Syntaxe Java

Info 211a

Alice Jacquot (d'après un cours de Guillaume Wisniewski)
jacquot@lri.fr

septembre 2020

Université Paris-Saclay & LRI

Présentation de Java

Rappel : qu'est-ce qu'un programme ?

Rappel : qu'est-ce qu'un programme ?

réponse cynique quelque chose qui contient des bugs

réponse classique instructions pour un ordinateur

meilleure réponse instructions pour un ordinateur compréhensibles par un être humain

$\Rightarrow$ langage de programmation

Qu'est-ce que Java ?

- un langage de programmation libre
- développé en 1995 par Sun puis racheté par Oracle (2009)
- syntaxe proche de celle du C et du C++
- aujourd'hui le langage le plus utilisé dans le monde de l'entreprise
- objectif : multi-plateforme « write once, run everywhere »

Java Virtual Machine

Machine virtuelle

- le compilateur Java génère du byte code et non de l'assembleur (langage machine)
- le byte code est exécuté sur une machine virtuelle : la JVM
- approche adoptée par la plupart des langages récents (.NET, python, ...)

Intérêt

- programmation de plus haut niveau (p.ex. plus de gestion manuelle de la mémoire)
- indépendance de la plate-forme : windows, linux, mac, web, téléphone, ...

Les avantages de Java

- multi-plateforme
- facilité de programmation (gestion de la mémoire)
- bonnes performances
- beaucoup d'outils pour aider le développement/débugage
- beaucoup de bibliothèques $\Rightarrow$ couvre tous les besoins courants
- langage universel : tous les informaticiens ont déjà fait du java

Les inconvénients

- verbeux (beaucoup de lignes pour des choses simples)
- peu de souplesse (peut-être considéré comme un avantage)
- coût d'entrée élevé

Pourquoi apprendre le java ?

Pour les (futurs) informaticiens

- langage le plus utilisé dans le monde de l'entreprise

Pour les autres

- langage de haut niveau
- principe de la programmation orientée objet

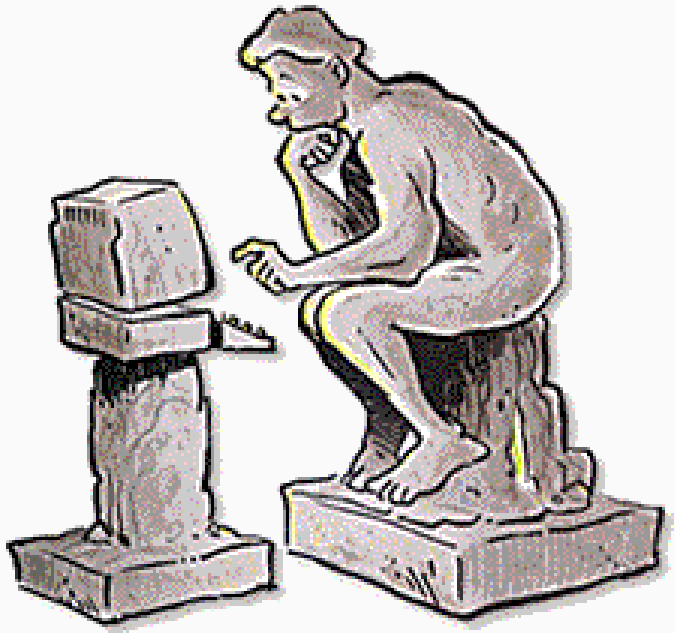
Les bases du langage Java

Disclamer

- La suite est un **rappel** rapide du cours de programmation impérative
- programmation impérative = exécution séquentielle d'instructions
- nouveauté = notion d'objets $\Rightarrow$ dans la partie suivante

Executive summary

Comment écrire un programme java ?



$\Rightarrow$ Faire comme si c'était du C...

Élément d'un langage informatique

des primitives : entités/instruction la plus simple qu'un langage permette de décrire ;

un moyen de combiner des primitives : pour construire des « actions » complexes à partir des primitives ;

un moyen d'abstraction : qui permet de nommer ces combinaisons et de les manipuler

Un premier programme (1)

Dans un fichier `Hello.java` (nom & majuscule impératifs) :

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("Hello world !");
5      }
6
7  }
```

compilation (`javac Hello.java`)
compile et crée un fichier `Hello.class`

exécution (`java Hello`) La JVM recherche dans `Hello.class` la méthode
`public static void main(String[])`
et l'exécute, si elle n'existe pas : erreur

Un premier programme (2)

Dans un fichier `Hello.java` (nom impératif) :

```
1  public class Hello {
2
3      public static void main(String[] args) {
4          System.out.println("Hello world !");
5      }
6
7  }
```

- seule partie utile : `System.out.println`
- instruction pour afficher une chaine de caractères
- tout le reste = particularités de java que l'on verra plus tard

Un programme plus utile (?)

```
1  public class Conv {
2
3      public static void main(String[] a) {
4          double tInF = 98.6;
5          double tInK = (tInF - 32.0) * (5 / 9) + 273.15;
6
7          System.out.println("t. in Kelvin: " + tempInK);
8      }
9
10 }
```

- convertit une température de degré Fahrenheit en Kelvin
- même instruction pour afficher
- variables

Flot d'exécution

paradigme impératif :

$\Rightarrow$ instructions exécutées de manière séquentielle

Comment concevoir un programme ?

Principe

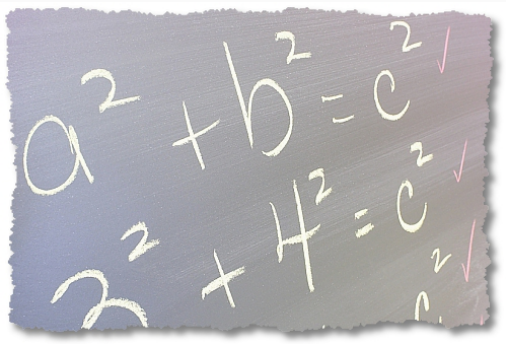
1. décomposer le problème en une suite d'instructions
2. « traduire » cette séquence d'instructions en java

Exemple

Pour trier un tableau :

- trouver le plus petit élément
- l'échanger avec l'élément en première position
- recommencer avec la partie non triée du tableau

Les variables (1)



- variable = nom pour des valeurs
- une variable doit d'abord être déclarée en spécifiant son type :

```
1 float maVariable;
```

- puis définie en lui donnant une valeur :

```
1 maVariable = 12.0;
```

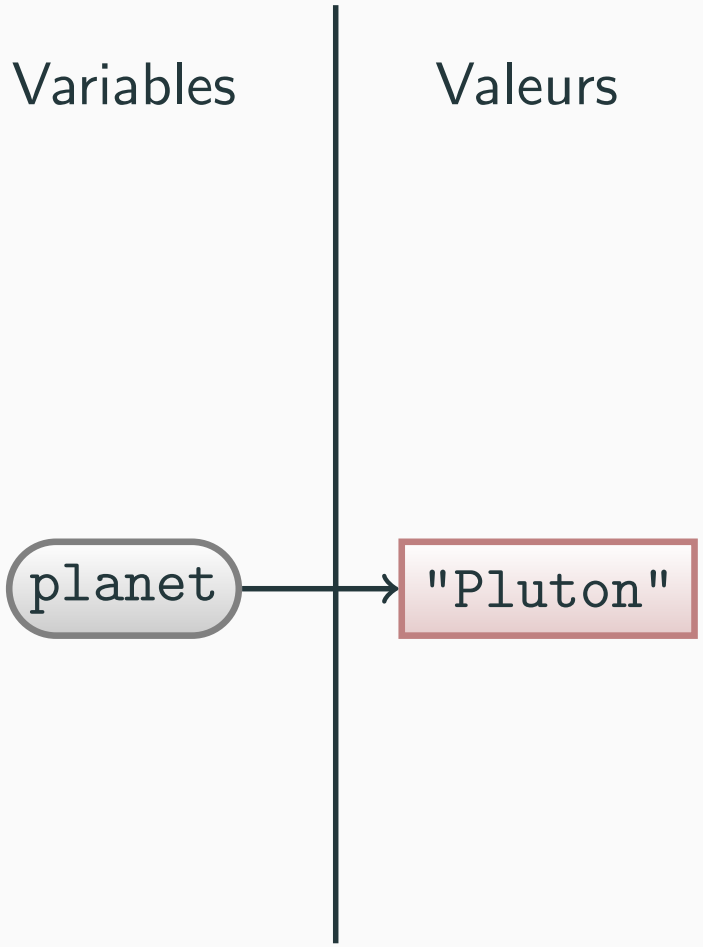
- en une instruction :

