

Le Langage Java

1^{re} 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
- 6 La gestion des erreurs et les types et les littéraux
- 7 La javadoc
- 8 Tester son code
- 9 Le survol des tableaux
- 10 Les variables locales et les expressions
- 11 Les conversions

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

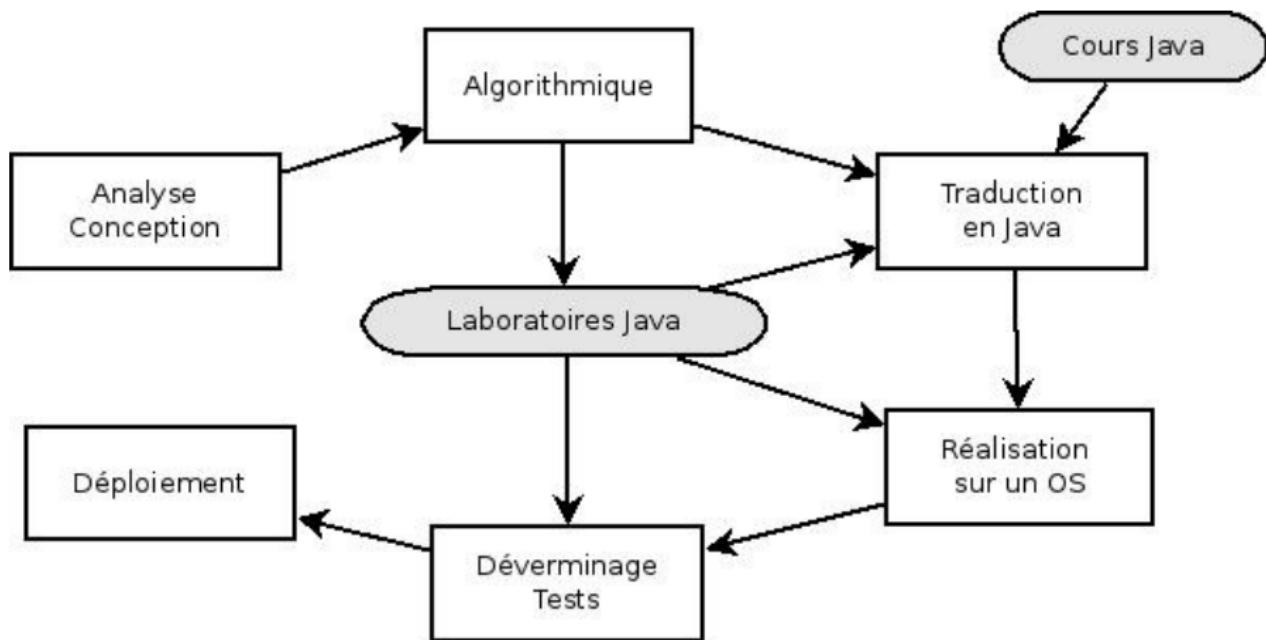


Crédit photo

Objectifs du cours

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

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 ?

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)

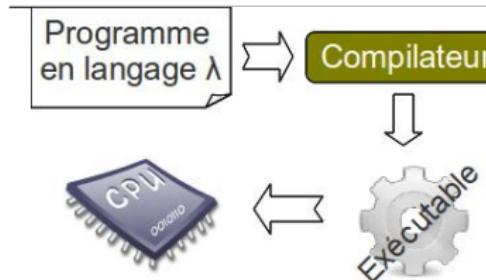
Un langage

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

Le problème de la traduction

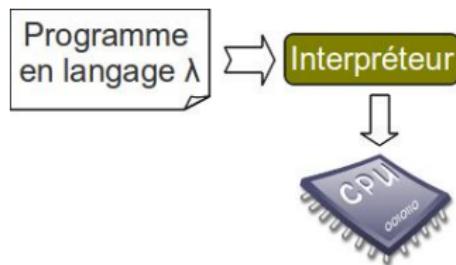
Un ordinateur ne comprend que le langage machine.
Nécessité d'une **traduction**

Compilation



*traduit d'une traite
avant l'exécution*

Interprétation



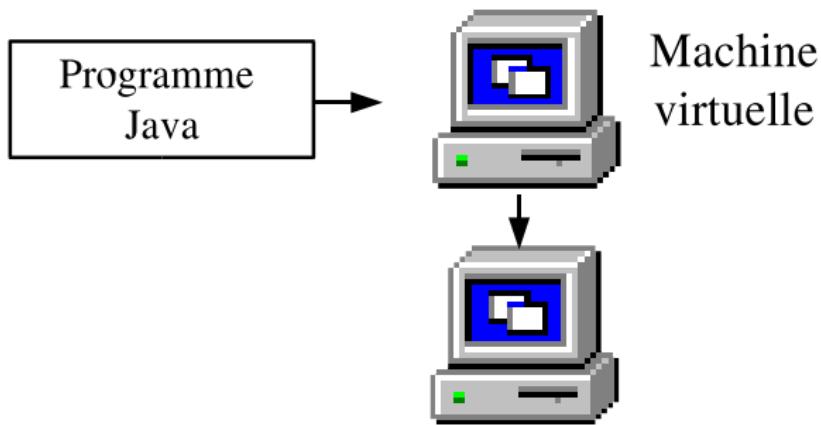
*traduit morceau par morceau
au moment de l'exécution*

Et Java ?

Compilé ou interprété ?

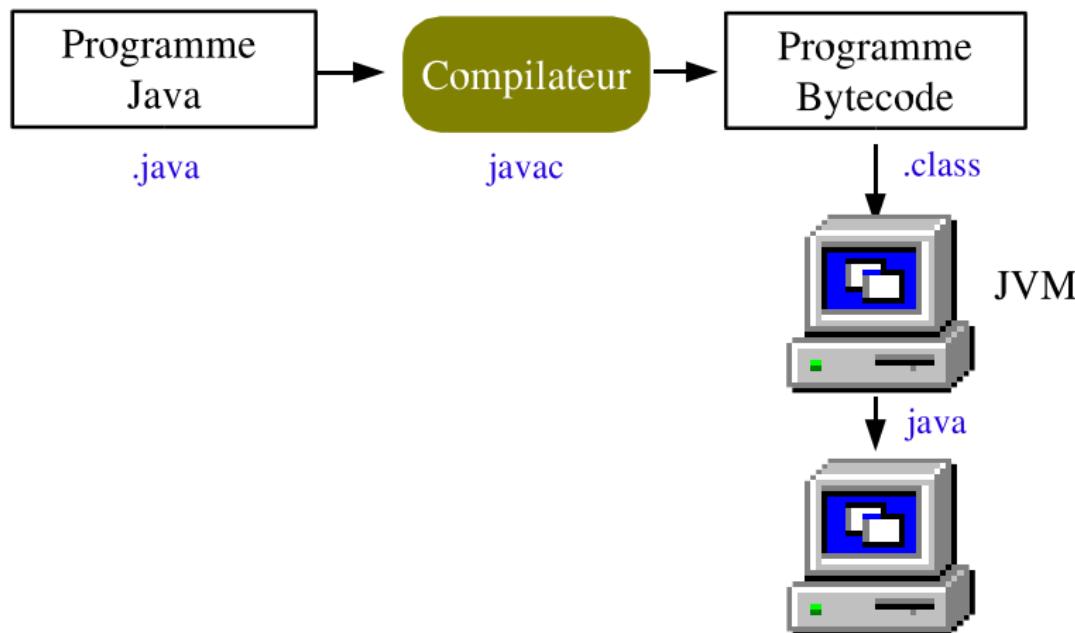
La machine virtuelle

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 notes ?

Séance 2

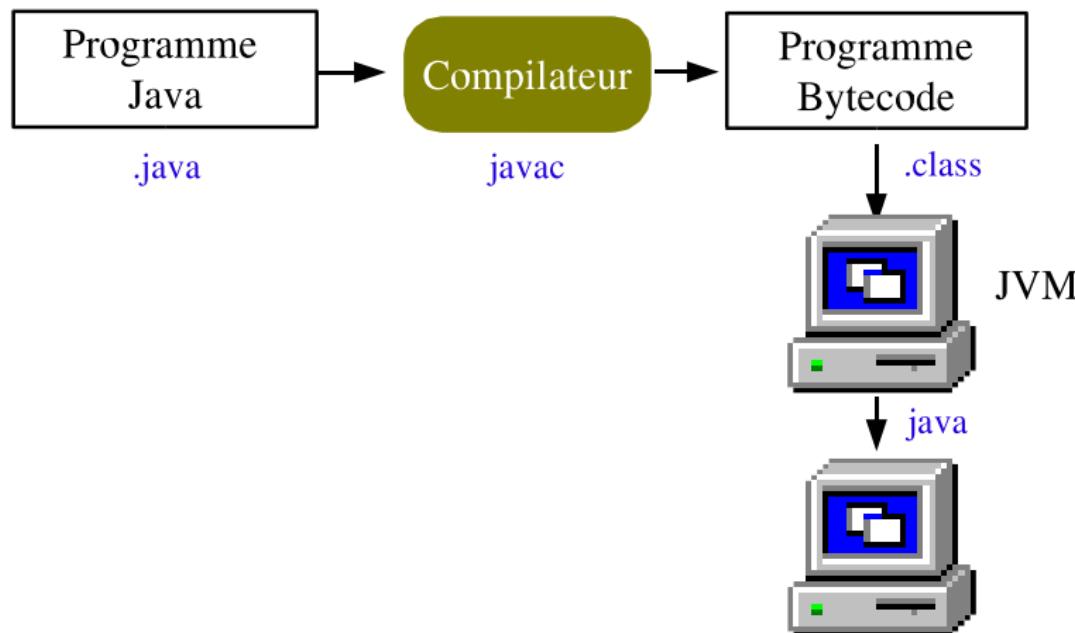
Développer en Java, premier survol

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



Credit photo

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



- credit 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  
    }  
}
```

Exemple

```
public class NomClasse {  
    public static void main(String [] args) {  
        System.out.println ("Bonjour\u00e7!");  
    }  
}
```

Les variables

Les types disponibles

En Algorithmique	En Java
Entier	int
Réel	double
Chaine	char , String
Booléen	boolean

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

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

ІБЛІОЛ МЕ І

ІБЛІОЛ МЕ І

ІБЛІОЛ МЕ І

Conventions de noms

Pour une variable :

- ▶ Tout mettre en minuscules
- ▶ Sauf les débuts de noms composés en majuscule

Pour une constante :

- ▶ Tout mettre en majuscules
- ▶ Utiliser _ pour séparer les mots

Dans tous les cas : être explicite

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/ou 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 ?

