Introduction à la programmation (Java)

(D’après des slides de Pierre Poulin); A ne pas distribuer

Buts

- Connaître certains principes (élémentaires) de la programmation
- Savoir écrire des programmes simples en Java
- Un peu de graphisme pour des petites applications
- Savoir structurer sa pensée (une machine exécute)
- Implantation/Implémentation d’algorithmes
- Connaître plus intuitivement les limites de la programmation (temps, difficultés, etc.)

Qu’est-ce qu’un programme ?

- Souvent une suite d’instructions (mais pas que; on trouve : définitions récursives, de donnés, de modules, d’objets, etc.)
- Exemple : une recette de cuisine, comment tracer un triangle équilatéral, trouver le déterminant d’une matrice, etc.
- Exemple (mauvais style, on verra plus tard pourquoi) :
  1. Faire A (un calcul)
  2. Faire B
  3. Tester une condition: si satisfaite aller à 2, sinon, aller à 4
  4. Faire C

Un programme est écrit dans un langage

- Langage machine/assembleur (add 12, …): bas niveau (rare)
- Langage haut niveau, souvent un mélange de :
  - Procédural : C, Fortran, Ada, etc.
  - Orienté Objet (OO) : Java, VB, C++, Python, etc.
  - Fonctionel : Ocaml, Lisp, Scala, etc.
  - Logique : Prolog, Coq, etc.

Les étapes

- Etudier le pb, l'environnement, les spécifications, les algos, les exigences, etc. (génie logiciel)
- Écrire un programme dans un (ou des) langage
- Compiler les sources (ou une partie) du programme :
  - Traduire le programme dans un langage de bas niveau (machine)
  - [éventuellement optimisations]
  - Produire un programme (code) exécutable (linker)
- Exécution
  - Charger le programme en mémoire (typiquement en tapant le nom du programme exécutable)
  - Charger les bibliothèques systèmes (de l'OS) nécessaires
  - Exécution

  Généralement, les compilateurs génères des exécutables pour les machines physiques+OS (Pcs, consoles, portables, routeurs, voitures/avions, etc.). On trouve aussi des machines virtuelles (Java) qui sont elles mêmes exécutées par des machines physiques. Il y a aussi la virtualisation mais on verra cela en Ing1/Ing2.

Termes

- Programme source, code source
  - Programme écrit dans un langage
- Code machine, code exécutable
  - Programme dans un langage de machine, directement exécutable par la machine
- Compilation (compilateur)
  - Traduire un code source en code exécutable
  - Ou en un code pour une machine virtuelle (lent)
- Interpréteur
  - Certains langages (Bash) n'ont pas besoin d'être traduit en code machine; Une « machine virtuelle » (pour le Bash, un shell) exécute le code source directement (encore plus lent en général)
  - La machine effectue la traduction sur la volée (*on the fly*), instruction par instruction, et l'exécute (Prolog, JavaScript)
  - Mixe des 2 (Python génère et compile/exécute du code C à la volé). Idem pour certaines machines Java

Programmation

- Syntaxe d’un langage
  - Comment formuler des instructions correctes, des bon types de données, etc. (grammaire)
- Sémantique (opérationnelle)
  - Comment les instructions réalisent/s’exécutent
- Erreur
  - de compilation: typiquement reliée à la syntaxe (tracer un cerKle) ou au typages (additionner des choux et des patates)
  - d’exécution (plus difficile à détecter et corriger): division par zéro, tracer un cercle de rayon négatif, correction finale (je voulais une raclette, j’obtiens de l’aligot), ou divergent (ne termine jamais)

Java

- Langage orienté objet
  - Notions de classes, héritage, …
- Beaucoup d’outils disponibles (packages)
  - JDK (*Java Development Kit*)
- Historique
  - Sun Microsystems
  - 1991: conception d'un langage indépendant du hardware
  - 1994: browser de HotJava, applets
  - 1996: Microsoft et Netscape commencent à soutenir
  - 1998: l‘édition Java 2: plus stable, énorme librairie
  - Pleins de nouvelles versions depuis...rachat par Oracle

Nous utiliserons que des versions LTS (longue terme support)

Java

- Compiler un programme en *Byte Code*
  - *Byte code*: indépendant de la machine
  - Interprété par la machine
- *javac programme.java*
  - Génère programme.class
- *java programme*
  - Lance le programme

Écrire un programme

```
public class Hello
{
    public static void main(String[] args)
    {
    // afficher une salutation
    System.out.println("Hello, World!");
    }
}
```

