Bases de la programmation

Université de Nice - Sophia Antipolis
Richard Grin
Version 1.2.5 – 24/9/11
http://deptinfo.unice.fr/~grin

Objectif du cours

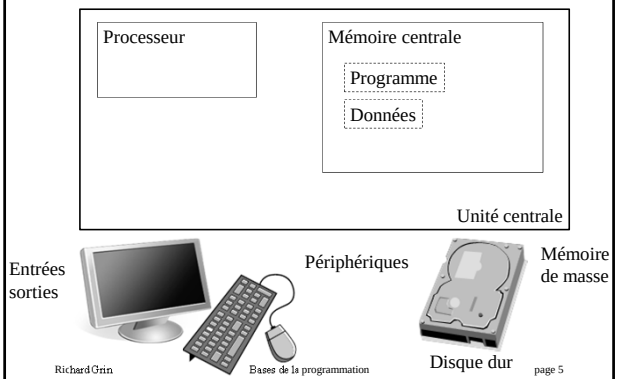
- Introduction à la programmation orientée objet pour pouvoir suivre le cours de POO de L3 Miage
- Aucune connaissance requise en programmation
- Les grandes étapes du cours :
  - programmation impérative
  - programmation structurée
  - les objets
- Langage utilisé : Java

Plan de cette partie

- Composants d'un ordinateur
- Programmes et langages informatiques
- Variables, instructions
- Premier programme en Java ; compilation, exécution
- Compilation, interprétation
- Exécution

Composants d'un ordinateur

(plus de détails dans le cours sur l'architecture des ordinateurs)



Utilisation des composants (1)

- Un programme est entré dans la mémoire centrale (tapé au clavier ou copié depuis le disque dur)
- L'utilisateur lance l'exécution du programme
- Les instructions du programme sont exécutées par le processeur
- Le programme utilise la mémoire centrale pour conserver des données calculées qui lui seront utiles plus tard dans l'exécution

Utilisation des composants (2)

- L'utilisateur peut transmettre des informations à l'aide du clavier ou de la souris
- Le programme peut aussi lire des informations sur le disque dur
- Les résultats de l'exécution sont affichés à l'écran ou enregistrés dans le disque dur (pour une session future d'utilisation)

Unités de mesure

- 1 bit : unité d'information de base, chiffre binaire 0 ou 1
- Octet : (*byte* en anglais) groupement de 8 bits
- K = 1000 = 10^3 ; Kilo ; Ko = 1000 octets
- M = 1 000 000 = 10^6 ; Mega ; Mo = 1000 Ko
- G = 1 000 000 000 = 10^9 ; Giga ; Go = 1000 Mo
- T = 1 000 000 000 000 = 10^{12} ; Tera ; To = 1000 Go
- Kb = Kilobyte = Ko ; Mb, Gb,…

Codage binaire (1)

- Base 2 ; pas 10 comme les nombres que l'on a l'habitude de manipuler
- 0 : 0
- 1 : 1
- 2 : 10 (1 « *deuzaine* »)
- 3 : 11
- 4 : 100 (1 *deuzaine* de *deuzaine*)
- 5 : 101
- …

Codage binaire (2)

- 2 bits permettent de coder 2^2 valeurs :
- 00, 01, 10, 11
- On pourrait par exemple coder les nombres décimaux 1, 2, 3, 4
- ou alors 0, 1, 2, 3
- ou alors -2, -1, 0, 1
- …

Calculs en binaire

- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 1 = 10 (« je retiens 1 » ou « je retiens une *deuzaine* »)
- 10 + 1 = 11
- 11 + 1 = 100
- 11 + 11 = 110

Codage

- 1 octet peut coder 2^8 = 256 valeurs différentes
- Les nombres de 0 à 257
- ou de -128 à 127
- ou 256 signes typographiques, par exemple, lettres majuscules et minuscules, chiffres, signes de ponctuation, symboles des opérations arithmétiques,…
- Exercice : démontrer que n bits peuvent coder 2^n valeurs différentes

Principales caractéristiques des composants

- Processeur : « intelligence » de l'ordinateur, il sait calculer et faire des choix (exécuter une instruction ou une autre suivant le contexte : si une valeur vaut 0, faire ceci, sinon faire cela)
- Appelé aussi CPU (*Central Process Unit*)
- Vitesse de quelques GHz (Hz = Hertz = nombre d'opérations élémentaires par seconde)
- Calculs avec des entiers ou « en virgule flottante »
- Peut contenir plusieurs cœurs pour faire des calculs en parallèle

Principales caractéristiques des composants

- Mémoire centrale : rapide (accès en nanosecondes, 10^{-9}) mais volatile (valeurs perdues entre 2 sessions de travail) ; capacités moyennes de quelques Go
- Mémoire périphérique : pas vraiment rapide (accès en millisecondes) mais non volatile ; appelée mémoire de masse car grosses capacités de centaines de Go à quelques To
  - Question : combien de fois plus lent que la mémoire centrale ?

