Programmation `Java` : cours 2

Valeur d'accueil et de reconversion en informatique (VARI1)
Daniel Porumbel (dp.cnam@gmail.com)
`http://cedric.cnam.fr/~porumbed/vari1/`

Plan

1. Des exemples Java, lecture de fichiers et exceptions
2. Bonnes pratiques de programmation et génie logiciel

Rappel : on défini une classe et on écrit à l'intérieur :

- uniquement des méthodes statiques
- variables globales $\approx$ des variables statiques de la classe

```
class Test{
    static int varGlobale;
    static int calculerCube(int x){
        return x*x*x;
    }
    ... static void main ...
}
```

ERROR non-static ... cannot be referenced from a static context

$\Longrightarrow$ vous avez oublié le mot static devant une méthode ou variable

Pour s'entrainer : Écrire une fonction qui renvoie la somme des valeurs positives d'un tableau.

Rappel : on défini une classe et on écrit à l'intérieur :

- uniquement des méthodes statiques
- variables globales $\approx$ des variables statiques de la classe

```
class Test{
    static int varGlobale;
    static int calculerCube(int x){
        return x*x*x;
    }
    ... static void main ...
}
```

`ERROR non-static ... cannot be referenced from a static context`

$\Longrightarrow$ vous avez oublié le mot static devant une méthode ou variable

Pour s'entrainer : Écrire une fonction qui renvoie la somme des valeurs positives d'un tableau.

Rappel : on défini une classe et on écrit à l'intérieur :

- uniquement des méthodes statiques
- variables globales $\approx$ des variables statiques de la classe

```
class Test{
    static int varGlobale;
    static int calculerCube(int x){
        return x*x*x;
    }
    ... static void main ...
}
```

`ERROR non-static ... cannot be referenced from a static context`

$\Longrightarrow$ vous avez oublié le mot static devant une méthode ou variable

Pour s'entrainer : Écrire une fonction qui renvoie la somme des valeurs positives d'un tableau.

Que fait-on si l'utilisateur saisit une bêtise ?

Voici un exemple de saisie clavier.
Et si l'utilisateur tapait « toto » au lieu de saisir un `double` ?

```
1 class TesterScanner{
2     public static void main(String[] args){
3         java.util.Scanner scanner;
4         scanner = new java.util.Scanner(System.in);
5         System.out.println("Saisir un double, svp");
6         double x;
7         x = scanner.nextDouble();
8         System.out.println("Vous avez saisi x="+x);
9     }
10 }
```

Solution : on mets la saisie clavier à la ligne 7 dans un bloc `try-catch`.

Que fait-on si l'utilisateur saisit une bêtise ?

Voici un exemple de saisie clavier.
Et si l'utilisateur tapait « toto » au lieu de saisir un `double` ?

```
1 class TesterScanner{
2     public static void main(String[] args){
3         java.util.Scanner scanner;
4         scanner = new java.util.Scanner(System.in);
5         System.out.println("Saisir un double, svp");
6         double x;
7         x = scanner.nextDouble();
8         System.out.println("Vous avez saisi x="+x);
9     }
10 }
```

Solution : on mets la saisie clavier à la ligne 7 dans un bloc `try-catch`.

Les Exceptions définissent la réaction aux conditions exceptionnelles rencontrées pendant l'exécution du programme

Le mot clé `catch` : indique une "mission de sauvetage", le code à exécuter s'il y a une exception dans le bloc défini par le mot clé `try`

```
try{
    int x = 9;
    int y = 0;
    int z = x/y;
    System.out.println("z="+z);
}catch(Exception e){
    System.out.println("Division par 0???");
}
```

Lire un fichier avec `Scanner`

On doit utiliser des objets des paquetages standard java :

- `java.io.File` : un fichier
- `java.util.Scanner` : le lecteur

```
import java.util.*;
import java.io.*;
class LectureFic{
  public static void main(String[] args)
                                    throws Exception{
    Scanner scan = new Scanner(new File("in.txt"));
    int a = scan.nextInt();
    int b = scan.nextInt();
    int c = scan.nextInt();
    System.out.println("La somme est "+(a+b+c));
  }
}
```

Lire un fichier avec `Scanner`

On doit utiliser des objets des paquetages standard java :