Nom de la classe

Une méthode

commentaire

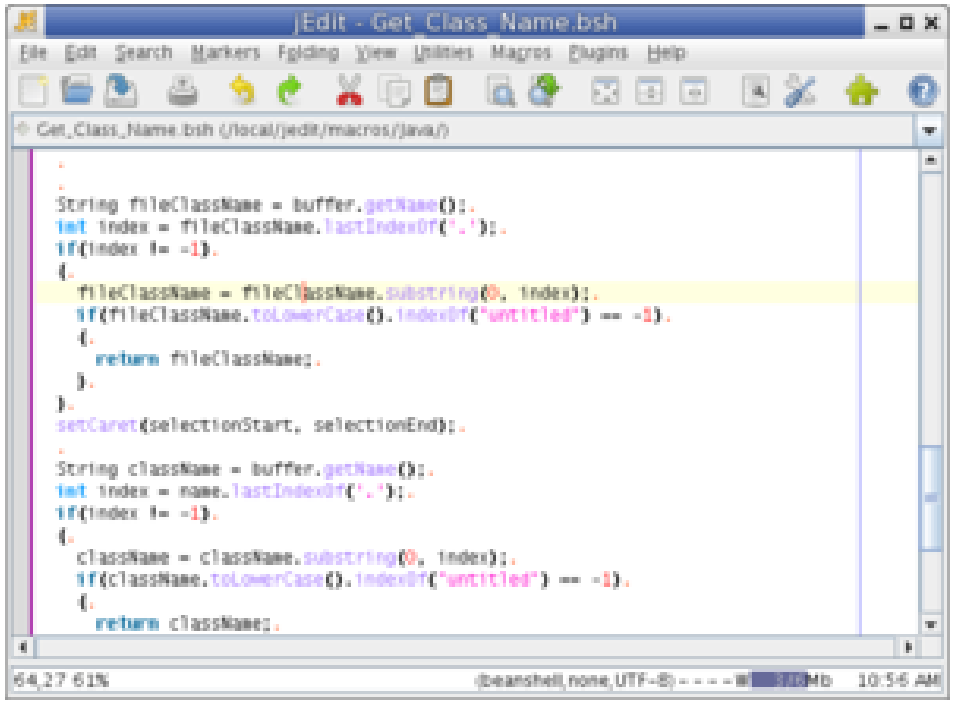
Une instruction

- Stocker ce programme dans le fichier Hello.java

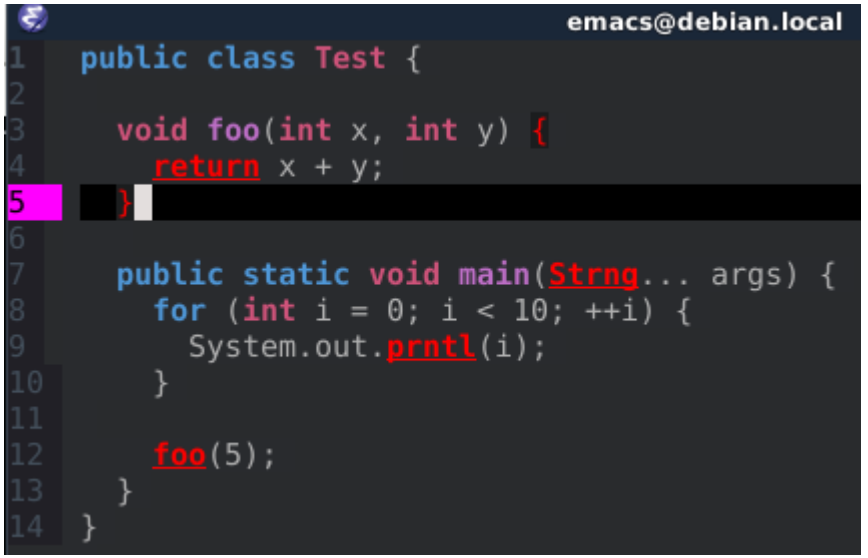
- IMPORTANT ≡ il existe des éditeurs de texte spécialisés pour la programmation (« emacs » (pas trop), Eclipse, NetBean etc.); Ceux-ci intègrent généralement des outils pour analyser le code (afficher les erreurs de statiques comme syntaxe/typage) mais aussi pour compiler et exécuter. Ce sont des IDE (Integrated Development Environment)

