Introduction à la programmation avec Java

UFR Sciences de Nice

Licence Math-Info 2006-2007

Module L1I1

Frédéric MALLET

Jean-Paul ROY

Ressources sur le langage Java

- "Conception objet en Java avec BlueJ : Une approche interactive" (2ème édition)
  par Barnes et Kolling, chez Pearson Education, 2005.
- "Informatique Industrielle et Java"
  par Mallet et Boéri, chez Dunod, 2003.
- La documentation en ligne des classes (ou « API ») :
  http://java.sun.com/j2se/1.4.2/docs/api/

Page Internet du cours :

http://deptinfo.unice.fr/~fmallet/L1I1/

La science informatique

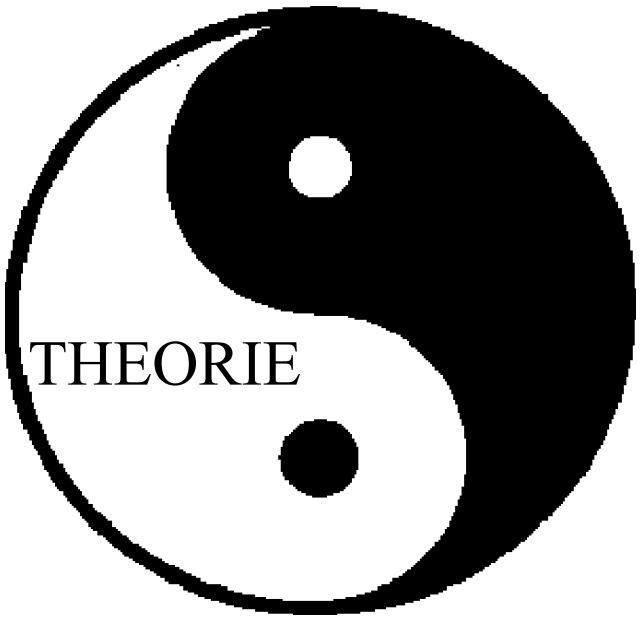
Objets et classes

Qu'est-ce que l'informatique ?

❶ Une DISCIPLINE SCIENTIFIQUE chargée du traitement des informations par ordinateur(s). Elle utilise :

- des **mathématiques** : modélisation, rigueur.

- de l'**électronique** :
digital = numérique = {0,1}.
Numériser le monde réel [textes, images, musique, réseaux].



- des **langages de programmation** :
piloter les ordinateurs à l'aide d'algorithmes.

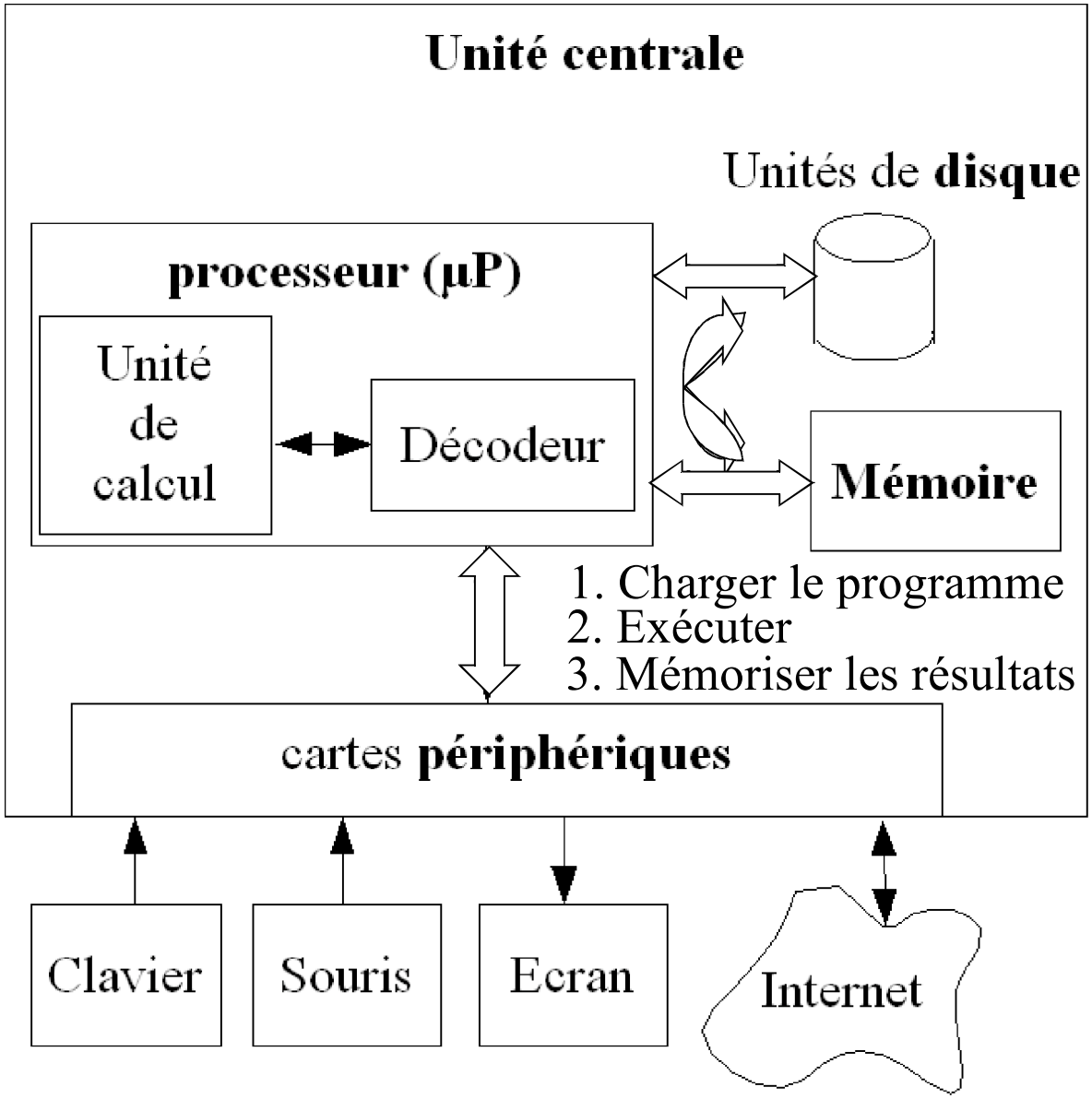
- de la **linguistique**, de la **psychologie**, de la **biologie**, de la **physique**, et bien d'autres disciplines annexes.