- `java.io.File` : un fichier
- `java.util.Scanner` : le lecteur

```
import java.util.*;
import java.io.*;
class LectureFic{
  public static void main(String[] args)
                                        throws Exception{
    Scanner scan = new Scanner(new File("in.txt"));
    int a = scan.nextInt();
    int b = scan.nextInt();
    int c = scan.nextInt();
    System.out.println("La somme est "+(a+b+c));
  }
}
```

D'autres méthodes pour lire un fichier texte

- `java.io.FileReader` : un fichier
- `java.io.BufferedReader` un objet capable de lire le fichier grâce à la fonction `readLine()`

```
java.io.BufferedReader objLecteur;
objLecteur = new java.io.BufferedReader
             (new java.io.FileReader("in.txt"));
//Lecture effective d'une ligne et affichage:
String ligne = objLecteur.readLine();
System.out.println(ligne);
```

D'autres méthodes pour lire un fichier texte

- `java.io.FileReader` : un fichier
- `java.io.BufferedReader` un objet capable de lire le fichier grâce à la fonction `readLine()`

```
java.io.BufferedReader objLecteur;
objLecteur = new java.io.BufferedReader
             (new java.io.FileReader("in.txt"));
//Lecture effective d'une ligne et affichage:
String ligne = objLecteur.readLine();
System.out.println(ligne);
```

La somme de 3 valeurs d'un fichier `in.txt`

Principe :

```
java.io.BufferedReader objLecteur;
objLecteur = new java.io.BufferedReader
             (new java.io.FileReader("in.txt"));
int somme = 0;
for(int i=0;i<3;i++){
     String ligne = objLecteur.readLine();
     somme = somme + Integer.parseInt(ligne);
}
```

Problème : certaines choses peuvent mal se passer :

1. Le fichier n'existe pas
2. Il n'y pas de valeurs entiers dans le fichier

$\Longrightarrow$ On est obliger de gérer ces Exceptions

La somme de 3 valeurs d'un fichier `in.txt`

Principe :

```
java.io.BufferedReader objLecteur;
objLecteur = new java.io.BufferedReader
             (new java.io.FileReader("in.txt"));
int somme = 0;
for(int i=0;i<3;i++){
     String ligne = objLecteur.readLine();
     somme = somme + Integer.parseInt(ligne);
}
```

Problème : certaines choses peuvent mal se passer :

1. Le fichier n'existe pas
2. Il n'y pas de valeurs entiers dans le fichier

$\Longrightarrow$ On est obliger de gérer ces Exceptions

La somme des 3 valeurs trouvées dans le fichier `in.txt`

```
public static void main(String[] args){
  java.io.BufferedReader objLecteur;
  try{
     objLecteur = new java.io.BufferedReader
                      (new java.io.FileReader("in.txt"));
     int somme = 0;
     for(int i=0;i<3;i++){
       String ligne = objLecteur.readLine();
        somme = somme + Integer.parseInt(ligne);
      }
  }catch(Exception e){
      System.out.println("Exception:"+e);
  }
}
```

Les paquetages Java

Nous avons déjà vu des classes :

java.io.FileReader pour ouvrir un fichier

java.io.BufferedReader pour lire à partir d'un fichier

java.util.Scanner pour lecture clavier

Ces classes sont définies dans des bibliothèques/paquetages Java. Il s'agit de `java.io` et `java.util`.
Pour utiliser ces paquetage sans problème, il suffit d'écrire au début du programme :

```
import java.io.*;
import java.util.*
```

La somme des 3 valeurs dans le fichier

```
import java.io.*;
import java.util.*;
class Lecture{
  public static void main(String[] args){
    BufferedReader objLecteur;
    try{
       objLecteur = new BufferedReader
                              (new FileReader("in.txt"));
       int somme = 0;
       for(int i=0;i<3;i++){
          String ligne = objLecteur.readLine();
          somme = somme + Integer.parseInt(ligne);
       }
        System.out.println(somme);
    }catch(Exception e){
        System.out.println("Exception:"+e);
    }
  }
}
```