Programmes et langages informatiques

Programme informatique

- Il sert à résoudre un problème (faire un calcul, exécuter des actions, aider à la prise de décision, dessiner,…)
- Il est écrit dans un langage qui contient des ordres que l'ordinateur peut « comprendre »
- Ce langage contient des instructions élémentaires qui disent à l'ordinateur ce qu'il doit faire

Langages informatiques

- Un ordinateur sans aucune logiciel ne comprend que le langage implanté dans son processeur
- Ce langage est de très bas niveau
- Les informaticiens écrivent des programmes en utilisant des langages de plus haut niveau (plus proches du langage humain)
- Des programmes, appelés compilateurs ou interpréteurs, traduisent ensuite ces programmes dans le langage du processeur

Code source – code exécutable

- Le code source est constitué des instructions d'un programme écrit dans un langage de haut niveau
- La traduction dans le langage de l'ordinateur s'appelle le code exécutable

Langages informatiques

- Il existe de très nombreux langages informatiques
- Ces langages peuvent être regroupés en quelques catégories principales, appelées des paradigmes de programmation
- Un paradigme est une façon de représenter, de modéliser une réalité

Programmation impérative

- La programmation impérative consiste à écrire un programme en donnant des instructions qui modifient l'état du programme (les données manipulées par le programme rangées dans des variables identifiées par des noms)
- Exemple :

```
lire x;
si (x > max)
   max = x;
…
```


Programmation (orientée) objet

- Manipule des objets qui s'envoient des messages
- Les messages envoyés aux objets peuvent être décrits en partie en utilisant la programmation impérative (variables et « si »)
- Exemple de code de l'objet « employé » :

```
[comptable], quel sera mon salaire le mois prochain ?
si (salaire < maDemande -100)
   [entreprise], je démissionne
```

Les objets entre [ ]

Les messages en italiques

- Le but final de ce cours est d'introduire au paradigme objet
- Pour apprendre les bases de la programmation, il est plus simple d'étudier tout d'abord le paradigme impératif

Instructions, variables

Instructions

- Les unités de programmation d'un programme informatique sont les instructions
- Les instructions sont regroupées en modules pour gérer la complexité et pour représenter les concepts utilisés par le langage (fonctions, objets ou messages par exemple)
- Dans le langage Java que nous allons utiliser, les instructions sont regroupées dans des blocs (délimités par des accolades), des méthodes (pour les messages) et des classes (pour les objets)

Instruction

- Une instruction comporte des mots-clés ou symboles définis par le langage et des expressions qui ont une valeur au moment de l'exécution
- Exemple :

```
if (x == 0)
    y = 5 * x;
```

- Les types d'instruction dépendent du langage qu'on utilise, en particulier du niveau d'abstraction du langage par rapport au processeur

Types d'instructions

- Pour les langages habituels (langage C par exemple) de type impératif les grands types d'instructions sont :
  - déclaration du type d'une variable
  - affectation d'une valeur à une variable
  - alternative : selon la valeur d'une expression, une séquence d'instructions est exécutée, ou une autre
  - répétition : une séquence d'instructions est répétée un certain nombre de fois

Variable

- Une variable correspond à un emplacement dans la mémoire centrale
- Une variable est utilisée par le programme pour enregistrer une valeur qu'il réutilisera dans la suite de son exécution
- Une variable est identifiée par son nom
- Affectation : enregistrer une nouvelle valeur dans une variable ; par exemple « x = 8; »

Convention sur les noms de variable en Java

- Peut contenir des lettres et des chiffres : **`x12`**
- Commence par une lettre minuscule : **`total`**
- Si composé de plusieurs mots, les mots internes commencent par une lettre majuscule : **`totalDesValeurs`**

Affectation

- x = *expression*;
- 2 étapes bien distinctes :
  1. calcul de l'*expression*
  2. rangement de la valeur calculée à l'emplacement mémoire désigné par la variable
- Exemple :

```
x = x + 1;
```


Exemple

```
x = 5;
y = x * 3;
if (y >= 15) {
  y = y + 10;
}
```

Quelle valeur aura **`y`** à la fin de ce code ?