```
Algorithme test ()  
    nombre1: Entier  
    Demander nombre1  
    Si nombre1 > 0 Alors  
        Afficher nombre1, "est positif"  
    Sinon  
        Si nombre1 = 0 Alors  
            Afficher nombre1, "est nul"  
        Sinon  
            Afficher nombre1, "est négatif"  
        Fin Si  
    Fin Si  
Fin Algorithme
```

Le « si-sinon-si »

L'exemple précédent s'écrirait mieux (extrait) :

```
Si nombre1 > 0 Alors
    Afficher nombre1, "est positif "
Sinon Si nombre1 = 0 Alors
    Afficher nombre1, "est nul"
Sinon
    Afficher nombre1, "est négatif "
Fin Si
```

ce qui donnerait en Java

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

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 »

```
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

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 "RUNS" 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.

RUNS

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 ?

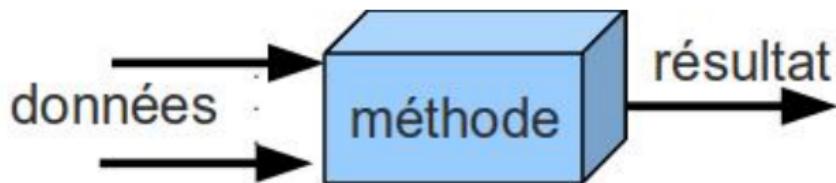
- ▶ \exists un **nom qui décrit tout ce qu'il fait**
- ▶ Il résout un **sous-problème** bien précis
- ▶ Il est **fortement documenté**
- ▶ Il est le plus **général possible**
- ▶ Il tient sur une page

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 resultat>;  
}
```

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 dans la même classe

```
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 dans la même classe

```
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 dans la même classe

```
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
- Les boucles (survol)

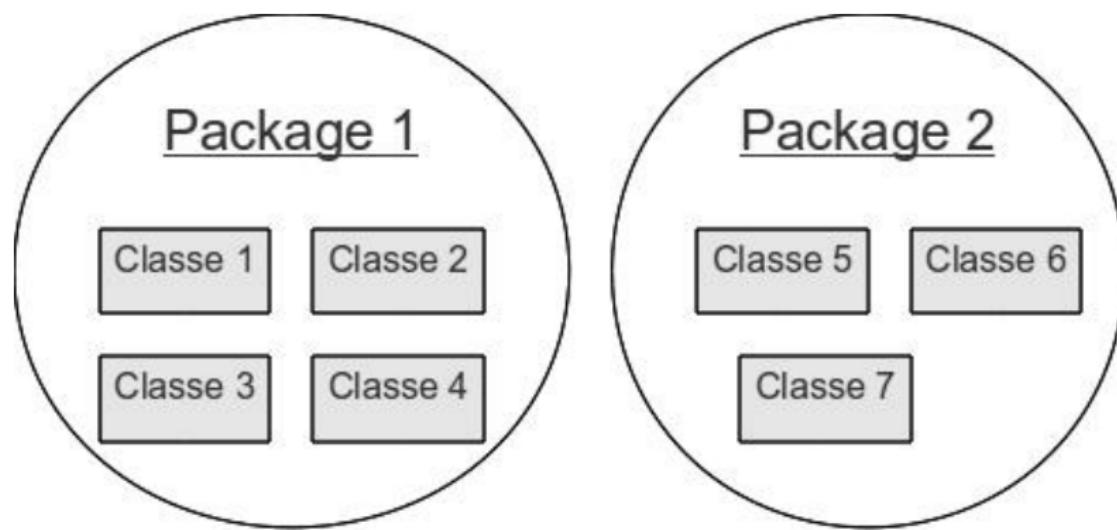
Rangement



Crédit photo

Le groupement en package

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



La notion de package

Un **package** donne un nom complet à une classe

- ▶ `mon.paquet.MaClasse`,
- ▶ `be.heb.esi.java1.MaClasse`,
- ▶ `java.util.Scanner`,
- ▶ `org.apache.struts2.components.Anchor`

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

Utilisation

Comment savoir comment utiliser les classes et méthodes ?

En lisant la **javadoc**

[All Classes](#)**Packages**[java.applet](#)[java.awt](#)[java.awt.color](#)[java.awt.datatransfer](#)[java.awt.dnd](#)[ButtonGroup](#)[ButtonModel](#)[ButtonUI](#)[Byte](#)[ByteArrayInputStream](#)[ByteArrayOutputStream](#)[ByteBuffer](#)[ByteChannel](#)[ByteHolder](#)[ByteLookupTable](#)[ByteOrder](#)[C14NMethodParameterSpec](#)[CachedRowSet](#)[CacheRequest](#)[CacheResponse](#)[Calendar](#)[Callable](#)[CallableStatement](#)[Callback](#)[CallbackHandler](#)[CallSite](#)[CancelablePrintJob](#)[CancellationException](#)[CancelledKeyException](#)[CannotProceed](#)[CannotProceedException](#)[CannotProceedHelper](#)[CannotProceedHolder](#)[CannotredoException](#)[CannotundoException](#)[CanonicalizationMethod](#)[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#)[Summary](#): [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail](#): [Field](#) | [Constr](#) | [Method](#)

java.util

Class Calendar

[java.lang.Object](#)[java.util.Calendar](#)**All Implemented Interfaces:**[Serializable](#), [Cloneable](#), [Comparable<Calendar>](#)**Direct Known Subclasses:**[GregorianCalendar](#)

```
public abstract class Calendar
extends Object
implements Serializable, Cloneable, Comparable<Calendar>
```

The `Calendar` class is an abstract class that provides methods for converting between a specific instant in time and a set of day-of-month, hour, and so on, and for manipulating the calendar fields, such as getting the date of the next week. An instance value that is an offset from the Epoch, January 1, 1970 00:00:00.000 GMT (Gregorian).

The class also provides additional fields and methods for implementing a concrete calendar system outside the package. This is protected.

Like other locale-sensitive classes, `Calendar` provides a class method, `getInstance`, for getting a generally useful object that returns a `Calendar` object whose calendar fields have been initialized with the current date and time:

```
Calendar rightNow = Calendar.getInstance();
```

A `Calendar` object can produce all the calendar field values needed to implement the date-time formatting for a particular language (e.g., Japanese-Gregorian, Japanese-Traditional). `Calendar` defines the range of values returned by certain calendar fields, as well as the fact that each month of the calendar system has value `MONTH == JANUARY` for all calendars. Other values are defined by the concrete subclasses and subclass documentation for details.

Getting and Setting Calendar Field Values

Crédit photo

Utilisation

On peut y lire le nom du package



et la description de la méthode

getInstance

```
public static Calendar getInstance()
```

Gets a calendar using the default time zone and locale. The `Calendar` returned is based on the current time in the default time zone with the default locale.

Returns:

a `Calendar`.

On verra comment produire une javadoc similaire pour son code

Comment créer mes propres
packages ?

Créer ses packages

Commande : **package** <nom du paquet>

```
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 nom.paquet.NomClasse
```

Cela a une incidence sur l'endroit où placer le **bytecode**

[All Classes](#)**Packages**[java.applet](#)[java.awt](#)[java.awt.color](#)[java.awt.datatransfer](#)[java.awt.dnd](#)[ButtonGroup](#)[Bu](#)[Bu](#)[By](#)[By](#)[By](#)[By](#)[By](#)[By](#)[By](#)[By](#)[ByteOrder](#)[C14NMethodParameterSpec](#)[CachedRowSet](#)[CacheRequest](#)[CacheResponse](#)[Calendar](#)[Callable](#)[CallableStatement](#)[Callback](#)[CallbackHandler](#)[CallSite](#)[CancelablePrintJob](#)[CancellationException](#)[CancelledKeyException](#)[CannotProceed](#)[CannotProceedException](#)[CannotProceedHelper](#)[CannotProceedHolder](#)[CannotredoException](#)[CannotundoException](#)[CanonicalizationMethod](#)[Prev Class](#) [Next Class](#) [Frames](#) [No Frames](#)[Summary](#): [Nested](#) | [Field](#) | [Constr](#) | [Method](#) [Detail](#): [Field](#) | [Constr](#) | [Method](#)

java.util

Class Calendar

java.lang.Object

java.util.Calendar

All Implemented Interfaces:

Connaitre l'API, c'est gagner du temps !

```
extends Object
implements Serializable, Cloneable, Comparable<Calendar>
```

The `Calendar` class is an abstract class that provides methods for converting between a specific instant in time and a set of date and time fields, such as `DAY_OF_MONTH`, `HOUR`, and so on, and for manipulating the calendar fields, such as getting the date of the next week. An instance of `Calendar` also contains a value that is an offset from the *Epoch*, January 1, 1970 00:00:00.000 GMT (Gregorian).

The class also provides additional fields and methods for implementing a concrete calendar system outside the package. These fields are `protected`.

Like other locale-sensitive classes, `Calendar` provides a class method, `getInstance`, for getting a generally useful object of type `Calendar`. This method returns a `Calendar` object whose calendar fields have been initialized with the current date and time:

```
Calendar rightNow = Calendar.getInstance();
```

A `Calendar` object can produce all the calendar field values needed to implement the date-time formatting for a particular language (e.g., Japanese-Gregorian, Japanese-Traditional). `Calendar` defines the range of values returned by certain calendar fields, as well as the meaning of the values. For example, the first month of the calendar system has value `MONTH == JANUARY` for all calendars. Other values are defined by the concrete subtypes of `Calendar`. See the documentation and subclass documentation for details.