Exemple

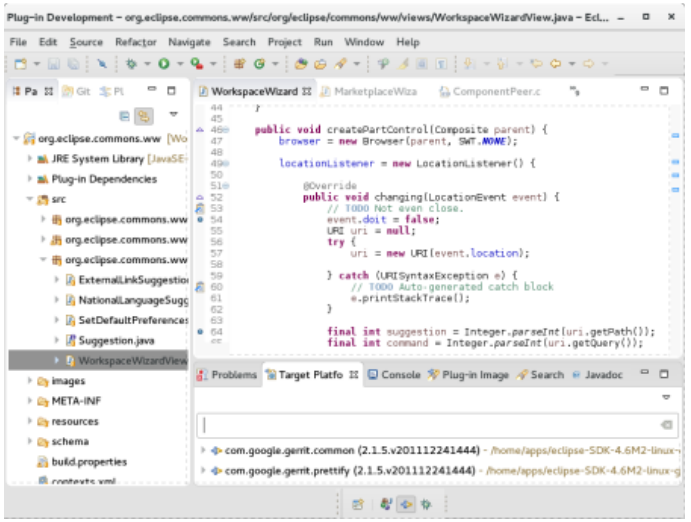
Jedit



Emacs



Eclipse

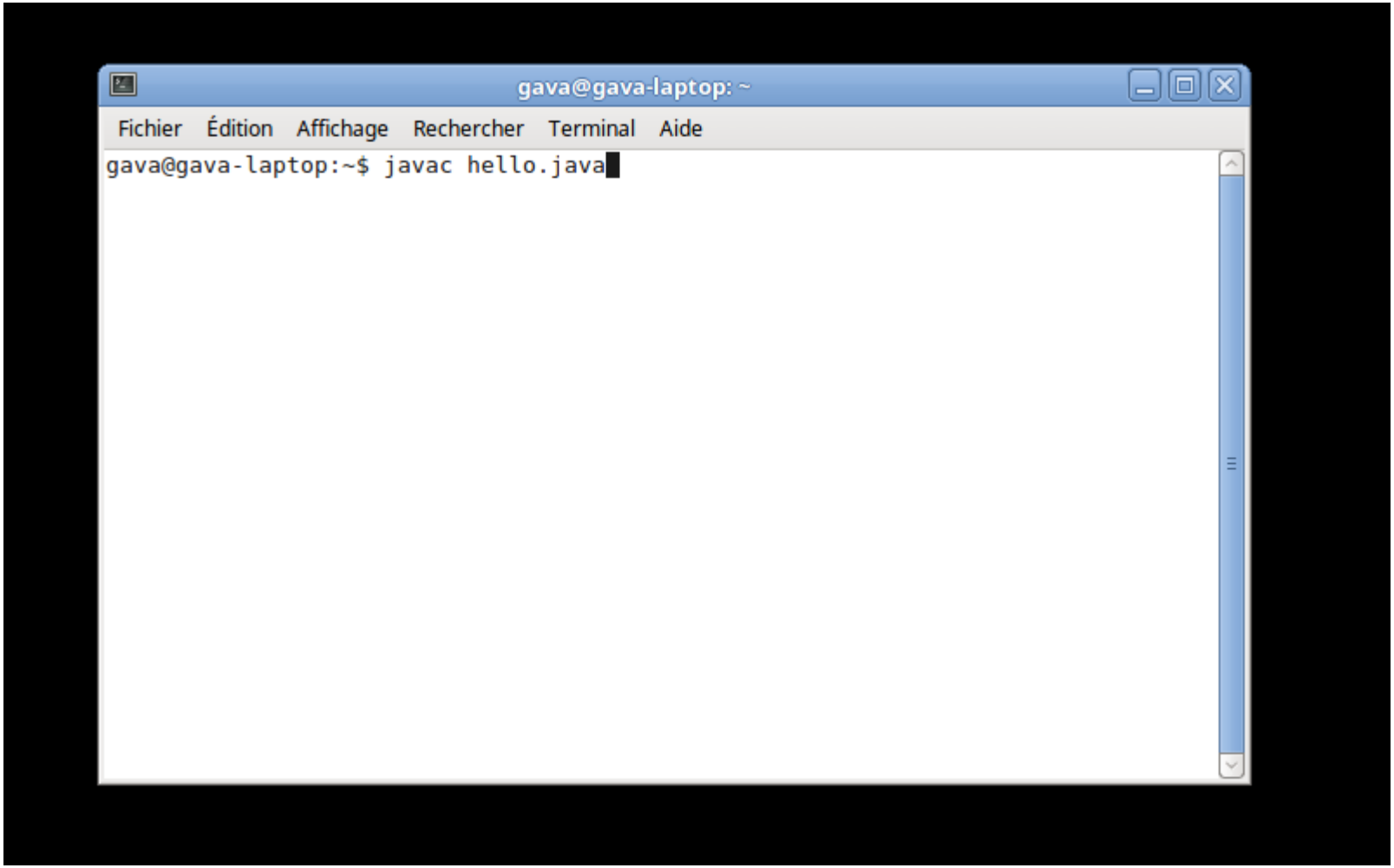


Lancer un programme (avec un shell=bash,dos,etc.)

- Compilation
  - *javac Hello.java*
  - Ceci génère *Hello.class*
- Lancer l'exécution
  - *java Hello*
- Résultat de l'exécution

  Hello, World!

Exemple


gava@gava-laptop: ~
Fichier Édition Affichage Rechercher Terminal Aide
gava@gava-laptop:~$ javac hello.java

Éléments de base dans un programme

- mots réservés: public class static void
- identificateurs: args Hello main String System out println
  - main String System out println: ont une fonction prédéfinie
- littéral: "Hello World!"
- ponctuation: { accolade } [crochet] (parenthèse)
- Commentaires
  - // note importante pour comprendre cette partie du code
  - /* … commentaires sur plusieurs lignes
    */

Classe

- Un programme en Java est défini comme une classe
- Dans une classe:
  - attributs, méthodes
- L'en-tête de la classe

  public class NomDeClasse
  - *public* = tout le monde peut utiliser cette classe
  - *class* = unité de base des programmes OO
- Une classe par fichier
- La classe NomDeClasse doit être dans le fichier NomDeClasse.java
- Si plus d'une classe dans un fichier *.java*, *javac* génère des fichiers *.class* séparés pour chaque classe

Classe

- Le corps

```
{
    …
}
```

- Contient les attributs et les méthodes
  - Attributs: pour stocker les informations de la classe
  - Méthodes: pour définir ses comportement, ses traitements, …
- Conventions et habitudes
  - nom de classe: NomDeClasse
  - indentation de { }
  - indentation de ...
  - Les indentations correctes ne seront pas toujours suivies dans ces notes pour des raisons de contraintes d'espace par PowerPoint…

Méthode: en-tête

- L'en-tête:

  public static void main(String[] args)
  - *main*: nom de méthode
  - *void*: aucune sortie (ne retourne rien)
  - *String[] args*: le paramètre (entrée)
    - *String[]*: le type du paramètre
    - *args*: le nom du paramètre
- Conventions
  - nomDeParametre
  - nomDeMethode
  - nomDAttributs
  - nomDObjet

Méthode: corps

- Le corps:

```
{
// afficher une salutation
System.out.println("Hello, World!");
}
```

- contient une séquence d'instructions, délimitée par { }
  - // afficher une salutation : commentaire
  - System.out.println("Hello, World!"): appel de méthode
- les instructions sont terminées par le caractère ;

Méthode: corps

- En général:

  nomDObjet.nomDeMethode(<liste des paramètres>)
  - System.out: l'objet qui représente le terminal (l'écran)
  - println: la méthode qui imprime son paramètre (+ une fin de ligne) sur un *stream* (écran)

    System.out.println("Hello, World!");
  - "Hello, World!": le paramètre de println
- La méthode main
  - "java Hello" exécute la méthode *main* dans la classe *Hello*
  - *main* est la méthode exécutée automatiquement à l'invocation du programme (avec le nom de la classe) qui la contient

Exercices

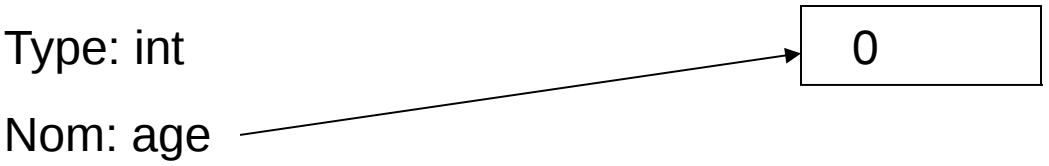
1) Recopiez, compilez puis exécutez le programme hello.java. Ensuite, testez aussi directement « java hello.java »...
2) Dans le programme « hello.java », testez les messages d'erreur du compilateur en modifiant chacun des mots du programme. Exemple, écrire « publiK », « Sistem », oubliez une parenthèse, ou une accolade, etc.
3) Écrire un programme qui affiche « hello » sur une ligne puis « ca ? » sur une deuxième ligne

