être la déclaration de la classe

Tout programme doit contenir une méthode main qui porte la signature ci-contre

Écrire à l'écran "Hello the World"

Fermer les accolades

# Première application

---

- Le mot **class** veut dire que nous allons définir une nouvelle classe **Java**, suivi du nom de cette classe.
- En Java, les majuscules et les minuscules sont considérés comme des caractères différents.
  - Les caractères « { » et « } » marquent le début et la fin du **bloc d'instructions** à réaliser par la classe.
- Le mot **main** indique que cette méthode est la méthode principale de la classe.
- Un interpréteur Java a pour fonction d'exécuter les **instructions** de la méthode principale **main**, du programme qu'on lui soumet.

# Première application

---

- Une méthode peut prendre des paramètres de types précis et renvoie éventuellement une valeur de type tout aussi précis.
- le mot **void** signifie que la méthode **main** ne renvoie aucune valeur.
- **args[ ]** est le paramètre d'entrée de type **String** de la méthode **main**.
- les mots **public** et **static** décrivent chacun une caractéristique de la méthode (**public** : méthode visible, **static** : spécifique à la classe et non aux objets)
- **System.out.println** est une commande permettant d'afficher la chaîne de caractère « **Hello the World** » sur la sortie standard de la machine qui est l'écran

# Structure d'une application Java

- Parmi les classes définies dans le fichier **CCC.java** il ne peut y avoir qu'une seule classe publique et ayant le même nom **CCC** que le fichier.
- Elle contient généralement la méthode **main()**

Fichier «**CCC.java**»

Définition d'une classe **AAA**

Définition d'une classe **BBB**

⋮

Définition d'une classe **CCC**

Méthode **main ( )**

# Compilation et exécution

---

- Pour exécuter, dans une console DOS ou UNIX, si j'ai un fichier HelloWorld.java pour la classe HelloWorld :
  - **javac HelloWorld.java**
    - Compilation du code java
    - Indication des erreurs de syntaxe éventuelles
    - Génération d'un fichier HelloWorld.class si pas d'erreurs
  - **java HelloWorld**
    - Java est la machine virtuelle
    - Exécution du bytecode (*HelloWorld.class*)
    - Nécessité de la méthode main, qui est le point d'entrée dans le programme

I.2.

# Mécanisme de programmation par objets

---

Instanciation

Envoi de message

Héritage

# Instanciación

---

L'instanciación est l'opération qui consiste à créer un objet à partir d'une classe.

En Java, le mot-clé **new** provoque une instanciación en faisant appel à un **constructeur** de la classe instanciée.

- Un constructeur est une méthode qui a le **même nom** que la classe
- Un constructeur n'a **pas de valeur de retour**
- **Plusieurs constructeurs** peuvent exister dans une même classe (avec des arguments différents)
- Il faut **au moins un constructeur** dans une classe pour en instancier des objets

# Constructeur de la classe Livre

---

```
public class Livre {  
    private String titre;  
    private String auteur;  
    private Lecteur emprunteur;  
    public Livre(String t, String a) {  
        titre = t;  
        auteur = a;  
        emprunteur = null;  
    }  
    public void setTitre(String t) {  
        titre = t;}  
    public String getTitre() {  
        return titre;}  
  
    public Date emprunte(Lecteur lec) {  
        ...  
    }  
}
```

Constructeur

# Instanciación de la clase Livre

---

```
public class Bibliotheque {  
    public void enregistreLivre(String titre, String ecrivain) {  
        Livre nouveauLivre;  
        nouveauLivre = new Livre(titre,ecrivain);  
        ajouteEnBibliotheque(nouveauLivre);  
    }  
  
    public void ajouteEnBibliotheque(Livre li) {  
        ...  
    }  
}
```

# Invocation de méthodes

---



livresEmpruntés?

Un point . sépare le nom de la méthode de l'objet sur lequel elle est invoquée.