Echanger la valeur de 2 variables

```
x = 10;
y = 15;
x = y;
y = x;
```

Quelles valeurs auront x et y à la fin de ce code ?

x | y
10 | 15

```
x = 10;
y = 15;
```


x | y
10 | 15

x = y; ??

x | y
15 | 15

```
x = y;
```


x | y
15 | 15

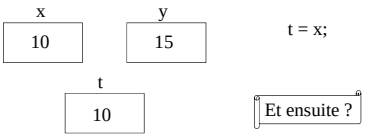
- La valeur 10 est perdue !
- Il aurait fallu la conserver avant d'exécuter « x = y »
- Comment conserver une valeur qui sera utilisée plus tard ?
- En conservant cette valeur en mémoire centrale
- Donc en l'affectant à une variable

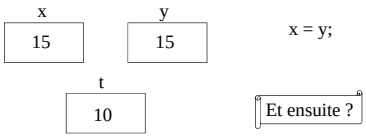
x | y | t
10 | 15 |

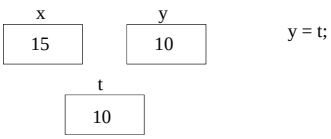
```
x = 10;
y = 15;
```

Par quelle instruction continuer ?

- Nouvelle variable t pour ranger la valeur de x qui va être « écrasée » par 15







« = » en Java et en mathématiques

- Ne pas confondre le « = » de l'affectation avec le « = » mathématique
- Il n'est pas symétrique :
  - à gauche doit se trouver un nom de variable qui désigne un emplacement mémoire
  - la droite peut contenir n'importe quelle expression qui calcule une valeur qui peut être rangée dans l'emplacement mémoire désigné par la gauche

Exemples

- a = a + 1 est une équation mathématique qui n'a pas de solution
- a = a + 1, en Java, incrémente de 1 la valeur de la variable a
- a + 5 = 25 n'a aucun sens en Java car « a + 5 » ne désigne aucun emplacement mémoire

Déclaration d'une variable

- En Java, toute variable doit être déclarée avant d'être utilisée
- La déclaration d'une variable indique le type des données qu'elle contiendra
- Une valeur entière est déclarée de type « **`int`** » (*integer* = entier en anglais)

Code pour échanger les valeurs de x et y

```
int x;
int y;
x = 10;
y = 15;
x = y;
y = x;
```

Qu'est-ce qu'il manque ?

Déclaration d'une variable

```
int x;
int y;
int t;
x = 10;
y = 15;
t = x;
x = y;
y = t;
```


Initialisation dans la déclaration

- Il est possible d'affecter une valeur initiale à une variable lors de sa déclaration :

```
int x = 10;
int y = 15;
int t = x;
x = y;
y = t;
```


Déclaration/initialisation de plusieurs variables

- Plusieurs variables de même type peuvent être déclarées (et même initialisées) dans une seule instruction :

```
int x, y;
int m = 5, n = 10;
```


Le type ne peut être changé

- En Java, le type d'une variable ne peut être changé
- Ce code est interdit :

```
int x;
…
double x;
…
```


Bloc d'instructions

- Presque tous les langages ont besoin de regrouper des instructions
- En Java les accolades permettent de regrouper plusieurs instructions en un bloc d'instructions

Exemple

```
int x = z + 3;
int y = w * 2;
int t;
if (x < y) {
  t = x;
  x = y;
  y = t;
}
```

Que fait ce code ?

Portée d'une variable

- La portée d'une variable désigne la portion du programme où la variable peut être utilisée
- Portée d'une variable en Java :
  - de l'endroit où elle a été déclarée
  - jusqu'à la fin du bloc où elle a été déclarée

Exemple

```
int x = …;
int y = …;
if (x < y) {
  int t = x;
  x = y;
  y = t;
}
int z = t + 2;
```

Portée de `t`

Erreur !