Variable

- Variable: contient une valeur
  - Nom de variable
  - Valeur contenue dans la variable
  - Type de valeur contenue
    - *int*: entier, *long*: entier avec plus de capacité
    - *Integer*: classe entier, avec des méthodes
    - *float*: nombre réel avec point flottant, *double*: double précision
    - *String*: chaîne de caractères ("Hello, World!")
    - *char*: un caractère en Unicode (‘a’, ‘$’, ‘é’, …)
    - *booleen*: true/false
- Définition générale

  Type nomDeVariable;

  Exemple: int age;



Visualisation

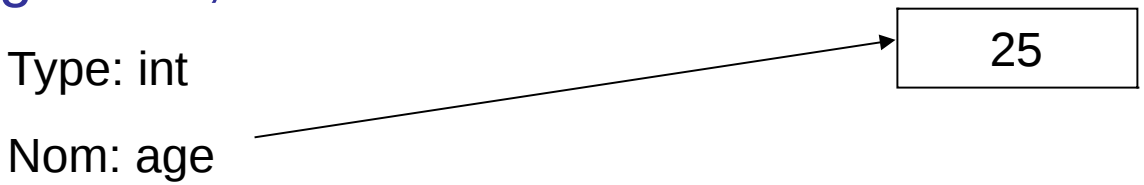
Maintenant visualiser nos programmes (que des simples, lecture clavier p. ex, ne fonctionne pas) :

http://www.pythontutor.com/

(et cliquer sur « Java Tutor »)

Modifier la valeur

- Affecter une valeur à une variable
- E.g. age = 25;

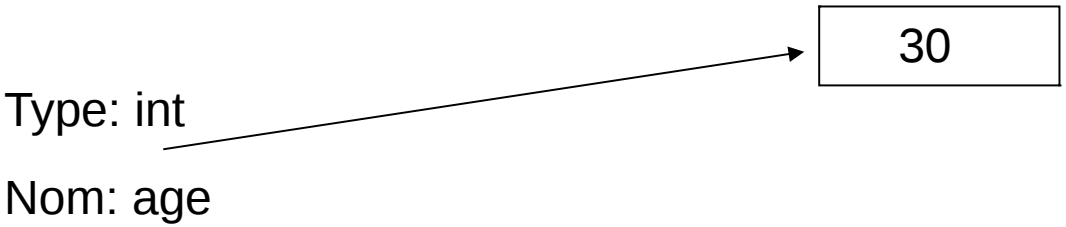


- Lire une variable

  age+5

- Les 2 combinés

  age=age+5



- Erreur si: age = "vingt cinq";
  - Type de valeur incompatible avec la variable

Exercices

1)Écrire un programme « age » (celui nous servira dans ses exercices à calculer votre age) avec 2 variables de noms « naissance » et « annee » et de type int, toutes les 2 initialisées à 0.

2)Faire afficher (4 lignes) :

Annee de naissance

0

Annee en cours

0

3) Essayer ensuite d'avoir

Annee de naissance : 0

Annee en cours : 0

Pour ce faire, utilisez println("hello "+10)

API

Java comporte bcp (vraiment) de bibliothèques, c'est-à-dire des class avec pleins de méthodes (savoir-faire) déjà faites pour vous. Exemple, graphisme, réseau, sécurité, I/O (input/ouput), etc. Regardons tous cela ensemble.

- https://docs.oracle.com/en/java/javase/11/
- Cliquez sur « API Documentation »
- Puis « java.base », « system », « PrintStream » de « out », « println(String x) »
- On regardez ensemble maintenant pour « in »
- Cela semble un peu difficile.
- Heureusement, il existe

int	**read**(byte[] b)	Reads some number of bytes from the input stream and stores them into the buffer array b.
int	**read**(byte[] b, int off, int len)	Reads up to len bytes of data from the input stream into an array of bytes.
byte[]	**readAllBytes**()	Reads all remaining bytes from the input stream.
int	**readNBytes**(byte[] b, int off, int len)	Reads the requested number of bytes from the input stream into the given byte array.

read

```
public int read(byte[] b, int off, int len) throws IOException
```

Reads up to len bytes of data from the input stream into an array of bytes. An attempt is made to read as many as len bytes, but a smaller number may be read. The number of bytes actually read is returned as an integer.

This method blocks until input data is available, end of file is detected, or an exception is thrown.

If len is zero, then no bytes are read and 0 is returned; otherwise, there is an attempt to read at least one byte. If no byte is available because the stream is at end of file, the value -1 is returned; otherwise, at least one byte is read and stored into b.

Lire des données

Récupérer une class (pour du savoir-faire)

```
import java.util.Scanner;

public class Main {
  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    System.out.println("Saisissez un entier : ");
    int i = sc.nextInt();
    System.out.println("Saisissez une chaîne : ");
    //On vide la ligne avant d'en lire une autre
    sc.nextLine();
    String str = sc.nextLine();
    System.out.println("FIN ! ");
  }
}
```

Nouvelle objet (instanciation)

Lire au clavier

Exercice (suite)

1) Modifier votre programme « age » pour qu'il affiche votre age (annee-naissance).
2) Testez votre votre code avec les combinaisons suivantes (bugs) :
    a) Naissance-annee
    b) naissance>annee
    c) Tapez une date de naissance + grande que 2019...Votre avis ?

Les comparaisons numériques

• Les **opérateurs de comparaison** usuels fonctionnent sur les nombres entiers et approchés :

Expression	*a pour valeur*	
3 > 5	→	false
3 < 5	→	true
5 < 2 + 1	→	false *(+ est prioritaire sur <)*

En cas de doute :
5 < (2 + 1)

• Les valeurs constantes true et false sont les **valeurs booléennes**. On peut afficher une valeur booléenne.

```
print(3 < log(2));
```

Run →

```
false
```

• Une **expression booléenne** est une expression dont la valeur est booléenne.