...

```
Lecteur lecteur1 = new Lecteur("Anne","Durand");  
Livre monLivre = new Livre("Germinal", "Zola");  
String titreDuLivre = monLivre.getTitre();
```

...

```
ListeLivres l1 = lecteur1.livresEmpruntés();  
for (int i =0; i < l1.nombre(); i++)  
    System.out.println(l1.getLivre(i).getTitre());
```



Anne Durand  
Lectrice



Arsène Deschamps  
Lecteur

# Exemple

```
public class Livre {
    private String titre;
    private Lecteur emprunteur;
    public boolean estEmprunte() {
        if (emprunteur == null) return
false;
        else return true;
    }
    public Date emprunte(Lecteur lec) {
        if ( this.estEmprunte() == true)
            return null;
        if ( lec.empruntPossible() ) {
            emprunteur = lec;
            lec.ajouteEmprunt( this );
            return new Date();
        } else return null;
    }
}
```

```
public class Lecteur {
    private Livre[ ] emprunts;
    private int nbEmprunts;
    ...
    public boolean empruntePossible() {
        if (nbEmprunts < 5) return true;
        else return false;
    }

    void ajouteEmprunt(Livre liv) {
        emprunts[nbEmprunts] = liv;
        nbEmprunts ++;
    }
}
```

# I.3.

## Java, plus précisément ?

---

Historique, objectifs,  
caractéristiques...

# Historique

---

- Développé par les laboratoires de recherche de Sun Microsystems au début des années 1990.
  - Développé comme une alternative au C/C++.
  - Langage principalement attribué à James Gosling
  - Devrait être petit et portable.
  - Ciblé pour du logiciel embarqué dans des produits de consommation (appareils électroniques intelligents).



# Historique

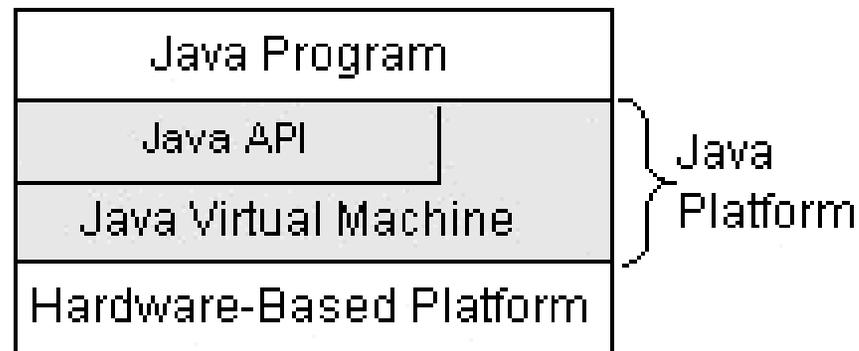
---

- Le nom Oak étant déjà commercialisé
  - Le nom de **JAVA** fut adopté lors d'un passage dans un café aux alentours.
  - **James Gosling, Arthur Van Hoff, and Andy Bechtolsheim**
  - **Just Another Vague Acronym**
- Java dévoilé en 1995
  - Netscape intègre l'exécution de Java dans son navigateur
  - Sun annonce un environnement de développement gratuit
  - Java fait l'objet d'un brevet ce qui permet à SUN de le contrôler
  - 1995 – 1996
    - Grand concours de rédaction d'applettes qui popularisent JAVA

# C'est quoi Java ?

---

- Java est un langage de programmation
  - Un programme Java est compilé et interprété
- Java est une plateforme
  - La « Java Platform » est constituée de :
    - La « Java Virtual Machine » (JVM)
    - Des interfaces de programmation d'application (Java API)



# L'API Java

---

- L'environnement Java propose une **API (Application Programmer's Interface)**
- L'API Java est structuré en libraires (packages).
  - Package : regroupement de classes ayant un lien logique entre elles
  - pour les utiliser dans d'autres classes
  - pour les « spécialiser »
- Pour programmer **efficacement**, une bonne connaissance de ces packages est **indispensable**
- Ne pas refaire ce qui a déjà été fait (d'ailleurs surement en mieux)