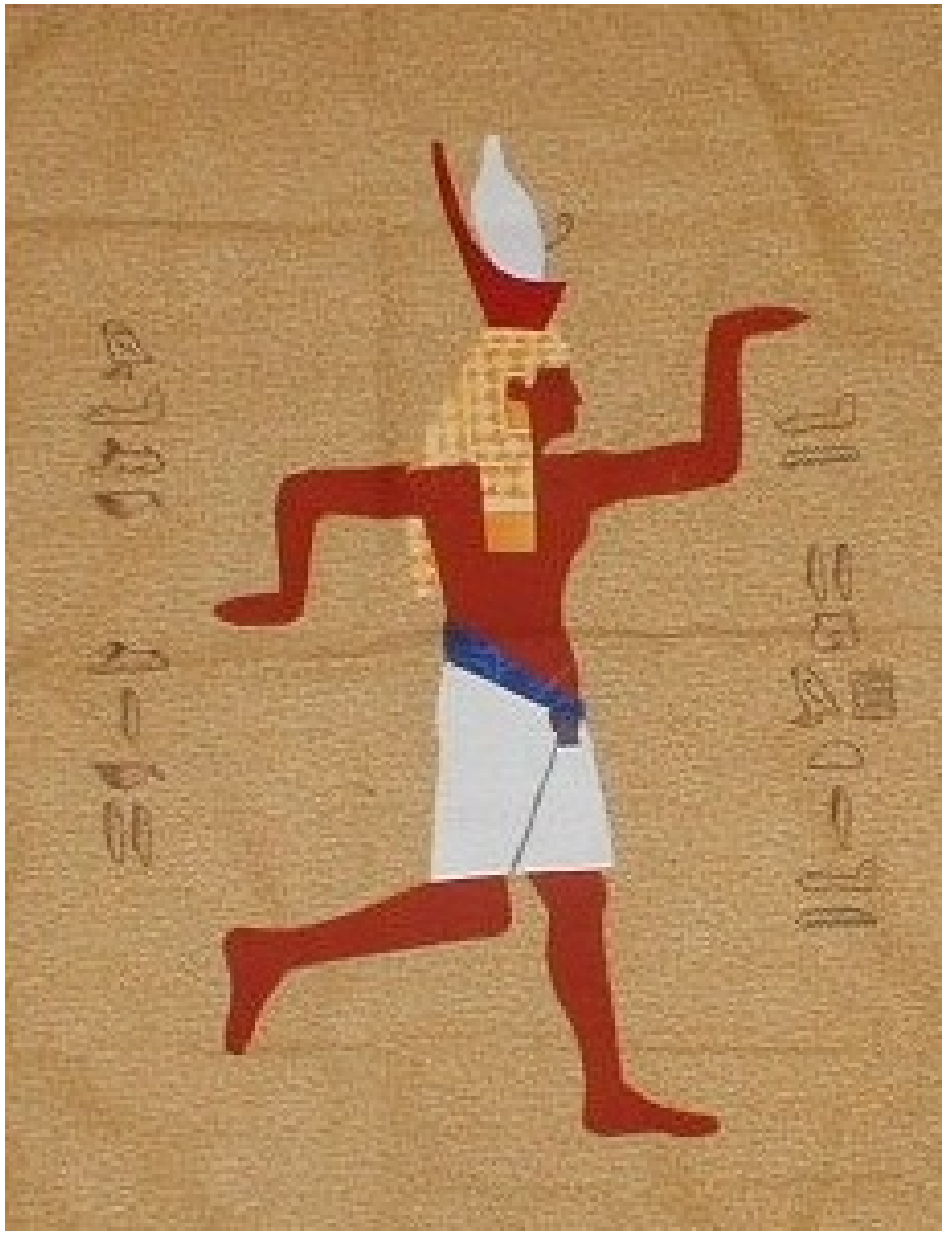
Plan

1 Des exemples Java, lecture de fichiers et exceptions

2 Bonnes pratiques de programmation et génie logiciel

On utilise des accolades dites égyptiennes

```
for(int i=0;i<n;i++){
    tab[i]=i;
}
```



Comparer la position des accolades avec les mains dans la figure !

Important : Le niveau de décalage/indentation d'une ligne doit indiquer son niveau d'imbrication

```
class Test{
|   public static void main(String[] args){
|   |   //Imbrication 2
|   |   for(int i=0;i<4;i++){
|   |   |   //Imbrication 3
|   |   |   for(int j=0;j<4;j++)
|   |   |       //imbrication 4
|   |   |       System.out.print("␣"+i*j);
|   |   |   //Imbrication 3
|   |   |   System.out.println("");
|   |   }
|   | //^cette colonne doit rester vide!!!
|   }
}
```

Important : Le niveau de décalage/indentation d'une ligne doit indiquer son niveau d'imbrication

```
class Test{
    public static void main(String[] args){
        //Imbrication 2
        for(int i=0;i<4;i++){
            //Imbrication 3
            for(int j=0;j<4;j++)
                //imbrication 4
                System.out.print("␣"+i*j);
            //Imbrication 3
            System.out.println("");
        }
    //^cette colonne doit rester vide!!!
    }
}
```

Si on monte sur la colonne d'une accolade fermante on trouve la ligne de l'accolade ouvrante!!!

