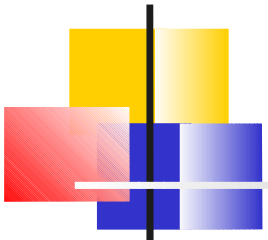


Java

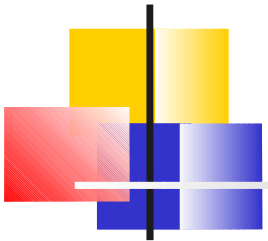
- Définitions
- Historique
- Caractéristiques
- Les bases du langage
- Structure des programmes

« Ecrire une fois, exécuter partout »



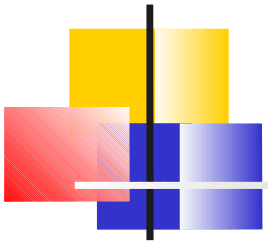
Qu'est ce que JAVA ?

- Trois composantes qu'il faut distinguer :
 - Un langage de programmation
 - Utilisé pour écrire les programmes et les applets
 - Une machine virtuelle
 - La MV exécute le byte code résultat de la compilation
 - Le byte code est un « langage machine » indépendant du CPU
 - Une plate-forme ou noyau des API
 - Correspond a un ensemble de classes prédéfinies qui existent sur chaque installation Java
 - Cette plate forme peut être étendue au moyen d'extension standard facultative



Un langage de Programmation

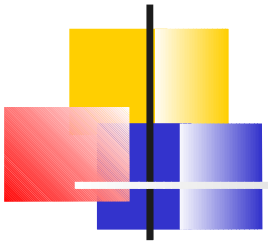
- Orienté objet
- Syntaxe similaire à C
- Débarrassé des « nuisances » et des complexités C++
- Simple et facilement accepté
 - Agréable à utiliser
 - Facilite la tâche des programmeurs pour écrire des applications robustes et sans bug.



Une Machine Virtuelle

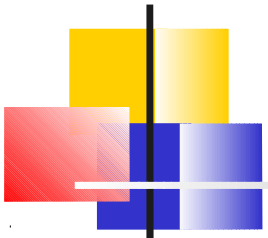
■ La Compilation

- Un compilateur est un programme qui va simplement prendre un fichier **.java** (format texte) et générer un fichier **.class** (binaire) qui contiendra le byte-code.
- Le byte-code est un langage machine « indépendant du CPU et de l'OS »
- Le compilateur (**javac**) va vérifier :
 - La syntaxe du programme.
 - Certaines erreurs algorithmiques (instructions jamais exécutées ...).
 - La présence et la cohérence des bibliothèques utilisées.



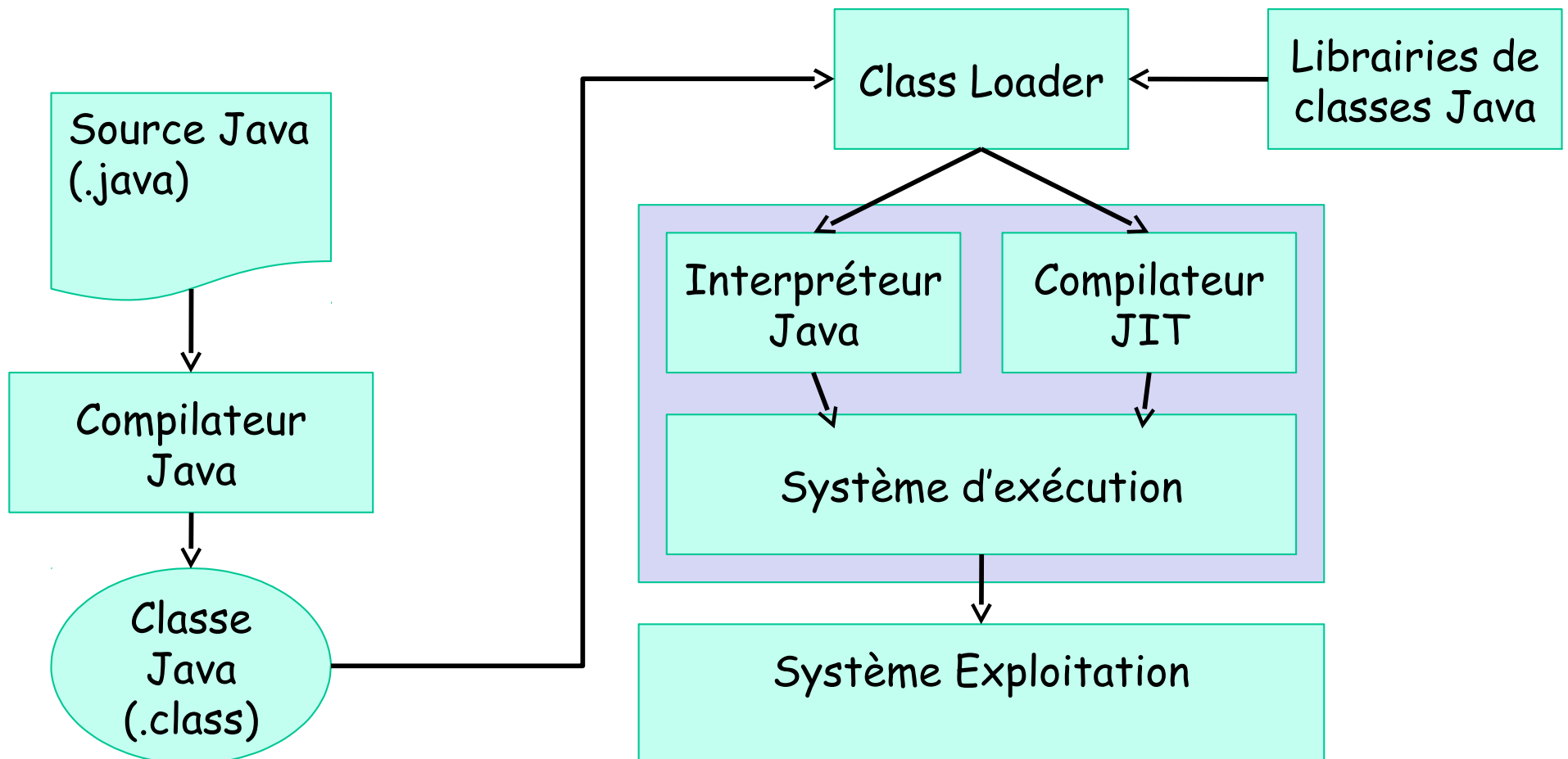
Une Machine Virtuelle

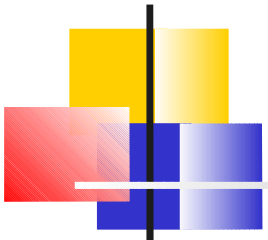
- La machine virtuelle est un interpréteur de Byte Code
 - sur toute machine sur laquelle une machine virtuelle existe (Windows - Unix - Linux - MacOS - Windows CE - Android).
 - Efficace, contrairement à d'autres interpréteurs car elle utilise :
 - Une technologie JIT (*Just-In-Time*) : compilation en code machine en temps réel.
 - Une technologie HotSpot : compilation des points chauds en code natif.
- Il est possible de ne pas utiliser la machine virtuelle :
 - Il existe des compilateurs natifs (performances aussi bonnes que des programmes C/C++).
 - Il existe des outils de transformations : traduction du langage Java vers d'autres langages (le code généré n'est pas optimisé).



