

## Cours n° 1

# Introduction à la programmation



# Sommaire

---

## **1. Langages de 3<sup>ème</sup> génération**

- Conceptualisation
- Traduction
- Compilation et interprétation

## **2. Syntaxe des langages de programmation**

- Notation BNF
- Diagrammes de Conway

## **3. Le langage Java**

- Historique
- Spécificités
- Unité de compilation

## **La programmation**

**Claude Delannoy**, «Programmer en Java», Eyrolles

**Vincent Granet**, « Algorithmique et programmation en Java », Dunod

**Peter Haggard**, «Mieux programmer en Java», Eyrolles

**Patrick Niemeyer & Joshua Peck**, « Java par la pratique », O'Reilly

## **Les API**

**David Flanagan**, «Java in a nutshell », O'Reilly

**David Flanagan**, «Exemples en Java in a nutshell », O'Reilly

## **Sites**

[java.sun.com](http://java.sun.com) (site officiel)

[java.sun.com/javase/downloads](http://java.sun.com/javase/downloads)

[java.developpez.com/cours/](http://java.developpez.com/cours/) (tutoriels)

## **Dépôt des cours et des devoirs**

<https://m1ilgii2007v2.googlecode.com/svn/trunk/>

Username : **TALmaster1** Password : **g8f4f2p3**

---

## Plan du cours

---

- 1) Introduction à la programmation
- 2) Données en Java
- 3) Traitements en Java
- 4) Objets, classes et héritage
- 5) Les exceptions
- 6) Les flots d'entrées-sorties
- 7) Les chaînes de caractères
- 8) Traitements de textes
- 9) Interfaces graphiques I
- 10) Interfaces graphiques II
- 11) Interfaces graphiques III
- 12) Modèles de documents
- 13) Tests avec JUnit

## Notion de programmation en informatique

---

### Définition de l'informatique

Science de l'information ; ensemble des techniques de la collecte, du tri, de la mise en mémoire, de la transmission et de l'utilisation des **informations traitées automatiquement** à l'aide de programmes (logiciels) mis en œuvre sur **ordinateur** [*dictionnaire Petit Robert*]

### Objectif en programmation

Permettre l'**automatisation** de « **tâches** » à l'aide d'**ordinateurs** (automates programmables)

### Production liée à l'activité de programmation

#### Programme / Logiciel (d'application)

Le **média** utilisé pour **décrire** l'automatisation de la tâche  
L'association d'une séquence d'**instructions** et de **données**  
**pouvant être traitées** (comprises et exécutées) par l'**ordinateur**



### Langages de « haut niveau »

- **Mnémoniques** sont proches du **langage naturel**
- **Formalisme proche** du **raisonnement** humain
- **Formalisme** de plus en plus **proche** de la **conceptualisation** des problèmes du programmeur
- Formalismes pour la **réutilisation** de portions de code

**Gain de productivité, de sûreté** du logiciel

### Dans la catégorie des langages impératifs

Fortran (1957/1962/1966...), Algol (1958/1960/1968), Cobol (1959/1974),  
Basic (1964), Pascal (1969),  
C (1972), Modula (1979), Ada (1983),  
Langages OO : C++ (1983), Ada95 (1995), Java (1995), C# (2000), ...  
Langages objets : SmallTalk (1972), Eiffel (1985)

## Paradigmes de programmation

---

### Programmation impérative : un paradigme de programmation

- Basé sur l'**exécution** de **blocs d'instructions** et sur une **mémoire** où les **instructions** peuvent laisser une trace de leur exécution en **modifiant** les **cellules mémoires**
- L'**état d'un programme** correspond aux **valeurs courantes de la mémoire**
- **Modèle** de haut niveau de l'**architecture matérielle** de **Von Neumann** : une zone mémoire et un processeur qui agit sur cette mémoire

### Remarque :

La **connaissance** de l'**architecture des ordinateurs** permet de situer les **ressources** de la machine et de construire des **programmes adaptés**

## Exemple d'évolution dans les langages

### Basic (à ses débuts)

```
1  REM  Commentaire
4  LET N = 10
7  IF (N = 0) THEN GOTO 15
10 ELSE LET N = N-1
11 REM Action à répéter
13 GOTO 7
15 END
```

### Java

```
/** Commentaire pour génération
    automatique de document
 */
for (int i=0; i<10; i++) {
    // Action à répéter
}
```