Langages typés

- Java est un langage typé : on doit déclarer le type d'une variable avant de l'utiliser
- Il existe d'autres langage non typés (Javascript, par exemple, pour programmer les clients Web) dans lesquels les types des variables ne sont pas indiqués dans le programme
- Les langages typés sont moins souples mais plus sûrs car davantage d'erreurs du développeur peuvent être détectées par le langage (par exemple une erreur dans le nom de la variable)

Afficher la valeur d'une variable (1)

- La « formule magique » (sera expliquée plus tard) pour faire afficher sur l'écran la valeur d'une variable x est
  `System.out.println(x);`
- Exemple :

```
x = 10;
System.out.println(x);
```

affichera 10 sur l'écran, suivi d'un passage à la ligne

Afficher la valeur d'une variable (2)

- `System.out.print(x);`
  affiche la valeur de la variable, sans passer ensuite à la ligne
- Exemple :

```
x = 10;
y = 20;
System.out.print(x);
System.out.println(y);
```

affiche 1020 sur l'écran, suivi d'un passage à la ligne

Afficher un texte

- **`System.out.print("Hello world");`** affiche « Hello world » sur l'écran
- **`System.out.println("Hello world");`** affiche « Hello world » sur l'écran, et passe à la ligne ensuite

Variable de type texte

- Une variable peut contenir un texte
- Il faut la déclarer de type **`String`** :

```
String nom = "Dupond";
```


Nombres à virgule

- Le type **`double`** indique une variable qui peut contenir un nombre à virgule (remplacée par un « . ») :

```
double largeur = 12.5;
```

- Le nombre de chiffres après la virgule n'est pas fixé ; le type **`double`** désigne des nombres à « virgule flottante »

Affichage texte et nombre

- Soit la ligne de code

```
int x = 10;
```

- Comment faire afficher « Valeur de x = 10 » en utilisant la variable x ?
-

```
System.out.print("Valeur de x = ");
System.out.println(x);
```

- En fait on peut aussi utiliser la concaténation des chaînes de caractères de Java :

```
System.out.println("Valeur de x = " + x);
```

(le nombre entier 10 est traduit par Java en **`String`**)

Commentaires en Java (1)

- Lorsqu'un programme contient du code qui n'est pas complètement évident il est important d'ajouter des commentaires pour l'expliquer
- En effet, un programme utile sera souvent modifié pour l'adapter à un changement de contexte ou pour corriger des erreurs ; le code doit donc être facilement compris par les développeurs futurs qui le modifieront

Commentaires en Java (2)

- 3 façons d'ajouter des commentaires
- En fin de ligne (ne peut couvrir plusieurs lignes) :

```
x = 12; // pour indiquer que ….
```

- Juste avant le code (une ou plusieurs lignes) :

```
/* Pour indiquer que … */
x = 12;
```

- Idem /* mais pour la javadoc (avant déclarations ou définitions) :

```
/** Pour indiquer que … */
int x = 12;
```

Commentaires en Java (3)

- Pour faire plus joli et pour la lisibilité :

```
/**
 * Pour indiquer que …
 *
 */
```

- « // » Peut aussi s'utiliser sur une seule ligne avant l'instruction ou le bloc d'instructions à commenter :

```
// Pour indiquer que ….
x = 12;
```


Premier programme en Java ; compilation, exécution

Le code source du premier programme

classe **HelloWorld**

```
public class HelloWorld {
  public static void main(String[] args){
    System.out.println("Hello world");
  }
}
```

- La classe Helloworld est **public**, donc le fichier qui la contient doit s'appeler

  HelloWorld.java

Le code source du premier programme

méthode **main**

```
public class HelloWorld {
  public static void main(String[] args){
    System.out.println("Hello world");
  }
}
```

- La classe **Helloworld** contient une seule méthode : **main**
- Signature de la méthode **main** : **String[] args**
  Un seul paramètre de type « tableau de chaînes de caractères » (étudié plus tard dans le cours)

Méthode

- Une méthode **static**, comme la méthode **main**, correspond à un message que la classe peut recevoir
- Si une classe reçoit un message, elle exécute le code contenu dans la méthode correspondante
- La méthode qui s'appelle **main** est particulière : c'est le message qui est envoyé à la classe si on lance son exécution
- Si on lance l'exécution de la classe **HelloWorld**, celle-ci va afficher « Hello World » sur l'écran

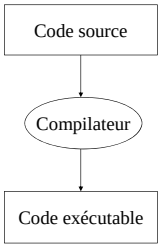
Compilation, interprétation

Compilation d'un code source

- Un code source ne peut être exécuté directement par un ordinateur
- Il faut traduire ce code source dans un langage que l'ordinateur (le processeur de l'ordinateur) peut comprendre
- Un compilateur est un programme qui effectue cette traduction

La compilation

Programme écrit par un développeur



Programme dans le langage de la machine

Exécution

- Lorsqu'un programme a été compilé, on peut faire exécuter le code exécutable
- Le code source n'est pas utilisé pour l'exécution ; on pourrait très bien le supprimer