Environnement de Compilation

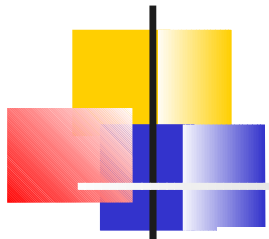
Environnement d'Exécution



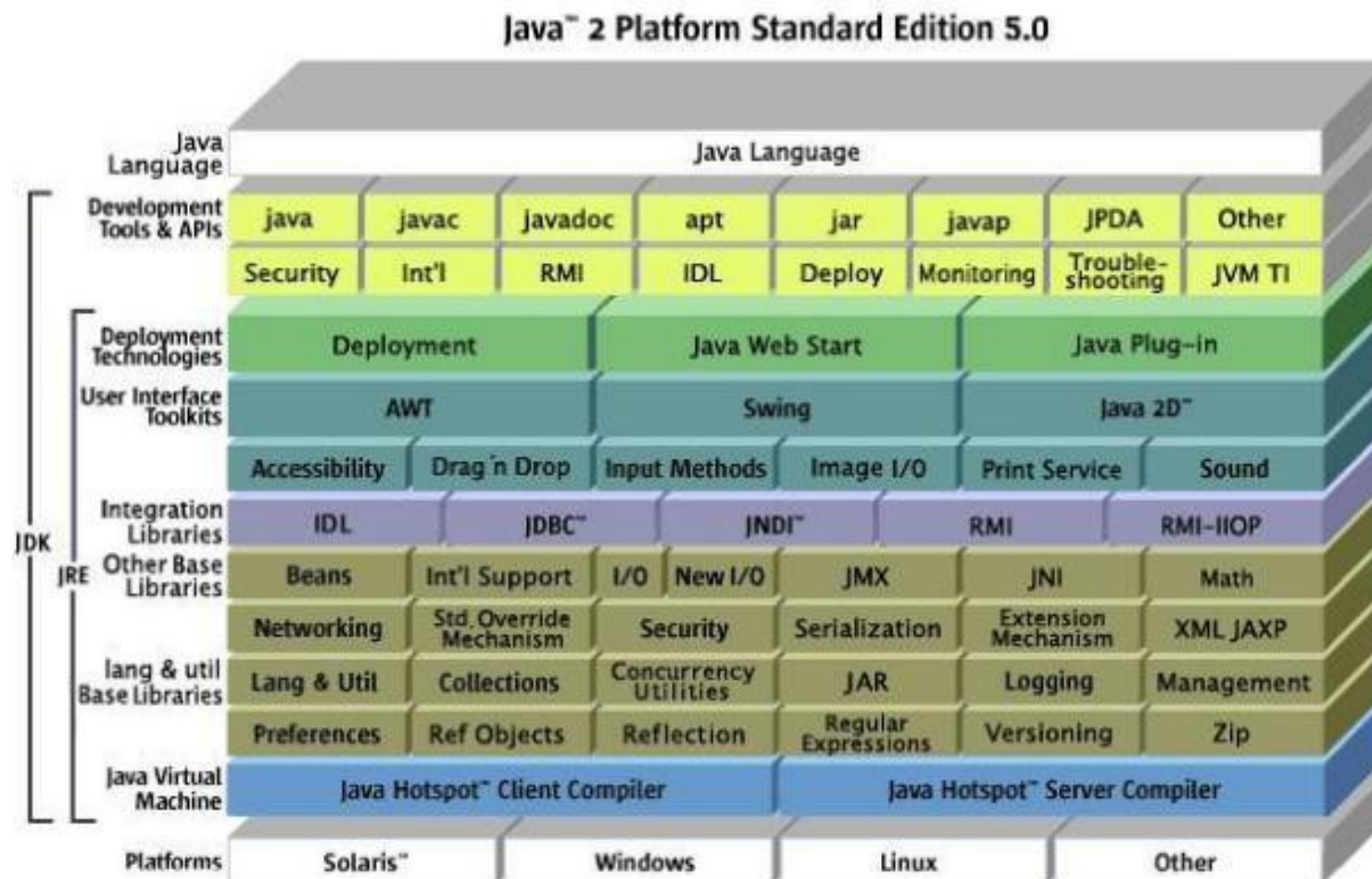


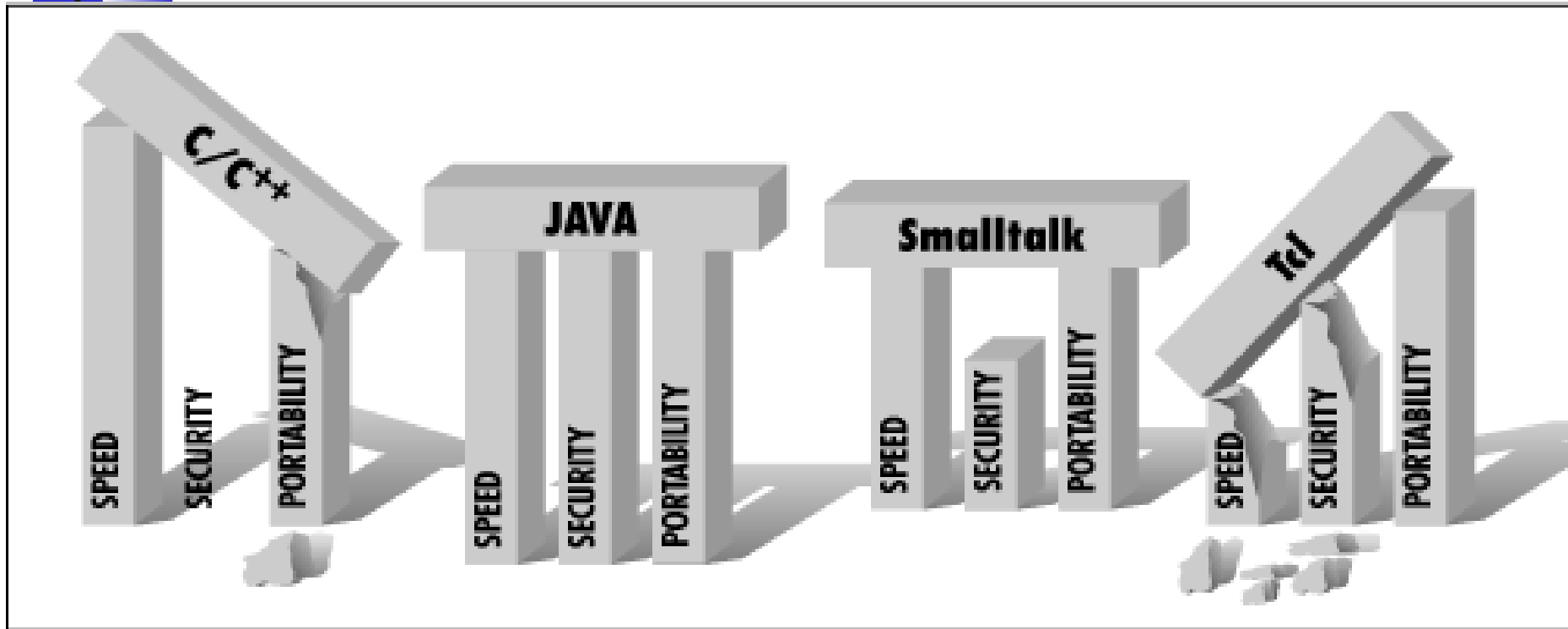
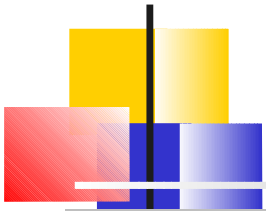
Une Plate-forme

- Tous les programmes écrits en Java dépendent
 - de l'ensemble des classes prédéfinies qui forment la plate-forme Java.
- Organisées en packages (paquetages) selon des thèmes/fonctionnalités
 - Entrées/Sorties (java.io)
 - Réseaux (java.net)
 - Interfaces Utilisateur (java.awt, javax.swing)
 - Accès aux bases de données (java.sql)
 - Invocation de méthodes d'objets distants (java.rmi)
 - Fonctions mathématiques (java.math)

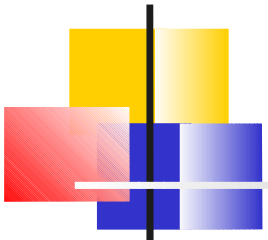


Une Plate-forme





« **Exploring Java** » by Patrick Niemeyer & Joshua Peck; 1-56592-184-271-9, 500 pages (est.) 2nd Edition July 1997 (est.)



A retenir

- SUN Microsystem est à l'origine de Java (aujourd'hui racheté par Oracle)

- La devise de SUN pour Java est
« Ecrire une fois, Exécuter partout »

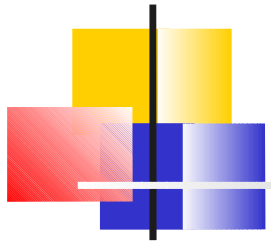
- Un danger pour Microsoft ?