## Conceptualisation

---

### Evolutions sensibles dans les langages

- Langages **sans structuration générale**  
Ex : les premières versions de Fortran (IV), Basic, ...
- Support des **structures de contrôle**
- Support des **fonctions/procédures**
- Support d'utilisation de **bibliothèques** standards ou « personnelles »  
Gros effort de développement des bibliothèques standard (e.g., C++, Java)  
dont les composants techniques
- Support des concepts **objet** (types abstraits de données)
- Support du concept de **généricité**  
patron (C++), générique (Ada), héritage (langage objets et orientés objet)
- Support de **sûreté d'utilisation**  
assertion, exception, pré-conditions, post-conditions

# Traduction

---

### Source et exécutable

De façon analogue au programme écrit en langage d'assemblage,  
Le **programme-source** écrit en langage « de haut niveau » devra être **traduit**  
en **langage-machine** pour être exécuté

### Deux types de traducteur

- **Compilateur** associé à un **éditeur de liens**
- **Interpréteur**

### Exemples :

**Lisp** (langage fonctionnel), peut être **interprété** ou **compilé**

**C++** est un langage **compilé**, jamais interprété

**Java** est un langage **semi-compilé** :

- compilé dans un byte code,
- interprété à l'exécution avec une machine virtuelle (JVM, Java Virtual Machine)

### Compilation (1/2)

---

#### Compilateur (*compiler*)

Traduit le programme-source dans son ensemble en un programme appelé le **code-objet**

#### Objectif

Produire un code-objet i) correct, ii) **optimisé** : le code-objet doit s'exécuter le plus vite possible

#### Etapas de compilation

- **Analyse lexicale** : découpage du programme en lexèmes
- **Analyse syntaxique** : vérification de la correction de la syntaxe du source
- **Analyse sémantique** : analyse des structures de données
- Transformation du code-source en **code intermédiaire**
- Application de techniques d'**optimisation** sur le code intermédiaire
- Transformation du code intermédiaire en **code-objet**, avec éventuellement l'insertion de **données de débogage**

### Compilation (2/2)

---

#### **Editeur de liens (*linker*)**

Généralement, le code-objet n'est exécutable qu'après une phase d'édition des liens

#### **Fonction**

Construire une **image mémoire** contenant l'ensemble des parties de code compilées séparément (modules/sous-programmes, fonctions de bibliothèques statiques, fonctions de bibliothèques dynamiques)

#### **Effort de développement d'un compilateur**

Important en raison de la complexité des compilateurs actuels (plusieurs années de développement, en équipe)

## Interpréteur

---

**Analyser, traduire et exécuter un programme-source dans un environnement généralement interactif**

### **Cycle d'un interpréteur**

- Lire et analyser une instruction
- Si l'instruction est syntaxiquement correcte, l'exécuter (sinon message d'erreur)
- Passer à l'instruction suivante

### **Remarque :**

Du fait de la phase d'interprétation, l'exécution d'un programme interprété est plus lente que le même programme compilé

### **Effort de développement d'un interpréteur**

Bien moins important que celui d'un compilateur

## Compilateur *versus* interpréteur

---

### Le plus de l'interpréteur

- **Facilite la mise au point des programmes**  
-en corrigeant progressivement les erreurs, -en évitant la phase de compilation (souvent longue)
- Très adapté au **développement rapide d'applications (RAD)**  
(e.g., **prototypes** d'applications) car permet rapidement de tester les applications

### Le plus du compilateur

- Un langage **compilé** permet un **développement plus efficace** des projets (de grande taille). Optimisation plus globale, traduction effectuée une seule fois et non pas à chaque utilisation
- Un langage **compilé** permet la **diffusion** des programmes sous forme **d'exécutable** (binaire) et non de code-source  
(protection de la propriété intellectuelle)

### Portabilité

s'il existe un **interpréteur/compilateur spécifique** à la plate-forme **cible**  
**Porter un langage interprété** sur une plate-forme cible est **moins coûteux** que de porter un langage compilé (cf. effort de développement)