❷ Mais l'informatique est aussi l'ensemble des TECHNOLOGIES permettant de parvenir à ce but.

- les **ordinateurs** [calculateurs électroniques],
- leurs **périphériques** [écrans, claviers, souris, scanners, modems, imprimantes, disques magnétiques, vidéo, etc.]
- les **réseaux** [Internet, Wifi] permettant aux utilisateurs d'ordinateurs de communiquer par « e-mail », de faire du « e-commerce », de gérer leur compte en banque, d'effectuer des recherches...
- etc.

Architecture d'un ordinateur

- Pentium d'Intel [PC, Mac], Athlon d'AMD [PC]
- Power-PC d'IBM et Motorola [anciens Mac]



Concevoir des algorithmes

• Un **algorithme** est une description explicite d'un traitement, d'un calcul. Exemple : les lasagnes

1. *Préparer* les feuilles de lasagne;
2. *Préparer* la sauce bolognaise (ou les légumes);
3. *Préparer* la béchamel;
4. *Répéter*
   - Mettre une couche de béchamel dans un plat;
   - Mettre une couche de sauce;
   - Recouvrir d'une feuille de lasagne;

   *tant qu'*il reste des ingrédients;
5. Faire cuire au four 30 minutes;

Les lasagnes. Remarques :

séquence

fonctions

- L'ordre des **actions** est important.
- Des actions sont décomposées en **sous-actions**.
  Ex: préparer la béchamel : mélanger beurre, farine, …
- Certaines actions sont conditionnelles.
  Ex: Préparer les feuilles de lasagne : **si** l'on est pressé, on les achète prêtes **sinon** on les prépare soi-même !
- On en envie d'exprimer des **répétitions**.
  Ex: **tant que** « il reste des ingrédients » **faire** …

structures de contrôle

De l'algorithme au programme

• Un algorithme doit être précis et traduisible dans un **langage de programmation** :

C, Fortran, Java, Pascal, Scheme, etc.

[des dizaines de langages !]

• Il n'existe **pas de langage universel**, seulement des langages mieux adaptés que d'autres à certaines tâches.

• Le langage de programmation permet de manipuler des « **variables** » [ex : le nombre de couches de lasagnes déjà en place]. Ces variables vont être stockées dans la **mémoire centrale** de l'ordinateur.

Les langages de programmation

- L'ordinateur (μP) ne comprend que le binaire !
- L'humain préfère un langage naturel : français, anglais…
- **Programmation = expression dans un langage informatique donné d'un algorithme résolvant un problème**.
- Nous avons choisi le **langage Java** pour introduire les notions élémentaires de programmation.
  - Mots et codages spécifiques = la **syntaxe** du langage
  - Notions non spécifiques = objet, variable, structure de contrôle, … que l'on retrouve dans les autres langages.
- **Compilation** = traduction d'un langage de haut-niveau (C, Java, etc) en langage$^{(\text{machine})}_{(\text{bytecode})}$ plus proche du processeur.

