

Chapitre 2: L'essentiel pour commencer

(NFA031 - Jour)

V. Aponte & P. Courtieu

Cnam

Comprendre les programmes

Programmes = Données + Instructions

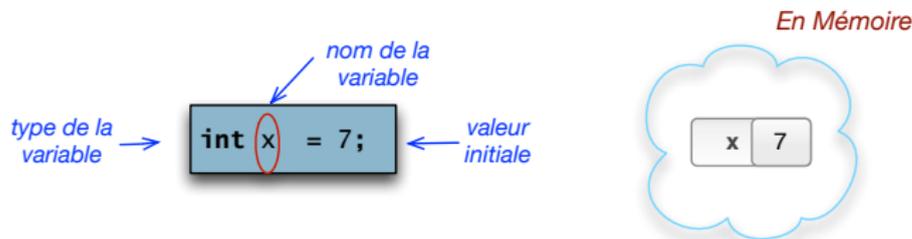
- **Données** \Rightarrow comprendre variables + types.
- **Instructions** \Rightarrow comprendre structures de contrôle + sous-programmes.

Comprendre les programmes (2)

Ecrire des programmes \Rightarrow comprendre :

- **syntaxe des instructions** :
 \Rightarrow *comment agencer mot-clés et symboles.*
- **sémantique des instructions** :
 \Rightarrow *comportement à l'exécution.*

Variables et types : un aperçu

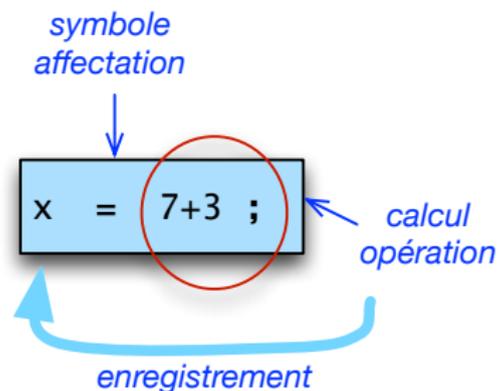


Variables : C'est quoi ? (**sémantique**) :

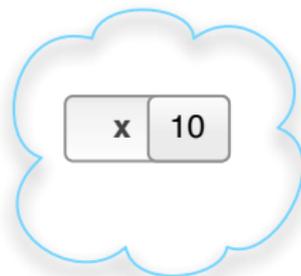
- un **emplacement mémoire** qu'on **nomme** (ici `x`) ;
- qui **contient une donnée** (ici `7`),
- et qui possède **type** (ici `int`) ;

et un type ? décrit la nature d'une donnée. Le type `int` signifie « nombre entier ».

Instructions : un aperçu (affectation)



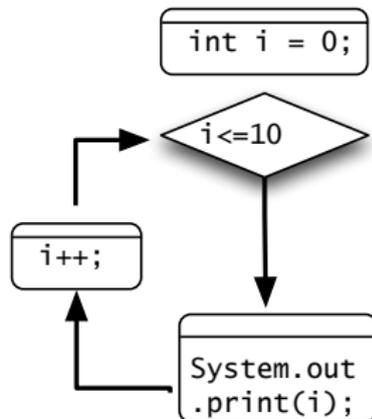
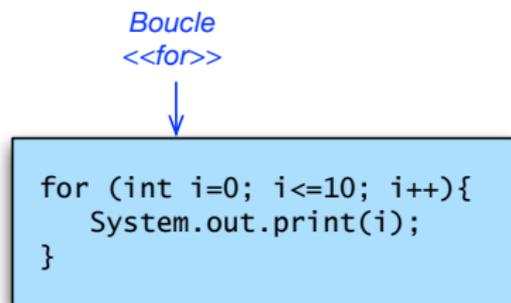
En Mémoire



Instruction C'est quoi ? **ordre à exécuter**.

Ici : « (1) calculer `7+3` ; (2) enregistrer en mémoire ».