## API Java : quelques classes standards

---

- **java.lang** : classes essentielles objet, types de base, processus
- **java.util** : structures de données (collections) listes, ensembles, hashtables, arbres, itérateurs
- **java.awt** : interface graphique (Abstract Window Toolkit) fenêtres, boutons, événements...
- **java.io** : entrées / sorties flot de données provenant de fichier ou buffer
- **java.net** : réseau URL, sockets
- **java.rmi** : objets distribués (Remote Method Invocation)
- **java.sql** : JDBC (Java Data Base Connectivity) connexion à une base de données relationnelle envoi de requêtes SQL, récupération des résultats

# La JVM Java

---

- Définit les spécifications hardware de la plateforme
- Lit le bytecode compilé (indépendant de la plateforme)
- Implémentée en software ou hardware
- La JVM définit :
  - Les instructions de la CPU
  - Les différents registres
  - Le format des fichiers .class
  - La pile des instructions
  - L'espace mémoire
- Trois tâches principales :
  - Charger le code (class loader)
  - Vérifier le code (bytecode verifier)
  - Exécuter le code (runtime interpreter)

# La JVM Java

---

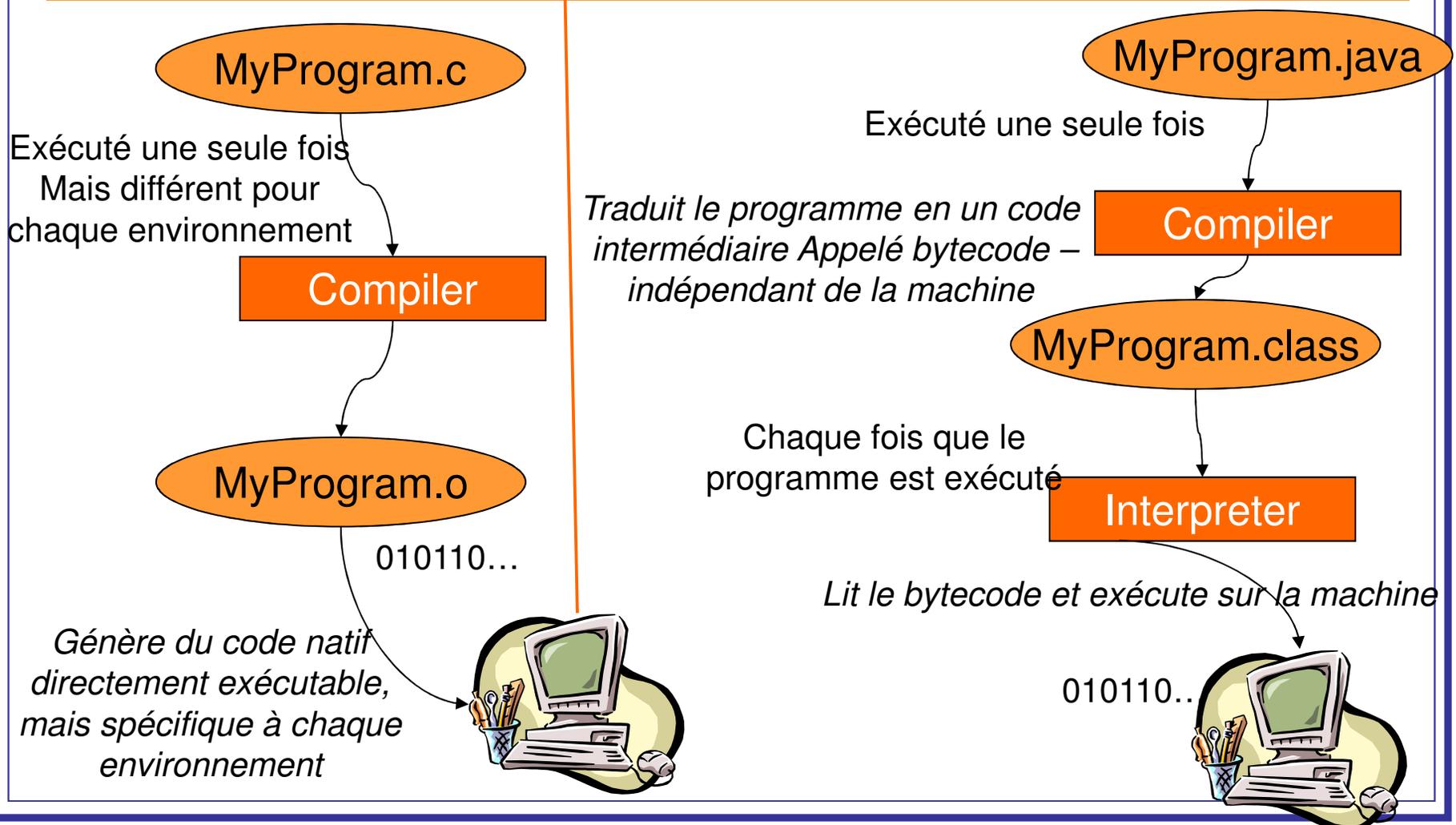
## ■ Avantages

- Développement plus rapide
  - **courte étape de compilation pour obtenir le byte-code,**
    - **pas d'édition de liens,**
    - **débogage plus aisé,**
  - **Le byte-code est plus compact que les exécutable**s pour voyager sur les réseaux.

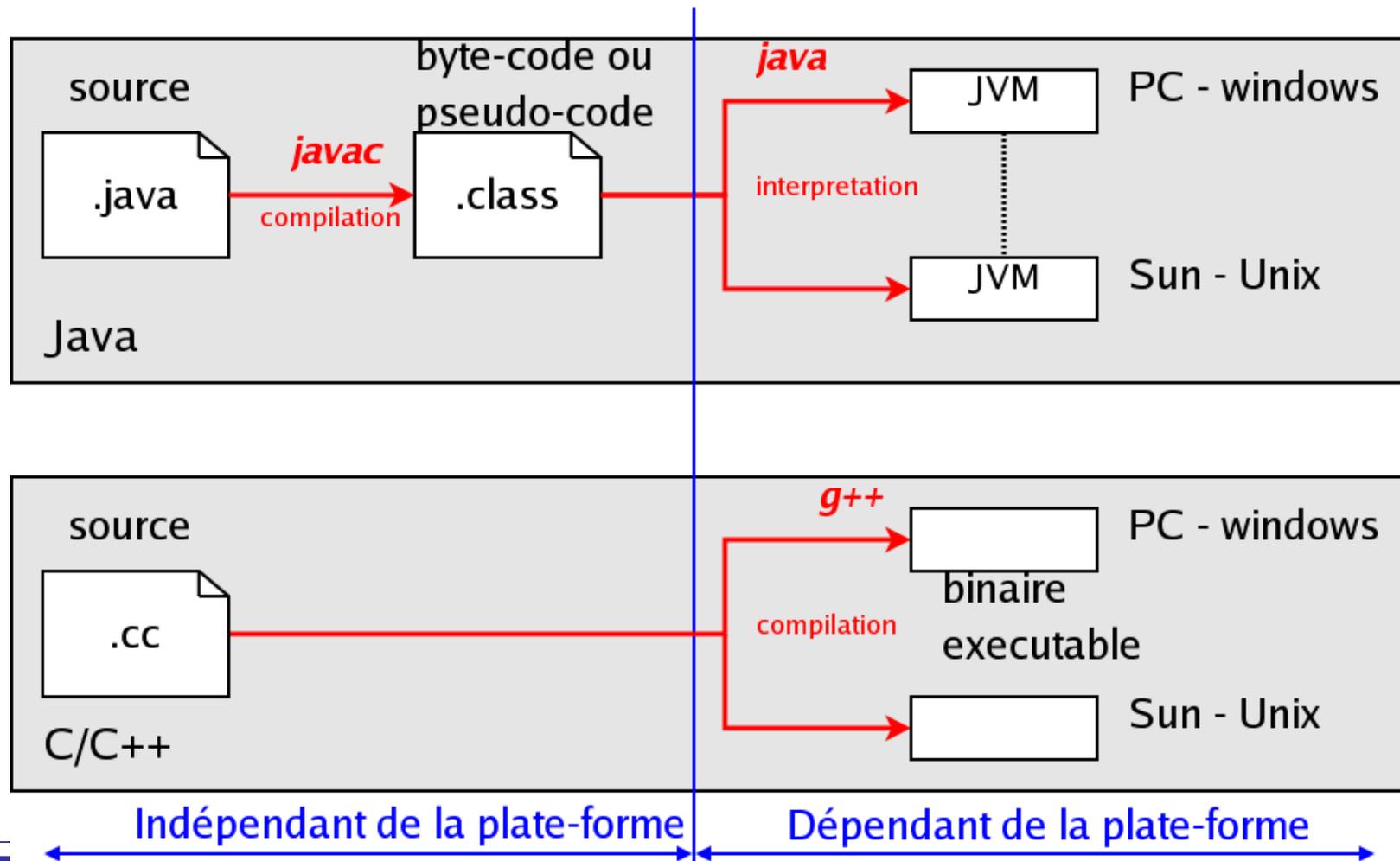
## ■ Inconvénients :