- Peut-être,

La plate-forme Java **réduit la dépendance** des programmeurs vis à vis du SE, et

augmente la liberté

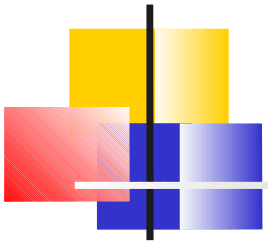
de choix de SE des utilisateurs finaux



Java

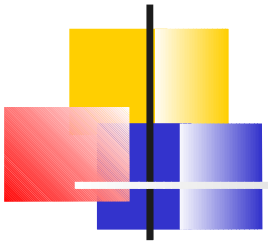
- Définitions
- Historique
- Caractéristiques
- Les bases du langage
- Structure des programmes

« Ecrire une fois, exécuter partout »



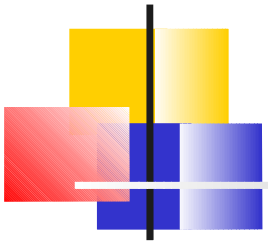
Les Grandes Dates

- Créé par Sun Microsystems en 1991 :
 - projet Green, équipe dirigée par Patrick Naughton et James Gosling pour concevoir un langage informatique limité, adapté aux appareils domestiques (commutateurs de câbles TV). Utilisation de C++ comme point de départ.
- En Mai 1995, présentation du navigateur HotJava.
- Fin 1995, Netscape rend son navigateur Netscape 2.0 compatible avec Java. Les principaux éditeurs (IBM, Symantec, Inprise, Microsoft) acquièrent la licence pour utiliser Java dans leurs produits.
- La première version (Java 1.02) est publiée en 1996.



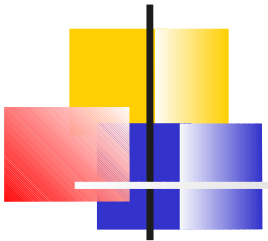
Les Grandes Dates

- La version 1.1 (1997) ajoute une gestion objet des évènements, l'internationalisation et les Java Beans.
- La version 1.2 (fin 1998) offre de nombreuses améliorations : Swing pour créer des interfaces graphiques, amélioration de la JVM.
 - A partir de la version 1.2, Sun désigne les versions sous le nom **Java 2**
- La version 1.4 apporte correction de bugs, amélioration de la JVM, ajout des assertions, amélioration de JDBC, etc..
- La version 1.5 (Java 5) a apportée beaucoup de nouveautés, un cours y sera consacré.
- La dernière version est la version 1.7



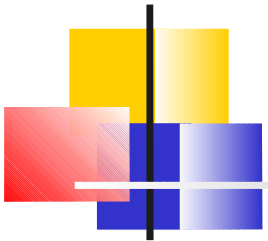
Bibliographie

- **Au coeur de Java 2, Vol. 1,**
 - Cay S. Horstmann & Gary Cornell, Sun Microsystems (trad. française chez CampusPress).
- **Java in a Nutshell,**
 - David Flanagan, éd. O'Reilly & Associates.
- **Thinking in Java,**
 - Bruce Eckel, éd. Prentice Hall, version PDF en ligne
- **La documentation de l'API,**
 - site web de Sun, <http://www.java.sun.com>.
- **Les groupes Usenet**
 - `comp.lang.java.*` et `fr.comp.lang.java` (notamment la FAQ).



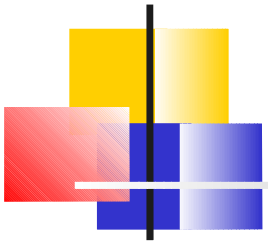
Java 1.0

- Première version publique de JAVA
 - 212 classes réparties en 8 paquetages dont :
 - `Java.lang` : Les types de base
 - `java.util` : les structures de données évoluées (collections)
 - `java.applet`,
 - `java.awt` : Interface graphique
 - `java.net` : Le réseau (socket, TCP, UDP)
 - `java.io` : Les Entrées /Sorties par flux
- Elle est toujours utilisée pour écrire des applets simples.



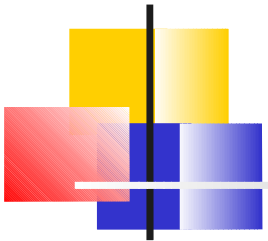
Java 1.1 (1997)

- De nouveaux « packages » dont
 - `java.lang.reflect` : introspection sur les classes et les objets
 - `java.beans` : composants logiciels
 - `java.sql (JDBC)` : accès homogène aux bases de données
 - `java.security` : signature, cryptographie, authentification
 - `java.serialisation` : sérialisation d'objets
 - `java.rmi` : Remote Method Invocation
 - `java.idl` : interopérabilité avec CORBA



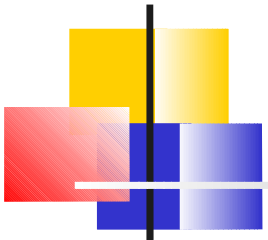
Java 1.1 (1997)

- La taille de la plate-forme double
 - ~500 classes (1.0 = 212)
 - 23 packages (1.0 = 8)
- Introduction des classes internes
- Modification significatives des API de la version précédente, par exemple IO et AWT
- Amélioration importante des performances de la MV
- Version aujourd'hui dépassée



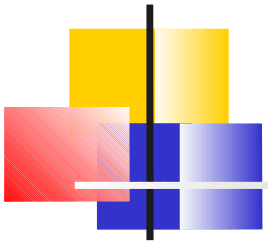
A partir de Java 1.2 => 2 (1998)

- La première révolution Java
- La plate-forme à triplé
 - ~1500 classes dans 59 packages
- En raison de l'importance des changements l'appellation devient « Plate-forme Java 2 »
- La plate-forme se décline en trois éditions :
 - J2SE : Java 2 Standard Edition
 - J2EE : Java 2 Enterprise Edition
 - J2ME : Java 2 Micro Edition



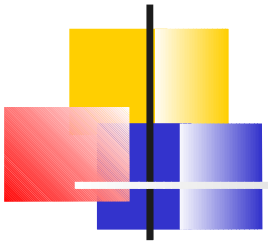
Version 1.4 (2002)

- Un nouveau gestionnaire d'entrée/sortie (NIO: New Input/Output).
- De nouvelles fonctionnalités dans JFC (gestion de la molette de la souris, mode graphique plein écran, drag&drop).
- Intégration de l'API pour XML (DOM, SAX, XSLT).
- Le support de l'IP V.6.
- Un outil de déploiement d'application clients en Java : Java WebStart.



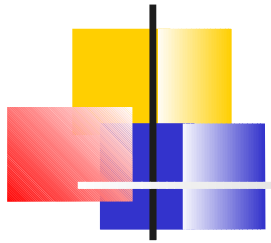
Version 1.5 ou 5 (2004)

- Nombreuses améliorations dans le langage :
 - types génériques
 - simplification des boucles for
 - autoboxing/unboxing
 - imports statiques
 - etc.
- La plate-forme, dans son édition standard, se décline en :
 - J2SE Development KIT (JDK 5.0)
 - J2SE Runtime Environment 5.0 (JRE 5.0)



Version 1.6 ou 6 (2006)

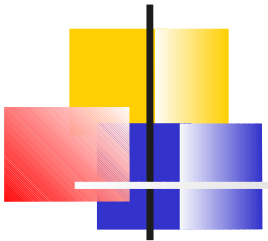
- Evolution mineure.
- Intégration de la classe SwingWorker.
- Splash Screen.
- Ajout du Look and feel GTK.
- Support des fichiers images PNG.



Java

- Définitions
- Historique
- **Caractéristiques**
- Les bases du langage
- Structure des programmes

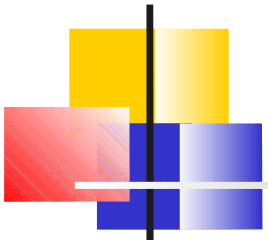
« Ecrire une fois, exécuter partout »



Les points forts de Java

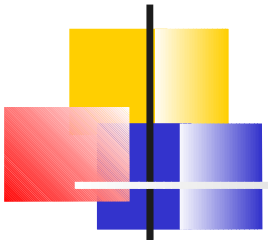
- Selon « The Java Language Environment White paper », Java doit être :
 - simple et familier,
 - orienté objet,
 - orienté réseaux,
 - robuste,
 - sécurisé,
 - d'architecture neutre,
 - portable,
 - interprété,
 - de performance élevée,
 - parallèle (multi-threadé).