Loop the loop !

Crédit photo

Instructions répétitives

Le Tant que :

```
while ( condition ) {  
    instructions  
}
```

Exemple :

```
int puissance = 1;  
while ( puissance < 1000 ) {  
    System.out.println ( puissance );  
    puissance = 2 * puissance;  
}
```

Exemple

```
import java.util.Scanner;
public class Exemple {
    /**
     * Affiche la somme d'entiers positifs entrés au clavier .
     * S'arrête dès qu'une valeur nulle ou négative est donnée.
     * @param args non utilisé
     */
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int nb;
        int somme = 0;
        nb = clavier.nextInt();
        while ( nb > 0 ) {
            somme = somme + nb;
            nb = clavier.nextInt();
        }
        System.out.println(somme);
    }
}
```

Instructions répétitives

Le **Pour** :

```
for ( int i=début; i<=fin; i=i+pas ) {  
    instructions  
}
```

Exemple :

```
for ( int i=1; i<=10; i=i+1 ) {  
    System.out.println ( i );  
}
```

Exemple

```
public class Exemple {  
    /**  
     * Affiche la somme des nombres pairs entre 2 et 100.  
     * @param args non utilisé  
     */  
    public static void main(String[] args) {  
        int somme;  
  
        somme = 0;  
        for ( int i=2; i<=100; i=i+2 ) {  
            somme = somme + i;  
        }  
        System.out.println (somme);  
    }  
}
```

Exemple

```
public class Exemple {  
    /**  
     * Affiche un compte à rebours à partir de 10.  
     * @param args non utilisé  
     */  
    public static void main(String[] args) {  
  
        for ( int i=10; i>=1; i=i-1 ) {  
            System.out.println (i);  
        }  
        System.out.println ("Partez\u25b6!");  
    }  
}
```

Instructions répétitives

`i++` est un raccourci pour `i=i+1`

Exemple :

```
for ( int i=1; i<=n; i++ ) {  
    System.out.println (i);  
}
```

Étude de cas

Lecture d'une donnée entière positive

Étude de cas

► Étape 1 : lire un entier

```
/**  
 * Lit un entier au clavier.  
 * Les valeurs non entières sont passées.  
 * @return l'entier lu.  
 */  
public static int lireEntier () {  
    Scanner clavier = new Scanner(System.in);  
    int nb;  
    // Tant que ce n'est pas un entier au clavier  
    while ( ! clavier.hasNextInt() ) {  
        clavier.next(); // le lire, le passer  
    }  
    nb = clavier.nextInt ();  
    return nb;  
}
```

Étude de cas

► Étape 2 : lire un entier positif

```
/**  
 * Lit un entier au clavier.  
 * Les valeurs non entières, nulles ou négatives sont passées.  
 * @return l'entier lu.  
 */  
public static int lirePositif () {  
    int nb;  
    nb = lireEntier ();  
    while (nb<=0) {  
        nb = lireEntier ();  
    }  
    return nb;  
}
```

Étude de cas

► Étape 3 : un exemple de main

```
/**  
 * Un exemple de main.  
 */  
public static void main(String[] args){  
    int nombreLu;  
  
    System.out.print("Entrez un entier positif : ");  
    nombreLu = lirePositif();  
}
```

Séance 6

La gestion des erreurs et les types et les littéraux

- Écrire du code robuste
- Gérer les erreurs
- Confiner les problèmes
- Les types et les littéraux
- Les types entiers
- Les types flottants
- Les booléens
- La chaîne de caractères



Environnement défaillant

Credit photo

Motivation

Un programme ne tourne pas dans un monde idéal

Il doit pouvoir **résister aux défaillances** de l'environnement

- ▶ On tente d'ouvrir un fichier qui n'existe pas
- ▶ L'utilisateur entre des données incorrectes
- ▶ ...

La gestion des erreurs

Exemple :

```
import java.util.Scanner;
public class Affiche {
    /**
     * Affiche un nombre entier lu au clavier .
     * @param args non utilisé
     */
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int nb;

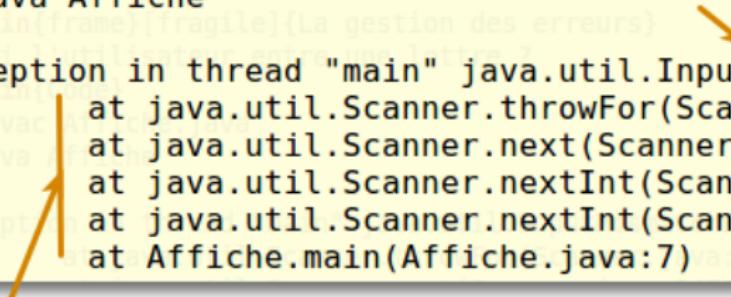
        nb = clavier.nextInt();
        System.out.println(nb);
    }
}
```

La gestion des erreurs

Et si l'utilisateur entre une lettre ?

```
> javac Affiche.java
> java Affiche
a
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:857)
        at java.util.Scanner.next(Scanner.java:1478)
        at java.util.Scanner.nextInt(Scanner.java:2108)
        at java.util.Scanner.nextInt(Scanner.java:2067)
        at Affiche.main(Affiche.java:7)
```

Nom de l'exception



Pile d'appels (par où il est passé)

Comment (essayer)
de gérer le problème ?

catch



La gestion des erreurs

L'instruction **try catch**

- ▶ **try** : contient les instructions qui **peuvent mal se passer**
- ▶ **catch** : contient le code qui est **en charge** de gérer le problème

Quand un problème se présente dans le **try**

- ▶ Le code du **try** est interrompu
- ▶ Le code du **catch** est exécuté
- ▶ Ensuite, on continue après le **try–catch**

La gestion des erreurs

Exemple : on affiche un message plus clair

```
package be.heb.esi.lgj1;
import java.util.Scanner;
public class Affiche {
    /**
     * Affiche l'entier lu au clavier ou un message si ce n'est pas un entier.
     * @param args inutilisé .
     */
    public static void main(String[] args) {
        Scanner clavier = new Scanner(System.in);
        int nb;
        try {
            nb = clavier.nextInt();
            System.out.println(nb);
        }
        catch(Exception e) {
            System.out.println("Ce n'est pas un entier!");
        }
    }
}
```

Confiner les problèmes - Motivation

Imaginons la situation suivante :

- ▶ Un programme demande un entier à l'utilisateur
- ▶ Il doit être positif
- ▶ L'utilisateur entre un nombre négatif
- ▶ Le programme ne le vérifie pas tout de suite

Confiner les problèmes - Motivation

On aura un problème :

- ▶ Un plantage
- ▶ Un résultat erroné
- ▶ Un effet indésirable (perte de données, ...)

Mais le problème va survenir :

- ▶ Plus tard dans le temps
- ▶ Plus loin dans le code

⇒ **Difficile** à comprendre et **corriger**

Confiner les problèmes

Besoin de **confiner** les problèmes

- Un problème est **détecté rapidement** avant qu'il ne se propage dans le reste du code

Cas pratique : vérifier les **paramètres**

- Si une contrainte est associée à un paramètre
 - Le vérifier en début de méthode
 - Que faire si pas valide ?

Lancer une exception

Confiner les problèmes

Lancer une exception

```
/*
 * Calcule la racine carrée d'un nombre.
 * @param nb le nombre dont on veut la racine carée.
 * @return la racine carrée de <code>nb</code>.
 * @throws IllegalArgumentException si <code>nb</code> est négatif.
 */
public static double racineCarrée(double nb) {
    if (nb<0) {
        throw new IllegalArgumentException("nb doit être positif !");
    }
    // Traitement normal. On est sûr que le paramètre est OK.
}
```

Confiner les problèmes

Exemple

```
try {  
    System.out.println( racineCarrée( val ) );  
} catch (Exception ex) {  
    System.out.println( "Calcul_impossible!" );  
}
```

On peut aussi préciser qu'on n'attrape **que** les
IllegalArgumentException

```
try {  
    System.out.println( racineCarrée( val ) );  
} catch (IllegalArgumentException ex) {  
    System.out.println( "Calcul_impossible!" );  
}
```

Java est un langage fortement typé

Légende urbaine ?

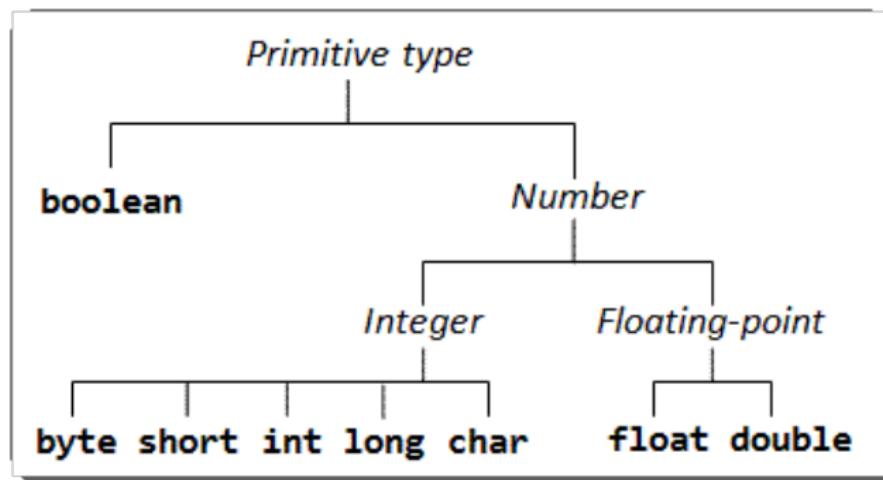
Les types

Toute donnée a un type

Quels types ?