- **L'interprétation du code ralentit l'exécution** de l'ordre de quelques dizaines de fois plus lent que C++
- Java est gourmand en mémoire

# Portabilité



# Portabilité



# Caractéristiques

---

- Totalemment portable : génère des octets de code après la compilation (Bytecode)
- C'est un langage interprété.
- Indépendant de l'architecture: JVM (Java Virtual Machine)
- Totalemment orienté objet.
- Ramasse-miettes automatique pour la gestion de la mémoire (pas besoin de delete pour détruire objets).
- Bibliothèques complètes(+/-) et normalisées.
- Facilités de communication réseau incluses.
- Gestion propre des exceptions.

# Caractéristiques

---

- Le langage Java est interprété :
  - Un programme Java n'est pas compilé en code machine ;
    - Il sera compilé en code intermédiaire interprété nommé **ByteCode**.
    - Lors de l'exécution le **ByteCode** sera interprété à l'aide d'une machine dite virtuelle **JVM** (**Java Virtual Machine**).
- Le langage Java est portable et indépendant des plates-formes :
  - Le code intermédiaire produit « **ByteCode** » est indépendant des plates-formes.
    - Il pourra être exécuté sur tous types de machines et systèmes à condition qu'ils possèdent l'interpréteur de code Java « **JVM** ».

# Caractéristiques

---

- Simple et familier
  - Apprentissage facile (syntaxe du langage C)
  - Développeurs opérationnels rapidement (syntaxe du langage C)
  - Suppression des concepts sources de bugs
    - pointeurs
    - include (fichier en-tête)
- Orienté Objet
  - Java ne permet d'utiliser que des objets (hors les types de base)
- Fiable et robuste (en mémoire et en exécution)
  - Sources d'erreurs limitées
    - typage fort (Ada),
    - pas de manipulations de pointeurs, etc. (qui est le cas du C++),
  - Vérifications faites par le compilateur facilitant une plus grande rigueur du code

# Caractéristiques

---

- Le langage Java est distribué :
  - Supporte des applications réseaux (protocoles de communication [java.net](#))
    - [URL](#) : permet l'accès à des objets distants
    - [RMI](#) : Remote Method Invocation
    - Programmation d'applications Client/Serveur : classe [Socket](#)
    - Manipulation de fichier local ou fichier distant identique : indifférence à la localisation
- Le langage Java est multithread :
  - JAVA permet l'exécution simultanée de plusieurs processus léger ([thread](#))
    - Classe [java.lang.thread](#) avec les méthodes permettant de :
      - [Démarrer](#), [Exécuter](#), [Stopper](#) ces processus.
    - contrôler les synchronisations et l'état cohérent des données.

# Versions de Java

---

**Java 1.0.2** (1996) : **le JDK : Java Development Kit** , 212 classes, 8 paquetages  
version minimale des browsers web.