ATTENTION !

- Les signes ≤ et ≥ des maths se notent <= et >= en programmation !

	a pour valeur	
3 >= PI	→	false
3 <= 2 + 1	→	true

- Le signe = des maths se note == en Processing/Java/C lorsqu'il sert à vérifier qu'une égalité est vraie.

pow(2,10) == 1024	→	true
PI == 3.1416	→	false
PI == 3.1415927	→	true

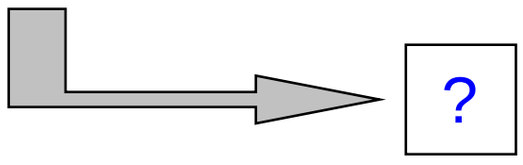
- Mais ne vous fiez pas au dernier exemple ! En principe **il est illusoire de demander l'égalité <u>exacte</u> entre deux nombres <u>approchés</u>** !

0.1+0.1+0.1+0.1+0.1+0.1+0.1 == 0.7	→	false

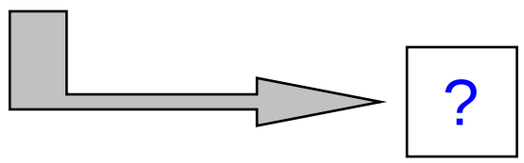
Distinguer = (affectation) et == (égalité)

- C'est l'une des erreurs majeures du débutant !

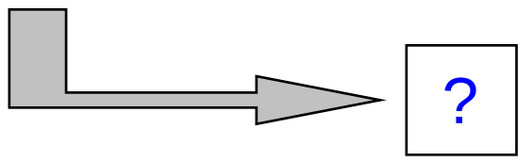
```
int x = 2, y = 3, z = 6;    // déclarations + initialisations
z = z + 1;     //  z  devient égal à z + 1, donc à 7
y == x + 1     // est-ce que y vaut x + 1, c'est à dire 3 ?
```




```
x == y + 1     // est-ce que x vaut y + 1, c'est à dire 4 ?
```




```
x + 2 == y - 1     // est-ce que x+2 vaut y-1, c'est à dire 4==2 ?
```



- Le résultat de x == y est bien un **booléen**.

L'expression conditionnelle if ... else ...

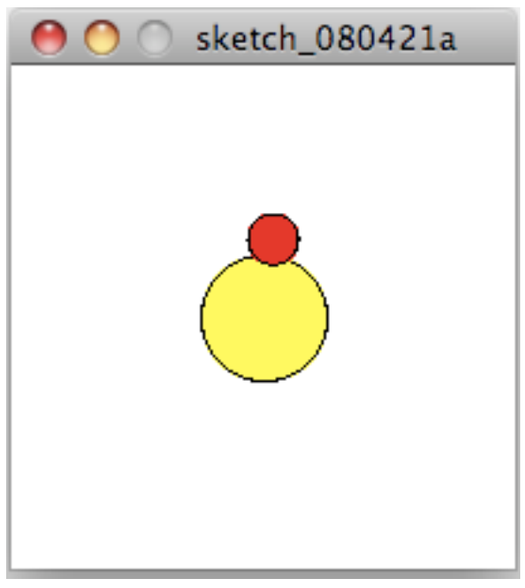
- Vitale, elle permet de prendre une décision à un moment donné.

SI ... ALORS ... SINON ...

if (...) { ... } else { ... }

- Exemple. Une **balle vibrante** rouge part du centre du canvas, elle suit un mouvement aléatoire.

- On utilise la méthode random(n) dont la signature est `float random(float n)` et dont le résultat est un nombre approché pseudo-aléatoire de [0;n[.
Par exemple, random(6) ∈ [0.0;6.0[au petit bonheur la chance...



- Version 1. On laisse la balle aller où elle veut...

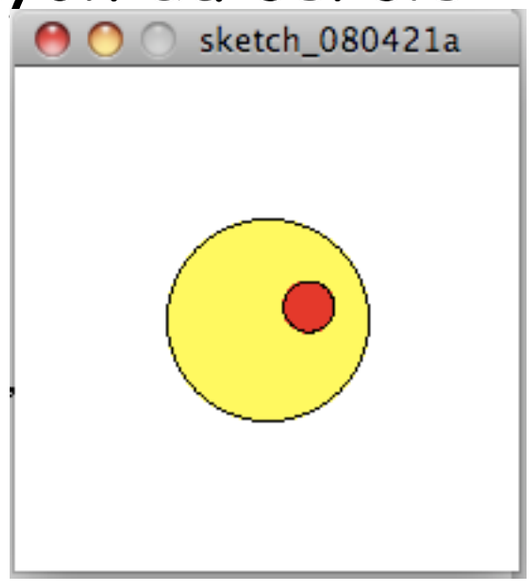
```
int largeur = 200, hauteur = 200;          // dimensions du canvas
int rayonBalle = 10;                       // rayon de la balle rouge
int rayonCercle = 50;                      // rayon du cercle jaune
float x, y;                                // position de la balle

void setup() {
  size(largeur, hauteur);
  x = largeur / 2;                         // position initiale
  y = hauteur / 2;                         // de la balle
}

void draw() {
  background(255,255,255);                 // couleur de fond du canvas
  fill(255,255,0);                         // remplissage du cercle en jaune
  ellipse(largeur/2, hauteur/2, 2 * rayonCercle, 2 * rayonCercle);
  fill(255,0,0);                           // remplissage de la balle en rouge
  ellipse(x, y, 2 * rayonBalle, 2 * rayonBalle);

  float dx = random(6) - 3, dy = random(6) - 3;    // déplacement aléatoire
  x = x + dx;                                      // et mise à jour de la position
  y = y + dy;
}
```