- ▶ **primitifs prédéfinis :**
 - entier, réel, booléen (logique)
- ▶ **références prédéfinis :**
 - tableaux, String, ...
- ▶ **références définis par le programmeur**

Les types primitifs



source : http://www3.ntu.edu.sg/home/ehchua/programming/java/J2_Basics.html



Les entiers

Credit photo

Les types numériques entiers

byte, short, int et **long** (**char** sera vu à part) :

- ▶ nombres signés (en complément à 2)
- ▶ codés sur 8-bit, 16-bit, 32-bit et 64-bit
- ▶ comprennent donc les valeurs
 - -128 à 127
 - -32768 à 32767
 - -2147483648 à 2147483647
 - -9223372036854775808 à 9223372036854775807

Un *littéral*, c'est la
représentation d'une valeur

Les littéraux entiers

Un littéral numérique **décimal**

- ▶ suite de chiffres
- ▶ _ éventuellement pour la lisibilité
- ▶ le suffixe (`I` ou `L`) : distingue un **int** d'un **long**
- ▶ pas de **byte** ou **short**, un littéral est **int** (ou **long**)

Les littéraux entiers

Un littéral numérique **octal**

- ▶ suite de chiffres de 0 à 7
- ▶ précédé de 0

Les littéraux entiers

Un littéral numérique **hexadécimal**

- ▶ suite de chiffres et lettres a,b,c,d,e,f
(minuscules/majuscules)
- ▶ précédé de **0x** ou **0X**

Les littéraux entiers

Un littéral numérique **binaire**

- ▶ suite de chiffres 0 et 1
- ▶ précédé de 0b ou 0B

Les littéraux entiers

Exemple La quantité 100 de type **int**

- ▶ 100
- ▶ 1_0_0
- ▶ 0144
- ▶ 01_44
- ▶ 0x64
- ▶ 0b110_0100



char
un type numérique particulier

Crédit photo

Le type numérique caractère

char

- ▶ caractère Unicode codé en UTF16
- ▶ entier non signé sur 16 bits
- ▶ assimilé à un entier
- ▶ un littéral de type **char**
un **caractère entre *single quote***
- ▶ les séquences d'échappement \n , \t , \' , \"



Les pseudo-réels

Crédit photo

Les types à virgule flottante

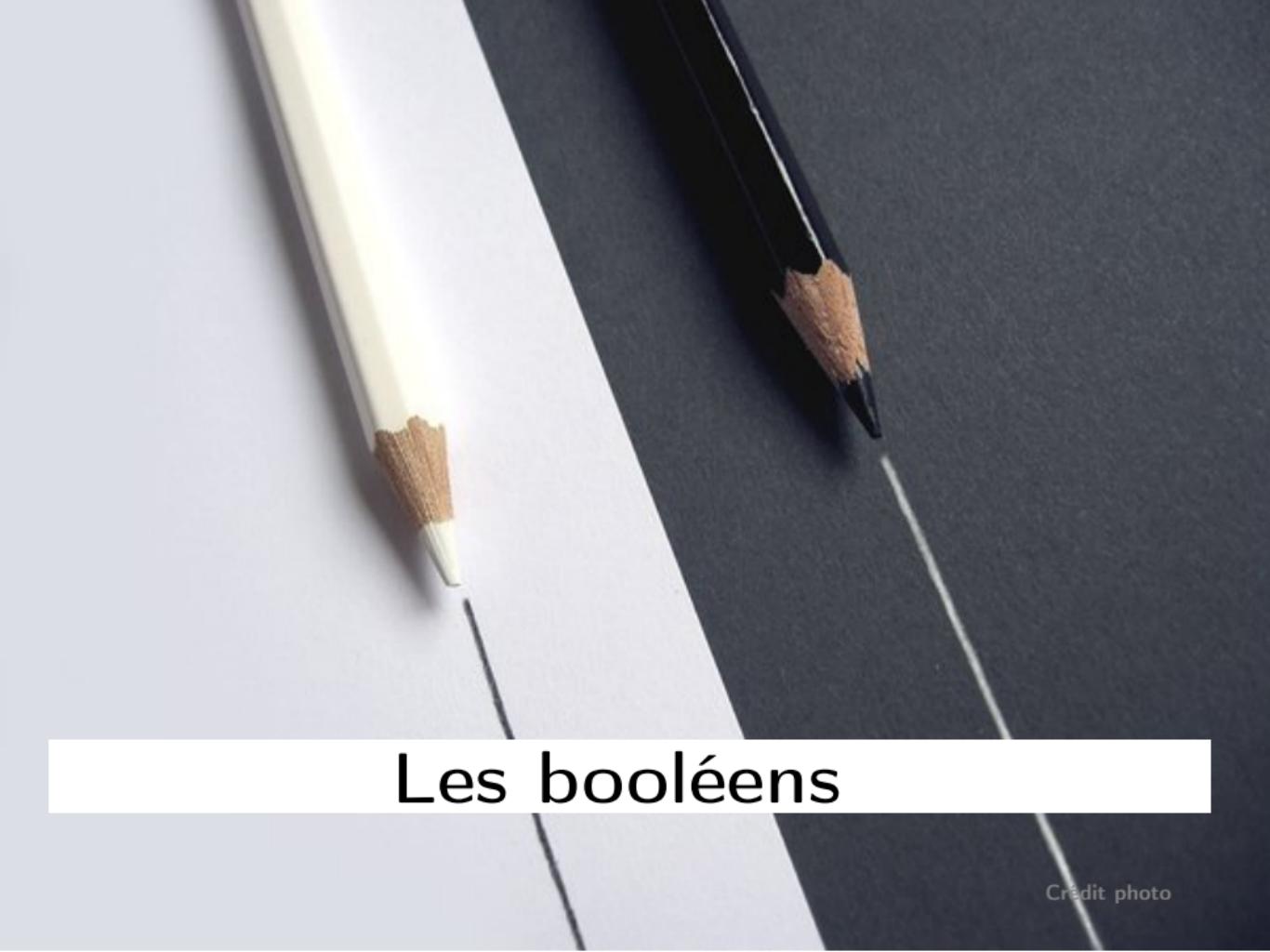
float, double

- ▶ respectent la norme IEEE754
- ▶ codés sur (respectivement) 32-bit, et 64-bit
- ▶ on utilisera plus souvent le type **double**
- ▶ **modélisation** de la notion mathématique

Les littéraux à virgule flottante

partie entière	.	partie décimale	E	exposant	suffixe
----------------	---	-----------------	---	----------	---------

- ▶ 4 parties optionnelles (mais pas ensemble)
- ▶ en l'absence de suffixe : un double
- ▶ exemples : 1.2E3, 1.F, .1, 1e-2d, 1f
- ▶ contre-exemples : 1, .E1, E1



Les booléens

Crédit photo

Le type booléen

boolean

- ▶ appelé aussi **logique**
- ▶ 2 valeurs : **true** (vrai) et **false** (faux)

La chaîne de caractères

La chaîne de caractères

String

- des caractères entre *double quote*
- **char** \neq String

Séance 7

La javadoc

- La documentation Java
 - javadoc
 - Les tags
 - Étude de cas

« *It's not a bug - it's an undocumented feature.* »
Author Unknown

La documentation



Crédit photo

Motivation

Pour qui ?
Qu'écrire ?

Motivation

Où mettre la documentation ?

- ▶ Avec le code
 - Plus facile pour le maintenir
 - Plus de chance de garder la synchronisation avec le code
- ▶ Mais le programmeur-utilisateur n'a pas à voir le code pour l'utiliser

Motivation



literate programming

- ▶ la documentation accompagne le code
- ▶ un outil extrait cette documentation pour en faire un document facile à lire
- ▶ toute la documentation suit la même structure, le même style
 - plus facile à lire



A surreal landscape featuring a large, dark, monolithic concrete structure in the foreground, resembling a giant's finger or a stylized doorway. The background is filled with numerous cherry blossom trees in full bloom, their branches heavy with pink flowers. The sky is a dramatic, cloudy expanse, with bright white clouds contrasting against a dark, overcast upper portion. The overall atmosphere is dreamlike and artistic.

code → *javadoc* → doc

Javadoc

javadoc

- ▶ Commentaire javadoc identifié par `/** ... */`

```
/**  
     Calcule et retourne le maximum de 2 nombres.  
*/
```

- ▶ documentation produite au format HTML
- ▶ On commente essentiellement
 - la classe : rôle et fonctionnement
 - les méthodes publiques : ce que ça fait, paramètres et résultats
- ▶ Se met **juste au dessus** de ce qui est commenté

Les tags

Utilisation de **tags** pour identifier certains éléments

Les plus courants :

- ▶ **@param** : décrit les paramètres
- ▶ **@return** : décrit ce qui est retourné
- ▶ **@throws** : spécifie les exceptions lancées
- ▶ **@author** : note sur l'auteur

Les tags

Exemple

```
/*
 * Donne la racine carrée d'un nombre.
 * @param nb le nombre dont on veut la racine carrée .
 * @return la racine carrée du nombre.
 * @throws IllegalArgumentException si le nombre est négatif.
 */
public static double sqrt( double nb ) {...}
```

- ▶ Les types sont déduits de la signature et ajoutés à la documentation
- ▶ La première phrase (terminée par un `.`) sert de résumé

Les tags

Résumé

Modifier and Type	Method and Description
static double	<code>sqrt(double nb)</code> Donne la racine carrée d'un nombre.

Détail

sqrt

```
public static double sqrt(double nb)
```

Donne la racine carrée d'un nombre.

Parameters:

nb - le nombre dont on veut la racine carrée.

Returns:

la racine carrée du nombre.

Throws:

`java.lang.IllegalArgumentException` - si le nombre est négatif.