Maintenance d'un programme

- En fait, il faut garder le code source pour le cas (certain) où le développeur voudrait modifier le programme
- Un programme est très souvent modifié
  - pour corriger les erreurs (son comportement n'est pas correct ou les résultats sont faux)
  - pour ajouter des fonctionnalités

JDK

- Lorsqu'on télécharge Java pour écrire des programmes, on récupère le JDK (*Java Development Kit*)
- Le JDK contient :
  - Des outils de développement (javac, java, javadoc,…)
  - De très nombreuses classes déjà écrites et que l'on peut utiliser quand on écrit des programmes ; par exemple la classe **`System`** (**`System.out.println("Hello world")`**)

Compilation avec *javac*

- Le JDK fournit le compilateur *javac (java compiler)*
- **`javac HelloWorld.java`**
  crée un fichier « **`HelloWorld.class`** » qui contient le code exécutable, et le place dans le même répertoire que le fichier « **`.java`** »
- Le fichier à compiler peut être désigné par un chemin absolu ou relatif :
  `javac util/Liste.java`

Exécution du programme

- Pour exécuter le programme exécutable (**`HelloWorld.class`**) on utilise la commande « *java* » fournie par le JDK
- **`java HelloWorld`**
  exécute le code de la méthode **`main`** de la classe **`HelloWorld`**

Exécution du programme

- Attention ! pas
  ~~**`java HelloWorld.class`**~~
  mais
  `java HelloWorld`
- **`HelloWorld`** est un nom de classe et pas un nom de fichier. Donc
  - on ne peut pas donner un chemin
  - pas de suffixe **`.class`**

Où doit se trouver le fichier .class ?

- **`java HelloWorld`**
  `HelloWorld.class` doit se trouver dans le *classpath*
- Le *classpath* peut recevoir une valeur avec l'option **`-classpath`** de la commande java :
  `java –classpath rep1/rep2 HelloWorld`
- Par défaut le *classpath* est le répertoire courant

Classpath

- L'option **`–classpath`** de java (et de javac) permet d'indiquer où la commande doit aller chercher les classes dont elle a besoin
- Cette option permet d'indiquer un ou plusieurs emplacements dans l'arborescence des fichiers
- Le séparateur entre plusieurs emplacements est « : » en Linux et « ; » en Windows

Exemples de classpath

- Sous Unix :
  `.:~/java/mesclasses1:~/mesclasses2`
- Sous Windows :
  `.;c:\java\mesclasses1;c:\mesclasses2`

Programme Java

- Il peut être composé d'une ou de plusieurs classes
- Toutes les classes qui se trouvent dans le *classpath* peuvent être utilisées par le programme (en plus des classes fournies par le JDK)

Exemple

- Dans le TP 2, une classe **`Console`** sera fournie
- Cette classe contient la méthode **`readInt()`** pour lire la valeur d'un entier au clavier
- Il suffira de mettre le fichier **`Console.class`** dans le répertoire courant pour pouvoir utiliser la classe **`Console`** :

```
int x = Console.readInt();
```

met dans la variable **`x`** la valeur tapée au clavier par l'utilisateur

Conventions pour les identificateurs Java

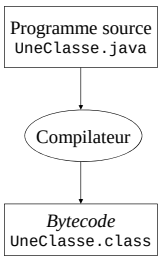
- Les noms de classes commencent par une majuscule : **`Cercle`**, **`Object`**
- Les noms de variable commencent par une minuscule
- Les mots contenus dans un identificateur commencent par une majuscule : **`UneClasse`**, **`uneMethode`**, **`uneAutreVariable`**

Compilation en Java → *bytecode*

- En Java, le code source n'est pas traduit directement dans le langage de l'ordinateur
- Il est d'abord traduit dans un langage appelé « *bytecode* », langage d'une machine virtuelle (JVM ; *Java Virtual Machine*) définie par *Sun*
- Ce langage est indépendant de l'ordinateur qui va exécuter le programme

La compilation fournit du *bytecode*

Programme écrit en Java



Programme en *bytecode*, indépendant de l'ordinateur

Exécution du *bytecode*

- Le *bytecode* doit être exécuté par une JVM
- Cette JVM n'existe pas ; elle est simulée par le programme java qui interprète le *bytecode :*
  - lit les instructions (en *bytecode*) du programme **`.class`**,
  - les traduit dans le langage natif du processeur de l'ordinateur
  - lance leur exécution

