

# Le Langage Java

## 1<sup>e</sup> année

J. Beleho (bej)   C. Leruste (clr)   M. Codutti (mcd)  
P. Bettens (pbt)   F. Servais (srv)   C. Leignel (clg)  
D. Nabet (dna)   J. Lechien (jlc)

Haute École de Bruxelles — École Supérieure d'Informatique

Année académique 2014 / 2015

# Liste des séances

- 1 Objectifs, évaluations et introduction
- 2 Développer en Java, premier survol
- 3 La gestion des erreurs et le survol des alternatives
- 4 Lisibilité et notions de modules
- 5 Notion de package et survol des structures répétitives

# Séance 1

## Objectifs, évaluations et introduction

- Objectifs
- Moyens
- Évaluations
- Concepts
- Traduction

« *I really hate this darn machine ; I wish that they would sell it.  
It won't do what I want it to, but only what I tell it.* »  
*Anonyme*

Notre objectif ...

... votre objectif



Crédit photo

# Objectifs du cours

- ▶ initiation à la programmation
- ▶ apprentissage de bons comportements
- ▶ implémentation sur un OS (*operating system*)

# Le Langage Java

## └ Objectifs, évaluations et introduction

### └ Objectifs

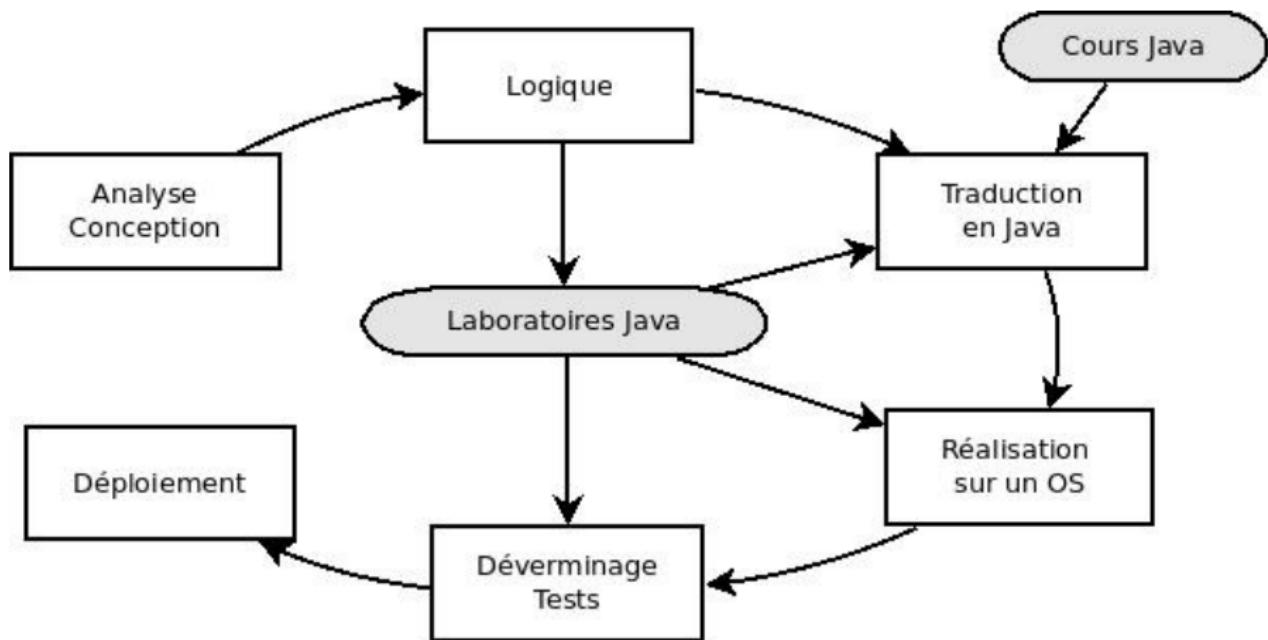
#### └ Objectifs du cours

## Objectifs du cours

- » initiation à la programmation
- » apprentissage de bons comportements
- » implémentation sur un OS (*operating system*)

1. Aborder des thèmes comme : lisibilité, robustesse, documentation, tests
2. Capacités de dévemrinage
3. Autonomie dans le travail
4. On peut aussi aborder le choix du langage. Pourquoi Java ?

# Liens avec les autres cours



# Supports et ressources

## Rien

- ▶ pas de syllabus ;
- ▶ pas de livre ;

## Quoique

- ▶ les slides sur github ;
- ▶ des liens, des documents, ... sur poÉSI
- ▶ un forum de discussion, (fora)



Évaluation à quelle sauce ?