## 1. LANGAGE DE 3<sup>ème</sup> GENERATION

# TIOBE Programming Community Index ([www.tiobe.com](http://www.tiobe.com))

Position Sep 2012	Programming Language	Ratings Sep 2012	Status
1	C	19.3%	A
2	Java	16.3%	A
3	Objective-C	9.8%	A
4	C++	9.1%	A
5	C#	6.6%	A
6	PHP	5.6%	A
7	Visual Basic	5.5%	A
8	Python	3.9%	A
9	Perl	2.3%	A
10	Ruby	1.7%	A

# Spécification des langages de programmation

---

### Langage de programmation

- **Lexique** (vocabulaire) incluant les mots réservés du langage
- **Syntaxe** (grammaire)
- **Sémantique**

Un **langage** de programmation est **spécifié**

- précisément
  - sans ambiguïté d'interprétation
- dans son **manuel de référence**

### Définition des aspects lexicaux et syntaxiques

Règles de grammaire hors-contexte (règles syntaxiques)

Deux formalismes usuels pour la description des règles syntaxiques

- Notation BNF
- Diagramme de Conway



## 2. SYNTAXE DES LANGAGES DE PROGRAMMATION

### Mots réservés du langage Java

abstract	boolean	break	byte	bvalue
case	cast	catch	char	class
const	continue	default	do	double
else	extends	false	final	finally
float	for	future	generic	goto
if	implements	import	inner	instanceof
int	interface	long	native	new
null	operator	outer	package	private
protected	public	rest	return	short
static	super	switch	synchronized	this
throw	throws	transient	true	try
var	void	volatile	while	

### Notation BNF (Backus Naur Form) (1/2)

#### Méta-langage constitué de

- **Méta-symboles**

::=	définition d'une catégorie syntaxique
<>	délimiteurs d'une catégorie (sans délimiteur : littéraux du langage)
	ou logique,
{ }	répétition d'item (0 ou plusieurs fois)
[ ]	item optionnel (0 ou une fois)

*Diverses extensions...  
" " délimiteurs des littéraux  
du langage*

- **Symboles terminaux**

Littéraux/mots/lexèmes du langage (mots réservés)  
cf. Transparents (mots réservés de Pascal et de Java)

- **Symboles non terminaux**

Catégories syntaxiques décrivant le langage

### Notation BNF (Backus Naur Form) (2/2)

#### Règle de formation d'un identificateur

*Un identificateur est une séquence composée de lettres (minuscules ou majuscules, à l'exception des caractères accentués), de chiffres, du caractère de soulignement (\_) ou du caractère dollar (\$) ; il ne doit pas commencer par un chiffre, les espaces ne sont pas autorisés.*