Utilité de la JVM

- Du code exécutable Java (fichier .class) peut facilement être transportable d'un ordinateur à l'autre, même s'ils ne possèdent pas le même type de processeur
- Très utile pour exécuter du code récupéré sur le Web
- Sans cette JVM il faudrait récupérer le code source et le recompiler (opération lourde si le programme contient beaucoup de code)

Langages compilés ou interprétés

- Java est un langage compilé : un programme Java doit être traduit en entier par un compilateur avant d'être exécuté
- Des langages n'ont pas d'étape de compilation ; on les appelle des langages interprétés
- Le code source est interprété pendant l'exécution par un interpréteur

Langages interprétés

- L'interpréteur est présent pendant l'exécution
- Il lit d'abord une instruction, puis la traduit dans le langage du processeur pour la faire exécuter
- Il lit ensuite l'instruction suivante pour la traduire et la faire exécuter et ainsi de suite…
- Comme un interprète qui traduit un discours au fur et à mesure qu'il est prononcé

Avantages et inconvénients de la compilation

- Les langages interprétés sont souvent plus souples mais moins sûrs que les langages compilés
- En effet, un grand nombre d'erreurs de programmation peuvent être détectées dans l'étape de la compilation, ce qui évite des erreurs pendant l'exécution
- L'exécution avec un langage interprété est sont souvent moins rapide, car elle est ralentie par la traduction pendant l'exécution

Question

- A quel moment avons-nous déjà rencontré un interpréteur dans ce cours ?

Erreurs détectées à la compilation (1)

- La compilation vérifie que les instructions du programme ont bien un sens (en accord avec la définition du langage utilisé) ; elle vérifie la syntaxe du programme
- Par exemple, le compilateur indiquera une erreur s'il manque un « ; » à la fin d'une instruction

Erreurs détectées à la compilation (2)

- Le compilateur vérifie aussi que le type des expressions est correct ; par exemple, l'instruction suivante provoquera une erreur à la compilation :

```
int x = "La valeur est " + y;
```

- Le code exécutable ne sera produit que si le code source ne contient aucune erreur (de syntaxe ou de typage)

Erreurs à l'exécution

- Le compilateur ne peut pas tout vérifier ; il vérifie que le programme a un sens mais il ne peut pas vérifier que ce sens est bien celui que le programmeur voulait lui donner
- L'exécution du programme peut donc ne pas donner le résultat prévu
- Des erreurs peuvent même empêcher le programme de fonctionner, par exemple si une valeur saisie par l'utilisateur provoque le calcul de la racine carrée d'un nombre négatif

Tests

- Il est difficile d'écrire un grand nombre de lignes correctement du premier coup
- Des méthodes de programmation donnent des recettes pour éviter au maximum les erreurs mais l'écriture de tests est indispensable pour éliminer le plus d'erreurs possibles avant l'utilisation en production du code

Votre environnement de développement

- Éditeur de texte pour taper le code source (emacs ou un autre, mais avec indentation automatique en java) ; il doit fournir du texte pur, pas du texte formaté comme le fait Word
- Compilateur pour transformer le code source en *bytecode* (javac)
- Interpréteur de *bytecode* pour exécuter le programme (java)

Complément sur les variables : les constantes en Java

Déclaration d'une constante

- Lorsqu'un programme utilise la même valeur constante à plusieurs endroits, il est bon de créer une constante nommée pour la désigner
- Exemple :

```
final static double PI = 3.14;
```

- Cette déclaration doit être mise en dehors de toute méthode ; il est d'usage de la mettre au début de la classe
- En Java la convention est de mettre le nom d'une constante en majuscule : **`UNE_CONSTANTE`**

Exemple

```
public class Cercle {
  public static final double PI = 3.14;

  public static void main(String[] args) {
    double rayon = …;
    double surface = PI * rayon * rayon;
    …
  }
}
```

Utilisation d'une constante

- Le programme ne pourra la modifier
- ~~`PI = PI + 1.2;`~~
  provoquera une erreur à la compilation

Visibilité d'une constante

- Si la définition de la constante est précédée de **`private`**, elle n'est utilisable que dans la classe
- Si elle est précédée de **`public`**, la constante est utilisable en dehors de la classe, en préfixant son nom par le nom de la classe ; par exemple :

```
x = 2 * Cercle.PI;
```


Avantages de la déclaration d'une constante

- Le nom de la constante et les éventuels commentaires documentent la valeur
- Il est facile de changer la valeur en ne modifiant qu'une seule instruction