Plein d'idées et (trop) de langages

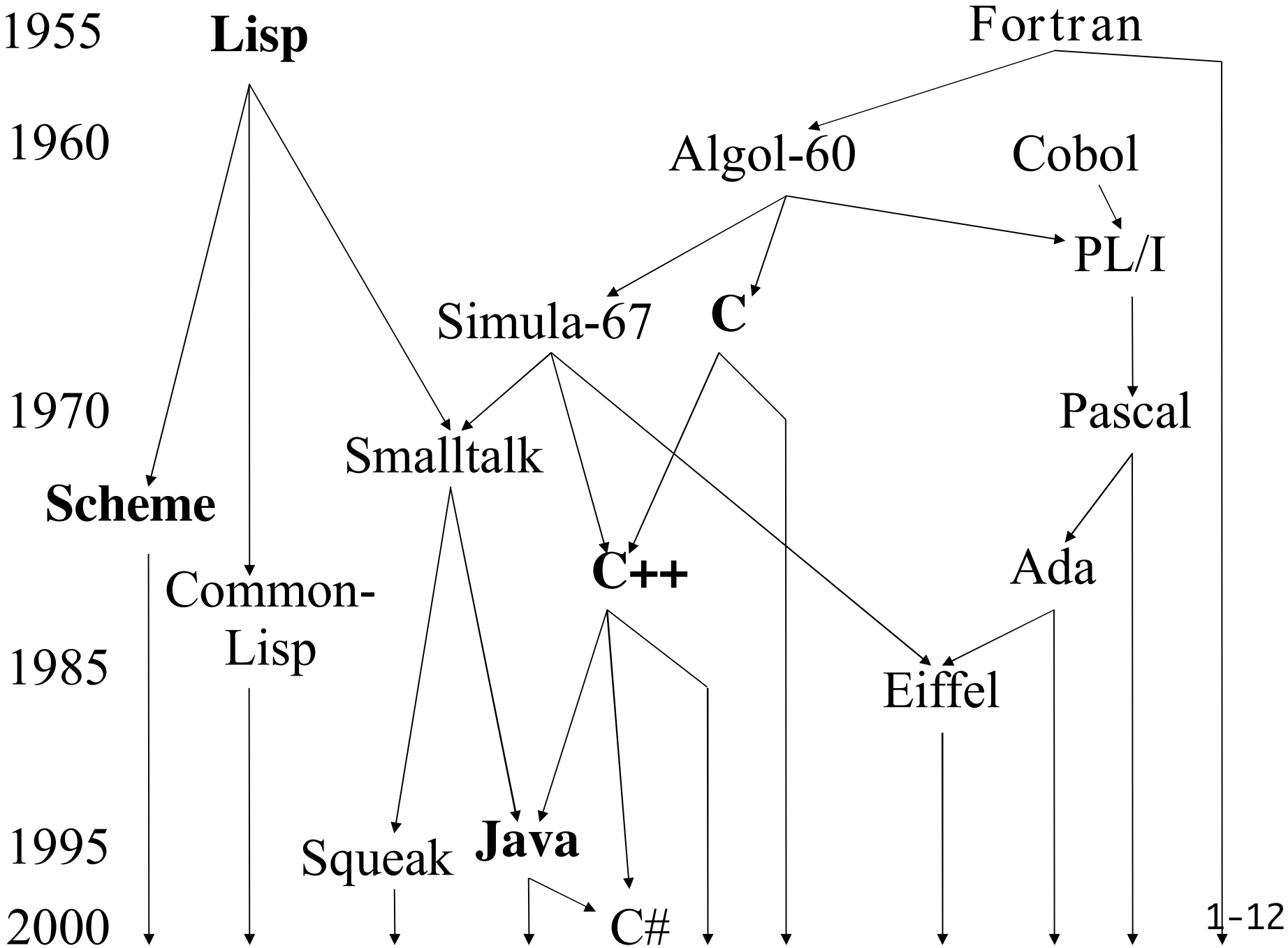
- ✔ Approche **impérative**

  Manipulation explicite de la mémoire par des instructions de haut niveau. Exécution séquentielle (temps).

- Approche **fonctionnelle**
  Composition de fonctions, récurrence. La mémoire est gérée de manière implicite (cf cours de 2ème année).

- Approche **logique**
  Règles décrivant des relations logiques entre les données et les résultats. Le « quoi » au lieu du « comment ».

- ✔ Approche **par objets** (sur-couche)
  Modélisation des données sous la forme d'objets, métaphore du monde réel. Ré-utilisation.


1955
1960
1970
1985
1995
2000
Lisp
Fortran
Algol-60
Cobol
PL/I
Simula-67
C
Pascal
Smalltalk
Scheme
C++
Ada
Common-
Lisp
Eiffel
Squeak
Java
C#

La Programmation Impérative

- **Programmation** = action de concevoir et de réaliser une solution (**algorithme** + **programme**) en vue d'une exécution automatique.
- **Impérative** = basée sur la notion d'état général représenté par un **ensemble de variables situées en mémoire**.
- **Programme impératif** = ensemble **d'instructions** (d'ordres) pour passer d'un **état initial** (énoncé + données) à un **état final** (solution) par un ensemble d'**actions élémentaires successives**.