Le code HTML

Peut contenir des balises HTML

Exemple :

```
/*
 * Indique si l'année est bissextile . Pour rappel :
 * <ul>
 *   <li>Une année qui n'est pas divisible par 4 n'est pas bissextile
 *     (ex: 2009)</li>
 *   <li>Une année qui est divisible par 4</li>
 *     <ul>
 *       <li>est en général bissextile (ex: 2008)</li>
 *       <li>sauf si c'est un multiple de 100 mais pas de 400 (ex: 1900, 2100)</li>
 *       <li>les multiples de 400 sont donc bien bissextiles (ex: 2000, 2400)</li>
 *     </ul>
 *   </ul>
 * Plus formellement, <code>a</code> est bissextile si et seulement si <br/>
 * <code>a MOD 400 = 0 OU (a MOD 4 = 0 ET a MOD 100 != 0)</code>
 * @param année l'année dont on se demande si elle est bissextile
 * @return vrai si l'année est bissextile
 */
```

Le code HTML

Ce qui donne

estBissextile

```
public static boolean estBissextile(int année)
```

Indique si l'année est bissextile. Pour rappel :

- Une année qui n'est pas divisible par 4 n'est pas bissextile (ex: 2009)
- Une année qui est divisible par 4
 - est en général bissextile (ex: 2008)
 - sauf si c'est un multiple de 100 mais pas de 400 (ex: 1900, 2100)
 - les multiples de 400 sont donc bien bissextiles (ex: 2000, 2400)

Plus formellement, a est bissextile si et seulement si

$$a \bmod 400 = 0 \text{ OU } (a \bmod 4 = 0 \text{ ET } a \bmod 100 \neq 0)$$

Parameters:

année - l'année dont on se demande si elle est bissextile

Returns:

vrai si l'année est bissextile

Production de la documentation

Commande javadoc

- ▶ `javadoc Temps.java`
- ▶ `javadoc *.java`
- ▶ `javadoc -d doc *.java`
- ▶ `javadoc --charset utf-8 *.java`
- ▶ ... et beaucoup d'autres options
(cf. documentation de javadoc)

Une bonne documentation

Une bonne *javadoc* décrit le **quoi** mais jamais le **comment**

- ▶ → Ne jamais parler de ce qui est privé
- ▶ Mauvais exemples :
 - *On utilise un for pour parcourir le tableau.*
 - *Pour aller plus vite, on stocke le prix hors tva dans une variable temporaire.*

Une bonne documentation

Ne pas écrire ce que javadoc écrit lui-même :

- ▶ Mauvais exemples :
 - *nb - un entier qui ...*
 - *La méthode sqrt ...*
 - *Cette méthode ne retourne rien.*
- ▶ Pour en savoir plus :

<http://www.oracle.com/technetwork/articles/java/index-137868.html>



Étude de cas

Séance 8

Tester son code

- Présentation
- Les tests
- Plan de tests
- JUnit
- Conclusion

« The best performance improvement is the transition from the nonworking state to the working state. » J. Osterhout

Tests



Credit photo

Présentation

Tous les programmes réels contiennent des **bugs** (erreurs, défauts)

- ▶ Parfois même **beaucoup**
- ▶ **Inacceptable**
 - **Inconfort** pour l'utilisateur
 - **Perte** de temps, d'argent, de données, de matériel
 - Voir **danger** pour la vie humaine

Présentation

Rappel des types d'erreurs

- ▶ À la **compilation**
- ▶ À l'**exécution**, le programme **s'arrête**
- ▶ À l'**exécution**, le programme fournit une **mauvaise réponse**

On pourrait aussi parler d'autres défauts

- ▶ Trop **lent**
- ▶ Trop gourmand en **mémoire**

« J'ai fait tourner le programme ;
il fonctionne très bien. »

Présentation

Pour produire un logiciel **sans bug** il faut

- ▶ Suivre une **méthodologie** éprouvée
 - Pour produire une première version avec peu de bugs (cf. *Analyse*)
- ▶ **Tester, tester** et ... **tester** encore !
 - Pour détecter ceux qui restent
 - Besoin d'outils pour nous aider
 - Le plus facile/rapide possible

Les tests

Il existe plusieurs sortes de tests : unitaires, d'intégration, fonctionnels, non-régression, ...

Nous nous intéressons aux tests **unitaires**

Pour en savoir plus : [http://fr.wikipedia.org/wiki/Test_\(informatique\)](http://fr.wikipedia.org/wiki/Test_(informatique))

Test unitaire :

Procédure permettant de tester le bon fonctionnement d'un module (une méthode)

A person with blonde hair, wearing a black tricorn hat with a red rose and a gold chain, and a red velvet jacket with gold embroidery, is looking down at a weathered, yellowed treasure map. The map features a grid system with 'N' (North) at the top, 'S' (South) at the bottom, 'E' (East) on the right, and 'W' (West) on the left. A large blue dot marks a specific location on the map. The background shows a vast, calm sea under a cloudy sky.

planning

Crédit photo

Plan de tests

- ▶ Planifier les tests
- ▶ Choisir les cas intéressants / judicieux
- ▶ Se baser sur les erreurs fréquentes

Expérience

Plan de tests - Exemple

Exemple

public static int max(int n1, int n2, int n3) ...

qui calcule la valeur maximale de trois nombres

- ▶ Que tester en plus du cas général ?
 - Le maximum est la première/dernière valeur
 - Présence de nombres négatifs

Plan de tests - Exemple

Plan de tests de la méthode *max*

#	nombres	résultat	ce qui est testé
1	1, 3, 0	3	cas général
2	1, -3, -4	1	maximum au début
3	1, 3, 11	11	maximum à la fin
4	-1, -3, -4	-1	que des négatifs

Plan de tests

Quand tester ? Le plus **souvent** possible

- ▶ Erreur plus facile à identifier/corriger
- ▶ Idéalement après chaque méthode écrite

Que tester ? **Tout**

- ▶ Le nouveau code peut mettre en évidence un problème dans le code ancien (**régression**)

JUnit

Comment tester ?

- ▶ Pas à la main : intenable
- ▶ Besoin d'un outil **automatisé**



JUnit

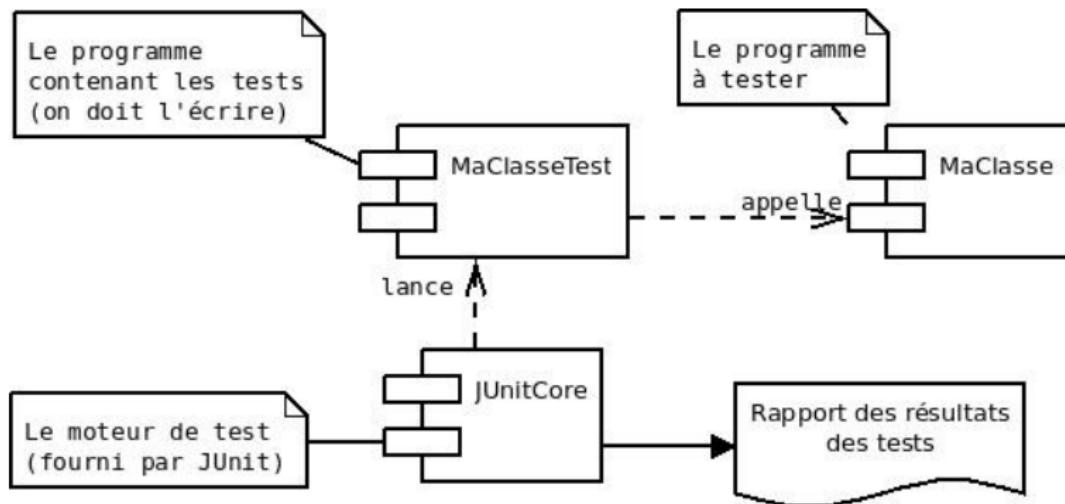
Crédit photo

JUnit - En pratique

La classe de test contient **une méthode** de test **par cas**

- ▶ Autonome (ne reçoit rien ne retourne rien)
- ▶ Contient des **affirmations**
 - Appel de la méthode à tester
 - **Comparaison** entre le résultat **attendu** et le résultat **obtenu**

JUnit



JUnit - En pratique

Exemple

```
@Test  
public void max_cas1() {  
    int n1 = 1, n2 = 3, n3 = 0;  
    assertEquals( 3, MaClasse.max(n1, n2, n3) );  
}
```

- ▶ Reconnu comme un test unitaire grâce à l'**annotation** `@Test`
- ▶ Pas de **static**
- ▶ `assertEquals` vérifie que les 2 valeurs sont identiques
- ▶ `assertTrue(val)`, `assertFalse(val)`, ...

JUnit - En pratique

Comment **lancer** les tests ?

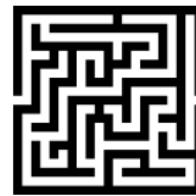
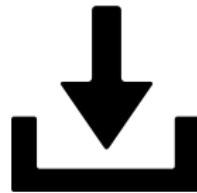
```
java org.junit.runner.JUnitCore mon.paquet.MaClasseTest
```

Résultats :

- ▶ Nombre de tests effectués / réussis
- ▶ Détail sur les tests ratés
 - Nom du test
 - Résultat obtenu comparé au résultat attendu

JUnit - En pratique

Où se trouve la classe `org.junit.runner.JUnitcore` ?