```
1 float maVariable = 12.0;
```

- on peut affecter à une variable soit une valeur, soit le résultat d'une opération
- remarque importante : convention java pour nommer les variables : `maVariable`, `uneAutreVariable`, ...

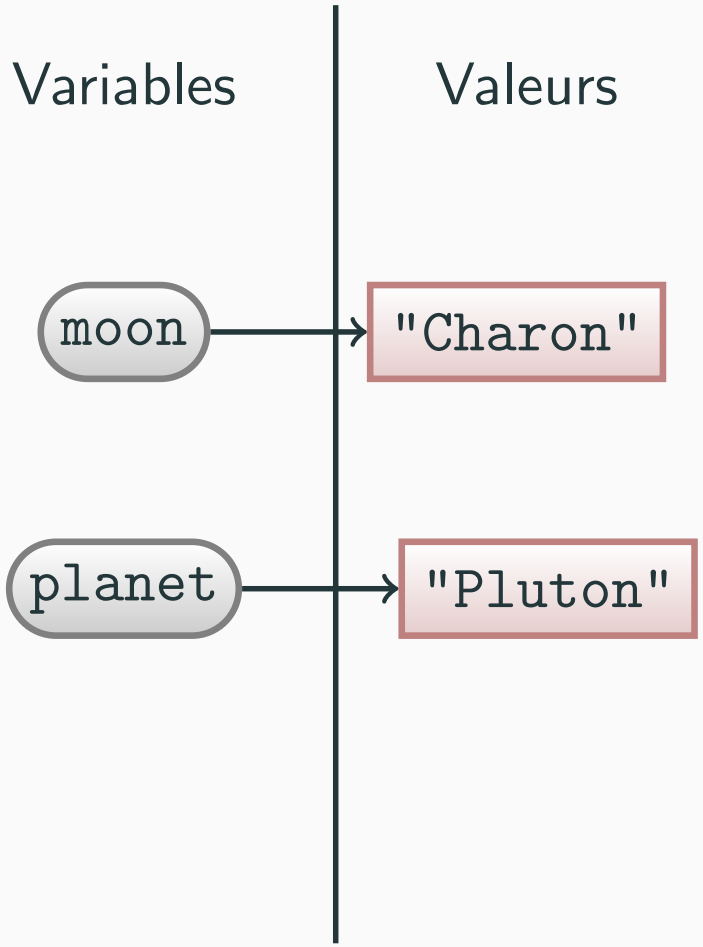
Les variables (2)

```
1  String planet = "Pluton";
```



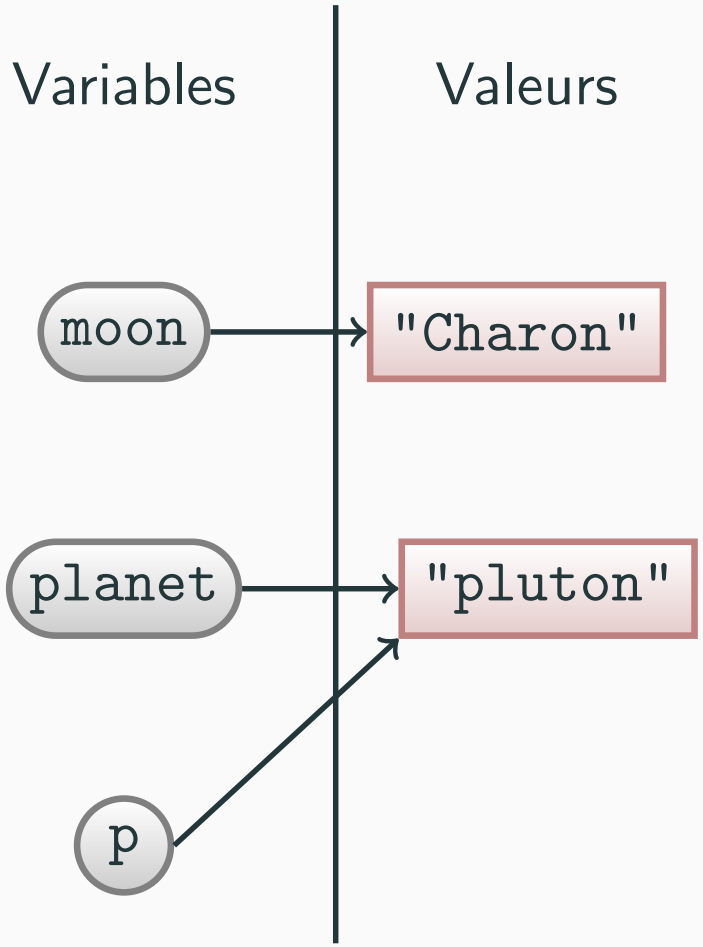
Les variables (2)

```
1  String planet = "Pluton";
2  String moon = "Charon";
```



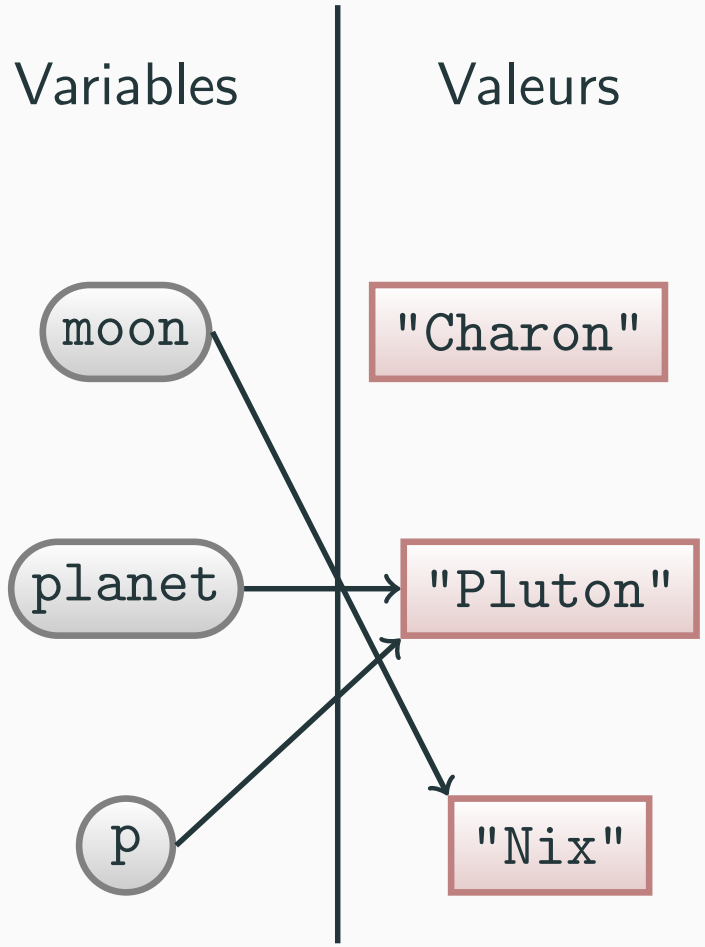
Les variables (2)

```
1  String planet = "Pluton";
2  String moon = "Charon";
3  String p = planet;
```



Les variables (2)

```
1  String planet = "Pluton";
2  String moon = "Charon";
3  String p = moon;
4  moon = "Nix";
```



Type

- un type défini :
  - un ensemble de valeurs
  - un ensemble d'opérations possibles
- exemple :
  - `int i` : `i` peut prendre toutes les valeurs de $\mathbb{N}$ + définition de l'addition, de la soustraction, ...
  - `String s` : `s` est une chaîne de caractères + définition de l'addition (concaténation) et interdiction de la multiplication :
- principaux types java : `int`, `double`, `char`, `boolean`

Types : exemples

```
1  String s1 = "bonjour";
2  String s2 = "tout le monde";
3
4  String s3 = s1 + " " + s2;
5  // s3 = "bonjour tout le monde"
6
7  s3 = s1 - s2;
8  // erreur de compilation : opération interdite
9
10 s3 = s1 + 1;
11 // ok : concaténation
12
13 s3 = s1 - 1;
14 // interdit
```

Les tableaux

- ensemble de valeurs de taille fixe + ordonné
- déclaration :