- Version 2. Je veux **empêcher son centre de sortir du cercle** jaune. Je teste la distance des centres et la compare au rayon du cercle :




```
void draw() {
  background(255,255,255);
  fill(255,255,0);
  ellipse(largeur/2, hauteur/2,2*rayonCercle,2*rayonCercle);
  fill(255,0,0);
  ellipse(x,y,2*rayonBalle,2*rayonBalle);
  float dx = random(6)-3, dy = random(6) - 3;
  float xSuiv = x+dx;        // la position suivante
  float ySuiv = y+dy;
  if (distance(xSuiv,ySuiv,largeur/2,hauteur/2) < rayonCercle) {
    x = xSuiv;               // si la position suivante est encore dans
    y = ySuiv;               // le cercle, je peux mettre à jour la position
  }                          // sinon je ne fais rien...
}

float distance(float x1, float y1, float x2, float y2) {
  return sqrt( (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) );
}
```

une méthode auxiliaire

- Dans l'exemple précédent, nous avons utilisé la forme :

SI ... ALORS ...

qui s'exprime en Java sous la présentation suivante :

```
if (condition) {
  ...
}
```

- Notez que la condition est une expression booléenne entre **parenthèses**, et que les instructions qui seront exécutées si la condition est vérifiée sont entre **accolades**.

- Une suite d'instructions entre accolades se nomme un **bloc**.

```
if (condition) {
  instruction1;
  instruction2;
  ...
}
```

La clause else

• Mais dans la plupart des cas, nous devrons prévoir ce qui se passe si la condition **n'est pas** vérifiée.

SI ... ALORS ... SINON ...

qui s'exprimera cette fois sous la présentation avec deux blocs :

```
if (condition) {
  instruction1;
  instruction2;
  ...
} else {
  instruction8;
  instruction9;
  ...
}
```

• Vous aurez bien noté que le mot "then" est sous-entendu !

La clause else

• Exemple. Je veux que le contour du cercle soit électrifié ! Si la balle **touche le cercle**, elle reçoit une **décharge** et se retrouve brutalement au centre !

```
if (distance(xSuiv,ySuiv,largeur/2,hauteur/2) < rayonCercle) {
    x = xSuiv;
    y = ySuiv;
} else {
    println("Aïe !");
    x = largeur/2;
    y = hauteur/2;
}
```

• L'oubli du else est aussi une erreur fréquente dans les programmes !

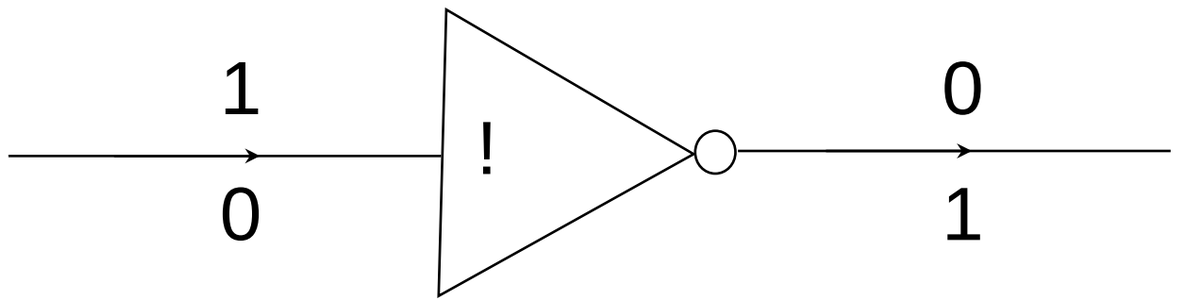
• Les instructions dans chaque bloc sont décalées par-rapport à la marge, pour bien montrer à quel niveau elles se trouvent. On dit qu'elles sont **indentées**. **Respectez l'indentation !**

Les opérateurs booléens ! && ||

• La **négation** d'une expression booléenne E se note !(E). Par exemple, la négation de x == y se note !(x == y), dont l'abréviation est x != y

• Attention, la négation de x < y est x >= y.

• Si E vaut true, alors !E vaut false, et inversement. Les électroniciens disent que le circuit not est un ***inverseur***.



la porte "NOT"

• Exemple : si le nombre entier n n'est pas premier :

```
if (!premier(n) {
  ...
}
```

avec

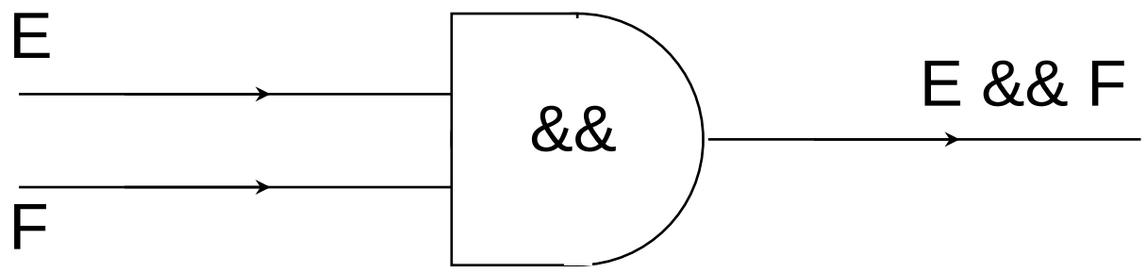
```
boolean premier (int n) {
 ...
}
```

• La **conjonction** $E \; et \; F$ de deux expressions booléennes E et F se note en Java E && F. Par exemple, pour exprimer que n est premier et que n est plus grand que 100, la condition s'écrira :

premier(n) && (n > 100)

• Une expression booléenne E && F vaut true si et seulement si E vaut true **et** F vaut true. Elle vaut false dans tous les autres cas.

• Les électroniciens schématisent un circuit ET de la manière suivante :



la porte "AND"

• COURT-CIRCUIT ! Attention, dans l'expression E && F, l'expression E est évaluée en premier. Si elle vaut false, le résultat du && est false sans avoir besoin d'évaluer F. Ceci est TRES IMPORTANT !