(<http://java.sun.com/docs/white/langenv/>)



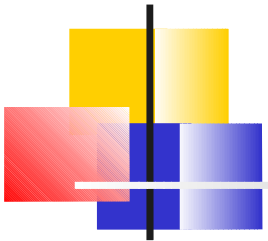
Java est un langage simple et familier

- Familier par sa parenté avec C :
 - structures de contrôle ;
 - types primitifs.
- Simple par rapport à C++ :
 - pas de fichier d'entête, pas de préprocesseur ;
 - pas d'arithmétique des pointeurs (ni de syntaxe) ;
 - pas de surcharge des opérateurs ;
 - pas de conversions de types sans contrôles ;
 - pas de sémantique de la valeur pour les objets ;
 - gestion automatique de la mémoire.



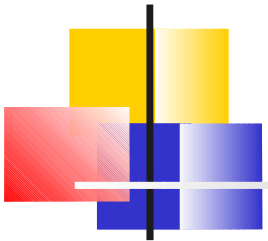
Java est un langage à objets

- Concepts-clés :
 - Modularité (notion de classe)
 - Extensibilité (relation d'héritage)
- Technique :
 - Classes, interfaces, paquetages.
 - Tout est classe (classes enveloppes pour les types primitifs).
 - Les objets sont accessibles par des références (pointeurs).
 - Héritage simple des classes.
 - Héritage multiple d'interfaces.
 - Polymorphisme et liaison différée.
 - Bibliothèque riche et « intégrée ».



Java est un langage distribué (réseau)

- But :
 - Exploiter simplement les ressources Internet.
- Moyens :
 - Module réseau (manipulation des sockets, protocoles usuels)
 - Code mobile :
 - Applets.
 - Servlets.
 - Appel à distance de méthodes :
 - Java pur : Module RMI (Remote Method Invocation).
 - Interopérabilité : Module Corba.



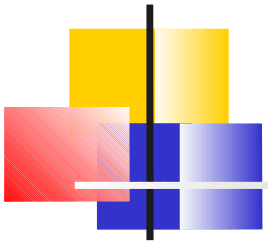
Java est un langage robuste

- But :

- éliminer les risques d'erreur (contexte de logiciel embarqué ou mobile).

- Moyens :

- Mécanismes sophistiqués de gestion des erreurs :
 - Mécanisme de typage « fort » (détection statique des erreurs).
 - Conversions de type contrôlées.
 - Détection dynamique des dépassements des bornes de tableaux.
 - Gestion des exceptions
- Mécanismes sophistiqués de gestion mémoire :
 - Contrôle de l'accès à la mémoire (pas de risque d'écrasement).
 - Libération « Automatique » de la mémoire (ramasse-miettes).



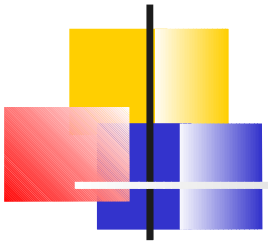
Java est un langage sécurisé

■ But :

- télécharger du code non fiable sur un réseau et l'exécuter dans un environnement sécurisé dans lequel il ne peut commettre aucun dégât.
 - Avec les applets, pas de virus, pas de lecture/écriture sur l'hôte
- Avec Java 2 extension de ces aspects sécurité aux applications
 - Tout code Java (Applet/servlet/Bean/application) peut être exécutée sous différents niveaux de permissions pour contrôler les accès aux système hôte.

■ Principe :

- Aucune confiance implicite
- Actuellement aucune autre plate-forme n'apporte autant de garanties de sécurité que la plate-forme Java



Java a une architecture neutre

- But :

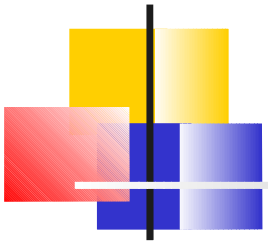
- Exécuter du code mobile dans un environnement hétérogène ou, plus simplement, exécuter le programme sur des machines différentes.

- Principe :

- Eviter les dépendances vis-à-vis :
 - du matériel ;
 - des couches logicielles : réseau, système d'exploitation, environnement graphique.

- Moyens :

- Utilisation d'une machine virtuelle (processeur abstrait).
- Définition d'une bibliothèque standard abstraite, instanciée pour chaque environnement (et tendre vers du pur Java).



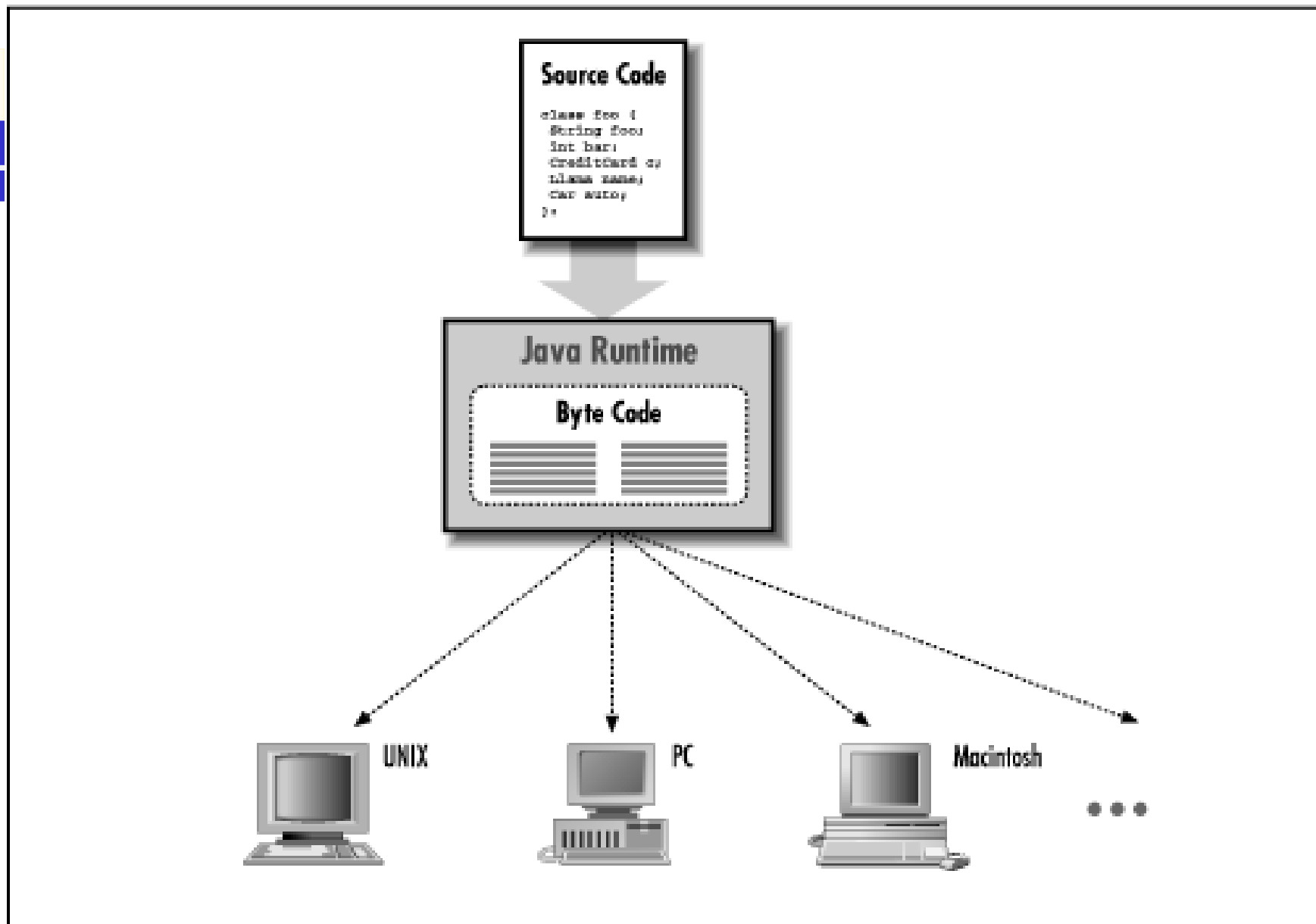
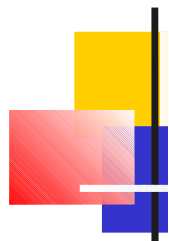
Java est un langage portable

■ But :

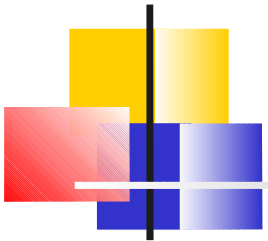
- Un même code compilé produit le même résultat sur toutes les architectures.
- Minimiser les modifications liées au portage de la machine virtuelle.