Les crises historiques du logiciel

- L'ordinateur permet de faire **plus et plus vite**

  => besoin d'un **méthode de conception** !

- Les années 70

  Trouver les erreurs = 90% du temps

  => **programmation structurée**

  (langages Pascal, C…)

- Les années 90

  Trop de temps consacré à **refaire ce qui existe**

  => **programmation orientée objet**

  (langages C++, Java, UML…)

Pas encore de solution complètement satisfaisante

La programmation impérative par objets

- Organiser les données d'un programme sous la forme d'**objets** : un cercle, un carré, un étudiant, un texte…
- Savoir **communiquer avec ces objets** en leur envoyant des **messages**.
- Chaque objet dispose d'un savoir-faire (ses **méthodes**).
- Un cercle sait se déplacer, s'agrandir… Un étudiant sait calculer la moyenne de ses notes…
- Les objets sont organisés en **classes** : la classe des carrés, la classe des étudiants, la classe des textes…
- Les classes sont organisées en hiérarchies : la classe des carrés pourrait être une **sous-classe** de la classe des formes géométriques…
- Ces hiérarchies sont au cœur des logiciels modernes !

Vous pénétrez dans le monde du numérique !

Codage et mémorisation

« *Mémoriser des données variées avec un amas de métal parcouru d'électricité* »

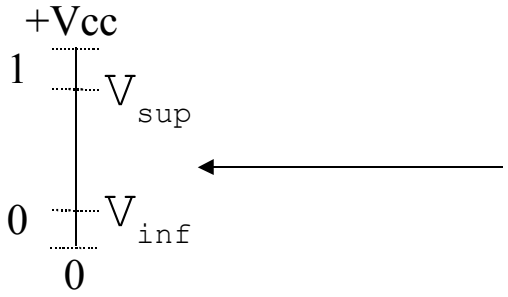
Numérique ou analogique ?

- **Analogique** : valeur = tension

  Échelle continue de valeurs => sensible au bruit
- **Numérique**

  Échelle discrète de valeurs : binaire, ternaire, ... ?

  BInary digiT = BIT (0 ou 1) : valeurs logiques



Mots de bits

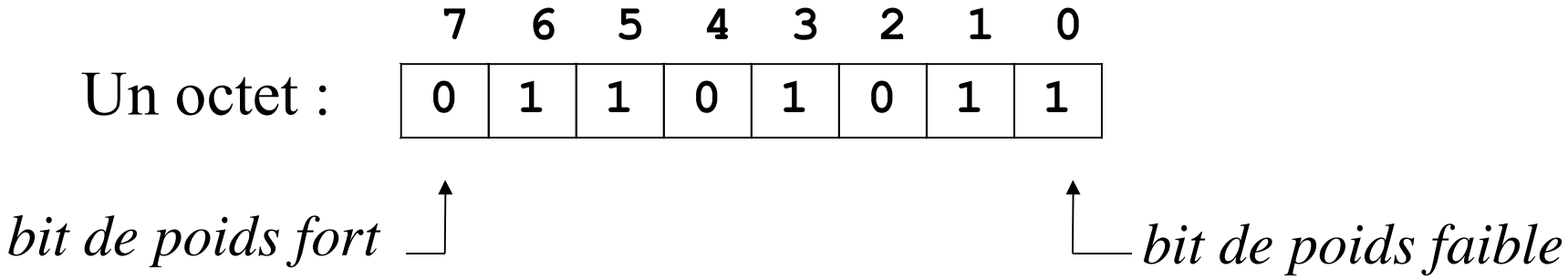
- 1 **bit** contient 0 ou 1
- 8 bits peuvent représenter $2^8 = 256$ valeurs : **octet**
- Un **mot-mémoire** sur une machine actuelle dont les adresses varient sur 32 bits est donc constitué de 4 octets.
- On peut ainsi représenter tous les ensembles FINIS de valeurs :

  Il suffit de prendre un nombre suffisant de bits !
- Le premier ordinateur utilisait la base 10

  Combien de lampes pour représenter 2006 en base 10 ?

Et les ensembles infinis comme **Z** ou **R** ?