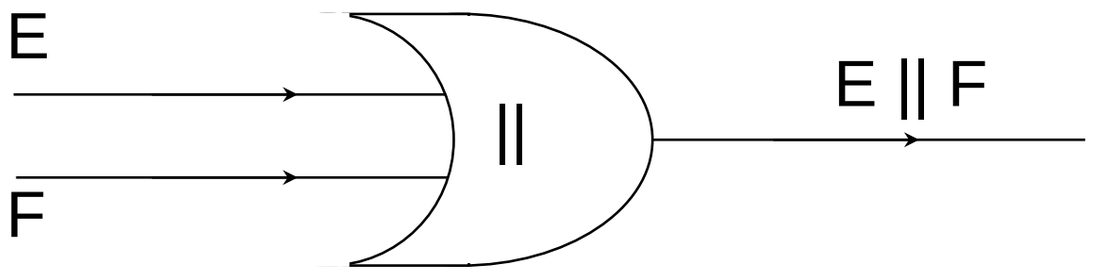
```
if ((x != 0) && (1/x > y)) ...
```

• La **disjonction** $E \text{ ou } F$ de deux expressions booléennes E et F se note en Java E || F. Par exemple, pour exprimer que n est premier ou qu'il est plus grand que 100, la condition s'écrira :

premier(n) || (n > 100)

• Une expression booléenne E || F vaut false si et seulement si E vaut false **et** F vaut false. Elle vaut true dans tous les autres cas.

• Les électroniciens schématisent un circuit OU de la manière suivante :

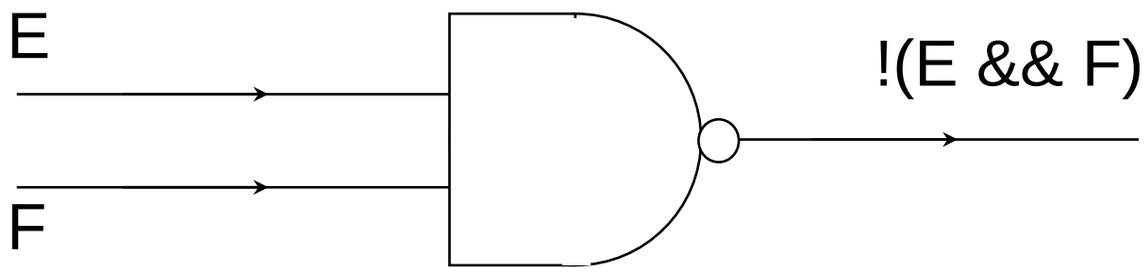


la porte "OR"

• COURT-CIRCUIT ! Attention, dans l'expression E || F, l'expression E est évaluée en premier. Si elle vaut true, le résultat du || est true sans avoir besoin d'évaluer F. Ceci est TRES IMPORTANT !

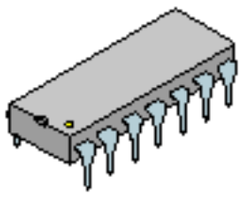
```
if ((x > 0) || (x < -2)) ...
```

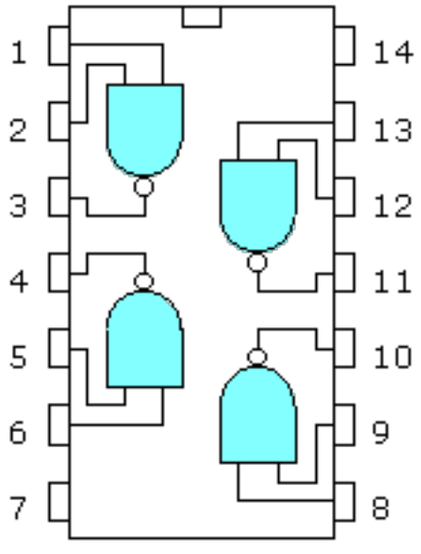
• On peut montrer [cf. TD] que **la seule porte NAND** [la négation du AND] **suffit à engendrer les autres portes** ! Cependant elle n'existe pas en Java ! Les électroniciens la connaissent bien.

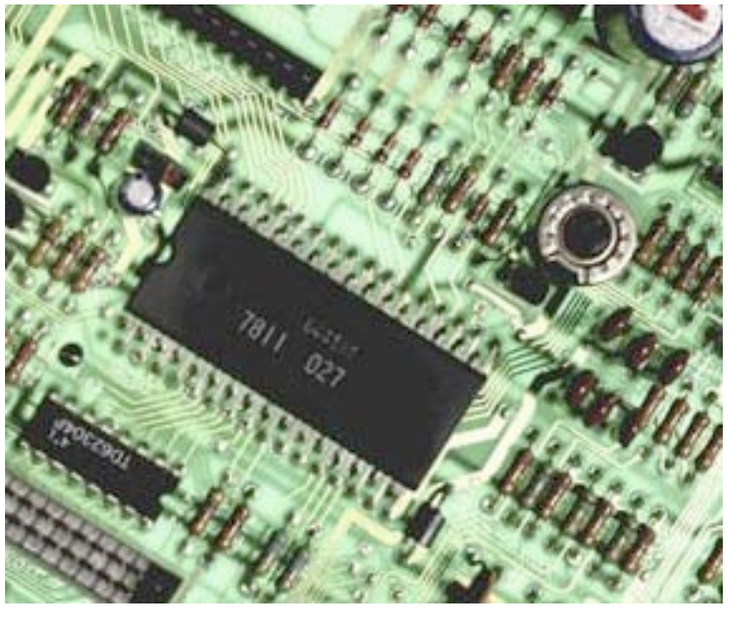


la porte "NAND"

• Les **portes logiques** sont de petits circuits électroniques qui forment, avec les transistors, le coeur de nos ordinateurs :







Les Méthodes

- Les langages objets ont des **méthodes**, les autres ont des **fonctions**

- Une **méthode** peut **avoir ou non un résultat** ! Une méthode sans résultat se contente d'effectuer une action (afficher du texte, déplacer une balle dans le canvas, modifier une variable, etc).

void méthode(int x)

↑ int ⟶ Ø

pas de résultat,
seulement une action !

float méthode(int x)

↑ int ⟶ float

un (seul) résultat,
de type float.