■ Moyens :

- Bibliothèque indépendante.
- Définition (sémantique) précise du langage :
 - taille, organisation physique données ;
 - valeurs par défaut, minimales, maximales ;
 - effets des opérateurs sur les données ;
 - ordre de calcul ;
 - effets des instructions sur la mémoire.

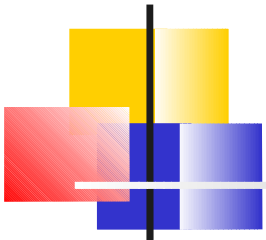


« **Exploring Java** » by Patrick Niemeyer & Joshua Peck; 1-56592-184-271-9, 500 pages (est.) 2nd Edition July 1997 (est.)



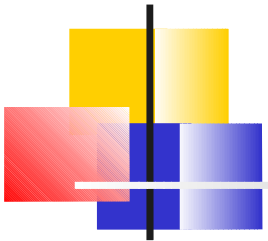
Java est un langage dynamique

- But : Accélérer le cycle de développement :
 - Eviter les recompilations inutiles (ex : nouvelle version bibliothèque).
 - Réduire les éditions de liens au strict nécessaire.
- Moyens :
 - Edition de liens dynamique.
 - Accès à la représentation interne des classes (réflexivité).
 - Chargement des classes à la demande (depuis une source locale ou répartie).



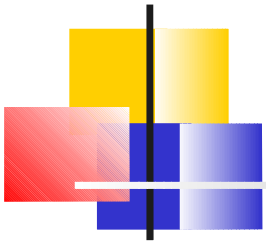
Java est un langage performant

- Byte code adapté pour être compilé à la volée (Just In Time) pour produire puis réutiliser le code associé à chaque instruction.
 - Cache mémoire pour éviter le chargement (et la vérification) multiple d'une même classe.
 - Compilation classique pour engendrer un programme propre à une architecture donnée avec édition de liens classique (mais perte de la mobilité).
 - Ramasse-miettes : processus indépendant de faible priorité
- **Remarque** : La performance d'un langage ne se mesure pas qu'à sa vitesse d'exécution, mais aussi au temps de développement requis.



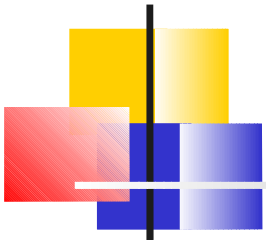
Java est un langage parallèle

- But : développement d'applications interactives :
 - Interfaces Homme/Machine.
 - Multimédia (graphisme, son, vidéo).
 - Calcul.
 - Entrées/Sorties.
- Moyens :
 - Processus légers : exécution parallèle, mémoire partagée.
 - Niveau langage et Machine virtuelle : contrôle plus fin.
 - Mécanismes de synchronisation : moniteurs, variables condition.
 - Système préemptif : le processus ne doit pas rendre la main (dépend de l'implémentation de la machine virtuelle).



Les idées fausses sur Java

- Java est une extension de HTML, il ne sert qu'à écrire des applications Web.
- Java est facile à apprendre.
- Java est un environnement de programmation facile à utiliser.
- Java deviendra un langage universel, multi-plates-formes.
- Java est interprété, donc trop lent pour les grosses applications.
- Les applets Java représentent un risque de sécurité.
- JavaScript est une version simplifiée de Java.
- Java élimine le besoin de scripts CGI.
- Java = C++
 - (c'est un langage presque purement objet, de plus haut niveau, plus proche de SmallTalk ou Ruby)



Les avantages clés de Java

- Programmation centrée réseau

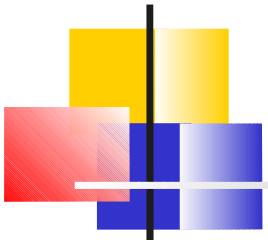
- Autre devise de SUN

- « Le réseau est l'ordinateur »

- L'accès aux ressources distribuées sur le réseau est incroyablement facilité par Java.
 - La création d'application utilisant des architectures clients/Serveur également.

- Des programmes dynamiques et extensibles

- Le code est organisé en Classes, enregistrées dans des fichiers séparés, chargées uniquement en cas de nécessité.
 - Un programme peut s'étendre dynamiquement en chargeant les classes dont il a besoin.
 - D'après le § précédent ces classes peuvent tout a fait être distribuées sur le réseau et chargées depuis différentes machines.



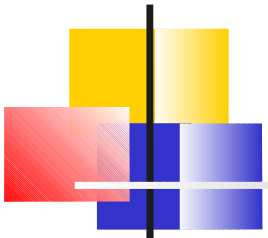
Les avantages clés de Java

■ Internationalisation

- Java et sa plate-forme sont conçu dès le départ avec un esprit d'ouverture sur le monde.
- C'est un langage qui possède des caractéristiques d'internationalisation au départ.
- Exemple : Java utilise les caractères **Unicode 16bits** qui représentent les alphabets phonétiques et les jeux de caractère idéographiques du monde entier.

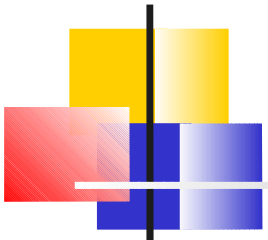
■ Performances

- Plus rapide que les langages de scripts,
- Au départ moins rapide que les langages compilés C/C++
- Aujourd'hui (depuis Java 2) les MV utilisant la technologie **JIT** sont extrêmement efficaces



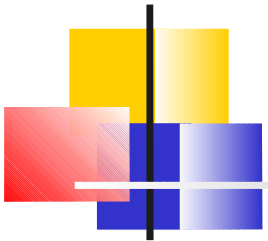
Les avantages clés de Java

- Efficacité du programmeur
 - L'équipe de développement de JAVA a examiné les aspects des langages C et C++
 - Déterminé les caractéristiques qui pourraient être éliminées dans le contexte de la programmation orientée-objet moderne.
 - Aspect familier de JAVA.
 - JAVA ressemble à C++. Les programmeurs familiers avec C, C++, Eiffel, ADA devraient apprendre JAVA facilement.
 - Les programmeurs l'aiment
 - Java est élégant et combiné à un jeu d'API puissant et souvent bien conçu.
 - Les programmeurs migrant vers Java produisent un code
 - Meilleur, moins buggé,
 - Permettant de réduire :
 - les temps de développement
 - les coûts associés



Des différences avec C++

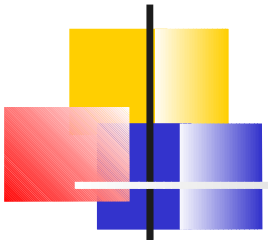
- Pas de structures ni d'unions
- Pas de types énumérés
- Pas de typedef
- Pas de préprocesseur
- Pas de variables ni de fonctions en dehors des classes
- Pas de fonctions à nombre variable d'arguments
- Pas d'héritage multiple de classes
- Pas de types paramétriques (template)
- Pas de surcharge d'opérateurs
- Pas de passage par copie pour les objets
- Pas de pointeurs, seulement des références



Structure d'un programme Java

- Programme Java = une ou plusieurs définitions de classes.
 - Une de ces classes au moins doit contenir une méthode de classe `main()` de prototype :

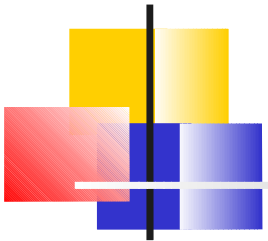
```
public static void main(String[] args);
```
 - La méthode `main()` ne renvoie rien, on utilisera la méthode de classe `System.exit(x)` pour renvoyer la valeur `x` au système d'exploitation.
 - Les paramètres de la fonction `main()` sont passés dans un tableau d'objets `String`. La longueur de ce tableau est consultable par son attribut `args.length`.
 - Contrairement à ce qui se passe en C, le premier élément de ce tableau est le premier paramètre



Exemple de programme : Echo.java

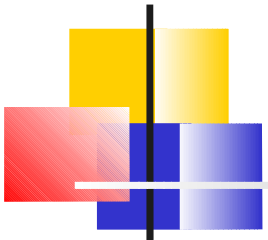
```
/** Programme affichant la liste des paramètres qui lui sont
    passés en ligne de commande. */
public class Echo {
    public static void main(String args[]) {
        if (args.length == 0) {
            System.err.println("Usage : java Echo param [param...]");
            System.exit(1);
        }
        for (int i = 0; i < args.length; i++)
            System.out.println("Param n° " + i + " = " + args[i]);
        System.exit(0);
    } // main
} // class Echo
```

Déclare une classe appelée `Echo`. Une simple méthode pour visualiser les paramètres sur la sortie standard.
Pour écrire : invoquer la méthode `println` de l'objet `out` (VC de `System`) `out` est une variable de classe de la classe `System` initialisée par une instance d'une classe de la hiérarchie `Stream`. On utilise la méthode `println` sur cet objet.