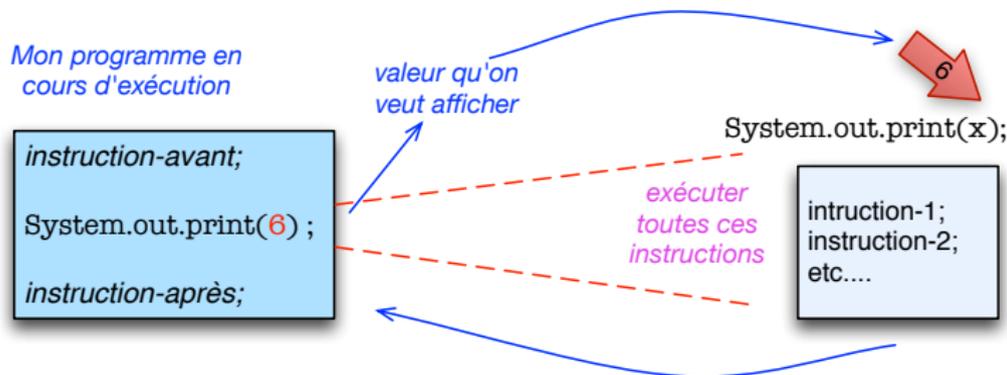
structure de contrôle : un aperçu (boucle)



C'est quoi ? Instruction complexe permettant de répéter, faire des choix, etc.

boucle Instruction de répétition.

Appel à sous-programmes : un aperçu



C'est quoi ? ordre d'exécution d'un **programme externe**. Ici on demande à `System.out.print` d'afficher 6.

sous-programme programme externe avec un nom. Exécuté quand ce nom est rencontré.

retour à la suite de mon programme.

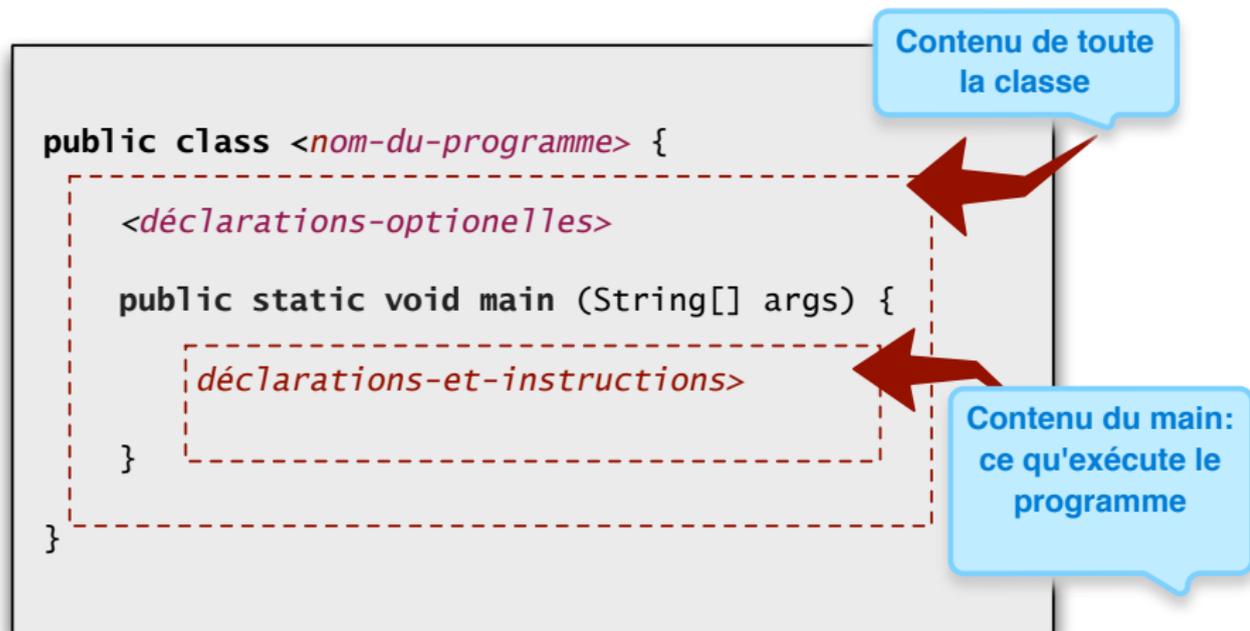
Les programmes en Java.