#### Règle BNF (indépendante du contexte)

`<identificateur> ::= <lettre> {<lettre> | <chiffre>}`

*Catégories syntaxiques :  
identificateur, lettre, chiffre*

`<lettre> ::= $|_|A|a|B|b|C|c|.....Z|z`

`<chiffre> ::= 0|1|2....9`

*Symboles terminaux en gras*

#### Règle récursive équivalente

`<identificateur> ::= <lettre> | <identificateur> [<lettre> | <chiffre>]`

*Exemples d'identificateurs :*

*Jhkjhj, JKK, nbClients, \_items, // valides*

*2emeJoueur, catégorie, @adresse // invalides*

#### Site à visiter :

<http://cui.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html>

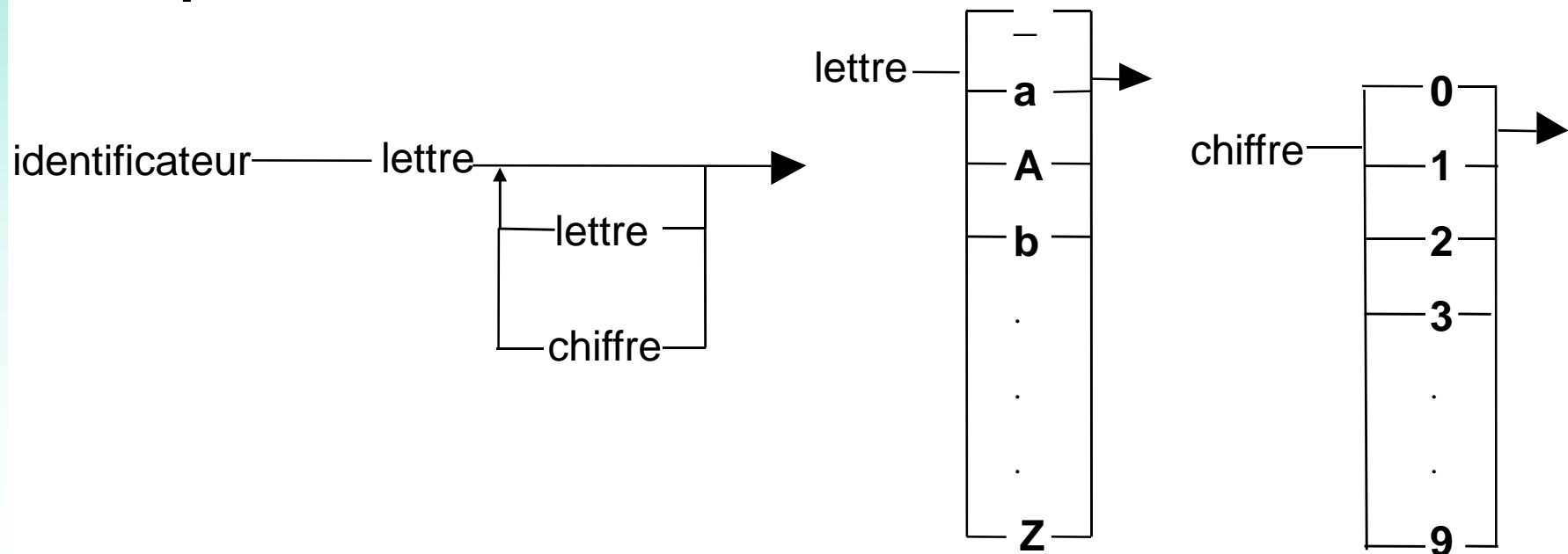
### Diagrammes de Conway

#### Expression graphique des règles syntaxiques

#### Principe de formation d'une catégorie syntaxique

Tout chemin allant de l'origine du diagramme à son extrémité (fléchée) engendre une expression de la catégorie appartenant au langage

**Exemple :** *diagramme syntaxique des identificateurs*



## Définition

---

**Environnement de programmation orienté objets adapté à la distribution d'applications sur Internet (Web Services)**

**Propriété de Sun Microsystems**

**Éléments de l'ensemble de programmation**

- Un langage de programmation

- Une définition d'une machine virtuelle (JVM)

- Un ensemble de classes réparties dans des API

- Un ensemble d'outils

## Historique

---

- 1990** Besoin d'interfaces graphique pour l'informatique personnelle (domotique)
- 1991** C++ en cours de normalisation. Création d'un langage dérivé nommé « Oak » par SunMicrosystems
- 1993** Sortie du 1<sup>er</sup> navigateur www (Mosaic de NSCA)
- 1995** « Oak » prend le nom de Java. Vente d'une licence à la société Netscape
- 1996** Java Developers Kit 1.0 (RMI, JDBC, JavaBeans)
- 1998** JDK 1.2 (Java 2), SunMicrosystems gagne son procès contre Microsoft
- 2002** J2SE 1.4 (Merlin)
- 2004** J2SE 1.5 (Tiger)
- 2007** Java SE 6 (Mustang), 3 777 classes et interfaces
- 2012** Java SE 7 (Dolphin), acquisition par Oracle en 2009

## Spécificités du langage Java

### Langage interprété (semi-compilé) adapté à la distribution d'applications sur Internet (Web Services)

- **Le portage d'un programme ne requiert ainsi que le portage de la machine virtuelle**  
Transformation d'un code Java dans un langage-machine « virtuel » (*byte code*).  
Pas de phase d'édition de liens. Interprétation du *byte code* par une machine virtuelle Java (JVM) : programme émulant les principales fonctionnalités d'un ordinateur
- **Portabilité sur les plates-formes possédant une JVM**  
Systèmes d'exploitation : Windows, Linux, Solaris  
Navigateurs web : Internet Explorer, Netscape, Mozilla
- **Robustesse**  
Interdiction des accès directs à la mémoire (*garbage collector*)  
Contrôle d'accès aux objets (indice de tableau)
- **Gestion de la sécurité**  
Trois niveaux de sécurité gérés par l'interpréteur (vérificateur de *byte code*, chargeur de classes, protection des fichiers et accès au réseau)
- **Multiprogrammation**  
Gestion et synchronisation de plusieurs tâches en concurrence
- **Applications réparties**  
API réseau intégrée (*applet, servlet, rmi*)