```
int[] t2;
String[] t3;
```

- création :

```
t = new int[5]; // t est un tableau de 5 int
int[] ti = {1, n, 4};
```

- la taille doit être précisée à la création + non modifiable
- les tableaux sont indexés à partir de 0 : `t[0] + t[1];`
- possibilité d'accéder à la taille d'un tableau : `t.length`

Note(s) pour plus tard

- les tableaux ne sont plus utilisés : on préfère les collections dynamiques `ArrayList`
- java fournit plein de méthodes de manipulation de tableaux (cf. la classe `Arrays`)

Portée d'une variable

- portée = partie du code dans laquelle la variable existe
- en java : portée = bloc :
  - création (réservation + initialisation de la mémoire) au début du bloc
  - destruction (libération de la mémoire) à la fin du bloc
- tout est fait automatiquement

Exemples

```
1       int a = 10;
2       for (int i = 0; i < 10; i++) {
3         int b = 12;
4         if (b % i == 0) {
5           System.out.println("ok");
6           int c = b % i;
7           System.out.println(c);
8         }
9       }
```

Exemples

```
1       int a = 10;
2       for (int i = 0; i < 10; i++) {
3         int b = 12;
4         if (b % i == 0) {
5           System.out.println("ok");
6           int c = b % i;
7           System.out.println(c);
8         }
9       }
```

Portée :

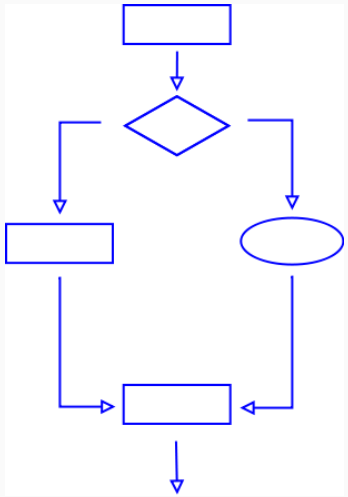
- a : 1–10
- b : 2–9
- c : 5–8
- i : 2–9

Structure de contrôle

Structures de contrôle

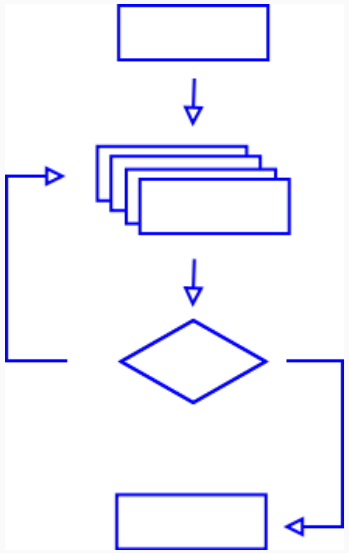
- programme informatique : exécution séquentielle d'instructions
- deux manière d'altérer l'exécution :

Sélection



Exécuter une série d'instructions plutôt qu'une autre

Répétition



Répéter l'exécution d'une série d'instructions

Structure conditionnelle (1/2)

L'instruction `if` teste une condition booléenne, si le résultat est vrai, le bloc d'instructions suivant la condition est exécuté, sinon il ne l'est pas :

```
1 if (condition) {
2   // bloc
3 }
```

Optionnellement, le premier bloc sera suivi du mot clé `else` et d'un second bloc exécuté seulement si la condition est fausse :

```
1 if (condition) {
2   // bloc 1
3 } else {
4   // bloc 2
5 }
```

Structure conditionnelle (2/2) I

```
1  public class TestIf {
2
3      public static void main (String[] args) {
4          int a = 2;
5          int b = 3;
6          int max = 0;
7
8          if (a < b) {
9              max = b ;
10         } else {
11             if (a == b) {
12                 System.out.println(a + " == " + b);
13                 System.exit(0);
14             }
```

Structure conditionnelle (2/2) II

```
15                max = a;
16            }
17            System.out.println("maximum = " + max) ;
18        }
19    }
```

Structure de répétition `for`

L'instruction `for` exécute une instruction `init` puis répète les instructions du bloc suivant l'instruction `for` tant que sa condition est vraie, à la fin de chaque répétition elle exécute son instruction `incr` :

```
for (init; condition; incr) {
  // bloc
}
```

Par exemple :

```
public class TestFor {

    public static void main (String[] args) {
        for (int i = 0; i < 3; i++) {
            System.out.println("i vaut " + i);
        }
    }
```

Exemple plus réaliste

L'exemple

Étant donné un ensemble/une population $x = (x_i)_{i=1}^n$, calculez :

- sa moyenne : $\bar{x} = \frac{1}{n} \times \sum_{i=1}^n x_i$
- son écart-type : $\sigma_x = \sqrt{\frac{1}{N} \times \sum_{i=1}^n (x_i - \bar{x})^2}$

Le contexte

- population (statistique) = ensemble (mathématique) = tableau (informatique)
- exemple : température de le semaine dernière :

lundi	mardi	mercredi	jeudi	vendredi	samedi	dimanche
20	21	22	25	26	25	21

Le code

```
1  int[] temp = {20, 21, 22, 25, 26, 25, 21};
2
3  // calcul de la moyenne
4  double acc = 0.0;
5  for (int i = 0; i < temp.length; i++) {
6      acc = acc + temp[i];
7  }
8  double moyenne = acc / temp.length;
9
10 // calcul de la déviation standard
11 acc = 0.0;
12 for (int i = 0; i < temp.length; i++) {
13     acc = acc + Math.pow(temp[i] - moyenne, 2);
14 }
15 double stddev = Math.sqrt(1.0 / temp.length * acc);
16
17 System.out.println("moyenne = " + moyenne + " stddev = " + stdde
```

Parcours de tableaux

Comme en C

```
1  public class TabFor {
2      public static void main(String[] a) {
3          int[] c = {1, 3, 5, 7};
4          for (int index = 0; index < c.length; index++) {
5              System.out.println(c[index]);
6          }
7      }
8  }
```

Structure `for each`

```
1  public class TabForEach {
2      public static void main(String[] a) {
3          int[] t = {1, 3, 5, 7};
4          for (int value : t) {
5              System.out.println(value);
```

Retour sur l'exemple précédent

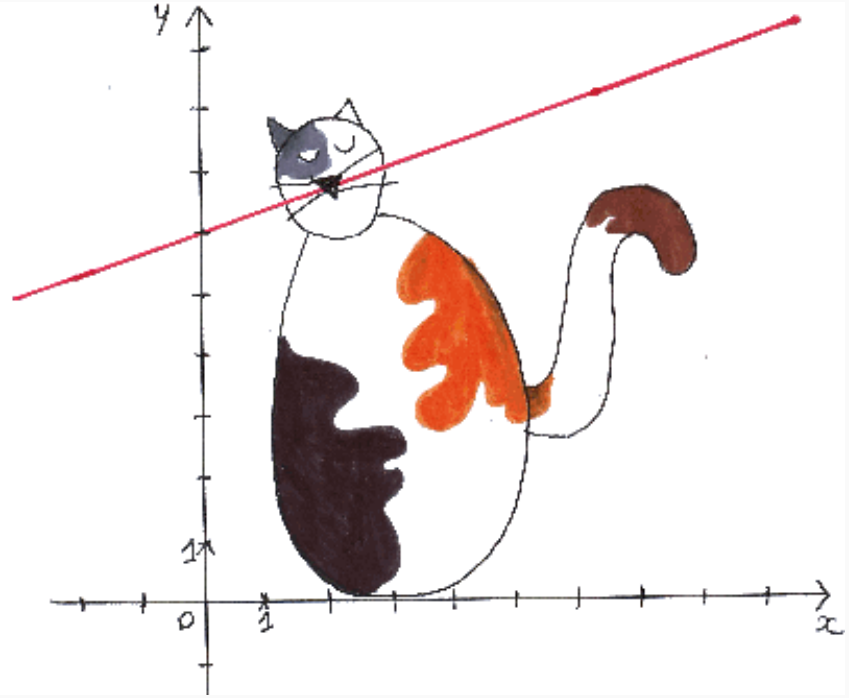
```java
// dans la méthode Main d'une classe Moyenne.java
int[] temp = {20, 21, 22, 25, 26, 25, 21};