Un premier programme

Calcul du carré d'un nombre entier lu au clavier

```
public class Carre {
    public static void main (String[] args) {
        int x;
        Terminal.ecrireStringln("Un_entier?_");
        x = Terminal.lireInt();
        int z = x * x;
        Terminal.ecrireStringln("Le_carre_de_" + x + "_est_" + z);
    }
}
```

Structure d'un programme Java



Structure d'un programme Java

Un programme Java est composé, au minimum de :

- Une «classe principale» :

```
public class <nom-du-programme>
```

- contenant un sous-programme ou «méthode principale» :

```
public static void main (String[] args)
```

- contenus classe principale et méthode main : délimités entre accolades { } .
- **public, class, static, void, main** ⇒ mots réservés : on ne peut pas les employer pour nommer variables, programmes, etc.

Structure d'un programme Java (2)

Nom programme : nom de la classe principale =

- nom programme en entier,
- nom du fichier contenant le programme + extension
.java.

Ex : le programme `Carre` doit se trouver dans le fichier
`Carre.java`

méthode `main` : obligatoire dans tout programme Java.

Exécuter `Carre`, c'est exécuter sa méthode `main`.

déclarations-instructions : variables et instructions à exécuter.

Comprendre Carre

```
public class Carre {  
    public static void main (String[] args) {  
        int x;  
        Terminal.ecrireStringln("Un entier?_");  
        x = Terminal.lireInt();  
        int z = x * x;  
        Terminal.ecrireStringln("Le_carre_de_" + x + "_est_" + z);  
    }  
}
```

ligne 3 : Déclaration des variables : x de type `int` ;

ligne 4 : affiche le message "Un entier ?"

ligne 5 : lecture d'un nombre, puis affectation dans x

ligne 6 : calcul du carré et affectation dans z , (déclaré au passage)

ligne 7 : affichage du résultat.

L'importance de commentaires

Les programmes sont lus par les développeurs, qui les font évoluer.

- la clarté et lisibilité du code est fondamentale ;
- les commentaires sont dès lors indispensables :
 - ▶ ils doivent décrire ce que le code fait, dans tous les cas, et sans ambiguïté.

```
/**  
 * Calcule et affiche le carré d'un nombre lu au clavier.  
 */  
public class Carre { ... }
```

Tous vos programmes doivent être commentés !

Les identificateurs (pour nommer)

Pour nommer les variables, les programmes, les sous-programmes.

Identificateur : mot qui commence par une lettre, ou par `_`, ou par `$`, suivi ou non de caractères parmi : `a ... z`, `A ... Z`, `0..9`, `_`, `$`, et caractères Unicode.

En plus, ils doivent respecter :

- Ne peut pas être un mot réservé : (`abstract`, `boolean`, `if`, `public`, `class`, `private`, `static`, etc).
- Caractères interdits : (`^`, `[`, `]`, `{`, `+`, `-`, ...).

Exemples : `a`, `id_a` et `X1` sont des noms de variables valides, alors que `1X` et `X-X` ne le sont pas.

3. Types de données en Java.

Les types

Type Description de la nature d'une donnée. Ex : un nombre entier (`int`), un nombre réel (`double`), un tableau, un booléen (`boolean`).

Constantes A chaque type correspond un ensemble de valeurs constantes. Ex : pour `int`, les constantes sont 1, 2, -4, etc.

Opérations Chaque type possède des **opérateurs** pour calculer avec ses valeurs. Ex : `+`, `-`, `*`, `/` sont des opérateurs pour `int` et `double`.

Les types en Java

- **Types primitifs** : pour les données élémentaires (une donnée « indivisible » : un entier, un booléen, etc).
 - ▶ en Java ils sont tous **prédéfinis** ;
 - ▶ **nous utiliserons** : `int`, `double`, `boolean`, `char` (il y en a d'autres).
- **Types référence** : ce sont les types des objets et des tableaux.
 - ▶ regroupent en général plusieurs données (ce n'est pas une donnée élémentaire)
 - ▶ certains sont prédéfinis : **nous utiliserons** `String`.

