

# Le Langage Java

## 1<sup>re</sup> année

J. Beleho (bej)   C. Leruste (clr)   M. Codutti (mcd)  
P. Bettens (pbt)   F. Servais (srv)   C. Leignel (clg)  
D.P. Bishop (bis)   S. Drobisz (sdr)

Haute École de Bruxelles-Brabant — École Supérieure d'Informatique

Année académique 2016 / 2017

## Séance 5

### Notion de package et types et littéraux

- Organiser le code
- Les types et les littéraux
- Les types entiers
- Les types flottants
- Les booléens
- La chaîne de caractères

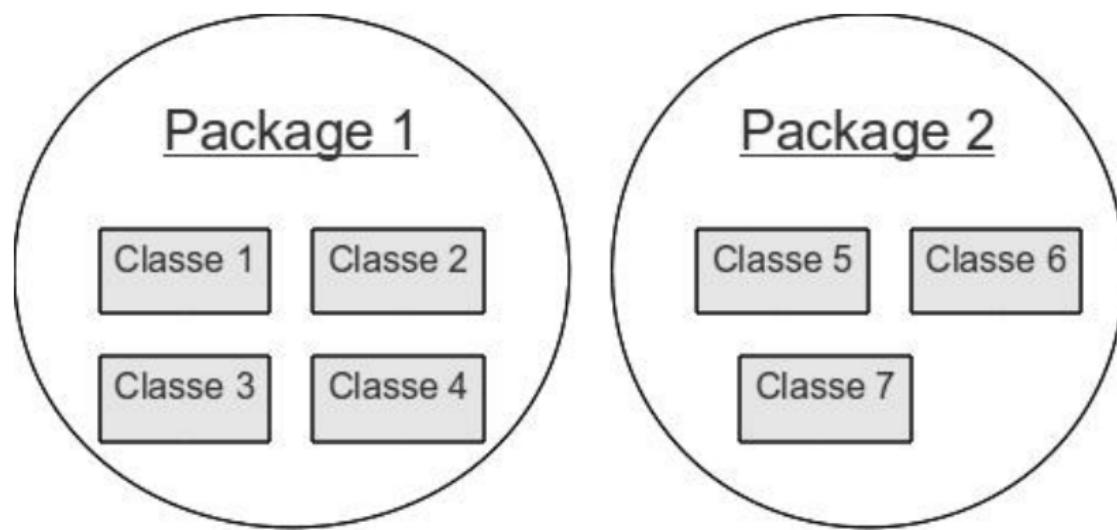
# Rangement



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.he2b.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](#)

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();
```

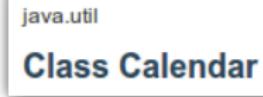
A `Calendar` object can produce all the calendar field values needed to implement the date-time formatting for a particular language (e.g., Japanese-Gregorian, Japanese-Traditional). `Calendar` defines the range of values returned by certain calendar fields, as well as the fact that each month of the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

Comment créer mes propres  
*packages* ?

# Créer ses packages

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