**Java 1.1.5** (1997) : 504 classes, 23 paquetages  
amélioration interface utilisateur (AWT), gestion des erreurs, cohérence du langage

**Java 1.2** (Java 2 en 1998) : 1520 classes, 59 paquetages  
Swing, Drag and drop, amélioration audio

**Java 1.3** (2001)  
J2SDK (nouveau nom) pour J2EE (Entreprise, serveur), J2ME (PDA)  
amélioration de la gestion mémoire, rapidité

**Java 1.4** (2002) : 2757 classes, 135 paquetages  
J2SDK → J2SE (nouveau nom),  
XML, expressions régulières, nouvelles E/S, accès à la mémoire vidéo (VolatileImage)  
pour l'accélération

etc...

# D'un point de vue technique

---

- **Les principaux outils de base de la JDK**
  - `javac` représente le compilateur de java: un programme écrit en java est transformé en bytecode.
  - `java` est l'interpréteur de java : c'est la commande à utiliser pour exécuter un programme java
  - `jdb` est le débogueur de java
  - `javap` permet de déassembler un fichier compilé
  - `javadoc` est un générateur de documentation. Il permet de générer de la documentation sur les programmes écrits en java.
- Documentation
  - en ligne : <http://java.sun.com/docs/>

# L'environnement de développement

---

## ■ Le compilateur

- Le compilateur `javac` permet de compiler un programme java (i.e un code source) en bytecodes java.
- La commande à utiliser pour compiler un programme est  
`javac [options] ClassName.java`
- A l'issue de cette commande, le compilateur `javac` génère un fichier `ClassName.class` afin qu'il soit ensuite interprété par la JVM (Java Virtual Machine)

## ■ L'interpréteur

- L'interpréteur `java` permet d'exécuter une application écrite en langage java (autre qu'une applet), plus spécifiquement un fichier `ClassName.class` (i.e le java bytecodes).
- Par le biais de l'interpréteur java, on peut faire passer des arguments à la fonction main
- La commande à utiliser pour exécuter un programme est  
`java [options] Classname <args>`

# L'environnement de développement

---

## ■ Le débogueur

- Le débogueur `jdb` permet de déboguer "en ligne » un programme (une classe)
- Il n'est pas facile à utiliser
- Pour pouvoir déboguer un programme (i.e une classe) il faut compiler la classe avec l'option `-g`
- La commande à exécuter pour l'utiliser est `jdb Classname`
- Il existe une aide pour le débogueur. Pour accéder à cette aide, il faut taper `help` ou `?`

## ■ Le générateur

- Le générateur de documentation `javadoc` permet de générer des documents HTML équivalents aux documents Java de SUN (i.e ayant la même structure)
- La commande à exécuter pour l'utiliser est `javadoc Classname`

# IDE (Integrated Development Environment)

---

## La JDK

- On peut télécharger la JDK à partir du site de Sun : [java.sun.com](http://java.sun.com)

## Autres outils de développement

Il existe plusieurs outils de développement dont la plupart sont payants.  
Néanmoins voici quelques uns gratuits :

- *Jedit* <http://www.jedit.org>
- *NetBeans* : <http://www.sun.com>
- *Eclipse* <http://www.eclipse.org>
- *Emacs (avec JDEE)*
  - <http://www.gnu.org/software/emacs/>
  - <http://jdee.sunsite.dk/>
- *JBuilder* : Windows, payant mais il existe des versions d'essai
- *Java Workshop*  
Windows/Unix, gratuit

# Avantages

---

- Ecrire une fois, exécuter partout !
- Sécurité.
- Exécution dans un navigateur Web (Applet).
- Gestion automatique de la mémoire (**garbage collector**)
- Programmation modulaire et dynamique.
- Lisibilité du code.
- Code compact (beaucoup est dans la JVM).
- **L'API.**
- Réseaux, interfaces, son, pont avec les bases de données en natif

# Inconvénients

---

- Interprété : lenteur
- Nécessite une JVM pour fonctionner.
- La gestion de la mémoire est inefficace dans la plupart des cas.
- Difficulté face aux applications gourmandes en mémoire.
- Moins de mécanismes objet que C++ pourtant plus ancien (héritage multiple)