JUnit - Exemple

Exemple

```
package be.heb.esi.java1;
public class Outil {
    public static int max(int n1, int n2, int n3) {
        int max = 0;
        if (n1 > max){
            max = n1;
        }
        if (n2 > max){
            max = n2;
        }
        if (n3 > max){
            max = n3;
        }
        return max;
    }
}
```

JUnit - Exemple

Exemple (suite)

```
package be.heb.esi.java1;

import org.junit.Test;
import static org.junit.Assert.*;

public class UtilTest {
    @Test public void max_cas1() {
        assertEquals(3, Util.max(1, 3, 0));
    }

    @Test public void max_cas4() {
        assertEquals(-1, Util.max(-1, -3, -4));
    }

    // + plus les cas 2, 3 et 5
}
```

JUnit - Exemples

```
$ javac Outil*.java  
$ java org.junit.runner.JUnitCore  
    be.heb.esi.java1.OutilTest
```

```
jUnit version 4.11
..E
Time: 0,013
There was 1 failure:
1) max_cas4(be.heb.esi.java1.UtilTest)
java.lang.AssertionError: expected:<-1> but was:<0>
    at org.junit.Assert.fail(Assert.java:88)
    at org.junit.Assert.assertNotEquals(Assert.java:743)
    at org.junit.Assert.assertEquals(Assert.java:118)
    at org.junit.Assert.assertEquals(Assert.java:555)
    at org.junit.Assert.assertEquals(Assert.java:542)
    at org.junit.Assert.assertEquals(Assert.java:13)
    at be.heb.esi.java1.UtilTest.max_cas4(UtilTest.java:13)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:483)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:47)
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:47)
```

java.lang.AssertionError: expected:<-1> but was:<0>

```
at org.junit.runners.ParentRunner$5.run(ParentRunner.java:250)
at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:63)
at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:236)
at org.junit.runners.ParentRunner.access$000(ParentRunner.java:53)
at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:229)
at org.junit.runners.ParentRunner.run(ParentRunner.java:309)
at org.junit.runners.Suite.runChild(Suite.java:127)
at org.junit.runners.Suite.runChild(Suite.java:26)
at org.junit.runners.ParentRunner$3.run(ParentRunner.java:238)
at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:63)
at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:236)
at org.junit.runners.ParentRunner.access$000(ParentRunner.java:53)
at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:229)
at org.junit.runners.ParentRunner.run(ParentRunner.java:309)
at org.junit.runner.JUnitCore.run(JUnitCore.java:160)
at org.junit.runner.JUnitCore.run(JUnitCore.java:138)
at org.junit.runner.JUnitCore.run(JUnitCore.java:117)
at org.junit.runner.JUnitCore.runMain(JUnitCore.java:96)
at org.junit.runner.JUnitCore.runMainAndExit(JUnitCore.java:47)
at org.junit.runner.JUnitCore.main(JUnitCore.java:40)
```

FAILURES!!!

Tests run: 2, Failures: 1

Crédit photo

Que faire pour tester qu'une méthode lance bien une exception ?

JUnit - Tester les exceptions

Imaginons une méthode `sqrt` pour calculer la racine carrée d'un entier

- ▶ Hypothèse : elle doit lancer une exception en cas de paramètre négatif

```
public static int sqrt(int val) {  
    if (val<0) {  
        throw new IllegalArgumentException(  
            "Pas de racine carrée pour un entier négatif");  
    }  
    // suite : calcul de la racine carrée  
}
```

JUnit - Tester les exceptions

Utilisation de l'**annotation**

```
@Test(expected=IllegalArgumentException.class)
public void sqrt_cas_négatif() {
    sqrt(-1);
}
```

- Le test est réussi si la méthode lance l'**exception** indiquée

« Faut-il *tout* tester ? »

Conclusion

*« Any program feature without
an automated test simply doesn't exist. »*

Kent Beck in *Extreme Programming Explained*.

- ▶ Le site de JUnit : <http://www.junit.org>
- ▶ La Javadoc de JUnit :
<http://junit.sourceforge.net/javadoc>

Séance 9

Le survol des tableaux

- Les tableaux (survol)
- Erreurs fréquentes

Avertissement

Nous présentons ici une vue **simplifiée** des tableaux en Java afin de **coller** à votre cours d'**algorithmique**.

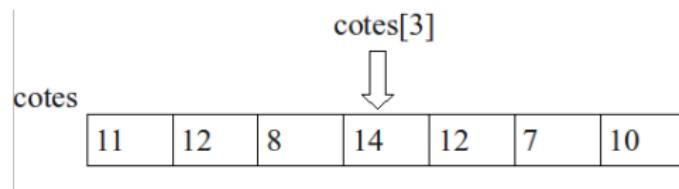
Nous aurons l'occasion d'être plus précis lors d'une prochaine leçon.



tableau
plusieurs variables de même type

Présentation

Nécessité de manipuler **plusieurs variables similaires** auxquelles on accède par un **indice**



Pourquoi pas plusieurs variables ?

Présentation

Écriture **compacte** et qui s'**adapte** à la taille

Exemple

En algorithmique, si **tab** est un tableau de N entiers

```
pour i de 0 à N-1 faire
    afficher tab[i]
fin pour
```

[]

Déclaration

Type [] identifier

Déclaration

Exemples

- ▶ `int []` est le type *tableau d'entiers*
- ▶ `String []` est le type *tableau de chaînes de caractères*

```
int [] cotes;  
String [] noms;
```

new

Création

identifier = new Type[taille]

Création

Exemple

- ▶ `new int[3]`
- ▶ `new String[taille]` où taille est défini

```
int [] entiers ;  
entiers = new int[3];
```

Remarque

La déclaration et la création peuvent être combinées

```
int [] entiers = new int[3];
```

{ }

Initialisation

```
identifier = new Type[] {x, x}
```

Initialisation

Exemple

- ▶ `new int[] {42, 17, -5}`
- ▶ `new String[] {"foo", "bar"}`

```
int [] entiers = new int[] {0x2A, 021, -5};  
String [] noms = new String[]  
    {"Victoria", "Melanie", "Melanie", "Emma", "Geri"};  
  
double[] réels ;  
réels = new double[] {4.2, -1};
```

Remarque

Par défaut, les éléments sont initialisés à `0` (numériques) ou `false` (booléens)

Création et initialisation

Cas particulier

Création du tableau et initialisation en une seule étape,
en **donnant ses valeurs**

```
int [] entiers = {0x2A, 021, -5};  
double[] pseudoRéels = {4.5, 1E-4, -4.12, Math.PI};  
  
double[] réels ;  
réels = {4.2, -1}; // FAUX
```

A close-up photograph of a grey cat sitting in a metal wheelbarrow. The cat is facing right, with its mouth open and tongue slightly out, possibly panting or meowing. Its eyes are partially closed. The background is a light-colored wall.

Accès aux éléments

[i]

Credit photo

Accès aux éléments

[i]

- ▶ 0 est l'indice de départ
- ▶ indices de **0** à **taille du tableau - 1**
- ▶ la taille du tableau est son nombre d'éléments

```
int [] entiers = {3, 14, 15};  
int entier = entiers [2]; // entier vaut 15  
entiers [1] = 85;  
entier = 0;  
entier = entiers [ entier+1]; //entier vaut 85
```

Accès aux éléments

Exemple

```
package be.heb.esi.lg1.tutorials.tableaux;

public class InitialisationTableau {
    public static void main(String[] args) {
        int[] entiers = new int[10];
        for(int i = 0; i < 10; i++) {
            entiers[i] = i;
        }
    }
}
```



Un tableau connaît
sa taille

`identifier.length`

Crédit photo

Taille

Exemple

```
int [] entiers = {4, 5, 6};  
int taille = entiers.length;  
System.out.println ( taille ); // écrit 3
```

Taille

Exemple

```
package be.heb.esi.lg1.tutorials.tableaux;

public class SimpleParcoursAscendant {
    public static void main(String [] args){
        int [] entiers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        for(int i = 0; i < entiers.length; i = i + 1) {
            System.out.println ( entiers [ i ]);
        }
    }
}
```

Taille

Exemple

```
package be.heb.esi.lg1.tutorials.tableaux;

public class SimpleParcoursDescendant {
    public static void main(String [] args){
        int [] entiers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
        for(int i = entiers.length - 1; i >= 0; i = i-1) {
            System.out.println ( entiers [ i ]);
        }
    }
}
```

Tableau et méthode

Un tableau peut être un paramètre d'une méthode.

Exemple : Afficher un tableau

```
public static void afficher ( int [] entiers ) {  
    for( int i = 0; i < entiers.length; i++ ) {  
        System.out.println ( entiers [ i ]);  
    }  
}
```

- L'appel pourrait être

```
int [] cotes = {12, 8, 10, 14, 9};  
afficher ( cotes );
```

Tableau et méthode

En Java, **passage de paramètre par valeur.**

Pour un tableau, cela signifie que l'on ne peut pas modifier le tableau dans son ensemble mais que l'on pourra modifier ses éléments.

Tableau et méthode

Exemple

```
public static void remplir( int [] entiers , int val ) {  
    for(int i = 0; i<entiers.length; i++) {  
        entiers [i] = val;  
    }  
}
```

- ▶ L'appel pourrait être

```
int [] cotes = new int[16]; // Ne pas oublier de le créer  
remplir( cotes , 20 );
```

Tableau et méthode

Exemple

```
public static void méthodeFausse( double[] réels ) {  
    double[] réelsDePassage = {4.2, -7, Math.PI};  
    réels = réelsDePassage; // INUTILE  
}
```

Quel que soit l'appel, le tableau que l'on passe en paramètre ne sera pas modifié

Tableau et méthode

Un tableau peut être une valeur de retour

Exemple : Créer un tableau avec valeur

```
public static int [] créer( int taille , int val ) {  
    int [] entiers = new int[ taille ];  
    for( int i = 0; i < taille ; i++ ) {  
        entiers [ i ] = val;  
    }  
    return entiers ;  
}
```

- ▶ L'appel pourrait être

```
int [] cotes = créer(16, 20);
```

Erreurs fréquentes

Exceptions

- ▶ **NullPointerException** : si vous essayez d'accéder à un élément d'un tableau qui n'a pas été créé (le tableau vaut **null** dans ce cas)
- ▶ **ArrayIndexOutOfBoundsException** : si vous donnez un indice qui n'existe pas (ex : **tab[10]** quand il n'y a que 10 éléments dans le tableau)

Séance 10

Les variables locales et les expressions

- Allocation mémoire
- Déclaration
- Conventions sur les noms
- Valeur initiale
- Le concept de «portée»
- Constantes
- Les expressions

Variable locale



Credit photo

Présentation

Désignation générique d'un **emplacement** de la **mémoire vive**

- ▶ **Possède un type**
- ▶ Ne peut contenir que des valeurs de ce type
- ▶ Allocation différente si type
primitif ou **référence**

Allocation mémoire

Pour un type **primitif**

- ▶ Indique la zone mémoire (sur la pile/*stack*) où se trouve la **valeur**

Nom variable

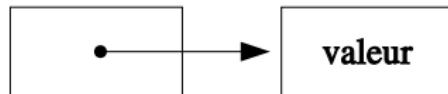
valeur

Allocation mémoire

Pour un type **référence** (ex : `String`, tableau)

- ▶ La zone mémoire contient l'**adresse** de la zone mémoire (sur le tas/*heap*) contenant la valeur (indirection)

Nom variable



Déclaration

Une variable déclarée dans un **bloc** ...
est locale à ce **bloc**

Block :

{ *BlockStatements*_(opt) }

BlockStatement :

LocalVariableDeclarationStatement
Statement

Déclaration

Déclaration

<type> <identifier> [= expression]

Exemples :

- ▶ **int** i;
- ▶ **String** nom, prénom;
- ▶ **boolean** ok=**true**, fini;
- ▶ **char** lettre , chiffre ='1';

Nom d'une variable

Identifier

Quel nom peut-on choisir ?