- **Extraire un (grand) sous-ensemble fini !**
- Ex: Codage des entiers signés
  - 8 bits = 256 valeurs, de -128 à +127 (en Java : *byte*)
  - 16 bits = 65536 valeurs, de -32768 à 32767 (*short*)
  - 32 bits = 2^{32} valeurs (*int*)
  - 64 bits = 2^{64} valeurs (*long*)
- Ex: Codage des nombres approchés (virgule flottante)
  - 32 bits = simple précision (*float*)
  - 64 bits = double précision (*double*)



$0*2^7 + 1*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = ?$

- Algorithme de calcul (**schéma de Hörner**) :
  1) Je prends une **variable `acc`** valant 0 ;
  2) Pour chaque bit b_i de la gauche vers la droite :

$$acc \rightarrow 2 * acc + b_i$$

Donc ici **`acc`** vaut à la fin : **107**

Sur le codage des nombres approchés…

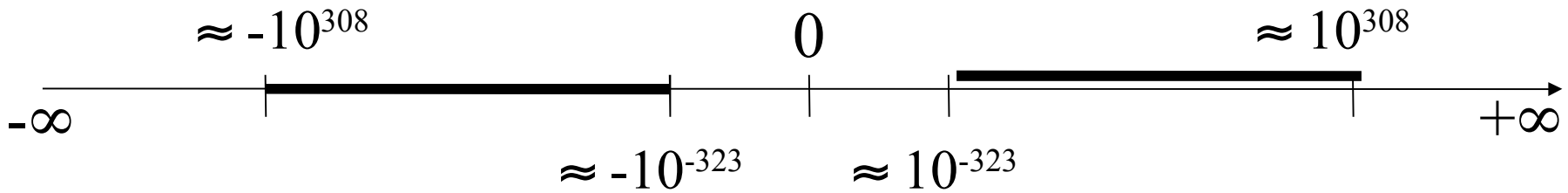
- Deux précisions : **simple** ou **double**.

Le codage **`float`** (32 bits)

$\sqrt{2} \approx 1.4142135$

Le codage **`double`** (64 bits)

$\sqrt{2} \approx 1.4142135623730951$



- Valeurs spéciales : **`Infinity`** (**`1.0/0.0`**) et **`NaN`** (**`0.0/0.0`**) !

Not A Number !

Mon premier contact avec les objets !

• Les **objets Java** modélisent les objets d'un problème donné. Ex: un cercle, un triangle, un carré…

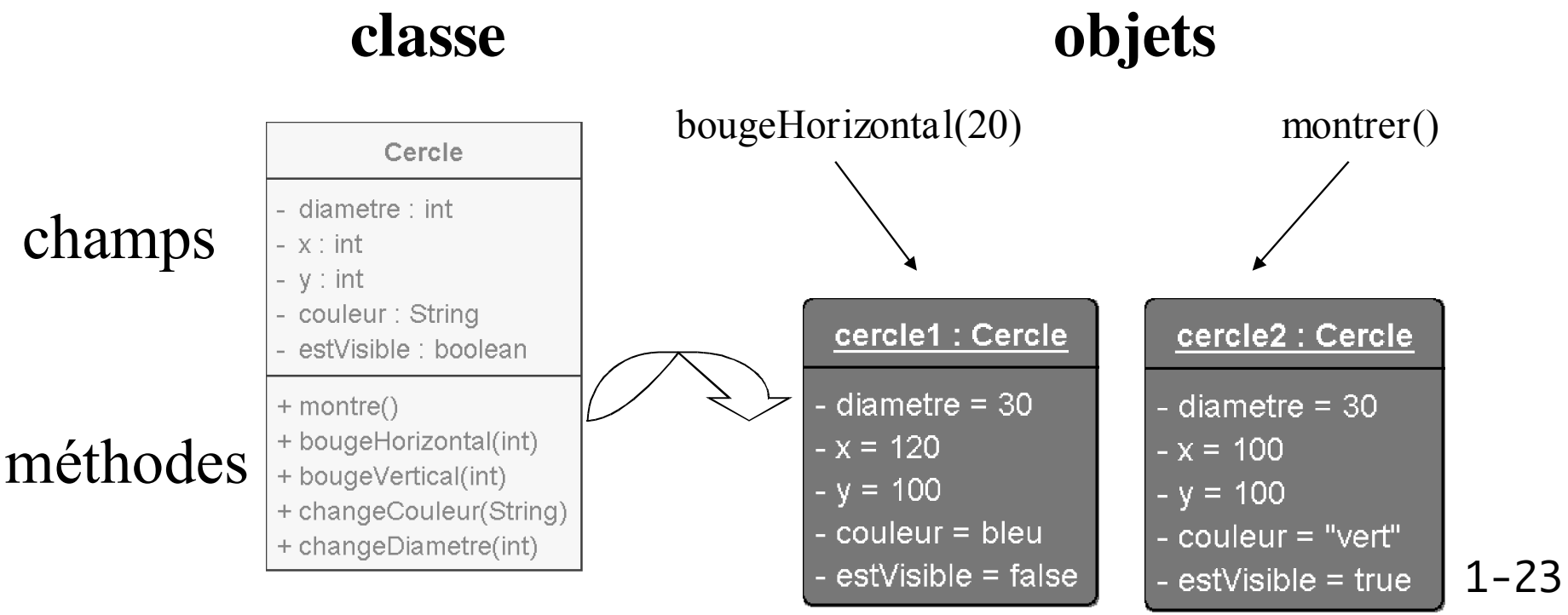
• Les objets sont créés à partir de **classes**. La classe décrit le type d'objet ; les objets sont des **instances de classes**.
Ex: un cercle donné est une instance de la classe ***`Cercle`***.