// calcul de la moyenne
double acc = 0.0;
for (int t :  temp) {
    acc = acc + t;
}
double moyenne = acc / temp.length;

// calcul de la déviation standard
acc = 0.0;
for (int t : temp) {
    acc = acc + Math.pow(t - moyenne, 2);
}
```

Les fonctions

Les fonctions



- bloc d'instructions avec un nom
- interruption du flot d'instructions
  - l'exécution continue au début de la fonction
  - à la fin : retour au point d'appel

Techniquement

- une fonction est caractérisée par :
  1. son nom
  2. ses paramètres
  3. son type de retour
- paramètre :
  1. au sens mathématique
  2. variables locales (à la fonction)
  3. initialisée à une valeur fixée lors de l'appel

En java

- il n'y a pas de fonctions en Java
- pour les besoins des exercices, on utilisera des méthodes statiques comme des fonctions
- c'est *moralement* la même chose
- définition :

```
1  public static int add(int a, int b) {
2  public static void test() {
3  ...
```

- appel :

```
1          MaClasse.add(2, 3);
```

où `MaClasse` est le nom de la classe dans laquelle est définie la méthode.

Exemple : maximum de deux nombres I

```
1  public class Max {
2
3      public static int max(int a, int b) {
4
5          if (a < b) {
6              return b;
7          } else {
8              return a;
9          }
10     }
11
12     public static void main (String[] args) {
```

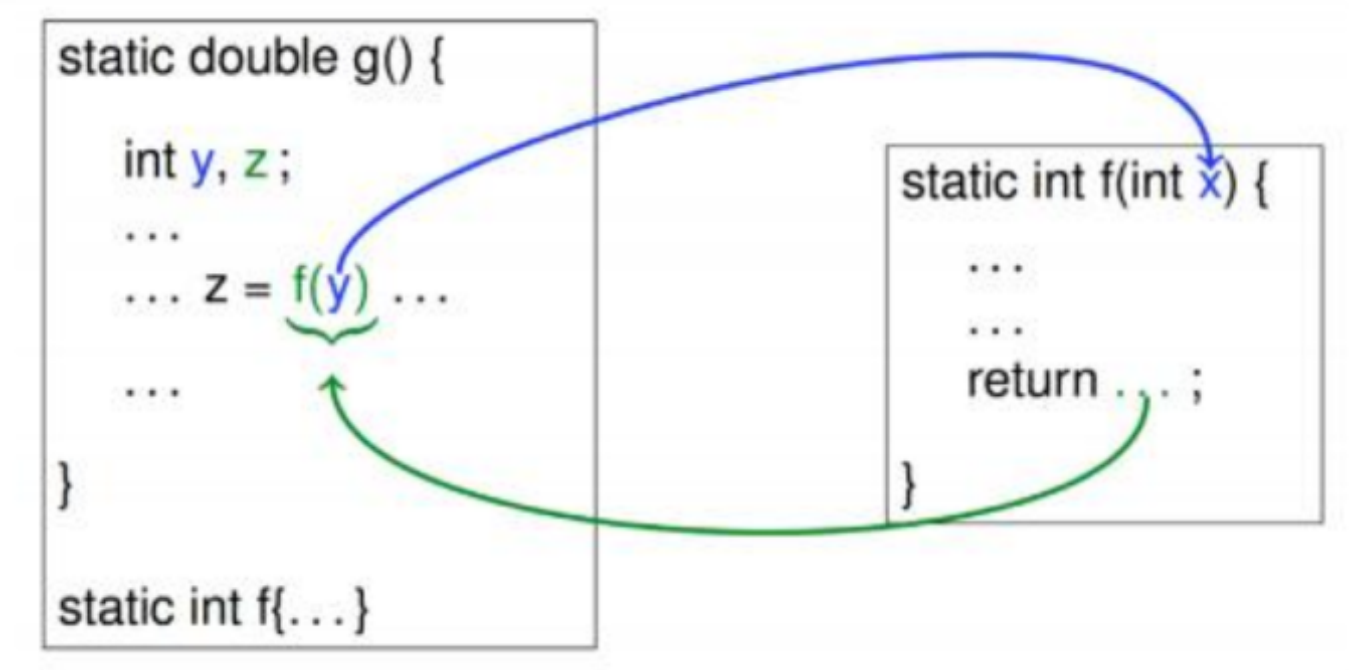
Exemple : maximum de deux nombres II

```
13
14              for (int i = 0; i < 5; i++) {
15                  System.out.println(max(i, i + 1);
16              }
17  }
```

Les 3 facettes d'une méthode

1. entête : déclaration de la méthode = nom + type de retour + type des paramètres
2. définition : entête + corps de la méthode
3. appel : l'endroit où l'on utilise la méthode ; nom de la méthode + arguments à lui passer

En image...


static double g() {
int y, z ;
. . .
. . . z = f(y) . . .
. . .
}
static int f{. . .}
static int f(int x) {
. . .
. . .
return . . . ;
}

Exemple n° 2 : maximum d'un tableau I

```
1  public class MaxTab {
2
3      public static int max(int[] tab) {
4
5          int m = tab[0];
6          for (int el : tab) {
7              if (el > m) {
8                  m = el;
9              }
10         }
11
12         return m;
```

Exemple n° 2 : maximum d'un tableau II

```
13        }
14
15        public static void main(String[] args) {
16            int[] t = {-3, 5, 2, 7, 1};
17            System.out.println(MaxTab.max(t));
18        }
19
20    }
```

Passage des paramètres

En général

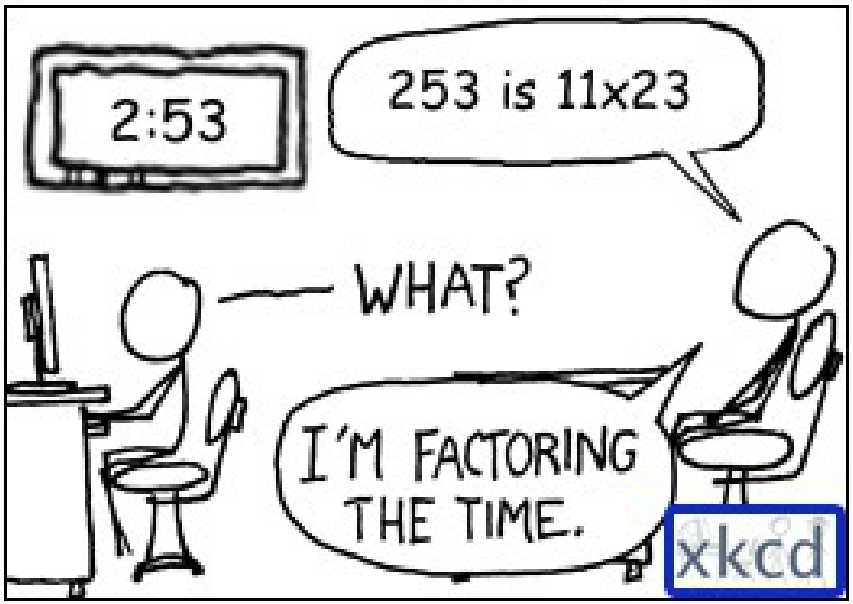
- passage par valeur : copie locale de la valeur, impossible de modifier le paramètre
- passage par référence : les modifications des paramètres sont visibles dans la méthode « mère » qui a appelé la méthode

En java

- type simple (int, float, ...) : passage par valeur
- type complexe (tableaux, objets) : passage par référence

À quoi ça sert ?

Intérêt n° 1 : factorisation



- pas besoin d'écrire plusieurs fois la même chose
- modularisation du code
  - programme complexe = « plein » de fonctions simples
- ré-utilisation du code

Intérêt n° 2 : abstraction

- améliorer la lisibilité du code
- plus facile/rapide de comprendre le nom d'une fonction que le code correspondant
- abstraire = cacher les détails de fonctionnement
- dire ce que l'on fait et pas comment on le fait

Exemple : intérêt de l'abstraction

Que fait le code suivant ?

```
1       r = n / 2;
2       while (abs( r - (n / r) ) > t) {
3         r = 0.5 * (r + (n / r));
4       }
5       System.out.println("r = " + r);
```

Et comme ça ?

Que fait le code suivant ?