- ▶ Règles Java
javaletter, \$, _, €
- ▶ Conventions
mixedCase, noms explicites

\$ _

Valeur initiale

Initialisation

- ▶ Par défaut les variables **ne** sont **pas** initialisées
- ▶ Initialisation avec n'importe quelle expression calculable à cet endroit là (à l'exécution)

Valeur initiale

Exemple

```
int poidsKilo = 20; // Un poids en kilos  
int poidsGramme = 1000*poidsKilo; // L'équivalent en grammes
```

```
int poidsKilo; // Un poids en kilos  
int poidsGramme = 1000*poidsKilo; // Erreur à la COMPILEATION
```

Scope (portée) d'une variable locale

Définition Le *scope* (*portée*) d'une variable indique la portion du programme où elle existe

Le *scope* d'une variable locale est

- ▶ le *block* de sa déclaration (entre {})
- ▶ dès sa propre initialisation

Scope (portée) d'une variable locale

Exemples : bon ou pas ?

```
{  
    int x = y;  
    int y = 1;  
}
```

```
{  
    int x;  
    int y = x;  
}
```

```
{  
    int x = 1, y = x;  
}
```

Constante

final

Valeur donnée

- ▶ Soit à la déclaration
- ▶ Soit par assignation ultérieure

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

Constante

Convention de nom différente

- ▶ Tout mettre en **majuscules**
- ▶ Utiliser _ pour séparer les mots

Exemples

```
final double PI = 3.1415;  
final int TAUX_TVA = 21;
```

A close-up photograph of a cat's face. The cat has large, wide-open green eyes with dark pupils. Its mouth is slightly open, revealing its tongue and two upper fangs. The fur is a mix of orange and white, with some darker spots on the nose and chin. The background is blurred.

Expression

BOGDAN BOEV
PHOTOGRAPHY
© 2012 ALL RIGHTS RESERVED

Crédit photo

Expression

Calcul faisant intervenir une ou plusieurs valeur(s) pour une opération déterminée.

Cette valeur peut, elle-même être une expression.

Définitions

Exemple : 1+2

- ▶ 1 et 2 sont les **opérandes**
- ▶ + est l'**opérateur**
- ▶ l'expression est de type **int**
- ▶ la valeur de l'expression est 3

Les expressions entières

Opérateurs entiers

+ - + - * / %

Opérandes pouvant intervenir

- un **littéral**
- une **variable**
- une **expression**

Les expressions entières

Les opérandes doivent être de **même** type (entier)
à *conversion près*

- ▶ Le type de l'expression est celui de ses opérandes
- ▶ Exemples :
 - `1 + 2` vaut 3 de type **int**
 - `1L + 2L` vaut 3 de type **long**
 - `3 / 2` vaut 1 de type **int**

i+i*2

Priorité et associativité

La stratégie d'évaluation se base sur

- ▶ la **priorité** d'un opérateur
- ▶ l'**associativité** des opérateurs de même priorité

priorité	opérateur	associativité
grande	- , + unaires *, /, %	↔
faible	- , + binaires	→

Priorité et associativité

Exercices

- ▶ $3 + 3 * 2 + 1 ?$
- ▶ $3 + 3 * 2 / - 4 + 5 \% 8 ?$

Les **parenthèses** permettent de **forcer** la stratégie,
expliciter l'ordre et **clarifier**.

Erreurs de calcul

La division par zéro

- ▶ Lance une exception (*ArithmeticException*)
- ▶ Pour l'instant, **arrête le programme** avec un message explicite

Le dépassement de capacité

- ▶ N'est **pas détecté** par la machine virtuelle
- ▶ Le résultat est tout simplement faux

Les expressions flottantes

Opérateurs flottants

+ - + - * / %

Opérandes (littéral, variable ou expression) du même type **flottant** (à conversion près)

Les expressions flottantes

Exemples : Donner la valeur et le type de

- ▶ `1.0 + 2.3`
- ▶ `1.0d - 2.3d`
- ▶ `7.0f / 3.5f`
- ▶ `1. / 3.`

Les expressions caractères

char est un type numérique entier

- ▶ **aucun opérateur** spécifique
 - ▶ Calculs possibles mais non recommandé
- Exemple** : 'a' + 'b'

Les chaines de caractères

Opérateur



- ▶ Un seul opérateur pour la **concaténation** de deux chaînes.
- ▶ Conversion si un des 2 opérandes n'est pas une chaîne.

Les expressions booléennes

Opérateurs booléens

! && ||

Tables de vérité

(ET)	true	false
true	true	false
false	false	false

(OU)	true	false
true	true	true
false	true	false

Les expressions booléennes

Tableau des priorités #2

priorité	opérateur	associativité
grande	- , + unaires, !	↔
	* , / , %	→
	- , + binaires	→
	&&	→
faible		→

Exemples

Comment évaluer ?

- ▶ **true && false || true**
- ▶ **false || false && ! true**
- ▶ **true || false && true**
- ▶ **false && (true || false)**
- ▶ **!(true || false) && true**

Les expressions relationnelles

Opérateurs de comparaison

< > <= >=

- Opérandes de type **numériques**
- Le **résultat** est du type **boolean**

Les expressions relationnelles

Opérateurs d'égalité

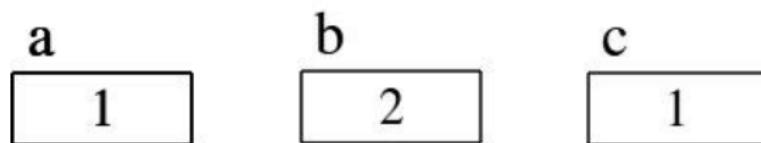
== !=

- ▶ Opérandes de **tout type**
- ▶ Sens différent si type primitif ou référence

Égalité de valeurs

Type primitif

Les **valeurs** sont comparées

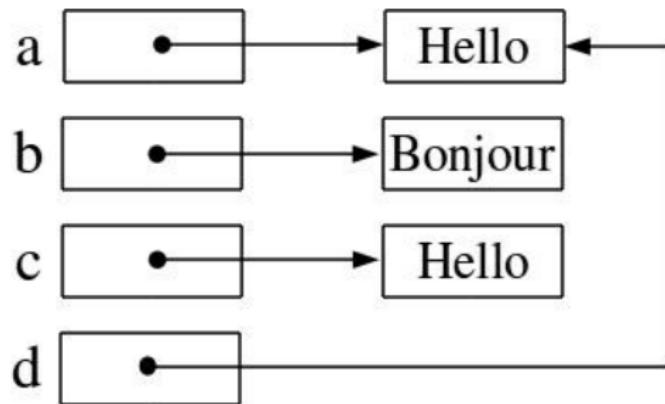


On a : $a == c$ mais $a != b$ et $b != c$

Égalité de valeurs

Type référence

Les **références** sont comparées



On a : $a \neq b$, $a \neq c$ mais $a == d$

Cas particulier du type String

Le compilateur réutilise l'espace pour les **littéraux** de type String

```
String s1 = "Hello";
String s2 = "Hello";
String s3 = "Hel";
s3 = s3 + "lo";
System.out.println(s1==s2); // Vrai
System.out.println(s1==s3); // Faux
s2 = "Bye";
System.out.println(s1==s2); // Faux
```

Egalité de valeur

equals()

```
{  
    String s1 = "Hello";  
    String s2 = "Hello";  
    String s3 = "Hel";  
    s3 = s3 + "lo";  
    System.out.println (s1.equals(s2)); // Vrai  
    System.out.println (s1.equals(s3)); // Vrai  
}
```

- ▶ Ne teste pas que les références sont identiques
- ▶ Mais bien que les **valeurs** référencées sont égales

L'expression conditionnelle

Opérateur conditionnel

? :

- ▶ Équivalent du **si-sinon** sous forme d'expression
- ▶ <condition> ? <expression> : <expression>

Exemples

- ▶ `heure < 12 ? "bonjour" : "bonsoir"`

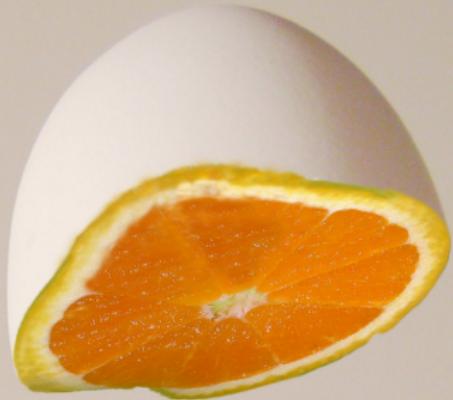
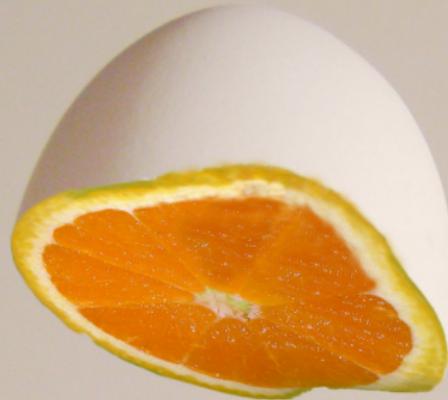
- ▶

```
int n = -4;
int abs = n > 0 ? n : -n;
```

Tableau des priorités et associativités

Tableau des priorités #3

priorité	opérateur	associativité
grande	- , + unaires, ! *, /, % -, + binaires <, >, <=, >= ==, !=	↔
	&&	⇒
		⇒
faible	?:	↔



Peut-on mélanger les types ?