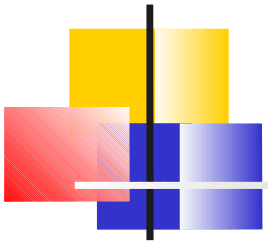
Exemple de programme : Echo.java

- Remarque :
 - Le nom du fichier doit être **le même** que celui de la classe publique (*une seule classe publique par fichier*).
- Compilation en bytecode :
 - `javac Echo.java`
- Interprétation du bytecode :
 - `java Echo`
- Production de la documentation :
 - `javadoc Echo.java`



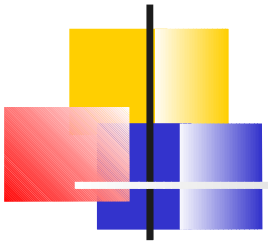
Utilisation de javadoc

- Outil permettant de documenter (spécifier) un paquetage, une classe, une méthode, une variable ou une constante.
 - Les membres privés n'apparaissent pas dans cette documentation.
 - Les commentaires **javadoc** doivent être placés avant ce qu'ils documentent.
 - Les commentaires **javadoc** sont compris entre la séquence `/**` et la séquence `*/`.
 - Ils peuvent contenir du texte simple, du code HTML et des balises **javadoc**.
 - La documentation produite est au format HTML et s'intègre dans le cadre de la documentation générale de l'API Java.



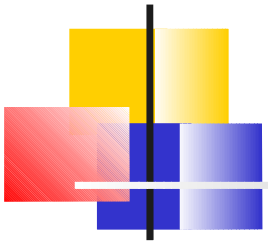
Utilisation de javadoc

- Les balises **javadoc** sont de la forme
`@nom description.`
- Elles sont classées en :
 - commentaires généraux ;
 - commentaires de paquetages et d'introduction ;
 - commentaires de classes ou d'interfaces ;
 - commentaires de méthodes ;



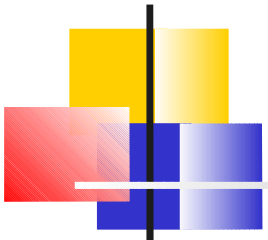
javadoc : Commentaires Généraux

- Les balises suivantes sont communes à tous les types de commentaires **javadoc** :
 - `@since texte` : pour décrire la version introduisant la fonctionnalité.
 - `@deprecated texte` : pour déconseiller cette fonctionnalité et proposer une solution de remplacement.
 - `@see lien` : pour ajouter un lien dans la rubrique « **See Also** » des classes et des méthodes. `lien` est :
 - `paquetage.classe#étiquette` (le plus utilisé) ;
 - `étiquette` (pour des URL externes) ;
 - `"texte"`.



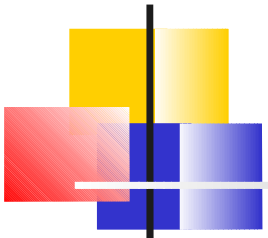
Commentaires de paquetages et d'introduction

- Pour produire un commentaire de paquetage, on ajoutera un fichier `package.html` dans le répertoire de celui-ci.
- Un commentaire d'introduction présente un fichier source. Il est placé dans un fichier `overview.html` situé dans le répertoire des sources.
- Le texte de ces fichiers sera compris entre les balises HTML `<BODY>` et `</BODY>`.
- Des fichiers modèles sont fournis sur le site de la documentation de **javadoc**.



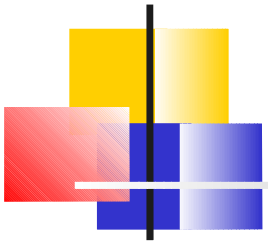
Autres Commentaires javadoc

- Commentaires de classes et d'interface
 - Doivent être placés après une instruction `import` et juste avant la définition de la classe/interface décrite.
 - Balises disponibles :
 - `@author nom`
 - `@version texte`
- Commentaires de méthodes
 - Doivent être placés juste avant la signature de la méthode décrite.
 - Balises disponibles :
 - `@param nom_param description_param`
 - `@return description`
 - `@throws description_exceptions`



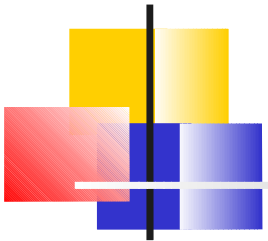
Exemple de code commenté

```
/** Une classe pour gérer les <i>fractions</i>.
    @author depinfo
    @version 0.99, 07/03/2013
 */
public class Fraction {
    private int num, den;
    /** Constructeur
        @param n entier représentant le numérateur
        @param d entier représentant le dénominateur
        @throws ArithmeticException si le dénominateur est nul
    */
    public Fraction(int n, int d) throws ArithmeticException { ... }
    /** Convertit une fraction en double
        @param aucun
        @return un double représentant la valeur de la fraction
    */
    public double toDouble() { ... }
} // Fraction
```

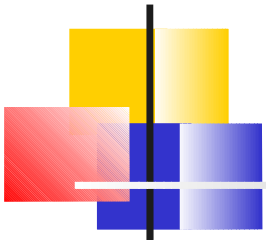
Les commentaires traditionnels

- Pour documenter le code, Java reconnaît deux types de commentaires :
 - Commentaires à la C (plusieurs lignes) : `/* ... */`
 - attention à ne pas rajouter un `*` à la suite du `/* ...`
 - Commentaires à la C++ (sur une ligne) : `//`
- On ne peut pas imbriquer des commentaires à la C (mais ils peuvent contenir des commentaires à la C++).
- Les commentaires à la C sont pratiques pour commenter un bloc de code lors des phases de débogage.



Conventions de nommage

- Sun a publié un document **Java Code Conventions**, disponible sur Internet (<http://www.oracle.com/technetwork/java/index-135089.html>).
- Java différencie les majuscules des minuscules.
- Tous les caractères Unicode sont autorisés : les identificateurs peuvent donc contenir des lettres accentuées, `étudiant`, `numéroLivre`, par exemple.
- La longueur des identificateurs n'est pas limitée.
- Les mots réservés sont tout en minuscules.



Conventions de nommage

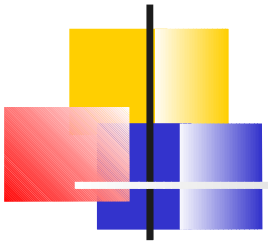
- Un nom de paquetage sera tout en minuscules (`monpaquetage`).
- Un nom de constante sera tout en majuscules (`MACONSTANTE`).
- Un nom de variable ou de méthode commencera par une minuscule.
- Les différents mots commencent par des majuscules, sauf le premier (`uneVariable`, `toString()`).
- Un nom de classe ou d'interface commencera par une majuscule. Les différents mots commencent par des majuscules (`MaClasse`, `FileInputStream`).
- Il y a (malheureusement) quelques exceptions... Par exemple :
`println()` au lieu de `println()`,
`Color.white` au lieu de `Color.WHITE`



Java

- Définitions
- Historique
- Caractéristiques
- Les bases du langage
- Structure des programmes

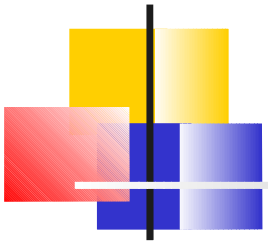
« Ecrire une fois, exécuter partout »



Exemple

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

- Déclare une classe appelée HelloWorld.
 - Une simple méthode pour visualiser la chaîne "Hello World" sur la sortie standard.
 - Pour écrire "Hello World" : invoquer la méthode `println` de l'objet `out` (VC de `System`)
 - `out` est une variable de classe de la classe `System` initialisée par une instance d'une classe de la hiérarchie `Stream`. On utilise la méthode `println` sur cet objet.