- Par exemple, les méthodes print(...), line(...), background(...) n'ont pas de résultat.
- Par contre, les méthodes log(...), sin(...), sqrt(...) renvoient un résultat.

- Notre méthode distance(...) de la page 7 renvoie un résultat.
- L'ordre de déclaration des méthodes n'a pas d'importance !

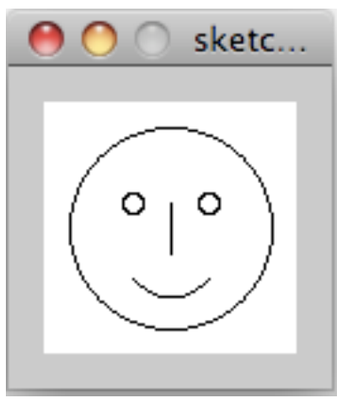
Les méthodes sans résultat

- Une méthode sans résultat se contente d'**effectuer une action**.

- Les plus célèbres en Processing sont setup() et draw(). La méthode **setup()** initialise l'animation, tandis que la méthode **draw()** dessine l'image courante et effectue les actions de mise à jour pour passer à l'image suivante.

- Voici une méthode qui affiche un *smiley* à l'écran, sans résultat. Aucune animation, donc seulement la méthode setup() :

```
void smiley() {
    fill(255,255,255);
    ellipse(50,50,80,80);
    ellipse(35, 40, 8, 8);
    ellipse(65, 40, 8, 8);
    line(50,40,50,60);
    bezier(35,70,45,80,55,80,65,70);
}
```




```
void setup() {
  size(100,100);
  background(255,255,255);
  smiley();
}
```

Une méthode ne fait rien si on ne l'invoque (appelle) pas !

Les méthodes avec résultat

- Une méthode avec résultat va **effectuer un calcul** et **renvoyer le résultat** de ce calcul à celui qui l'a appelée/invoquée.

- Le renvoi du résultat à l'appelant se fait avec l'instruction **return** qui abandonne immédiatement la fonction, avec le résultat sous le bras !

```
float foo(int x) {
  float res;
  if (x>0) {
    res = x;
  } else {
    res = (x + sin(x)) / 2;
  }
  return res;
}
```

signature

***déclaration** du résultat (bien typé)*

***calcul** du résultat*

***retour** du résultat*

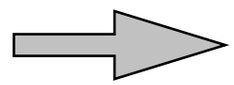
```
void setup() {
  println("foo(5) = " + foo(5));
  println("foo(-5) = " + foo(-5));
}
```

Run

```
foo(5) = 5.0
foo(-5) = -2.0205379
```

- On peut parfois simplifier le schéma précédent :

```
float moyenne(float x, float y) {
  float res;
  res = (x + y) / 2;
  return res;
}
```



```
float moyenne(float x, float y) {
  return (x + y) / 2;
}
```

- Une factorielle fac(n) par **récurrence sur n** :

```
int fac(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * fac(n - 1);
  }
}
```

0! = 1

n! = n x (n-1)! si n> 0

```
void setup() {
    println("fac(10) = " + fac(10));
    println("fac(17) = " + fac(17));
}
```


```
fac(10) = 3628800
fac(17) = -288522240
```

!

Université
Nice
Sophia Antipolis

Variables et méthodes

3 types de variables

```
int largeur = 100, hauteur = 100;


void cercle_inscrit(int gauche, int droite, int haut, int bas) {
  float x = moyenne(gauche, droite);
  float y = moyenne(haut, bas);
  ellipse(x, y, gauche - droite, haut - bas);
}

void setup() {
  size(largeur, hauteur);
  cercle_inscrit(0, largeur, 0, hauteur);
}
```

3 types de variables

```
// largeur et hauteur sont des champs
int largeur = 100, hauteur = 100;

// gauche, droite, haut, bas sont des paramètres
void cercle_inscrit(int gauche, int droite, int haut, int bas) {
    // x et y sont des variables locales
  float x = moyenne(gauche, droite);
  float y = moyenne(haut, bas);
  ellipse(x, y, gauche - droite, haut - bas);
}

void setup() {
  size(largeur, hauteur);
  cercle_inscrit(0, largeur, 0, hauteur);
}
```

Portée et durée de vie d'une variable

- La **portée (visibilité) d'une variable** est la partie du code source d'où l'on peut accéder à sa valeur.
  - La portée des *champs* est la totalité des méthodes définies.
  - La portée des *paramètres* et des *variables locales* est le bloc de déclaration (i.e. méthode ou accolades les plus proches)

- La **durée de vie d'une variable** correspond à la période pendant laquelle elle subsiste avant sa destruction.
  - Les *champs* gardent leur valeur au cours de la vie du programme.
  - Les *paramètres* et les *variables locales* ont une durée de vie limitée au seul appel du bloc (durée de l'appel de la méthode).

3 types de variables

```
// largeur et hauteur sont des champs
int largeur = 100, hauteur = 100;                                   largeur, hauteur

// gauche, droite, haut, bas sont des paramètres
void cercle_inscrit(int gauche, int droite, int haut, int bas) {    gauche, droite,
                                                                    haut, bas
  // x et y sont des variables locales
  float x = moyenne(gauche, droite);                                x
  float y = moyenne(haut, bas);                                     y
  ellipse(x, y, gauche - droite, haut - bas);
}

void setup() {
  size(largeur, hauteur);
  cercle_inscrit(0, largeur, 0, hauteur);
}
```