• Un objet possède des **champs** ou **attributs** décrits dans le texte de sa classe.

Ex: ***`Cercle`*** déclare pour chaque cercle un diamètre, un centre (x,y), une couleur, s'il est ou non visible, etc.
Ex: ***`cercle1`*** est un ***`Cercle`*** bleu de diamètre 30 à la position (10,20), ***`cercle2`*** est un autre ***`Cercle`***, vert, de diamètre 20 …

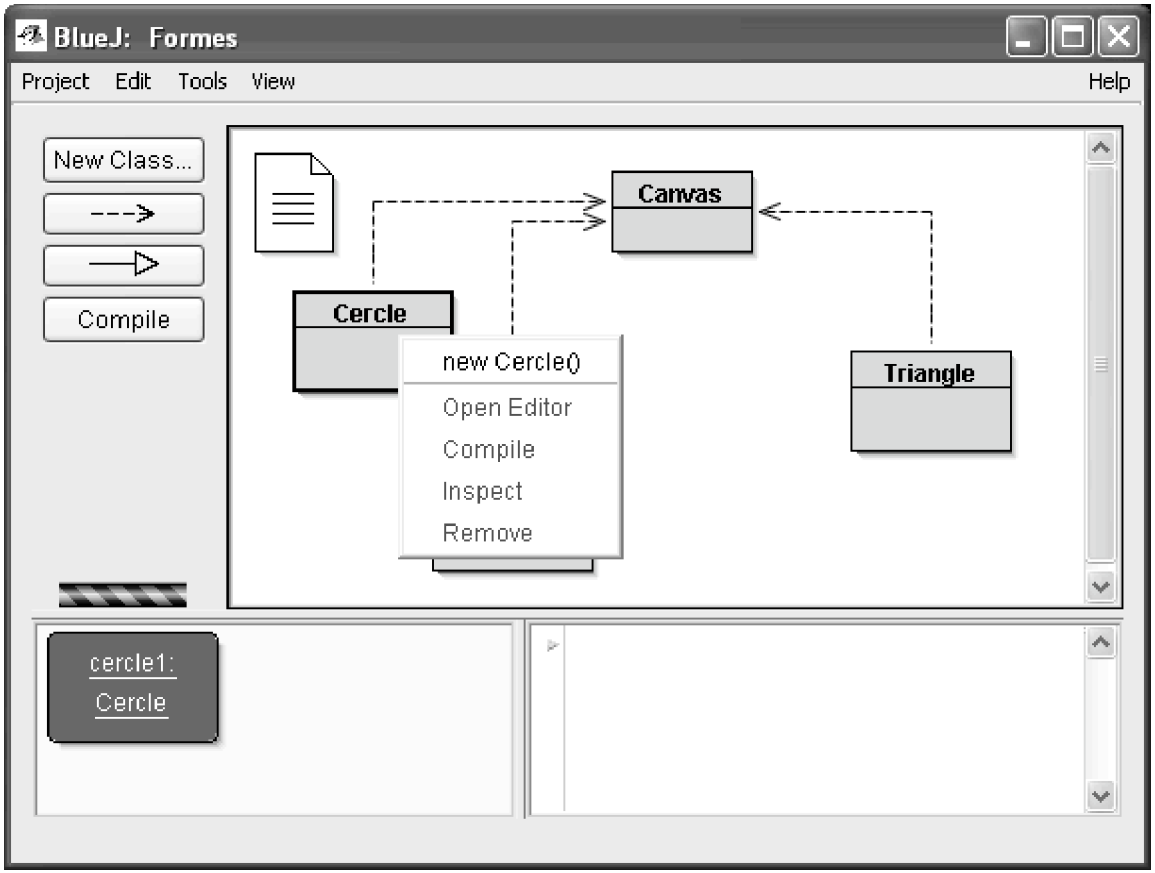
Les messages acceptés par un objet

• Une classe déclare aussi des **méthodes**, c'est-à-dire des messages compris par ses objets.
Ex : un ***`Cercle`*** peut apparaître (***`montre`***), être déplacé horizontalement (***`bougeHorizontal`***), etc…