```
1    public static double squareRootApproximation(n) {
2      double r = n / 2;
3      while (abs( r - (n/r) ) > t) {
4        r = 0.5 * (r + (n/r));
5      }
6      return r;
7    }
8    System.out.println("r = " + squareRootApproximation(r));
```

Conclusion (locale)

Lorsque l'on écrit :

```
1 double res = 3 * 3 * 3;
```

ou

```
1   double res = x * x * x;
```

notre programme est capable de calculer des cubes mais notre langage n'a pas accès au concept de « élever un nombre à la puissance 3 ».

La notion de fonction permet de construire des abstractions en nommant des motifs (c.-à-d. des séquences d'instructions)fréquents et de manipuler directement ces abstractions.

À plus long terme...

Ce que l'on a vu

Deux rôles :

1. programmeur concepteur : conçoit une fonction $\rightarrow$ doit connaitre parfaitement le fonctionnement de celle-ci et garantir l'exactitude de celle-ci
2. programmeur utilisateur : utilise une fonction $\rightarrow$ n'a besoin que de connaitre l'entête de la fonction

Programmer c'est communiquer !

- définition d'une API (Application Programming Interface) = liste des services (fonctions) fournis par un concepteur à un utilisateur
- programmation orientée objet = faciliter la communication entre concepteurs et utliisateurs

Les objets

Définition

une classe est un type de données abstrait

Définition

une classe est un type de données abstrait

- c'est un type

Définition

une classe est un type de données abstrait

- c'est un type
  - un ensemble de valeurs

Définition

une classe est un type de données abstrait

- c'est un type
  - un ensemble de valeurs
  - un ensemble d'opérations

Définition

une classe est un type de données abstrait

- c'est un type
  - un ensemble de valeurs
  - un ensemble d'opérations
- abstrait

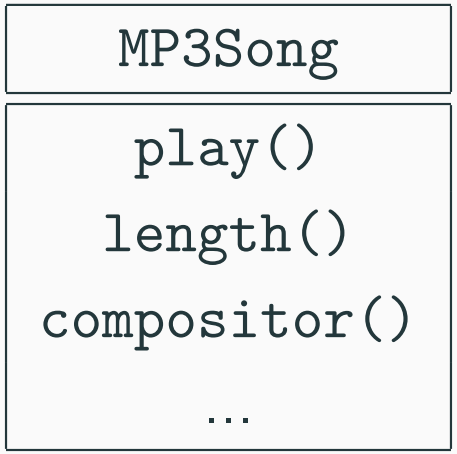
Définition

une classe est un type de données abstrait

- c'est un type
  - un ensemble de valeurs
  - un ensemble d'opérations
- abstrait
  - on ne sait pas comment il fonctionne

Plus précisément

Voici une (représentation d'une) classe :



En pratique :

- un nom
- interface = ensemble des opérations autorisées
- domaine = ?
  - encapsulation
  - domaine « suffisant » pour accomplir les tâches
  - sujet d'un cours futur

Comment utiliser une classe ?

- une classe n'est pas utilisable directement $\Rightarrow$ abstrait
- création d'une instance particulière de la classe = objet

```
1  MP3Song mySong = new MP3Song("le_tube_de_la_rentrée.mp3");
```

- l'instance/objet `mySong` = une chanson particulière (enfin un fichier !)
- la classe `MP3Song` = modèle générique de toutes les chansons possibles

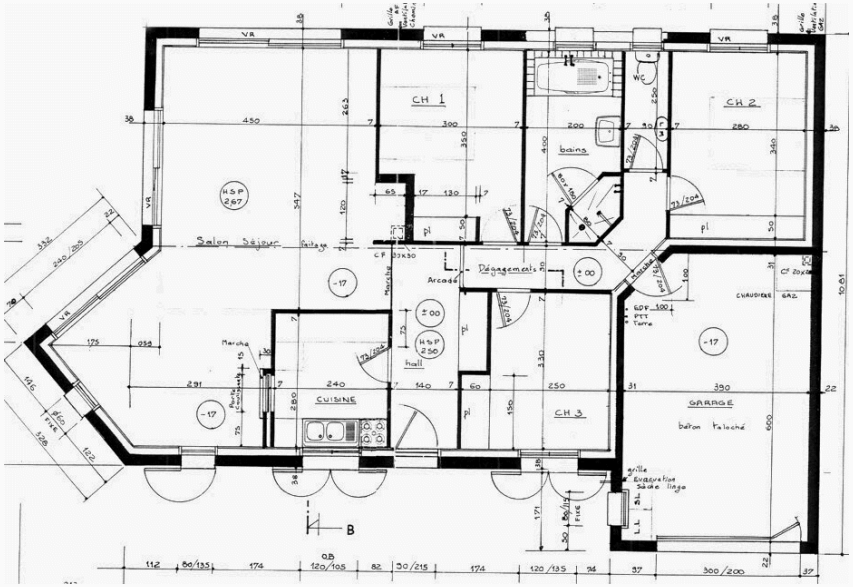
Bilan : classes et objets (1)

classe

objet/instance

Bilan : classes et objets (2)

classe



objet/instance

Création d'un objet

```
1  MP3Song mySong = new MP3Song("le_tube_de_la_rentrée.mp3");
```

- 2 étapes :
  1. déclaration `MP3Song mySong`
  2. définition `new MP3Song(...)`
- comme pour les variables (presque)
  - normal ! une classe est un type
- `mySong` est une référence
- initialisation par un constructeur et ses paramètres
- constructeur = même nom que la classe

Comment utiliser un objet ?

La syntaxe magique :

```
1    res = maReference.methode(parametre);
```

- `methode` = une des méthodes de l'interface
- envoie un message à `maReference`
- action sur `maReference`
- le type de retour et les paramètres sont spécifiés dans l'interface

Exemple : extraction de noms

Document :

```
Vainqueur étape n° 3 : Tyler Farrar
Vainquer étape n° 4 : Cadel Evans
Vainqueur étape n° 5 : Mark Cavendish
```

Objectif : extraire un tableau avec tous les noms propres en majuscules :

```
1  int[] res = {"FARRAR", "EVANS", "CAVENDISH"};
```

Remarque préliminaire : chaine de caractères

- les chaines de caractères (`String`) sont des objets ≪ normaux ≫...
- ... mais très utiles
- l'utilisation de `new` est facultative :

```
1 String s1 = new String("Bonjour");
2 String s2 = "Bonjour";
```

```java
public class StringExtract {

    public static void main(String[] s) {

        String doc =
"Vainqueur étape 3 : Tyler Farrar\nVainqueur étape 4 : Cade

        // cf. cours prochain pour ne pas
        //avoir à fixer la taille
        String[] res = new String[3];

        int i = 0;
        for (String line : doc.split("\n")) {
            String[] elements = line.split(" ");
```

```
15
16                  int pos = elements.length - 1;
17                  String winnerLastName = elements[pos];
18
19                  res[i] = winnerLastName.toUpperCase();
20                  i += 1;
21              }
22
23              for (String name : res) {
24                  System.out.println(name);
25              }
26          }
27      }
```

Attention


```
res = maRef.maMethode(a, b, 12)
```

maMethode n' est pas une fonction

Méthode $\neq$ fonction

- fonction = transfert du flot d'instruction
- méthode = modifie un objet
    - toujours liée à une référence

Parenthèse importante

Tester l'égalité de deux variables

- types simples : ==
- types complexes : `second.equals(secondReference)`

En particulier

Le code suivant :