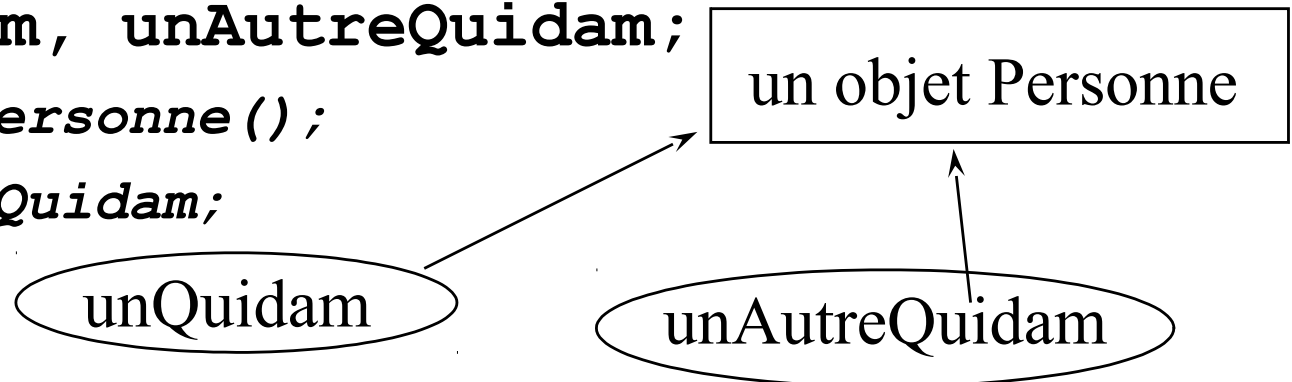
Les types de Java

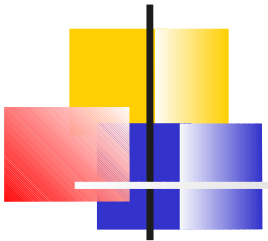
- Tout est objet en Java.
 - Sauf les types primitifs. (byte, short, int, long, float, double, char, boolean)
 - Les objets sont stockés par référence.
 - Les types primitifs sont stockés par valeur.
 - Java en propose une version encapsulée
 - Mécanisme d'autoboxing avec Java 1.5

Personne unQuidam, unAutreQuidam;

unQuidam = new Personne();

unAutreQuidam=unQuidam;

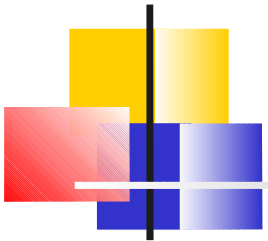




Les entiers et les réels

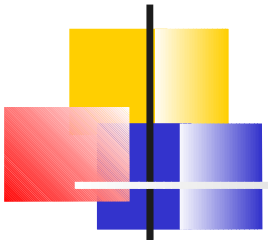
- Les types `integer` ont des longueurs fixes:
 - 8-bit en `byte` - 16 bits en `short`
 - 32 bits en `int` - 64 bits en `long`

pas de type `unsigned` pour les entiers en Java.
- Les types réels :
 - 32 bits pour les `float`
 - 64 bits pour les `double`.
 - Une valeur littérale à virgule flottante, est un `double` par défaut;
 - il faut la "caster" `float` pour qu'elle prenne le type `float`.



Les caractères et les booléens

- Une donnée du type `char` en Java, est un caractère **Unicode** sur 16 bits.
 - Les caractères Unicode sont des valeurs non-signées sur 16 bits.
`char myChar = 'Q';`
un type **Unicode** (valeur non-signée sur 16 bits) qui est initialisé à la valeur **Unicode** du caractère Q.
- Java a ajouté un type `boolean` comme un type primitif.
 - Un booléen Java est un type distinct
 - Contrairement au C, le booléen ne peut être converti en un type numérique.



Les objets JAVA

- Tous les autres types sont des objets.
 - Tous les objets Java doivent être alloués dynamiquement et ne peuvent exister statiquement dans le code

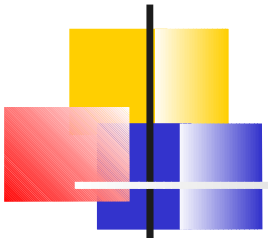
```
int tentativeDeTableauStatic [2];
```

Can't specify array dimension in a declaration.

```
int tentativeDeTableauStatic [2];
```

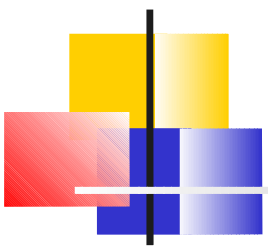
^

1 error



Les tableaux Java sont des objets

- On peut déclarer et allouer des tableaux de tout type.
 - `Point [] myPoints`
 - une référence à un tableau de point non initialisée.
- Pour allouer le nombre de cellules il faut :
 - `myPoints = new Point[10];`
 - 10 références à point sont ainsi allouées à `null`.
- L'attribut `length` donne la longueur du tableau :
 - `myPoints.length`
 - donne le nombre des éléments dans `myPoints`.
 - `howMany = myPoints.length`
 - donne la valeur 10 à la variable `howMany`.
- Les tableaux sont indexés par des `int`.
 - Le début c'est 0. En cas de débordement : exception.

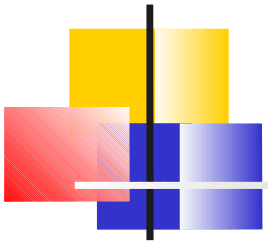


Les tableaux

- Les tableaux de char ne sont pas des Chaînes de caractères.

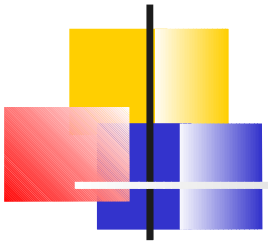
```
char tab[] = "ii" ;
```

Incompatible type for declaration. Can't convert java.Lang.String to char []



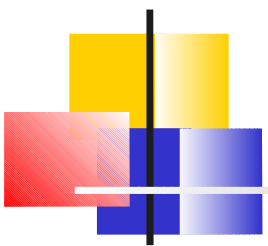
Purifier C/C++

- Le code source en JAVA est simple et lisible.
 - Il n'y a plus de préprocesseur, plus de `#define`, de `typedef`,
 - plus de recours aux fichiers `header`,
- Plus de
 - Surcharge des opérateurs, Opérateur de portée `::`
 - Structures et unions, fonctions
 - Héritage multiple, pointeurs, conversions implicites, `goto`
 - Problèmes de mémoire
- Les programmeurs peuvent lire et comprendre le code
 - le modifier et le réutiliser plus facilement et plus rapidement.



Plus de

- Plus besoin de structures ou d'unions avec les classes;
 - Vous pouvez déclarer une classe avec les variables appropriées pour obtenir un effet similaire.
- Java n'a plus de fonctions.
 - Tout ce qu'on peut faire au moyen d'une fonction peut être fait
 - par définition d'une classe
 - par création de méthodes pour cette classe.
- Plus de forçage de type
 - Java supprime le forçage de type automatique de C et C++.
 - Pour forcer le type d'une donnée en un type qui perdrait de la précision, il faut le faire explicitement en utilisant un **cast**.



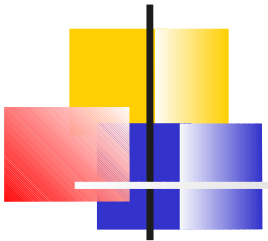
Plus de

- Plus d'héritage multiple

- L'héritage multiple, et tous les problèmes que cela comporte, a été supprimé de Java.
- Les caractéristiques désirables de l'héritage multiple sont permises grâce aux **interfaces**.