Crédit photo

# Évaluation

Évaluation de l'unité d'enseignement (UE),  
une cote pour toutes les activités d'apprentissage (AA) :

DEV1  
ALG - JAV - LAJ



programmer  
*[proh-gram-er]*

an organism that turns caffeine into software

geek.

Crédit photo

# Définitions

Définissons les concepts suivants :

- ▶ Un **programme** ?
- ▶ **Programmer** ?
- ▶ Un **langage de programmation** ?
- ▶ Différence entre **langue** et *langage* ?

# Un programme

La seule chose dont est capable un ordinateur est de réaliser extrêmement rapidement des instructions élémentaires

Toute tâche qu'on veut lui confier doit donc être préalablement décrite comme une suite séquentielle d'instructions (un programme)

# Le Langage Java

- └ Objectifs, évaluations et introduction
  - └ Concepts
    - └ Un programme

## Un programme

La seule chose dont est capable un ordinateur est de réaliser extrêmement rapidement une suite d'instructions élémentaires

Toute tâche qu'on veut lui confier doit donc être préalablement décrite comme une suite séquentielle d'instructions (un programme)

1. [http://en.wikipedia.org/wiki/Source\\_lines\\_of\\_code](http://en.wikipedia.org/wiki/Source_lines_of_code)  
donne des exemples de taille de programmes

# Un langage

Une classe de langages est adaptée à une classe de problèmes . . . et ces problèmes évoluent dans le temps . . .

# Le Langage Java

- └ Objectifs, évaluations et introduction
  - └ Concepts
    - └ Un langage

## Un langage

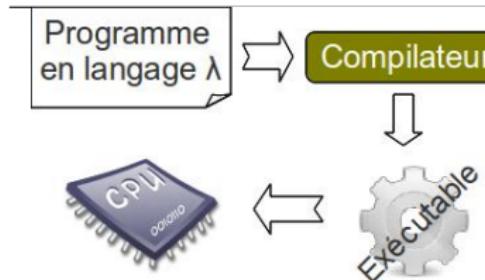
Une classe de langages est adaptée à une classe de problèmes ... et ces problèmes évoluent dans le temps ...

1. On peut aborder ici l'historique des langages : machine, assembleur, haut niveau, structuré, orienté objet, fonctionnels, orientés aspects...

# Le problème de la traduction

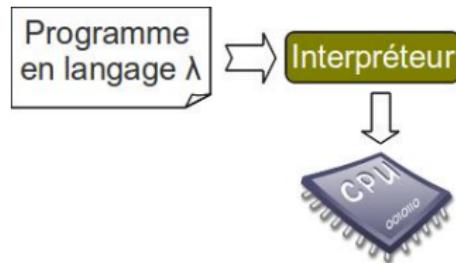
Un ordinateur ne comprend que le langage machine.  
Nécessité d'une **traduction**

## Compilation



*traduit d'une traite  
avant l'exécution*

## Interprétation



*traduit morceau par morceau  
au moment de l'exécution*

# Le Langage Java

## └ Objectifs, évaluations et introduction

### └ Traduction

#### └ Le problème de la traduction

#### Le problème de la traduction

Un ordinateur ne comprend que le langage machine.  
Nécessité d'une traduction



traduit d'une traite  
avant l'exécution



traduit morceau par morceau  
au moment de l'exécution

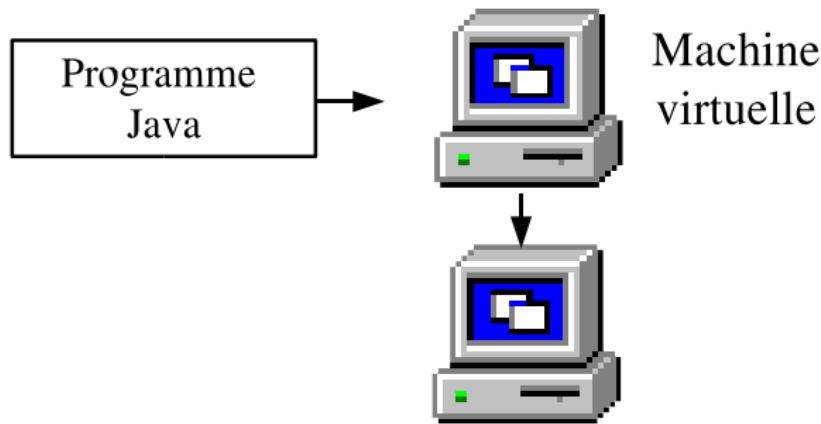
1. Faire une comparaison des avantages/inconvénients

# Et Java ?

Compilé ou interprété ?