```
1       String nom = "Elli";
2       if (nom == "Elli") {
3         System.out.println("victoire");
4       }
```

a un résultat indéterminé.

Comment trouver la classe qui nous intéresse ?

1. java 7 propose 3 977 classes
2. une infinité de bibliothèques

   $\Rightarrow$ il n'y a qu'à chercher

   (1^{re} partie des TP)
3. on peut créer ses propres classes

   2^{e} partie des TP

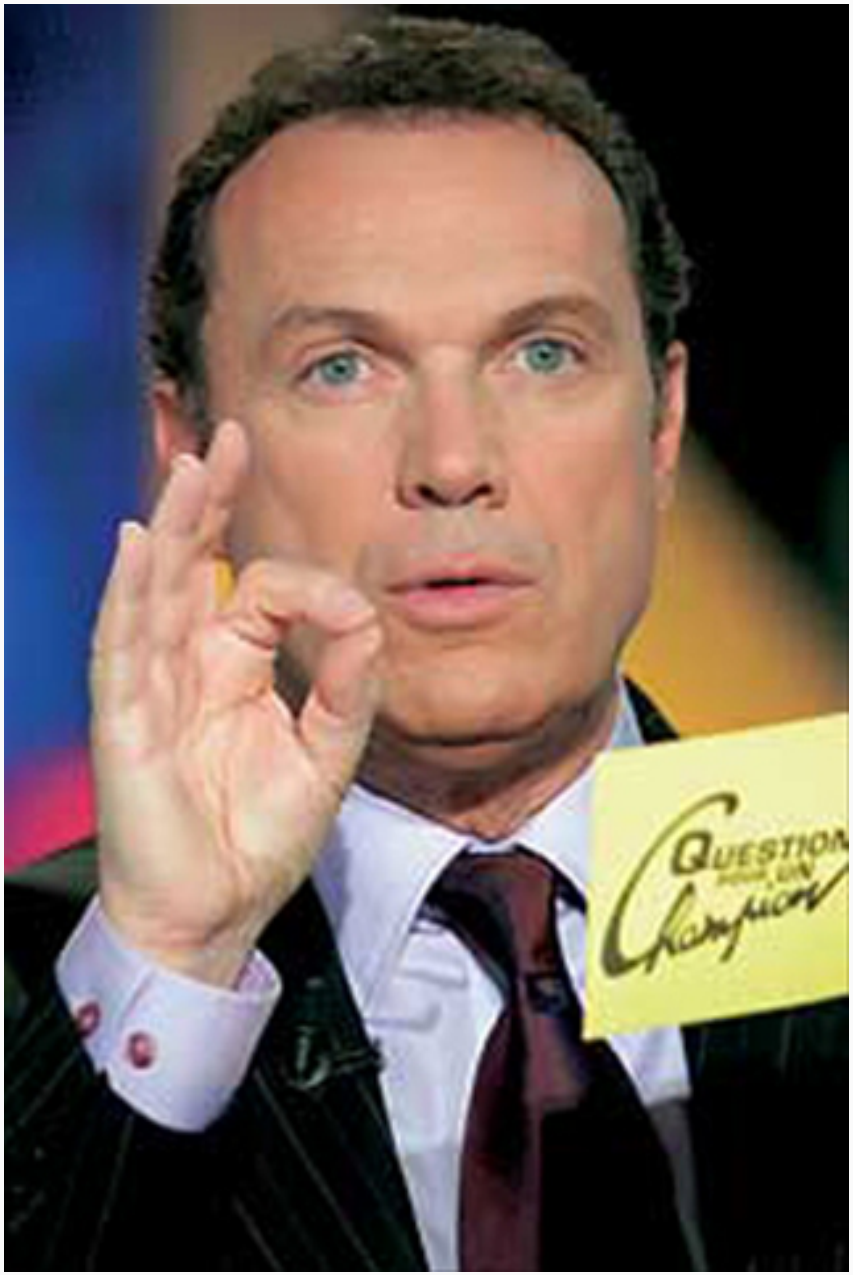
Utiliser une classe existante

- google + nom d'une classe ⇒ javadoc de la classe
- google + javadoc ⇒ ensemble des classes disponibles
- bibliothèque + javadoc ⇒ ensemble des classes d'une bibliothèque

Method Summary	
char	**charAt**(int index) Returns the char value at the specified index.
int	**compareTo**(String anotherString) Compares two strings lexicographically.
int	**compareToIgnoreCase**(String str) Compares two strings lexicographically, ignoring case differences.
String	**concat**(String str) Concatenates the specified string to the end of this string.
boolean	**contains**(CharSequence s) Returns true if and only if this string contains the specified sequence of char values.
boolean	**contentEquals**(CharSequence cs) Returns true if and only if this String represents the same sequence of char values as the specified sequence.

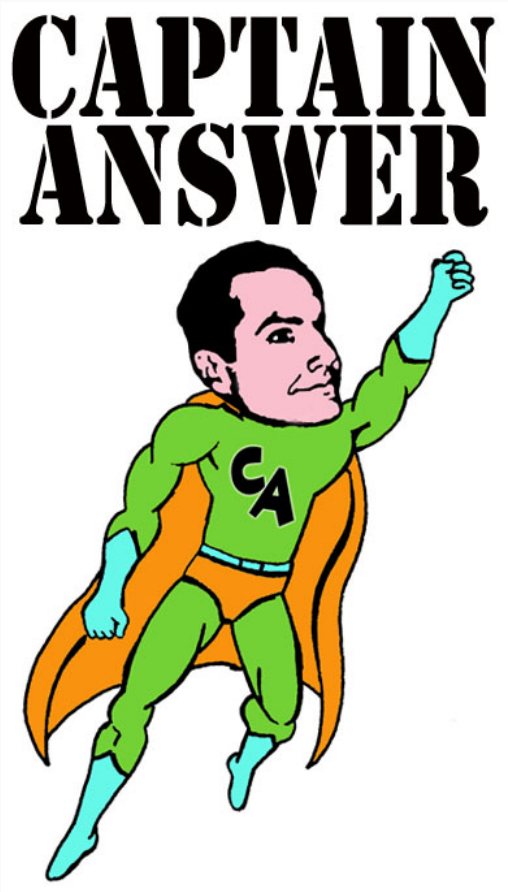
Mise en forme du code

Question préliminaire



Combien de lignes de code un programmeur produit en moyenne par jour ?

And the winner is...

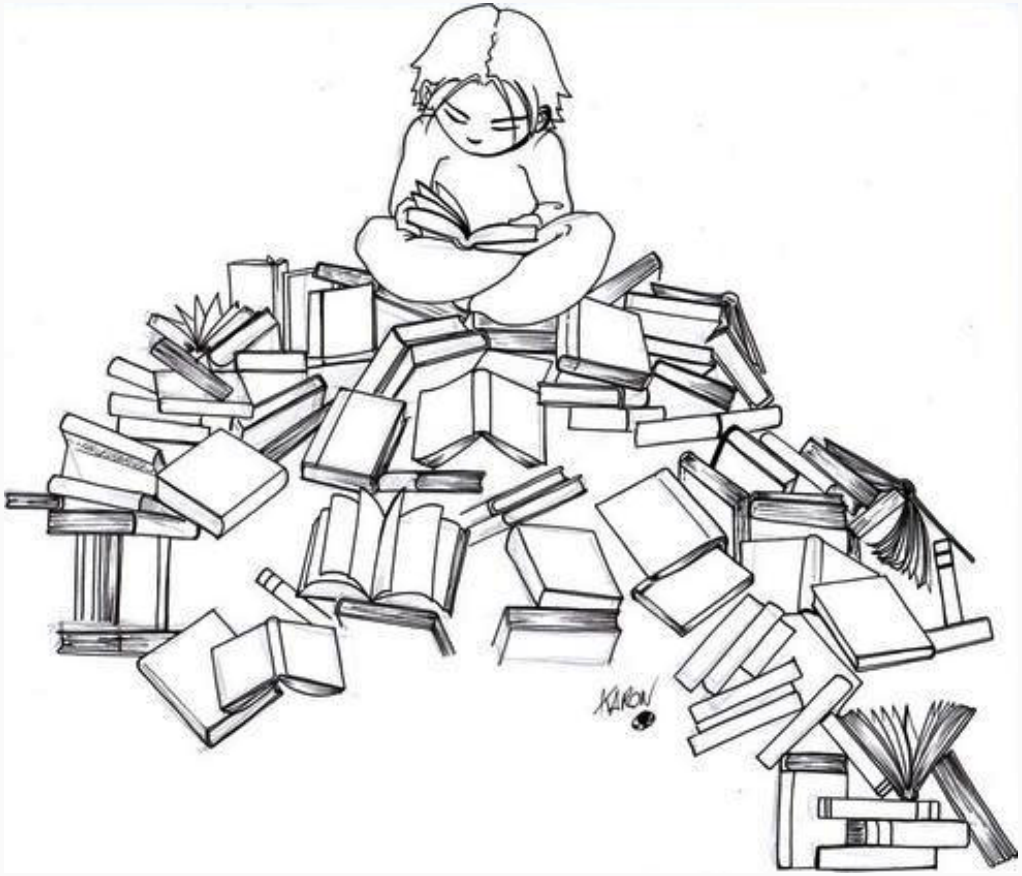


Les meilleurs programmeurs écrivent en moyenne :

10

lignes de code par jour

Conséquence

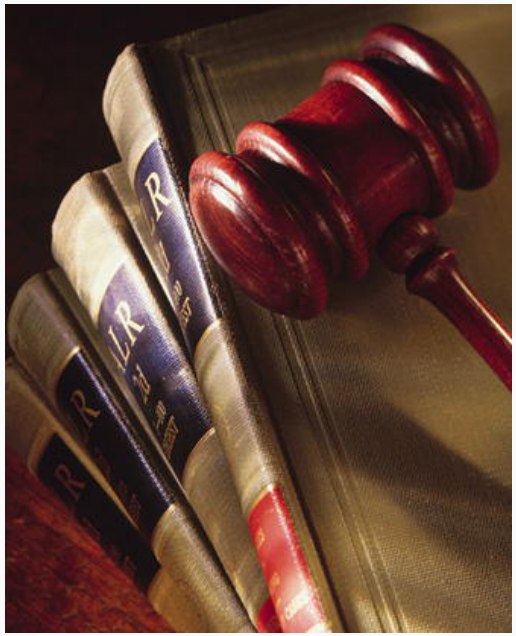


Le code source est fait pour être lu

Le code source est
fait pour être lu