- L'accolade fermante doit rester seule sur sa ligne!!!

Important : Le niveau de décalage/indentation d'une ligne doit indiquer son niveau d'imbrication

```
class Test{
    public static void main(String[] args){
        //Imbrication 2
        for(int i=0;i<4;i++){
            //Imbrication 3
            for(int j=0;j<4;j++)
                //imbrication 4
                System.out.print(" "+i*j);
            //Imbrication 3
            System.out.println("");
        }
    //^cette colonne doit rester vide!!!
    }
}
```

Imprimer cette diapo avant de faire des TPs!!!

Si on monte sur la colonne d'une accolade fermante on trouve la ligne de l'accolade ouvrante!!!

- L'accolade fermante doit rester seule sur sa ligne!!!

Une commande Linux utile : `indent`

- Une indentation correcte, dans le style `GNU` du langage `C`

```
indent  Prog.java
```

- Indentation proche de celle qu'on utilise dans le cours

```
indent -kr -i4 -nut Prog.java
```

kr Utiliser le style de Kernighan & Ritchie, les inventeurs du C

i4 quatre espaces = un niveau d'imbrication

nut pas de TAB

Le résultat n'est pas toujours parfait, car la commande `indent` pense qu'on lui donne un programme `C++`

C’est la fin du cours. Je laisse les diapos qui suivent juste pour information...

Le compilateur Java veut tout anticiper !

```
static double calcImpots(double revenu){
    if(revenu<9710)
        return 0;
    if(revenu>=9710)
        return (revenu-9700)*0.14;
}
```

Ce code `Java` ne compile pas : *missing return value* ! Il aurait du compiler et il compile en C++

Bjarne Stroustrup : Je veux donner un avantage aux bons programmeurs au lieu d'offrir une protection contre les erreurs des programmeurs médiocres

---

I want to give good programers an advantage instead of protecting mediocre programmers from making all sorts of stupid mistakes.

Le compilateur Java veut tout anticiper !

```
static double calcImpots(double revenu){
    if(revenu<9710)
        return 0;
    if(revenu>=9710)
        return (revenu-9700)*0.14;
}
```

Ce code `Java` ne compile pas : *missing return value* ! Il aurait du compiler et il compile en C++ [1]

Bjarne Stroustrup : Je veux donner un avantage aux bons programmeurs au lieu d'offrir une protection contre les erreurs des programmeurs médiocres

---

I want to give good programers an advantage instead of protecting mediocre programmers from making all sorts of stupid mistakes.

1. Enlever `static` ; d'autres langages (Python) imposent l'indentation pour compiler

Le compilateur Java veut tout anticiper !

```
static double calcImpots(double revenu){
    if(revenu<9710)
        return 0;
    if(revenu>=9710)
        return (revenu-9700)*0.14;
}
```

La remarque est un peu trop dure. Si on avait écrit `if(revenu>9710)...` le programme aurait compilé en C++. Mais pour `revenu=9710` il aurait renvoyé une valeur non-determinée

$\Longrightarrow$ une erreur très difficile à détecter, des problèmes que pour ceux qui gagnent exactement 9710.

des programmeurs médiocres

I want to give good programers an advantage instead of protecting mediocre programmers from making all sorts of stupid mistakes.

1. Enlever `static` ; d'autres langages (Python) imposent l'indentation pour compiler

Maximum 100 caractères par ligne

- Le `Google Java Style Guide` demande au maximum 100 caractères par ligne. [2]
- Le `Linux Kernel coding style` demande 80 caractères par ligne [3]

Deux raisons pour ces limitations

- Les yeux doivent bouger d'une fin de ligne pour trouver le début de la ligne suivante : difficile avec $>$100 lignes
  - Plus facile à lire un livre de poche qu'un texte d'une largeur d'un écran de 30 pouces
- On doit limiter le niveau maximal d'imbrication

---

2. `google.github.io/styleguide/javaguide.html#s4.4-column-limit`
3. `www.kernel.org/doc/html/v4.10/process/coding-style.html`

Écrire de belles fonctions

Recommandations du style noyau `Linux`,[4] pour qu'on puisse comprendre ce qu'on a écrit il y a 2 semaines.