- Plus de Pointeurs

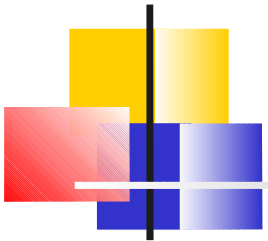
- **Rappel** : les pointeurs sont une des caractéristiques qui permettent aux programmeurs d'introduire des bugs dans leur code.
 - Les structures enlevées, la transformation des tableaux et des chaînes en objets, les pointeurs sont inutiles.
 - On accède aux tableaux par leurs indices arithmétiques.



Plus de

- Plus de `goto`, mais des `breaks` Multi-niveaux
 - `JAVA` n'a pas de `goto`. Au niveau de `break` ou `continue`, vous pouvez placer des labels.

```
test: for (int i= fromIndex; i + max1 <= max2; i++) {  
    if (charAt(i) == c0) {  
        for (int k = 1; k < max1; k++) {  
            if (charAt(i+k) != str.charAt(k)) {  
                continue test;  
            }  
        } /* end of inner for loop */  
    }  
} /* end of outer for loop */
```



Plus de gestion de la mémoire

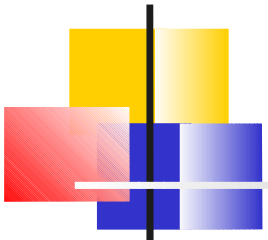
- C et C++ => problèmes de gestion de mémoire:
 - **allouer** de la mémoire, **libérer** de la mémoire
 - garder la trace de quelle mémoire peut être libérée.
- La gestion de mémoire explicite a prouvé être une source de bugs, crashes, et performances médiocres.
 - Si vous ne deviez retenir qu'une chose du L3 c'est celle-ci
 - Les pointeurs, `malloc` et `free` n'existent plus.
 - Java a un opérateur `new` pour allouer la mémoire pour les objets, pas de `free`.
 - La MV garde trace du statut de l'objet et réclame automatiquement de libérer la mémoire quand les objets ne sont plus utilisés.



Java

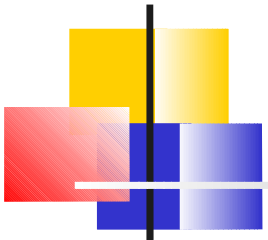
- Définitions
- Historique
- Caractéristiques
- Les bases du langage
- Structure des programmes

« Ecrire une fois, exécuter partout »



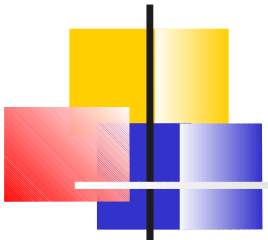
Travailler avec Java

- Récupérer le JDK
 - Site officiel
(<http://www.oracle.com/technetwork/java/index.html>)
- Récupérer un décompilateur
 - Par exemple jad (<http://www.varaneckas.com/jad>)
- Récupérer un Ide (éventuellement)
 - Eclipse (<http://www.eclipse.org/>)



Structure des programmes Java

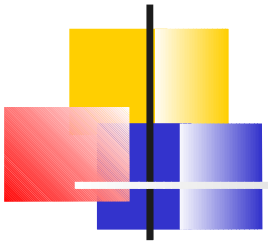
- Une application Java est composée :
 - d'un ou plusieurs fichiers source (.java) définissant une ou plusieurs classes
 - une classe `public` au maximum par fichier
 - Le fichier source **doit porter** le même nom que cette classe publique
 - une classe, au moins, **doit implémenter** la méthode
`main(String arg[])`
- Quel que soit l'IDE Java utilisé, on dispose d'au moins deux outils :
 - Un compilateur Java qui produit le «**bytecode**»
 - Une «**machine virtuelle**» capable d'exécuter le bytecode



Un exemple

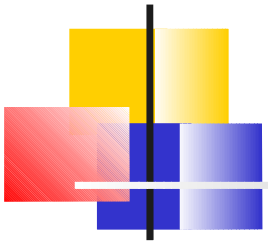
```
/**
 * This program computes the factorial of a number
 */
public class Factorial {                                // Define a class
    public static void main(String[] args) {            // The program starts here
        int input = Integer.parseInt(args[0]);          // Get the user's input
        double result = factorial(input);               // Compute the factorial
        System.out.println(result);                    // Print out the result
    }                                                    // The main() method ends here

    public static double factorial(int x) {              // This method computes x!
        if (x < 0)                                       // Check for bad input
            return 0.0;                                  // if bad, return 0
        double fact = 1.0;                              // Begin with an initial value
        while(x > 1) {                                  // Loop until x equals 1
            fact = fact * x;                             // multiply by x each time
            x = x - 1;                                   // and then decrement x
        }                                                // Jump back to the start of loop
        return fact;                                    // Return the result
    }                                                    // factorial() ends here
}                                                        // The class ends here
```



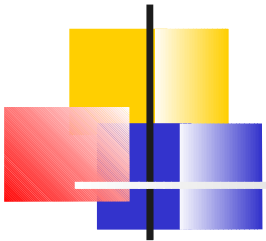
Les Packages (1)

- Un package regroupe un ensemble de classes sous un même « **espace de nom** ».
 - Les noms des packages suivent le schéma : `name.subname ...`
 - Une classe `Watch` appartenant au package `time.clock` doit se trouver dans le fichier `time/clock/Watch.class`
- Les packages permettent au compilateur et à la JVM de localiser les fichiers contenant les classes à charger.
- L'instruction `package` indique à quel package appartient la ou les classe(s) de l'unité de compilation (le fichier).



Les Packages (2)

- Les répertoires contenant les packages doivent être présents dans la variable d'environnement **CLASSPATH**
- En dehors du package, les noms des classes sont :
`packageName.className`
- L'instruction `import packageName ;` permet d'utiliser des classes sans les préfixer par leur nom de package.
- Toutes les API sont organisées en package dans la plateforme (`java.lang`, `java.io`, ...)



Les Packages (3)

CLASSPATH = \$JAVA_HOME/lib/classes.zip;\$HOME/classes

[~dedieu/classes/graph/2D/Circle.java](#)

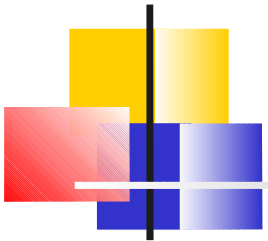
```
package graph.2D;  
public class Circle()  
{ ... }
```

[~dedieu/classes/graph/3D/Sphere.java](#)

```
package graph.3D;  
public class Sphere()  
{ ... }
```

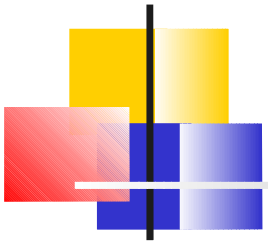
[~dedieu/classes/paintShop/MainClass.java](#)

```
package paintShop;  
  
import graph.2D.*;  
  
public class MainClass()  
{  
    public static void main(String[] args) {  
        graph.2D.Circle c1 = new graph.2D.Circle(50)  
        Circle c2 = new Circle(70);  
        graph.3D.Sphere s1 = new graph.3D.Sphere(100);  
        Sphere s2 = new Sphere(40); // error: class paintShop.Sphere not found  
    }  
}
```



Les outils

- Des machines virtuelles :
 - Kaffe, cacao, harissa, ...
- Des convertisseur :
 - c2j, j2c, Tabo...
- Des décompilateur :
 - Mocha, Crema, Jad
- Des générateur de parseurs :
 - JavaCC, JavaCUP
- Des profilers :
 - Hyperprofiler, ProfileViewer



Le JDK : Java Development Kit

- `javac` : compilateur de sources java
- `java` : interpréteur de byte code
- `appletviewer` : interpréteur d'applet
- `javadoc` : générateur de documentation (HTML, MIF)
- `javah` : générateur de header pour les méthodes natives
- `javap` : désassembleur de byte code
- `jdb` : debugger
- `javakey` : générateur de clés pour la signature de code
- `rmic` : compilateur de stubs RMI
- `rmiregistry` : "Object Request broker" RMI
- `jar` : pour archiver/compresser les classes dans un seul fichier