```
#include                                    <math.h>
#include                                   <sys/time.h>
#include                                   <X11/Xlib.h>
#include                                  <X11/keysym.h>
                                          double L ,o ,P
                                         ,_=dt,T,Z,D=1,d,
                                         s[999],E,h= 8,I,
                                         J,K,w[999],M,m,O
                                        ,n[999],j=33e-3,i=
                                        1E3,r,t, u,v ,W,S=
                                        74.5,l=221,X=7.26,
                                        a,B,A=32.2,c, F,H;
                                        int N,q, C, y,p,U;
                                       Window z; char f[52]
                                     ; GC k; main(){ Display*e=
 XOpenDisplay( 0); z=RootWindow(e,0); for (XSetForeground(e,k=XCreateGC (e,z,0,0),BlackPixel(e,0))
; scanf(‘‘%lf%lf%lf’’,y +n,w+y, y+s)+1; y ++); XSelectInput(e,z= XCreateSimpleWindow(e,z,0,0,400,400,
0,0,WhitePixel(e,0) ),KeyPressMask); for(XMapWindow(e,z); ; T=sin(O)){ struct timeval G={ 0,dt*1e6}
; K= cos(j); N=1e4; M+= H*_; Z=D*K; F+=_*P; r=E*K; W=cos( O); m=K*W; H=K*T; O+=D*_*F/ K+d/K*E*_; B=
sin(j); a=B*T*D-E*W; XClearWindow(e,z); t=T*E+ D*B*W; j+=d*_*D-_*F*E; P=W*E*B-T*D; for (o+=(I=D*W+E
*T*B,E*d/K *B+v+B/K*F*D)*_; p<y; ){ T=p[s]+i; E=c-p[w]; D=n[p]-L; K=D*m-B*T-H*E; if(p [n]+w[ p]+p[s
]== 0|K <fabs(W=T*r-I*E +D*P) |fabs(D=t *D+Z *T-a *E)> K)N=1e4; else{ q=W/K *4E2+2e2; C= 2E2+4e2/ K
 *D; N-1E4&& XDrawLine(e ,z,k,N ,U,q,C); N=q; U=C; } ++p; } L+=_* (X*t +P*M+m*l); T=X*X+ l*l+M *M;
  XDrawString(e,z,k ,20,380,f,17); D=v/l*15; i+=(B *l-M*r -X*Z)*_; for(; XPending(e); u *=CS!=N){
                                    XEvent z; XNextEvent(e ,&z);
                                        ++*((N=XLookupKeysym
                                          (&z.xkey,0))-IT?
                                          N-LT? UP-N?& E:&
                                          J:& u: &h); --*(
                                          DN -N? N-DT ?N==
                                          RT?&u: & W:&h:&J
                                           ); } m=15*F/l;
                                           c+=(I=M/ l,l*H
```

Conséquence (encore...)



- convention de code pour garantir la lisibilité
- pas vérifiée par le compilateur...
- ... mais par tous les autres programmeurs...
- ... et vos professeurs

Code Conventions for the Java Programming Language

This *Code Conventions for the Java Programming Language* document contains the standard conventions that we at Sun follow and recommend that others follow. It covers filenames, file organization, indentation, comments, declarations, statements, white space, naming conventions, programming practices and includes a code example.

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

The Code Conventions for the Java Programming Language document was revised and updated on April 20, 1999.

View HTML
Download HTML*~62K*
Download PDF*~127K*
Download PostScript*~151K*

`http://www.oracle.com/technetwork/java/codeconv-138413.html`

ou : google + java + code + conventions

Convention de code Java (1)

- nom de variable : `monNomDeVariable`
- nom de fonction : `monNomDeFonction`
- nom de classe : `MonNomDeClasse`

Convention de code Java (2)

- ouverture des accolades sur la même ligne (précédé d'un espace)
- fermeture des accolades sur une nouvelle ligne
- le code doit toujours être indenté
  - ouverture d'une accolade : tabulation supplémentaire
  - fermeture d'une accolade : une tabulation de moins
- les signes binaires ($+$, $=$, ...) sont précédés et suivis d'un espace
- les signes unaires ($++$) sont suivis d'un espace, mais pas d'espace avant
- pas d'espaces autour des parenthèses

Gestion des erreurs en Java

Exceptions

- en Java toutes les erreurs lors du déroulement du programme donne lieu à des exceptions qui arrête le programme
- informations contenues dans une exception :
  - nom de l'erreur ($\simeq$ cause) ;
  - ligne où l'erreur s'est produite ;
  - pile d'appels = liste des méthodes successives qui ont été appelées.
- informations déterminées par la jvm $\Rightarrow$ capacité d'introspection

Exemple

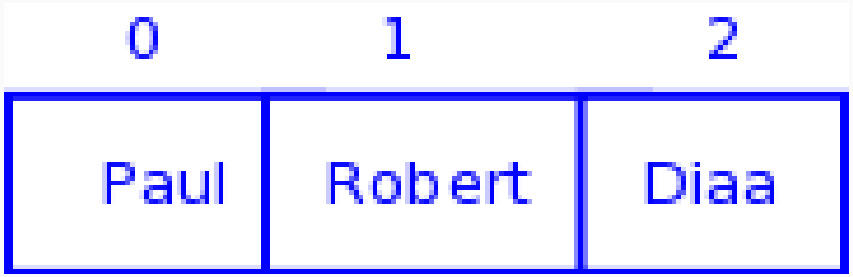
```
~/Desktop > java Test
Exception in thread "main" java.lang.ArithmeticException:
         / by zero
        at Test.maMeth(Test.java:4)
        at Test.main(Test.java:8)
```

Objets et collections

Notre problème

On veut représenter une collection d'objets

Outils classique : tableaux

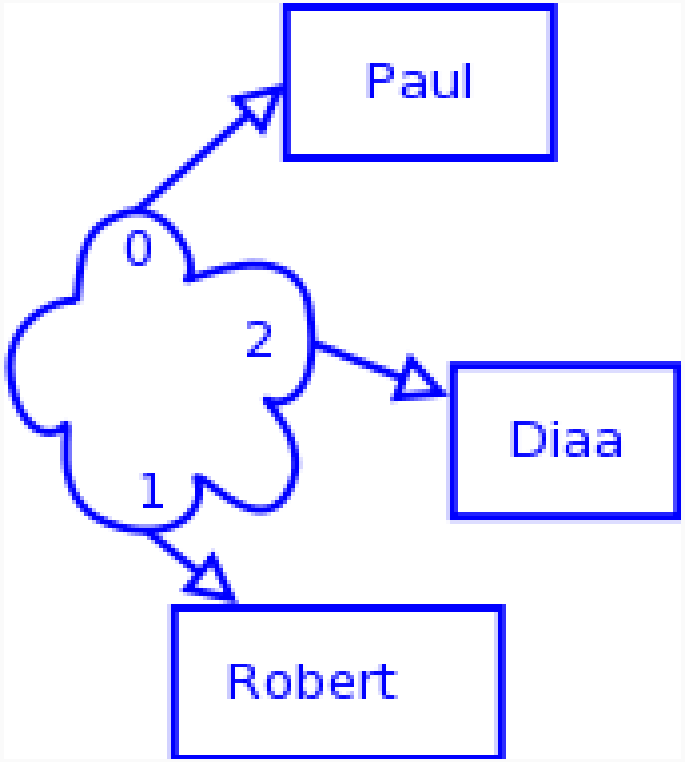


- inconvénient :
  - il faut connaitre la taille à la création
  - impossible d'ajouter ou de supprimer un élément
  - peu de méthodes de manipulation
- remarque : représentation directement liée à la manière dont les données sont stockées en mémoire

Changement de point de vue

- important = ce que fait la structure de données = interface
- important $\neq$ = comment elle le fait

Nouvelle vision



$\Rightarrow$ association entre un indice et un objet = liste

la classe `ArrayList` pour manipuler une association de ce type

La classe `ArrayList`

cf. javadoc

Particularité : les classes génériques

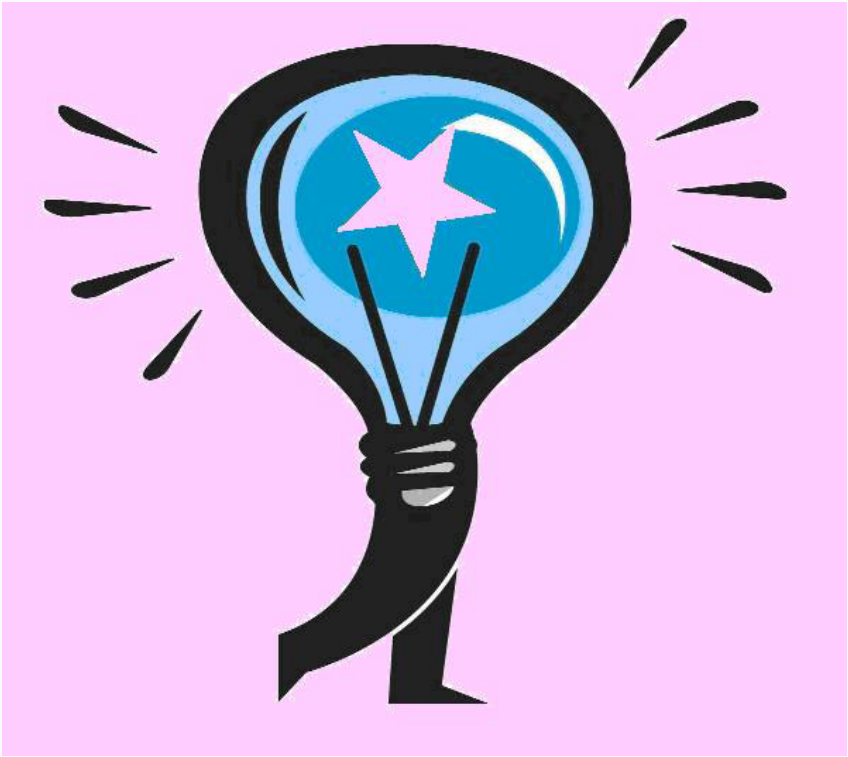
- les `ArrayList` permettent de stocker une collection d'objets homogènes (= de même type)
- comme un tableau, mais dynamique
- le type des objets est un paramètre de la classe
- syntaxe particulière ⇒ classe générique
- `ArrayList<E>` ⇒ `E` est le nom d'une classe (pas d'un type !)
- `E` est utilisé pour définir la signature des méthodes de la classe :