- Les fonctions doivent être courtes et douces, et faire qu'une seule chose (traduction littéraire de *Functions should be short and sweet, and do just one thing*.)
- Une fonction ne doit pas dépasser 20-30 lignes.
- Maximum 5-7 variables locales. Le cerveau est capable de gérer en même temps environ 7 objets/concepts.[5]

4. `www.kernel.org/doc/html/v4.10/process/coding-style.html`

5. Voir "The Magical Number Seven, Plus or Minus Two : Some Limits on Our Capacity for Processing Information", George A. Miller, 1956, un des plus cité papiers en psychologie.

Bien nommer les variables/fonctions c'est un art

- `Java` utilise souvent des noms de variables comme `compteurDuNombreDeFichiers`, `averageStringSize`
- Le style `C/C++` utilise souvent des abréviations comme `nbFics`, `avgStrSz`
  - si vous pensez qu'une abréviation est ambigüe, ajouter un commentaire au début du fichier pour décrire la variable

Exemple : convertir chaine de caractères → entier

Java : `Integer.parseInt(str)`

C : `atoi(str)` //Ascii To Integer

$\Longrightarrow$ on peut avoir plus d'informations sur une ligne de 80 caractères en `C/C++`

- On peut appeler 4-5 fois `atoi()` dans une condition `if`

Bien nommer les variables/fonctions c'est un art

- `Java` utilise souvent des noms de variables comme `compteurDuNombreDeFichiers`, `averageStringSize`
- Le style `C/C++` utilise souvent des abréviations comme `nbFics`, `avgStrSz`
  - si vous pensez qu'une abréviation est ambigüe, ajouter un commentaire au début du fichier pour décrire la variable

Exemple : convertir chaine de caractères → entier

Java : `Integer.parseInt(str)`

C : `atoi(str)` //Ascii To Integer

$\Longrightarrow$ on peut avoir plus d'informations sur une ligne de 80 caractères en `C/C++`

- On peut appeler 4-5 fois `atoi()` dans une condition `if`

Bien nommer les variables/fonctions c'est un art

- `Java` utilise souvent des noms de variables comme `compteurDuNombreDeFichiers`, `averageStringSize`
- Le style `C/C++` utilise souvent des abréviations comme `nbFics`, `avgStrSz`
  - si vous pensez qu'une abréviation est ambigüe, ajouter un commentaire au début du fichier pour décrire la variable

! Une bonne pratique : on peut utiliser des variables de $>$ 10 caractères uniquement pour :

- les variables rarement utilisées (pour ne plus avoir besoin de comprendre/chercher l'abréviation)
- pour de petits programmes.

Majuscules et minuscules

On commence avec une minuscule :

- un nom de variable
- un nom de fonction
- un nom de paquetage (un paquetage comporte que des minuscules)

On commence avec une majuscule :

- un nom de classe
- des constantes (`static final int MAX_INT=23243423`)

Modularité 1 : réaliser une longue tache

- Une tache récurrente dans les jeux video consiste à vérifier si deux objets s'intersectent.
- tester si les rectangles ci-dessous (de même taille) s'intersectent
  - $(x1A, y1A, x2A, y2A)$
  - $(x1B, y1B, x2B, y2B)$.

Une solution parfaitement fonctionnelle mais mauvaise !

```
if !(((x1A<x1B && x2A>x1B) && (y1A<y1B && y2A>
    y1B)) || ((x1A<x2B && x2A>x2B) && (y1A<y1B &
    & y2A>y1B)) || ((x1A<x1B && x2A>x1B) && (y1A
    <y2B && y2A>y2B)) || ((x1A<x2B && x2A>x2B) &
    & (y1A<y2B && y2A>y2B)))
        return false;
    else
        return true;
```

Modularité 1 : réaliser une longue tache

- Une tache récurrente dans les jeux video consiste à vérifier si deux objets s'intersectent.
- tester si les rectangles ci-dessous (de même taille) s'intersectent
  - $(x1A, y1A, x2A, y2A)$
  - $(x1B, y1B, x2B, y2B)$.