### 3. LE LANGUAGE JAVA

## Code-source

```
package cours01;
import util.*;

public class NombreMots {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String s, mots[];

        System.out.println("Calcul du nombre de mots dans une phrase");
        s = Keyboard.getLigne("Entrer le texte de la phrase : ");

        // segmentation de la phrase en mots
        mots = s.split(" ");

        System.out.println("cette phrase comporte " + mots.length + " mots");
        for (int i = 0; i < mots.length; i++)
            System.out.println(mots[i]);
        }
    }
```



## Unité de compilation : unité de base

---

### Unité de compilation

Source pouvant être compilé indépendamment de tout autre source

### Structure d'un programme

Un **ensemble d'unités de compilation** usuellement organisé en **fichiers textes** (physiques)

### Structure minimale d'un programme

Une **unité de compilation unique** organisée en un **fichier texte unique**

**Unité de compilation**  $\equiv$  **fichier texte**

**Unité de compilation** contenant un **point d'entrée**  
(du programme)

- Une **définition de fonction** appelée fonction principale (***main***)
- **Fonction** qui sera **exécutée/évaluée en premier**
- Le **corps** de la fonction délimité par { et } (bloc)  
contient les instructions à exécuter (chaque instruction se terminant par ;)

## Unité de compilation : unité de base

### Bloc (d'instructions)

- **Regroupe** un certain nombre (éventuellement nul) d'**instructions**
- Délimité par les accolades { (début de bloc) et } (fin de bloc)
- Un **bloc** peut **contenir d'autres blocs** (imbriqués)

**Exemple :**

```
{ } // bloc vide, sans instruction
{ x=0 ; } // une instruction, les accolades peuvent
// alors être omises ≡ x=0 ;
{ x=0 ; y=0 ; } // bloc de deux instructions
```

### Flot d'exécution

Les **instructions** d'un code-source sont **exécutées en séquence**

- i) en partant de la première ligne de code
- ii) de « gauche » à « droite »
- iii) de « haut » en « bas »

## Point d'entrée du programme en Java

### 1) **public static void main(String args[])**

Méthode publique (**public**) et statique (**static**) **main** d'une classe ne retournant pas de résultat (**void**)

**Paramètres de la fonction main**

**Paramètres d'exécution du programme**

**args[]** : tableau de chaîne de caractères (**String**)  
argv[0] indique le premier argument (exécutable non compris)  
argv[n] le nème argument (chaîne) de la commande

(**Tableaux et String**, notion différée)

### 2) **public class *NomClasse***

La **méthode main** est incluse dans une **unité de compilation** : un fichier physique ***NomClasse.java*** contenant une **classe publique**

Le nom du fichier physique est construit à partir du **nom de la classe** (commençant obligatoirement par une **majuscule**) suivi de l'extension **.java**

## Visibilité des bibliothèques

### 1) **Import NomPaquetage.\*;**

Importation des classes d'un paquetage nommé **Nompaquetage**  
Permet la visibilité de toutes les classes du paquetage  
et en conséquence l'utilisation de toute méthode publique de ses classes

**Remarque** : il est possible de désigner au lieu de `import`  
`NomPaquetage.*` ; `import NomPaquetage.NomClasse`  
Pour restreindre la visibilité à la seule classe **NomClasse** du **paquetage**

### 2) **package NomPaquetage;**

Déclare la classe **X** (du fichier-source où apparaît cette instruction)  
dans le paquetage **NomPaquetage**

### Une autre unité de compilation, par l'importation

### **Import NomPaquetage.\*;**

D'importer les classes (dont **X**) du paquetage **NomPaquetage**  
et d'utiliser les méthodes publiques (entre autres) de la classe **X**

## Commandes de compilation et d'exécution

- Par convention, les **fichiers sources** ont pour extension **.java**
- Commande de **compilation** d'un fichier source nommé ***MonProgramme.java*** :

`Javac MonProgramme.java`

**Remarque** : le résultat de la compilation est le fichier objet (en byte code) nommé ***MonProgramme.class***

- Commande pour **exécuter/interpréter** le programme :

`Java MonProgramme`

**Remarque** : C'est le fichier objet (en byte code) ***MonProgramme.class*** qui est interprété