```
package mon.paquet;  
public class NomClasse {  
    // Nom complet : mon.paquet.NomClasse  
}
```

# 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**

Par convention le nom de package correspond au nom de domaine inversé afin de permettre l'unicité.

be.esi-bru.pbt.acme

[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 this class 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 this class 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 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.

### Getting and Setting Calendar Field Values

Crédit photo

*Java est un langage fortement typé*

Légende urbaine ?

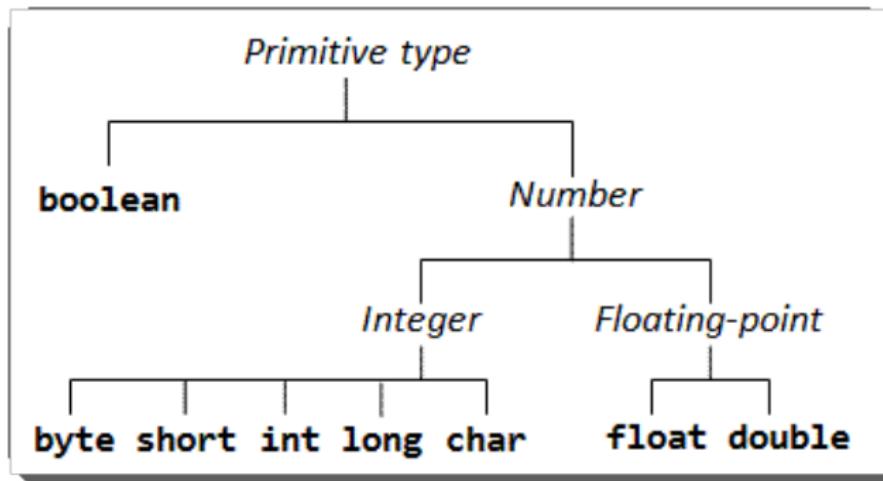
# Les types

Toute donnée a un type

Quels types ?

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

# Les types primitifs



source : [http://www3.ntu.edu.sg/home/ehchua/programming/java/J2\\_Basics.html](http://www3.ntu.edu.sg/home/ehchua/programming/java/J2_Basics.html)

A close-up, low-angle shot of a row of colorful bicycles parked side-by-side. The frames are painted in various vibrant colors including red, blue, green, yellow, pink, and black. Each bicycle features a red rectangular rear light mounted on the frame. The background is blurred, showing more of the same colorful bicycles extending into the distance.

## 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

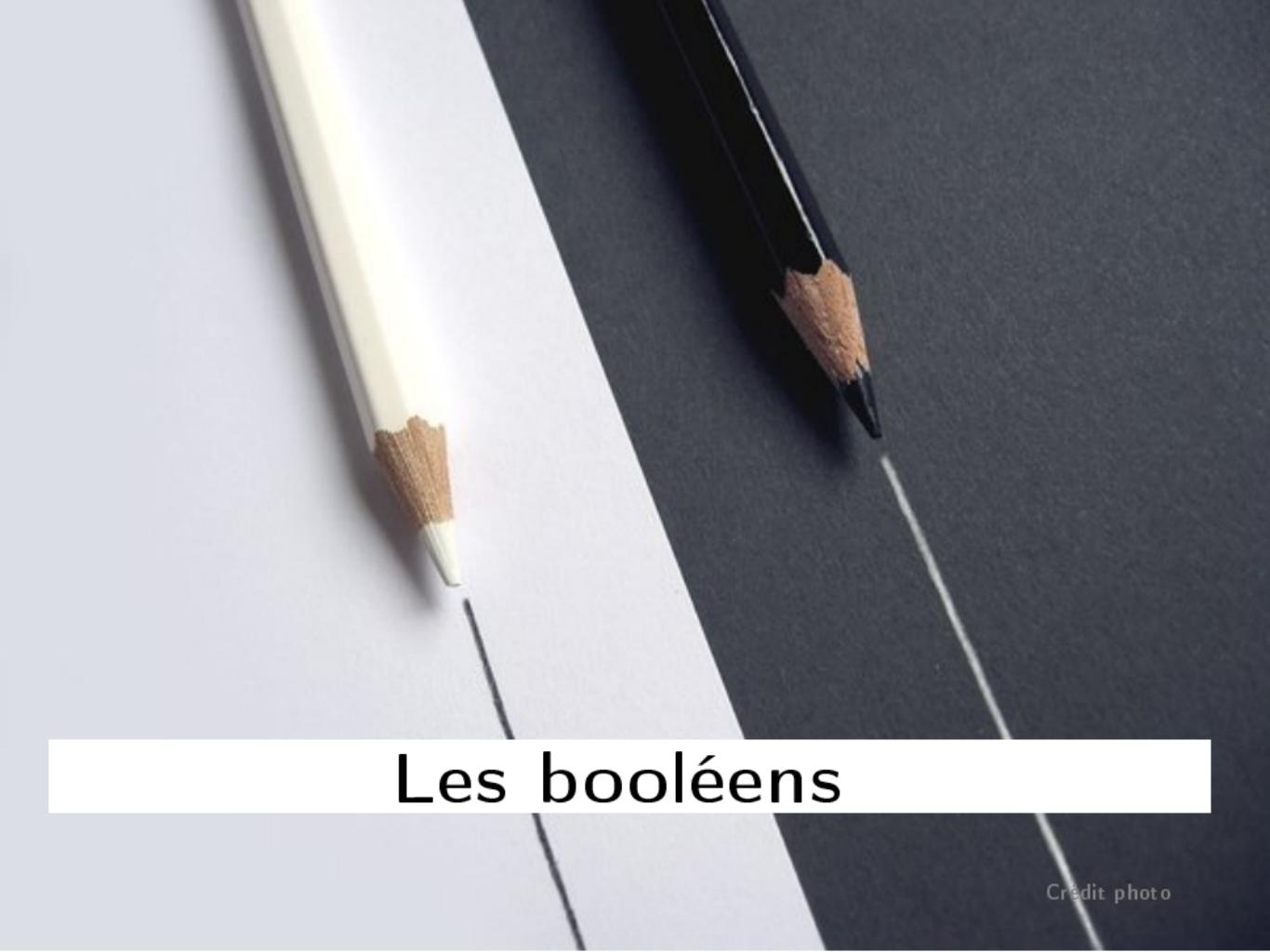
## **float, double**

- ▶ 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

# Crédits

Ces slides sont le support pour la présentation orale de l'unité d'enseignement **DEV1-JAV** à HE2B-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 