Une solution parfaitement fonctionnelle mais mauvaise !

```
if !(((x1A<x1B && x2A>x1B) && (y1A<y1B && y2A>
    y1B)) || ((x1A<x2B && x2A>x2B) && (y1A<y1B &
    & y2A>y1B)) || ((x1A<x1B && x2A>x1B) && (y1A
    <y2B && y2A>y2B)) || ((x1A<x2B && x2A>x2B) &
    & (y1A<y2B && y2A>y2B)))
        return false;
    else
        return true;
```

Modularité 2 : décomposition de la tache

Comparer ce code avec :

```
if (pointInt(x1B,y1B,x1A,y1A,x2A,y2A))
    return true;
if (pointInt(x2B,y1B,x1A,y1A,x2A,y2A))
    return true;
if (pointInt(x1B,y2B,x1A,y1A,x2A,y2A))
    return true;
if (pointInt(x2B,y2B,x1A,y1A,x2A,y2A))
    return true;
return false;
```

A retenir : La programmation modulaire permet de décomposer (factoriser) une grosse tache en plusieurs taches plus petites.

Avantages et inconvenients de la modularité

- La modularité rend le code plus lisible !
- La modularité est parfois sacrifiée pour l'efficacité ou pour rendre la compréhension plus difficile (obfuscation)
- Le code peut aussi être généré automatiquement

Exemple : le code `html` du site `www.google.fr`

```
<!doctype html><html itemscope="" itemtype="http:
google.j.b=(!!location.hash&&!!location.hash.matc
||(google.j.qbp==1);(function(){google.c={c:{a:tr
function(a,b,c){google.timers[a]||google.startTic
!0};google.c.u=function(a){var b=google.timers.lo
function(a){google.c.c.a&&google.afte&&google.tic
h||b.ctrlKey||b.shiftKey||b.altKey||b.metaKey||H(
B(b),g=(g.type||g.tagName).toUpperCase(),(g=32==(
typeof b[p]&&"srcElement"!==p&&"target"!==p&&(c[p
```

Avantages et inconvenients de la modularité

- La modularité rend le code plus lisible !
- La modularité est parfois sacrifiée pour l'efficacité ou pour rendre la compréhension plus difficile (obfuscation)
- Le code peut aussi être généré automatiquement

Exemple : le code `html` du site `www.google.fr`

```
<!doctype html><html itemscope="" itemtype="http:
google.j.b=(!!location.hash&&!!location.hash.matc
||(google.j.qbp==1);(function(){google.c={c:{a:tr
function(a,b,c){google.timers[a]||google.startTic
!0};google.c.u=function(a){var b=google.timers.lo
function(a){google.c.c.a&&google.afte&&google.tic
h||b.ctrlKey||b.shiftKey||b.altKey||b.metaKey||H(
B(b),g=(g.type||g.tagName).toUpperCase(),(g=32==(
typeof b[p]&&"srcElement"!==p&&"target"!==p&&(c[p
```

Il existe encore des dizaines de langages

Je vais juste noter qu'il est puéril de considérer qu'il y aurait un seul langage ou un seul outil de programmation comme le seul et unique meilleur outil pour tout le monde et pour tout problème. Si quelqu'un prétend maitriser le langage parfait, il est soit une personne stupide, soit un bon vendeur, soit les deux.
Bjarne Stroustrup (concepteur du C++)

I'll just note that I consider the idea of one language, one programming tool, as the one and only best tool for everyone and for every problem infantile. If someone claims to have the perfect language he is either a fool or a salesman or both.

Langages les plus modernes : utiliser souvent des fonctions standard déjà-écrites

Avantage On n'a pas besoin de re-inventer la roue

Avantage Les classes standard sont souvent très bien écrites

Inconvenient on peut sacrifier l'efficacité

Inconvenient on nous impose un raisonnement assez figé, (presque) pensée unique

- même la classe `String` nous impose une manière particulière de travail : on n'est pas censé changer une lettre d'une chaîne déjà construite

`String` est immutable mais `char[]` ne l'est pas.

Langages les plus modernes : utiliser souvent des fonctions standard déjà-écrites

Avantage On n'a pas besoin de re-inventer la roue

Avantage Les classes standard sont souvent très bien écrites

Inconvenient on peut sacrifier l'efficacité

Inconvenient on nous impose un raisonnement assez figé, (presque) pensée unique

- même la classe `String` nous impose une manière particulière de travail : on n'est pas censé changer une lettre d'une chaîne déjà construite

`String` est immutable mais `char[]` ne l'est pas.