Le logiciel BlueJ

`www.bluej.org`



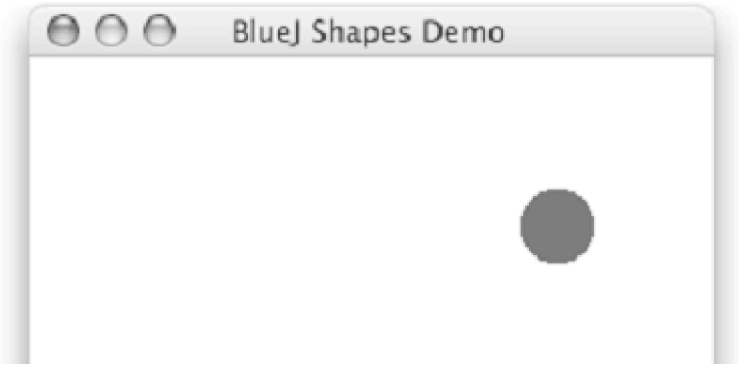

```
Cercle cercle1 = new Cercle();
```

Soit ***`cercle1`*** l'objet cercle obtenu comme nouvelle instance de la classe ***`Cercle`***.

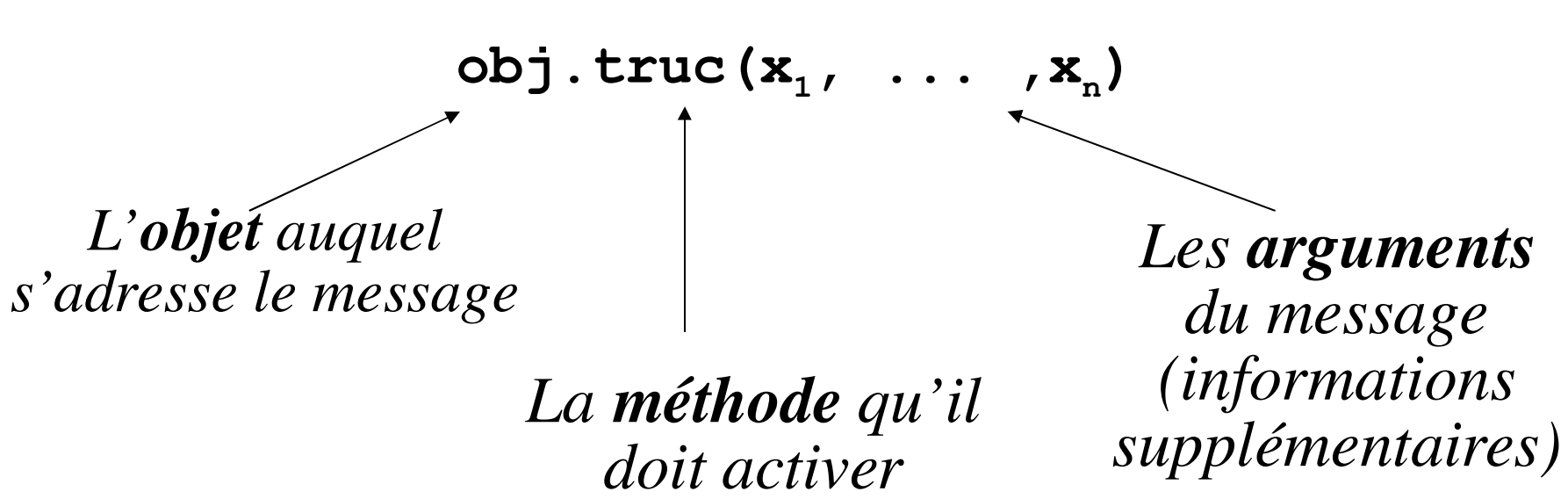
Appel de méthode (⇔ envoi de message)

```
cercle1.montre() ;

cercle1.bougeHorizontal(100) ;
```



• On communique avec un objet **en lui envoyant un message** (*en invoquant l'une de ses méthodes*). Cela amène l'objet à réaliser une action (et peut-être à renvoyer un résultat).



- Une méthode possède une **signature** :

```
void montre()

void bougeHorizontal(int distance)
```

Le type de la valeur renvoyée

Le nom de la méthode

Ici un seul paramètre et son type

Types de données

• Tout paramètre possède un **type**. Le type définit les catégories de valeurs qui peuvent être associées à un paramètre. Ex: le paramètre **`distance`** est de type **`int`**.

• En Java il y a deux sortes de types :

- les **types primitifs** (nombres, caractères, booléens).

- les **classes** d'objets (**`Cercle`**, **`Triangle`**, **`String`**…).

```
void bougeHorizontal(int distance)
void changeTaille(int hauteur, int largeur)
void changeCouleur(String couleur)
```

La classe String : les textes

• Une **chaîne de caractères** (mot, portion de phrase) est une instance de la classe **`String`**.

```
"Elle ajoute 243 !"        "2005"        "rouge"
```

```
cercle1.changeCouleur("rouge") ;
```

• L'opérateur + permet de calculer la **concaténation** de deux chaînes de caractères :

```
cercle1.changeCouleur("rou" + "ge") ;
```

L'ETAT d'un objet

• Chaque objet dispose d'un **état** représenté par les valeurs stockées dans ses **champs**.