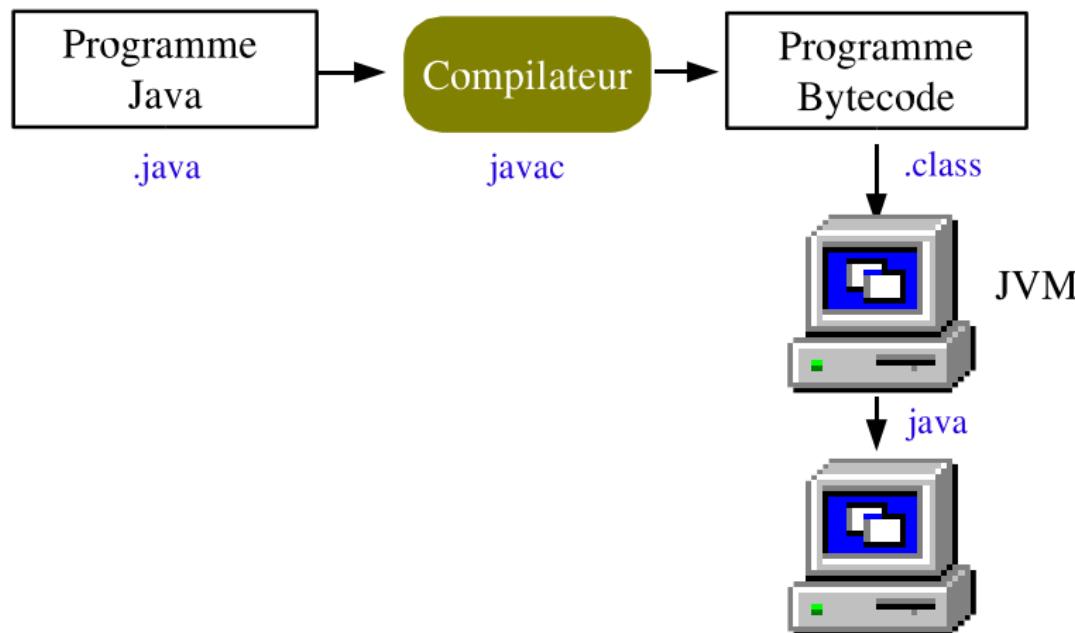
# La machine virtuelle

Java a une approche mixte la **machine virtuelle Java** (JVM)



D'abord compilé  
ensuite interprété

# La machine virtuelle



## *timeline Java*



Crédit photo

# Historique de Java

**92** SUN crée oak (systèmes embarqués).

Auteur : James Gosling

**94** Adapté à Internet grâce aux **applets**.

Devient Java

**96** Première version stable et gratuite de JDK

**98** Sortie de Java 2

**05** Version 1.5 de Java 2

**09** Oracle rachète Sun (et donc Java)

**11** Version 1.7 (Java 7, en GPL)

**14** Version 1.8 (Java 8)



- ▶ Java est-il installé sur ma machine ?
- ▶ Puis-je commencer à écrire un programme Java ?
- ▶ Qu'ai-je pris comme note ?

# Le Langage Java

- └ Objectifs, évaluations et introduction
  - └ Traduction



- » Java est-il installé sur ma machine ?
- » Puis-je commencer à écrire un programme Java ?
- » Qu'ai-je pris comme note ?