Exemple tableau de `char`

```
public static void main(String[] args){
    // tableau de char
    char[] x = {65, 66, 67, 13, 66};//13=aller à
                                    //la première case
                                    //sans saut de ligne
    x[1]     = 65;
    System.out.println(x);                  //->BAC

    // String
    String y = "ABC";
    System.out.println(y.charAt(1));//->B
    //Changer y[1] = 'B' avec String imutable?
    char[] z = y.toCharArray();
    z[1]     = 65;
    y        = new String(z);
    System.out.println(y);                  //->AAC
}
```

En théorie, il y de nombreuses régles sur Internet :

- ne pas utiliser de GO TO (pour sauter à une autre ligne)
  - Le but est d'éviter la "programmation spaghettis", mais ça existe en C++
  - James Gosling a crée la JVM originale avec le support de GO TO, mais il finalement effacé cette fonctionnalité car il trouve qu'on n'en a pas besoin. `http://stackoverflow.com/questions/2545103/is-there-a-goto-statement-in-java`
- ne pas utiliser de variables globales
- mettre toujours les accolades pour l'action d'un `if` ou `for`

Exemple de `GO TO` utile en `C/C++`

Les suggestions de codage du logiciel `midnight commander`[a] indiquent :

- Les GOTOs sont mauvais mais ils peuvent bien améliorer la lisibilité lorsqu'ils indiquent un point unique de sortie d'une fonction.

  GOTOs are evil, but they can greatly enhance readability [..] when used as the single exit point from a function.

```
if(profit<10000){
    impots= 0;
    goto finfunction;
}
if(profit<20000){
    impots= 0.1*profit;
    goto finfunction;
}
if(profit<30000){
    impots= 0.2*profit;
    goto finfunction;
}
finfunction:
    println(impots);
    return impots;
```

[a] Voir `midnight-commander.org/wiki/Hacking`. Le style de codage du noyau `Linux` donne des conseils similaires.

Sur “ne pas utiliser de variable globale”

L’idée essentielle à retenir est l’objectif général : la clarté. La règle “pas de variable globale” est là uniquement à cause du fait que les variables globales rendent le code moins clair.

Cependant, comme beaucoup de règles, les gens retiennent la règle et non pas le objectif final de la règle.

J’ai vu des programmes qui doublent la taille du code simplement pour éviter le mal des variables globales ;

Tom West, forum `http://stackoverflow.com/questions/484635/are-global-variables-bad`

Pas de surcharge d'opérateur sous Java

En *C++*, le code ci-après permet d'appeller une méthode de la classe Frac à l'aide de l'opérateur $+$;

```
Frac x, y, z;
z = x + y;
```

Sous Java, il faut écrire qq chose comme `z=x.ajouter(y)`.

J'ai pris la décision de ne pas utiliser la surcharge d'opérateurs en Java comme un choix personnel parce que j'ai vu trop de gens en abuser sous C++.

I left out operator overloading as a fairly personal choice because I had seen too many people abuse it in C++.

James Gosling (le concepteur du langage Java). Source :

www.gotw.ca/publications/c_family_interview.htm

Pas de surcharge d'opérateur sous Java

En *C++*, le code ci-après permet d'appeller une méthode de la classe Frac à l'aide de l'opérateur + ;

```
Frac x, y, z;
z = x + y;
```

Sous Java, il faut écrire qq chose comme `z=x.ajouter(y)`.

J'ai pris la décision de ne pas utiliser la surcharge d'opérateurs en Java comme un choix personnel parce que j'ai vu trop de gens en abuser sous C++.

I left out operator overloading as a fairly personal choice because I had seen too many people abuse it in C++.

James Gosling (le concepteur du langage Java). Source :

www.gotw.ca/publications/c_family_interview.htm

La surcharge d'opérateurs sous C++

De nombreuses décisions C++ trouvent leurs racines dans mon aversion à forcer les gens faire les choses d'une manière particulière. J'ai souvent été tenté d'interdire des fonctionnalités qui me déplaisaient, mais je me suis abstenu de le faire, parce que je ne pense pas que j'aie le droit d'imposer ma vision sur les autres.
[. . .]
C++ est délibérément conçu pour supporter une variété de styles plutôt que d'imposer un voie/pensée unique.