• Un objet de la classe **`Cercle`** possède 5 champs →

```
class Cercle {
   int diametre;
   int xPosition;
   int yPosition;
   String couleur;
   boolean estVisible;

        < texte de la classe >
}
```

• La valeur du champ **`diametre`** d'un objet **`toto`** de la classe **`Cercle`** s'obtiendra par l'expression :

```
toto.diametre
```

La « notation pointée »

- Le point « . » possède ainsi plusieurs significations dans le langage Java :

  - Il représente la virgule dans un nombre approché : **`3.1416`**
  - Il sépare le nom d'un objet du message (méthode) qui lui est envoyé : **`cercle1.changeCouleur("jaune");`**

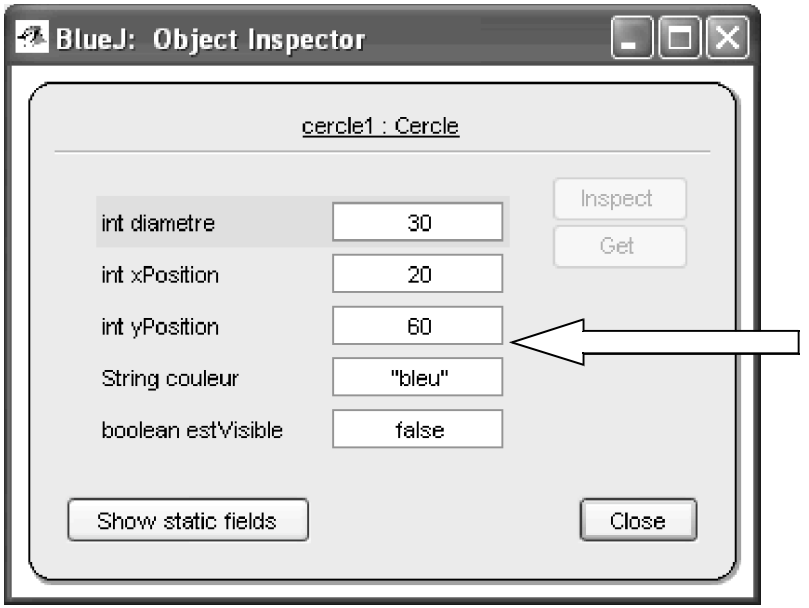
Hé, **`cercle1`**, change ta couleur en jaune !

  - Il sépare le nom d'un objet du nom d'un de ses champs :

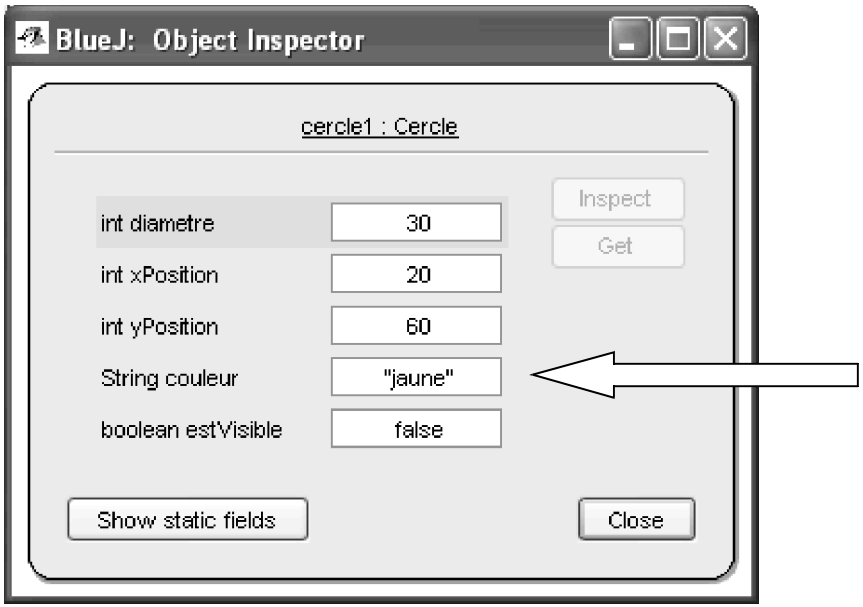
`cercle1.diametre`

Le diamètre de **`cercle1`**

• Lorsqu'on envoie un message à un objet en invoquant l'une de ses méthodes, il se peut que l'état de l'objet change !




```
cercle1.changeCouleur("jaune") ;
```



Le type primitif boolean : les « booléens »

• Un booléen est l'une des deux valeurs de vérité de la logique classique :

true	**false**
vrai	*faux*

• Attention, ce ne sont pas des objets !

• Voici le code Java de la méthode **`montre()`** de la classe **`Cercle`** :

```
void montre() {
   // changement d'état:
   this.estVisible = true;
   // appel de la méthode dessine sur l'objet courant:
   this.dessine();
}
```