1. Introduire les termes : JRE, JDK, ME, SE, EE
2. En fait, il y a un slide là-dessus dans la séance 2

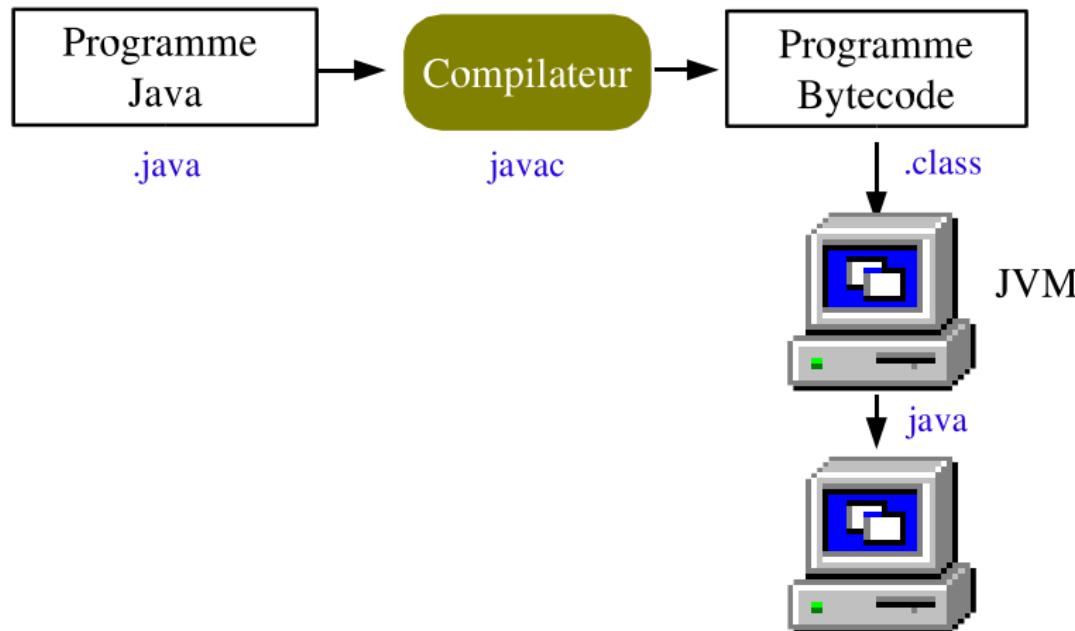
# Séance 2

## Développer en Java, premier survol

- La machine virtuelle
- Les outils de développement
- Algorithmes séquentiels (survol)
- Structure générale d'un programme
- Constantes
- Conventions
- Commentaires

Java est **compilé** puis **interprété**.  
L'interpréteur Java est la machine virtuelle (JVM)  
Le langage de bas niveau interprété par la JVM est le  
**bytecode**

# La machine virtuelle



# Avantages et inconvénients



# Exemple : premier programme

Prenons un exemple (*fichier Hello.java*)

```
// Mon premier programme
public class Hello {
    public static void main(String[] args) {
        System.out.println("Bonjour !");
    }
}
```

Compilons-le \$ javac Hello.java

On obtient la version compilée, le bytecode (*Hello.class*)

On peut l'exécuter \$ java Hello

Bonjour !

# Fourbir ses armes



Crédit photo

# Les outils de développement

## Les éditions de Java

- ▶ **Java SE** (édition standard)
- ▶ **Java ME** (édition mobile - plus léger)
- ▶ **Java EE** (édition entreprise - plus complet)

Où trouver `javac` et `java` ?

**JRE** (*Java Runtime Environment*)

**JDK** (*Java Development Kit*)



Éditer  
Compiler  
Exécuter

# Les outils de développement

## 1

- ▶ Un éditeur avec coloration syntaxique : gVim, Notepad++, nano, ...
- ▶ Gestion manuelle des noms et emplacements des fichiers
- ▶ Compilation et exécution en ligne de commande

# Les outils de développement

## 2

- ▶ Un *E*nvironnement de *D*éveloppement *I*ntégré :  
*Netbeans*, *Eclipse*, ...
- ▶ Intègre tout le processus de développement



- Crédit photo

# Structure générale du programme

```
$cat NomClasse.java
```

```
public class NomClasse {  
    // insert code here  
}
```

**Attention** Java est sensible à la casse

# La méthode principale

```
$cat NomClasse.java
```

```
public class NomClasse {  
    public static void main(String[] args) {  
        // insert code here  
    }  
}
```

# Les variables

## Les types disponibles

En Logique	En Java
Entier	<b>int</b>
Réel	<b>double</b>
Chaine	<b>String</b>
Caractère	<b>char</b>
Booléen	<b>boolean</b>

## Exemple de déclaration

```
int nb1;
```

# L'assignation et les calculs

L'assignation se fait via le symbole

=

```
nb1 = 1;
```

Opérateurs :

+ - \* / %

# Exemple

```
$cat Moyenne.java
```

```
public class Moyenne {  
    public static void main(String[] args) {  
  
        double nombre1;  
        double nombre2;  
        double moyenne;  
  
        nombre1 = 34345;  
        nombre2 = -3213213;  
        moyenne = (nombre1 + nombre2) / 2;  
        System.out.println(moyenne);  
    }  
}
```

# Exemple

```
$cat Moyenne.java
```

```
public class Moyenne {  
    public static void main(String[] args) {  
  
        int nombre1 = 34345;  
        int nombre2 = -321321;  
        double moyenne;  
  
        // division réelle car un des 2 opérandes est réel  
        moyenne = (nombre1 + nombre2) / 2.0;  
        System.out.println ("La moyenne est " + moyenne);  
    }  
}
```

# Lire au clavier

*Les applications modernes préfèrent les lectures dans des champs de saisies.*

Dans une console, voici une manière de faire

## Exemple

```
import java.util.Scanner;  
// ...  
Scanner clavier = new Scanner(System.in);  
// ...  
nombre1 = clavier.nextInt();
```