Crédit photo

Un mot sur les conversions

Exemples

- ▶ Calcul mélangeant les entiers et les réels
 - Les entiers sont convertis en réels
 - Ex : `3.2/2` vaut 1.6 de type **double**
- ▶ Assigner un entier à un réel
 - L'entier est converti en réel
 - Ex : `double d = 1; //d vaut 1.0`
- ▶ Assigner un réel à un entier
 - Refusé . . . sauf si demandé explicitement (**casting**)
 - Ex : `int i = 1.2; // refusé`
 - Ex : `int j = (int) 1.6; // j vaut 1`

Séance 11

Les conversions

- Présentation
- Dans les expressions
- Lors d'une assignation
(et d'un return)
- Lors d'un appel de méthode
- Avec les chaînes de caractères
- Le casting
- Récapitulatif



Conversions

Crédit photo

Conversions

Java impose que les types correspondent.
Parfois le compilateur convertit implicitement.

Quelles sont les règles précises ?

- ✓ **double d = 1;**
- ✗ **int i = 1.0;**

Contextes et sortes de conversions



8 groupes



5 contextes

Où en sommes-nous ?

- Présentation
- **Dans les expressions**
- Lors d'une assignation (et d'un return)
- Lors d'un appel de méthode
- Avec les chaînes de caractères
- Le casting
- Récapitulatif

Les conversions dans les expressions



Expressions

- ▶ Expressions **numériques**
- ▶ Adapte le type des opérandes
- ▶ Conversion **élargissante de type primitif**

Les conversions dans les expressions



Conversion élargissante de type primitif (widening primitive conversion)

byte

→ **short**

→ **int** ← **char**

→ **long**

→ **float**

→ **double**

Les conversions dans les expressions

- ▶ Opérateurs binaires, opérateurs unaires
- ▶ Le moins large et au minimum **int**

Les conversions dans les expressions

Exemples

```
System.out.println( 7 / 2. )
```

```
System.out.println( 7. / 2. )
```

```
System.out.println( 1 <= 1.0 )
```

```
byte b = 2; short s = 1; char c = '3'; int i = 4;
```

```
... s + i ...
```

```
... s + 5L ...
```

```
... c - s ...
```

```
... 2 <= i ...
```

```
... +s ...
```

```
... -b ...
```

```
... -'A' ...
```

```
... t[s] ...
```

```
... t['1'] ...
```

Où en sommes-nous ?

- Présentation
- Dans les expressions
- **Lors d'une assignation (et d'un return)**
- Lors d'un appel de méthode
- Avec les chaînes de caractères
- Le casting
- Récapitulatif

Conversion lors d'une assignation



Assignment

- ▶ Adapte le type de l'expression au type de la variable
- ▶ Opérateurs : `=`, `+=`, `-=`, `*=`, `/=`, `%=`
- ▶ Permet la **conversion élargissante**
- ▶ Mais aussi la **conversion arrondissante**

Conversion lors d'une assignation



Conversion arrondissante de type primitif (narrowing primitive conversion)

double

→ **float**

→ **long**

→ **int**

→ **short**

↔

char

→ **byte**

↙ ↘

À chaque étape : **perte de précision** possible

Conversion lors d'une assignation

Dans le cadre d'une assignation

- ▶ Elles ne sont pas toutes permises
- ▶ **si et seulement si**
 - La variable est de type **byte**, **short**, ou **char**
 - L'expression est
 - constante
 - de type **byte**, **short**, **char** ou **int**
 - sa valeur est représentable dans le type de la variable

Si ce n'est pas le cas, erreur à la compilation

Conversion lors d'une assignation

Exercice : identifiez les instructions correctes

```
long l1 = 12;
long l2 = 'a'+1;
short s1 = 12;
short s2 = 1+2;
byte b1 = 123245;
byte b2 = s1+1;
byte b3 = 21L;
```

```
char a ='a';
a += 1;
a = a+1;
```

Conversion de la valeur de retour

Situation similaire pour la **valeur de retour** d'une méthode

- Le type de l'expression accompagnant l'instruction **return** doit pouvoir être ramené au *ResultType*

Exemple

```
public static long add( int opérandeGauche, int opérandeDroite) {  
    return opérandeGauche + opérandeDroite;  
}
```

Où en sommes-nous ?

- Présentation
- Dans les expressions
- Lors d'une assignation (et d'un return)
- Lors d'un appel de méthode**
- Avec les chaînes de caractères
- Le casting
- Récapitulatif

Conversion lors d'un appel de méthode



Appel de méthode

- ▶ Conversion des **paramètres** effectifs
- ▶ **Conversion élargissante** possible

Exemple

```
public static int add( int opérandeGauche, int opérandeDroite) {  
    return opérandeGauche + opérandeDroite;  
}
```

les arguments lors de l'appel peuvent être **byte**, **short**, **char** ou
encore **int**

Conversion lors d'un appel de méthode

Attention à la **surcharge** (*overloading*)

Exemple

```
public static int max(long op1, long op2) {...}  
public static int max(int op1, int op2) {...}
```

Quelle méthode est choisie avec cet appel ?

```
short s1=1, s2=2;  
max(s1,s2);
```

Conversion lors d'un appel de méthode

Exemple

```
public static double op( double opérandeGauche, double opérandeDroite) {  
    return opérandeGauche * opérandeDroite;  
}
```

```
public static int op( int opérandeGauche, int opérandeDroite) {  
    return opérandeGauche / opérandeDroite;  
}
```

- ▶ Que retourne `op (3.,2.)` ?
- ▶ Que retourne `op(3,2)` ?
- ▶ Que retourne `op (3.,2)` ?

Conversion lors d'un appel de méthode

Cas particulier

- ▶ Parfois, il n'y a pas de méthode plus spécifique.
- ▶ **Exemple**

```
public static int max( int op1, long op2 ) {...}  
public static int max( long op1, int op2 ) {...}
```

Accepté mais certains appels seront ambigus
(erreur à la compilation)

- `max(1,2L)` // 1ère méthode
- `max(1L,2)` // 2ème méthode
- `max(1,2)` // ambigu

Où en sommes-nous ?

- Présentation
- Dans les expressions
- Lors d'une assignation (et d'un return)
- Lors d'un appel de méthode
- Avec les chaines de caractères
- Le casting
- Récapitulatif

Conversion en chaines de caractères



Chaine

Opérateur **+** avec un opérande de type **String**

Exemples

▶

```
System.out.println ("1+1\u00b3\u00b3" +2);
String s = "Pi\u00b3\u00b3" + 3.1415;
```

▶ Que donnera ceci ?

```
System.out.println ("1"+2+3);
System.out.println (1+2+"3");
```

Où en sommes-nous ?

- Présentation
- Dans les expressions
- Lors d'une assignation (et d'un return)
- Lors d'un appel de méthode
- Avec les chaînes de caractères
- **Le casting**
- Récapitulatif

Le *casting*



Casting

Conversion **explicite**

Casting :
(Type) Expression

Le *casting*



- ▶ élargissante
- ▶ arrondissante
- ▶ (un)boxing
- ▶ identique

Le *casting*

Remarques sur les conversions **arrondissantes**

- ▶ Aucune contrainte, on tronque

Exemple

byte b = (byte) 256;

- ▶ $256 = \boxed{0 \mid 0 \mid 1 \mid 0}$ (**int** codé sur 4 bytes)
- ▶ $\implies b = \boxed{0} = 0$

Le *casting*

Ne **peut pas** être utilisé pour

- ▶ Convertir en chaîne
 - `(String) monEntier` **interdit**
 - Écrire `""+monEntier` à la place
- ▶ Convertir une chaîne qui contient un numérique
 - `(int) "12"` **interdit**
 - Écrire `Integer.parseInt("12")` à la place

Exemples de *casting*

Exemples



Rappel des règles de priorité

Tableau des priorités et associativités

priorité			associativité
forte	post unaires	(params), ., expr++, expr--	\Rightarrow
	pré unaires	(Type), ++expr, --expr,	
		- , + , ! , new	$\Leftarrow\Rightarrow$
	multiplicatif	* , / , %	\Rightarrow
	additif	- , +	\Rightarrow
	relationnels	< , > , <= , >=	\Rightarrow
	égalité	== , !=	\Rightarrow
	et	&&	\Rightarrow
	ou		\Rightarrow
	condition	:?	$\Leftarrow\Rightarrow$
faible	assignations	=, +=, -=, *=, /=, %=	$\Leftarrow\Rightarrow$

Récapitulatif



Il y a **8 sortes** de conversions

- ▶ Élargissante / arrondissante de type primitif
- ▶ Conversion en chaîne de caractères
- ▶ Conversion identique
- ▶ Boxing / Unboxing
- ▶ Élargissante / arrondissante de type référence

Récapitulatif



Il y a **5 contextes** de conversion

- ▶ La promotion (calcul) numérique
- ▶ L'assignation
- ▶ Le casting
- ▶ La chaîne de caractères
- ▶ L'appel de méthode

Récapitulatif

	élargis.	arrondi	chaine	ident.
promotion num.	✓			✓
assignation	✓	✓ (*)		✓
chaine			✓	✓
casting	✓	✓		✓
méthode	✓			✓

(*) : sous certaines conditions

Crédits

Ces slides sont le support pour la présentation orale de l'unité d'enseignement **DEV1-JAV** à HEB-ÉSI

Crédits

Les distributions Ubuntu et/ou debian
du système d'exploitation **GNU Linux**.

LaTeX/Beamer comme système d'édition.

Git et GitHub pour la gestion des versions et le suivi.

GNU make, rubber, pdfnup, ... pour les petites tâches.

Images et icônes

deviantart, flickr, The Noun Project 