Many C++ design decisions have their roots in my dislike for forcing people to do things in some particular way [...] Often, I was tempted to outlaw a feature I personally disliked, I refrained from doing so because I did not think I had the right to force my views on others.
[. . .]
C++ is deliberately designed to support a variety of styles rather than a would-be "one true way".

Source : The Desing and Evolution of C++ (1.3 General Background) par ***Bjarne Stroustrup***

(concepteur du C++)

Un peu d'histoire

Le langage `Java` a été créé par James Gosling en 1995, influencé par :

- `C++`, créé par Bjarne Stroustrup en 1983 qui est une importante extension du langage `C` crée par Ritchee en 1972 (au début, `C++` s'appelait `C with classes`)
- `C++` a été influencé par `Simula`, le premier langage orienté Objets (1965 !) qui avait proposé :
  - la notation `<objet>.<méthode>`
    - penser à `str.length()`, `lecteurClavier.nextInt()`, etc
  - l'héritage
    - penser que toute classe `Java` hérite la classe `Object`
  - la restriction d'accès (public, privé)
  - encore d'autres idées utiles, voir la page suivante du livre [Steve Lohr, *Go To : The Story of the Math Majors, Bridge Players, Engineers, Chess Wizards, Maverick Scientists and Iconoclasts—the Programmers who created the software revolution*]

Norway in the 1960s. Simula was developed for studying the flow of things through networks – people through train ticket booths, products through manufacturing lines, ships through harbors, data through computer programs. It was the work of two scientists at the Norwegian Computer Center in Oslo, Kristen Nygaard and Ole-Johan Dahl, from 1962 and 1967.

Simula grew out of a discipline known as "operations research," which championed the application of statistical methods such as simulation to analyze and solve organizational problems. On paper, it seemed a neutral technique for improving everyday efficiency by smoothing traffic flows and the like. But in practice, there was a whiff of social engineering to this branch of management science. Human activity was an operation to be researched, and once understood, controlled. Perhaps not surprisingly, for a while the Soviet Union became an enthusiastic user of Simula, running on the government's Ural mainframes.

In Norway, Nygaard grew concerned that his creation was being used to organize work to the detriment of workers. The result, he observed, was "more routine work, less demand for knowledge and a skilled labor force, less flexibility in the workplace, more pressure." Nygaard's political sympathies lay with the Nowegian trade unions, leaving him with "a moral dilemma," he said. "I realized that the technology I had helped to develop had serious consequences for other people, especially the people that I had come to identify with politically." Nygaard went to the powerful Norwegian trade unions and helped them push for "data and technology agreements" with corporations and the government that gave the unions a say in the introduction and use of computer technology.

Stroustrup was interested in the technology of Simula, not its politics. The "classes" he saw in Simula were categories of data types, and indeed, the language that became C++ was initially called "C with Classes." Besides classes, Simula had the related concepts of inheritance and objects, which enable the programmer to compose software out of well-defined modules. A programming language, as a "natural" language like English or French, is a medium for expressing ideas, and Stroustrup found that Simula provided an elegant framework for expressing complex programming ideas clearly.

The Simula concepts help the programmer to label different kinds of data, organize the data in logical hierarchies, and then let knowledge flow from one data class to another related data class. For example, the programmer might create a class called "employees." Each member of the class is called an "object," so Mary Smith is an object, for programming purposes. The programmer will declare that each class member, or object, has certain attributes (name, age, Social Security number, salary, etc.). There are certain things, called "methods," that can be done with or to each class member (say, hiring, firing, increasing salary, or changing a medical benefits plan). There can be subclasses – "manager" might be a subclass of the "employee" class, for example. That new class of objects can inherit the characteristics of the parent class. So the programmer is only required to specify what makes "managers" different, while the general characteristics of employees are inherited.

This kind of programming with data objects arranged in logical hierarchies is known as object-oriented programming, and Simula was the first object-oriented language. And while Simula itself was always a niche language, its influence was far-reaching because it introduced the object-oriented technique to programming. Later, Alan Kay and others at Xerox's Palo Alto Research Center took the concept further with Smalltalk, an object-oriented language that helped bring point-and-click graphic computing to work stations. And Java, the popular Internet programming language, is object-oriented software.