# Lire au clavier - Exemple

```
$cat Test.java
```

```
import java.util.Scanner;

public class Test {
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        double nombre1;
        double nombre2;
        double moyenne;

        nombre1 = clavier.nextDouble();
        nombre2 = clavier.nextDouble();
        moyenne = (nombre1 + nombre2) / 2.0;
        System.out.println(moyenne);
    }
}
```

# Lire au clavier

Pour lire...	on écrit...
un entier	<code>nextInt()</code>
un réel	<code>nextDouble()</code>
un booléen	<code>nextBoolean()</code>
un mot	<code>next()</code>
une ligne	<code>nextLine()</code>
un caractère	<code>next().charAt(0)</code>

# Constante locale

Une **constante** s'écrit grâce à **final**

## Exemple

```
final int X = 1;  
final int Y;  
Y = 2*X;  
X = 2; // Erreur : possède déjà une valeur  
Y = 3; // Idem
```

## Conventions d'écriture

Ієропол мс I

Ієропол мс I

Ієропол мс I

# Le commentaire

Plusieurs manières d'ajouter un commentaire

```
// Commentaire sur une ligne  
/* Commentaire sur  
plusieurs lignes */
```

## Séance 3

### La gestion des erreurs et le survol des alternatives

- L'erreur est humaine
- Alternatives (survol)

*« There are two ways to write error-free programs;  
only the third one works. » Alan J. Perlis*

# Processus d'écriture d'un programme

Édition / compilation / exécution

... *et tout va bien*

# Quels types d'erreurs ?



Crédit photo

# Quels types d'erreurs ?

Quels types d'erreurs ?

- ▶ Les erreurs de **compilation**
- ▶ Les erreurs d'**exécution**

```
public Class Hello{  
    public static void main (string[] args){  
        System.out.println("Hello");  
    }  
}
```

```
$ javac Hello.java  
Hello.java:1: class, interface, or enum  
expected  
public Class Hello  
          ^
```

# Les erreurs de compilation



- ▶ Compiler souvent
- ▶ Apprendre à reconnaître **rapidement** les erreurs fréquentes
- ▶ Lire / comprendre les messages du compilateur

```
public class Division{  
    ...  
}
```

```
$ javac Division.java  
$ java Division  
Exception in thread "main"  
java.lang.ArithmetricException: / by zero  
at Division.main(Division.java:7)
```

# Les erreurs d'exécution



- ▶ Apprendre à reconnaître **rapidement** les erreurs fréquentes
- ▶ **Déboguer** son code ; la méthode de l'homme pauvre et le débogueur
- ▶ Mettre des tests en œuvre

# Alternatives



# Instructions de choix

## Le Si

```
if ( condition ) {  
    instructions  
}
```

## Le Si-sinon

```
if ( condition ) {  
    instructions  
} else {  
    instructions  
}
```

# Exemple

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int nombre1;

        nombre1 = clavier.nextInt();
        if (nombre1 < 0) {
            System.out.println(nombre1 + " est négatif");
        }
    }
}
```

# Exemple

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int nombre1;

        nombre1 = clavier.nextInt();
        System.out.println(nombre1 + " est un nombre");
        if (nombre1 < 0) {
            System.out.println(" négatif");
        } else {
            System.out.println(" positif");
        }
    }
}
```

# Exercice

Comment traduire cet algorithme ?

```
MODULE test ()  
    nombre1: Entier  
    LIRE nombre1  
    SI nombre1 > 0 ALORS  
        ECRIRE nombre1, "est positif"  
    SINON  
        SI nombre1 = 0 ALORS  
            ECRIRE nombre1, "est nul"  
        SINON  
            ECRIRE nombre1, "est négatif"  
        FIN SI  
    FIN SI  
FIN MODULE
```

# Expressions booléennes

Les comparateurs

< > <= >= == !=

Les opérateurs booléens

&& (et) || (ou) ! (non)

# Exemple

```
import java.util.Scanner;
public class Exemple {
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int nombre1;

        nombre1 = clavier.nextInt();
        if ((nombre1 % 2) == 0) {
            System.out.println ("Le nombre est pair");
        } else {
            System.out.println ("Le nombre est impair");
        }
    }
}
```

# Exemple

```
import java.util.Scanner;
public class Exemple {
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int âge;

        âge = clavier.nextInt();
        if ( âge<21 || âge>=60 ) {
            System.out.println (" Tarif ↴réduit ↴ !");
        }
    }
}
```