Quelques types prédéfinis en Java (int, double)

`int` nombres entiers représentables en machine sur 32 bits (31 plus le signe) $\{-2^{31}, \dots, 2^{31}\}$.

Constantes : `-7`, `0`, `1`, `2`....

Opérateurs : `+`, `-`, `*`, `/`, `%`, `==`, `>`, `<`, `>=`, `!=`

Attention : **division entière** si tous les opérandes sont des entiers !

```
/* Exemples */  
int x=2; int y = 6;  
x = x+y*3;
```

`double` nombres à virgule (à précision double, 64 bits).

Constantes : `0.0`, `0.1`, ...`18.58` ...

Mêmes opérateurs, sauf modulo.

Quelques types en Java (boolean)

`boolean` modélise les deux valeurs de vérité en logique.

Constantes : `true` et `false`.

Opérateurs : `&&` (et), `!` (négation), `||` (ou).

```
/* Exemples */
boolean a = false;
boolean b;
b = true || false;    // b ← true
b = true && false;    // b ← false
b = !a;               // b ← true
```

Type char

`char` caractères Unicode (sur 16 bits). Constantes : les caractères entourés de guillemets simples, p.e : ' a ', ' 2 ', ' @ ' ;
Opérateurs : de comparaison (=, <, >, !=,...)

```
/* Exemples */  
char a = 'c';  
Terminal.ecrireChar(a); // affiche c  
Terminal.ecrireChar('a'); // affiche a  
Terminal.ecrireChar('\t'); // affiche une tabulation
```

Type String

`String` chaînes de caractères.

Constantes suites de caractères entourées de guillemets doubles :
"entrer une somme en euros :?", "coucou", "a"
..., y compris la chaîne vide ""

Opérateurs + est l'opérateur de concaténation.

```
/* Exemples */  
int x = 3;  
Terminal.ecrireString("a");           // affiche a  
Terminal.ecrireString("a" + "b");     // affiche ab  
Terminal.ecrireString("x=␣" + x);     // affiche x = 3
```

Opérateurs de comparaison

Comparent deux expressions **de même type** (primitif).
Le résultat est un booléen.

==	égalité
<	plus petit
>	plus grand
>=	plus grand ou égal
<=	plus petit ou égal
!=	(différent)

Les expressions

Une expression c'est ...

- une **valeur constante**, ou une variable, ou opération composée d'opérateurs et opérandes.
 - ▶ **opérandes** : valeurs constantes, variables, ou appels de fonctions.
- **Sa valeur** est le résultat d'appliquer les opérateurs sur les opérandes.

Expression	Valeur calculée	Particularités
7	7	expression constante
$7 + 6.5$	13.5	expression arithmétique
"x" + "y"	"xy"	expression de concaténation sur des chaînes.
$x + 3$?	dépend de la valeur de x en mémoire
Math.min(3,4)	3	appel à une fonction
Terminal.lireInt()	valeur saisie	appel à une fonction

Exemples d'expressions

Supposons qu'en mémoire : $x=3$, $a=true$, $b =false$, $c='x'$.

Expression	Valeur
$7 > 4$	true
$7 \geq 7$	true
$7 \neq 7$	false
$'a' < 'b'$	true
$c > 'c'$	true
$c == 'c'$	false
$!a$	false
$a \ \&\& \ !b$	true
$x < 10$	true
$(x > 10) == ('b' == c)$	true

Contre-exemples :

- `int x = 1;`
- `System.out.println(5);`

Où trouver les expressions ?