```
1 boolean add(E e);
2 E get(int index);
```

Exemple

```
1  ArrayList<MP3Song> playlist = new ArrayList<MP3Song>();
2
3  playlist.add(new MP3Song("chanson1.mp3"));
4  // maChanson est une référence de type MP3Song
5  playlist.add(maChanson);
6  // ...
7
8  for (MP3Song song : playlist) {
9    System.out.println(song);
10 }
11
12 if (playlist.contains(taChanson)) {
13   System.out.println("on aime la même chanson !");
14 }
15
```

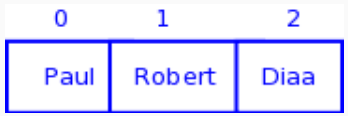
Ce qu'il faut retenir

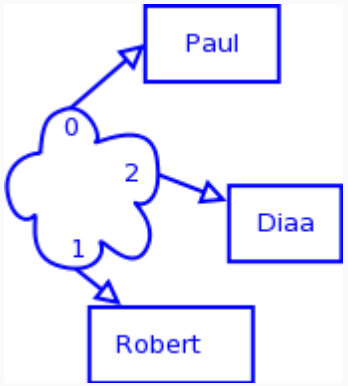


- possibilité d'ajouter/supprimer un élément
- accéder au i^e élément
- parcourir tous les éléments
- méthode de manipulation (tester la présence d'un élément, nombre d'éléments, ...)

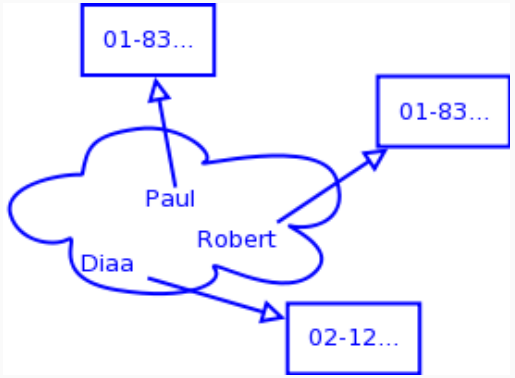
Un autre objet : les `HashMap`

- les listes : $\Rightarrow$ à un entier (indice) on associe un objet

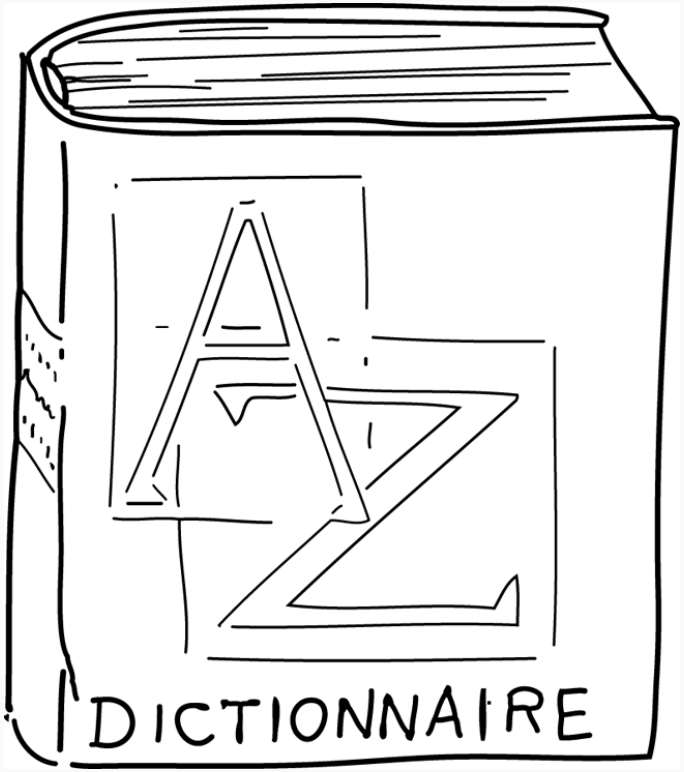




- généralisation : à un objet, on associe un objet



Un peu de vocabulaire



Une table de hachage

- = hashmap
- = dictionnaire
- = tableau associatif

associe à une clé *(key)* une valeur *(value)*.

javadoc de hashmap

Exemple

```
1       // création de l'objet
2       HashMap<String, String> annuaire = new HashMap<String,
3
4       // association d'un tél à une personne
5       annuaire.put("Guillaume", "06-...");
6       annuaire.put("Elli", "07-...");
7       // retrouver le tél. associé à un nom
8       String tel = annuaire.get("Elli")
9       // l'opération inverse n'est pas possible
10
11      // parcourir toutes les clés
12      for (String key : annuaire.keySet()) {
13        System.out.println(key)
14      }
```

Remarque

Pour utiliser `HashMap` et `ArrayList`, il faut ajouter :

- `import java.util.HashMap;`
- `import java.util.ArrayList;`

au début du fichier (cf. cours sur les packages)