# Le « selon-que »

## Première forme

```
switch(produit) {  
    case "Coca" :  
    case "Sprite" :  
    case "Fanta" :  
        prixDistributeur =60;  
        break;  
    case "IceTea" :  
        prixDistributeur =70;  
        break;  
    default :  
        prixDistributeur =0;  
        break;  
}
```

- ▶ Notez le **break**
- ▶ Possible avec : entiers, caractères et chaînes

# Le « selon-que »

Deuxième forme : la logique suivante

Selon que

nb > 0 : Écrire " positif "

nb = 0 : Écrire "nul"

autres : Écrire " négatif "

Fin selon que

s'écrit en Java

```
if (nb>0) {  
    System.out.println (" positif ");  
} else if (nb==0) {  
    System.out.println ("nul");  
} else {  
    System.out.println (" négatif ");  
}
```

# Séance 4

## Lisibilité et notions de modules

- Écrire du code lisible
- Écrire du code illisible
- Refactorisation
- Code modulaire  
(survol)

*« Any fool can write code that a computer can understand.  
Good programmers write code that humans can understand. »*  
*Martin Fowler*

```
public class h{public static void  
main(String[]  
args){System.out.println("Hi");}}
```

# Lisibilité du code

Un code est **souvent lu** ;

- ▶ lorsqu'il est écrit / mis au point ;
- ▶ correction de bug ;
- ▶ évolution du code ;

⇒ **La lisibilité est essentielle**

# Lisibilité du code : indentation

## 1

**Indenter** correctement son code

# Lisibilité du code : choix des noms

## 2

### Bien choisir le **nom des variables**

```
int u=clavier.nextInt(),n=clavier.nextInt(),
t=clavier.nextInt();
double p=u*n*(1+t/100.0);
System.out.println(p);
```

```
package esi.java.cours;

import java.util.Scanner ;

public class CalculPrixVente {
    public static void main ( String[] args ) {
        Scanner clavier = new Scanner ( System.in ) ;
        double àPayer;
        int prixUnitaire = clavier.nextInt();
        int nombreArticles = clavier.nextInt();
        int tauxTVA = clavier.nextInt();

        àPayer = prixUnitaire * nombreArticles * (1 + tauxTVA/100.0);

        System.out.println(àPayer);
    }
}
~
```

# Lisibilité du code : décomposition

## 3

### Décomposer les expressions trop longues

```
àPayer = prixUnitaireHTVA * (1 + tauxTVA/100.0) * nombreArticles;  
  
prixUnitaireTTC = prixUnitaireHTVA * (1 + tauxTVA/100.0);  
àPayer = prixUnitaireTTC * nombreArticles;
```

# Lisibilité du code : constantes

## 4

### Utiliser des **constantes**

```
final double TAUX_TVA = 0.21;
```

# Illisibilité du code

-1

Surcharger de **commentaires**

*Un commentaire explique ce que le code fait mais pas comment il le fait*

A close-up photograph of a neon sign. The word "RULE" is written in a bold, sans-serif font, with each letter composed of a glowing yellow neon tube. The letters are slightly curved and overlap each other. The background is a solid, vibrant red color, which provides a strong contrast to the yellow neon. The overall effect is bright and eye-catching.

RULE

D'autres conventions ?

# Conventions d'écriture

## Conventions d'écriture Java

- ▶ <http://www.oracle.com/...codeconv...>

# La refactorisation

## Refactorisation

*Improving the design of existing code*

JUnit VCS (git/svn)



Découper le code

Crédit photo

# Découper du code

## Pourquoi ?

- ▶ Pour le réutiliser
- ▶ Pour scinder la difficulté
- ▶ Pour faciliter le déverminage
- ▶ Pour accroître la lisibilité
- ▶ Pour diviser le travail

# Découper du code

## Comment ?

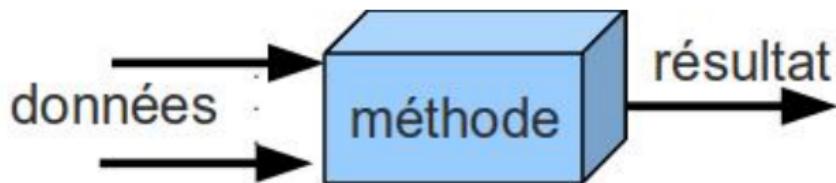
- ▶  $\exists$  un **nom qui décrit tout ce qu'il fait**
- ▶ Il résout un **sous-problème** bien précis
- ▶ Il est **fortement documenté**
- ▶ Il est le plus **général possible**
- ▶ Il tient sur une page