Un peu partout ! Essentiellement :

- membre droit d'une affectation ou initialisation,
- arguments d'un appel à un sous-programme.

```
/* Expressions dans toutes les lignes de ce code! */  
int x = 1;  
int y = x+2;  
boolean a = false;  
b = true || false;  
if (y>x)  
    x = Terminal.lireInt();  
else  
    y = Math.min(x*2,y);
```

Opérateurs d'incrément et décrement

Applicables sur tous les types numériques et sur `char`

```
i++;   équivaut à i=i+1;  
i--;   équivaut à i=i-1;
```

Opérateurs d'affectation :

```
x -= y;   équivaut à   x=x-y;  
x += y;   équivaut à   x=x+y;   (aussi avec String)  
x *= y;   équivaut à   x=x*y;  
x /= y;   équivaut à   x=x/y;
```

Opérateurs conditionnel

Permet d'obtenir une valeur résultat au moyen d'une conditionnelle.

```
<expr-bool> ? <expression1> : <expression2>;
```

- si <expr-bool> est vrai, renvoie <expression1>,
- sinon renvoie <expression2>

```
anneeSuivante = (mois == 12) ? (annee+1) : annee;
```

`anneeSuivante` est `(annee+1)` si on est en décembre, égale à `annee` sinon.

Précédence des opérateurs

Les expressions composées de plusieurs opérateurs sont évaluées de la gauche vers la droite, selon des règles de précedence indiquant la priorité des opérateurs les uns par rapport aux autres.

`2 + 3 * 4` équivaut à `2 + (3*4) = 14`
et non pas à `20`

`2 + 3 > 7` s'évalue en **true**

`2 + (3 > 7)` produit une erreur

Précédence des opérateurs

!	négation
* / %	multiplication, division, modulo
+ -	addition, soustraction
> < >= <= == !=	comparaison
&&	booléens
=	affectation

4. Variables et affectation.

Déclaration de variables

Avant d'utiliser une variable on doit la déclarer.

Syntaxe

```
<type-de-la-variable> <nom-variable>;  
<type-de-la-variable> <nom-variable> = <expression>
```

```
char reponse;  
int x = 0;
```

- type de la variable, puis son nom (identificateur),
- optionnellement, on donne une valeur initiale,
- les types de la valeur et de la variable doivent correspondre.

Déclaration de variables (sémantique)

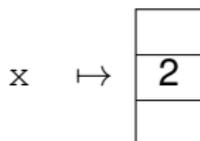
Exemple :

```
int x = 2;
```

Sémantique :

- on réserve (mémoire) un emplacement suffisant pour stocker une valeur du type de la variable (ici 32 bits),
- on y met la valeur (si elle est donnée). Elle doit être du bon type.

Mémoire



Utiliser une variable

Equivaut à utiliser la valeur qu'elle contient en mémoire.

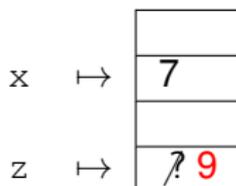
Elle doit être **déclarée** et **initialisée**.

```
int x = 7; int z;  
z = x + 2;
```

Calculer $x+2 \Rightarrow$ chercher en mémoire la valeur de x .

$x+2 \Rightarrow 7+2 \Rightarrow 9$.

Mémoire



Instruction d'affectation

Syntaxe

```
<nom-variable> = <expression>;
```

```
int x = 3;      int y = 7; // declarations  
x = x + y;     // affectation
```

Sémantique

- **A la compilation** : types à droite et à gauche doivent être compatibles.
- **Exécuté en 2 temps** : (1) calculer valeur à droite ; (2) copier résultat dans (emplacement) pour variable à gauche.

Exercice pour tester votre compréhension

Ecrire un programme Java qui :

- échange les valeurs de deux variables x (initialisée avec 2) et y (initialisée avec 10), toutes deux de type entier.
- Votre programme doit afficher les valeurs de ces deux variables avant et après l'échange.
- dessinez les changements dans la mémoire pour toutes les variables du programme.

5. Les sous-programmes en Java (méthodes).

Qu'est-ce qu'un programme (séquentiel) ?

Une recette, comme en cuisine. Une étape après l'autre.