En Java on dit **méthode** et pas **module**

# Appel de méthode

# Appel d'une méthode

Une méthode est une **boite noire**



Pour l'utiliser, il faut connaître :

- ▶ son nom ;
- ▶ ce dont elle a besoin ;
- ▶ ce qu'elle retourne ;
- ▶ mais **pas comment** elle fait ;

Pas de **comment**?  
Un simple **quoi**



# Appel d'une méthode

À partir du code d'une **autre classe**

- ▶ NomClasse.nomMéthode(...)
- ▶ **Exemples :**

```
double racine = Math.sqrt(4.0);
double aléatoire = Math.random();
int nb = -10;
int absolu = Math.abs(nb);
```

*Un appel de méthode au sein de la classe ne nécessite pas de répéter le nom de la classe*

# Définition d'une méthode

# Définition d'une méthode

## Syntaxe

```
public static <typeRetour> nomMéthode ([paramètre, paramètre, ...]) {  
    // instructions  
    <return résultat>;  
}
```

# Définition d'une méthode

**Exemple** : la moyenne de 2 réels

```
public static double moyenne ( double nb1, double nb2 ) {  
    double moyenne = (nb1 + nb2) / 2.0;  
    return moyenne;  
}
```

- ▶ Appel possible (si dans la même classe)

```
double cote = moyenne(12.5, 17.5);
```

# Définition d'une méthode

**Exemple** : la valeur absolue

```
public static int absolu ( int nb ) {  
    int abs;  
    if (nb<0) {  
        abs = -nb;  
    } else {  
        abs = nb;  
    }  
    return abs;  
}
```

## ► Exemples d'appels

```
int résultat = absolu(4);  
int écart = -10;  
int écartAbsolu = absolu(écart);
```

# Définition d'une méthode

## Exemple :

```
public static void présenter (String nomPgm) {  
    System.out.println ("Programme " + nomPgm);  
}
```

### ► Exemple d'appel

```
présenter ("moyenne de 2 nombres");
```

# Définition d'une méthode

## Exemple :

```
public static int lireEntier () {  
    Scanner clavier = new Scanner(System.in);  
    int nb;  
    System.out.println ("Entrez un nombre entier!");  
    nb = clavier.nextInt ();  
    return nb;  
}
```

### ► Exemple d'appel

```
int nb = lireEntier();
```

# Commentaire d'une méthode

## Documentation de Math.abs

### abs

```
public static int abs(int a)
```

Returns the absolute value of an int value. If the argument is not negative, the argument is returned. If the argument is negative, the negation of the argument is returned.

Note that if the argument is equal to the value of `Integer.MIN_VALUE`, the most negative representable int value, the result is that same value, which is negative.

**Parameters:**

a - the argument whose absolute value is to be determined

**Returns:**

the absolute value of the argument.

# Commentaire d'une méthode

Il est essentiel de commenter chaque méthode.

**Exemple** : la valeur absolue

```
/**  
 * Calcul de la valeur absolue.  
 * @param nb le nombre dont on veut la valeur absolue.  
 * @return la valeur absolue de <code>nb</code>  
 */  
public static int absolu ( int nb ) {  
...  
}
```

# Un exemple complet

```
import java.util.Scanner;

public class MaxEntiers {

    /**
     * Donne le maximum de 2 nombres.
     * @param nb1 le premier nombre.
     * @param nb2 le deuxième nombre.
     * @return la valeur la plus grande entre <code>nb1</code> et <code>nb2</code>
     */
    public static int max ( int nb1, int nb2 ) {
        int max=0;
        if (nb1 > nb2) {
            max = nb1;
        } else {
            max = nb2;
        }
        return max;
    }
}
```

( ... )

# Un exemple complet

```
/*
 * Lit un nombre entier.
 * Le nombre est lu sur l'entrée standard (le clavier).
 * @return le nombre entier lu.
 */
public static int lireEntier () {
    Scanner clavier = new Scanner(System.in);
    System.out.println ("Entrez un nombre entier!");
    return clavier.nextInt ();
}

/*
 * Affiche le maximum de 2 nombres entrés au clavier.
 * @param args pas utilisé .
 */
public static void main ( String [] args ) {
    int max; // Le max des nombres lus
    int nb1, nb2; // Chacun des nombres lus
    nb1 = lireEntier ();
    nb2 = lireEntier ();
    max = max(nb1,nb2);
    System.out.println ("max=" + max);
}
```

# Passage de paramètres

# Passage de paramètres

En algorithmique 3 passages de paramètres :

- ▶ en entrée, en sortie, en entrée-sortie

En Java, uniquement **par valeur**

- ▶ = la valeur est copiée dans le paramètre
- ▶  $\simeq$  paramètre en entrée

## Séance 5

### Notion de package et survol des structures répétitives

- Organiser le code

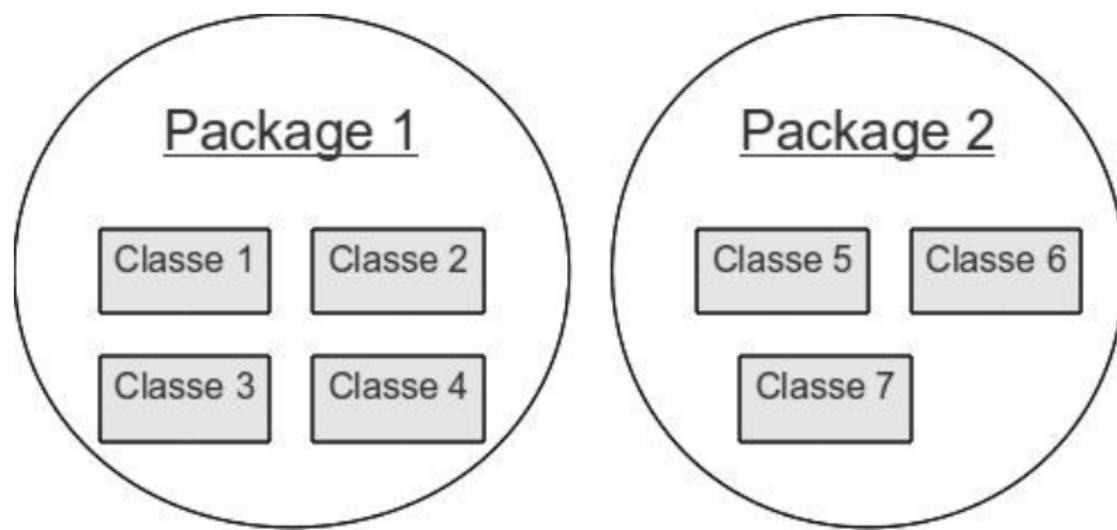
# Rangement



Credit photo

# Le groupement en package

Toutes les classes de l'**API** Java sont regroupées logiquement ... **package**



# La notion de package

## Un **package**

- ▶ Regroupe les classes liées
- ▶ Permet l'unicité des noms de classe
  - nom complet / qualifié : `monPackage.MaClasse`

## Nom d'un package

- ▶ identifiants séparés par des points .
- ▶ tout en minuscules
- ▶ adresse internet inversée (unicité)
- ▶ ex : `be.heb.esi.java1, org.apache.struts.action`

# Utilisation

Pour utiliser une classe

- ▶ mettre le nom **qualifié** (complet)

```
java.util.Calendar now = java.util.Calendar.getInstance();
```

- ▶ ou utiliser **import** qui crée un raccourci

```
import java.util.Calendar;
public Test {
    ...
    Calendar now = Calendar.getInstance();
    ...
}
```

- ▶ Cas particulier : le package **java.lang** est importé implicitement

- Exemple : on peut tout de suite écrire

```
double racine = Math.sqrt(1.21);
```

# Utilisation

Comment savoir comment utiliser les classes et méthodes ? En lisant la **javadoc**

# Utilisation

On peut y lire le nom du package

et la description de la méthode

# Créer ses packages

Pour placer une classe dans un package, la commande est  
**package nom\_package;**

- ▶ Doit être la **première instruction** du fichier
- ▶ Exemple :

```
package be.heb.esi.java1;  
public class Test {  
    // Nom complet : be.heb.esi.java1.Test  
}
```

# Créer ses packages

Qu'est-ce qui va changer en pratique ?

- ▶ La compilation ne change pas :

```
javac NomClasse.java
```

- ▶ L'exécution change :

```
java nomPackage.NomClasse
```

- ▶ Contraintes sur l'endroit où placer le **bytecode**
  - Lié à la notion de **CLASSPATH**
  - Sera détaillé au laboratoire

# Réutiliser du code

Si on doit coder quelque chose, c'est **peut-être déjà fait**

- ▶ ➡ autant le réutiliser
  - Gain de temps
  - Probablement mieux écrit
- ▶ Importance de **connaître l'API**  
(en tout cas les classes principales)

# Crédits

Ce document a été produit avec les outils suivants

- ▶ Les distributions **Ubuntu** et/ou **debian** du système d'exploitation **Linux**
- ▶ **LaTeX/Beamer** comme système d'édition
- ▶ **Git** et **GitHub** pour la gestion des versions et le suivi des corrections
- ▶ Les outils **make**, **rubber**, **pdfnup**, ...