- 1 Pelez et hachez finement 1/4 oignon(s).
- 2 Dans un bol :
 - 1 battez 2 oeufs entiers
 - 2 ajoutez 1 branche de basilic ciselé
 - 3 ajoutez 1/4 cube de bouillon de volaille.
 - 4 ajoutez sel et poivre à votre goût
 - 5 mélangez bien au fouet.
- 3 Dans une poêle :
 - 1 faites chauffer l'huile d'olive
 - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage râpé laissez cuire 7 min.

Écrire une recette \neq faire la cuisine

- Une recette écrite par un auteur :
 - ▶ est un morceau de texte
- Une exécution par un cuisinier :
 - ▶ qui lit la recette
 - ▶ réalisation des actions décrites

Écrire une recette \neq faire la cuisine

- Un programme écrit par un programmeur :
 - ▶ est un morceau de texte
- Une exécution par un processeur :
 - ▶ qui lit le programme
 - ▶ réalisation des actions décrites
- 1 programme, ∞ exécutions différentes
 - ▶ Paramètres différents
 - ▶ inputs différents (clavier, souris, fichier, etc)

1 programme, ∞ exécutions différentes

Ne pas confondre :

- Le programmeur qui programme :
 - ▶ prévoit chaque cas possibles à l'avance
 - ▶ décrit la (sous-)recette à suivre dans chacun des cas
- Le processeur qui exécute
 - ▶ ne fait que suivre la « recette » pas à pas

Appel à sous-programme = exécuter une recette

« externe »

```
Terminal.ecrireString("Bonjour!");
```

= appel au sous-programme `Terminal.ecrireString`

- 1 Séparez les feuilles de la salade
- 2 Lavez les feuilles
- 3 **Faites une vinaigrette (p.12)**
- 4 égouttez les feuilles
- 5 ...

Appel de sous-programme I

Salade verte : *(recette courante)*

Séparez les feuilles de la
salade

Lavez les feuilles

Faites une vinaigrette

égouttez les feuilles

...

Vinaigrette : *(recette externe)*

versez l'huile dans un bol

versez le vinaigre

versez la moutarde

battre jusqu'à émulsion

salez poivrez

Appel de sous-programme I

Salade verte : *(recette courante)*

1. Séparez les feuilles de la
salade

Lavez les feuilles

Faites une vinaigrette

égouttez les feuilles

...

Vinaigrette : *(recette externe)*

versez l'huile dans un bol

versez le vinaigre

versez la moutarde

battre jusqu'à émulsion

salez poivrez

Appel de sous-programme I

Salade verte : *(recette courante)*

1. Séparez les feuilles de la salade
2. Lavez les feuilles

Faites une vinaigrette

égouttez les feuilles

...

Vinaigrette : *(recette externe)*

versez l'huile dans un bol

versez le vinaigre

versez la moutarde

battre jusqu'à émulsion

salez poivrez

Appel de sous-programme I

Salade verte : *(recette courante)*

1. Séparez les feuilles de la salade
2. Lavez les feuilles
3. **Faites une vinaigrette**
égouttez les feuilles
...

Vinaigrette : *(recette externe)*

versez l'huile dans un bol
versez le vinaigre
versez la moutarde
battre jusqu'à émulsion
salez poivrez

- arrête l'exécution de la recette actuelle

Appel de sous-programme I

Salade verte : *(recette courante)*

1. Séparez les feuilles de la salade
 2. Lavez les feuilles
 3. **Faites une vinaigrette**
- égouttez les feuilles
- ...

Vinaigrette : *(recette externe)*

4. versez l'huile dans un bol
- versez le vinaigre
- versez la moutarde
- battre jusqu'à émulsion
- salez poivrez

- arrête l'exécution de la recette actuelle
- démarre l'autre recette

Appel de sous-programme I

Salade verte : *(recette courante)*

1. Séparez les feuilles de la salade
2. Lavez les feuilles
3. **Faites une vinaigrette**
égouttez les feuilles
...

Vinaigrette : *(recette externe)*

4. versez l'huile dans un bol
5. versez le vinaigre
versez la moutarde
battre jusqu'à émulsion
salez poivrez

- arrête l'exécution de la recette actuelle
- démarre l'autre recette

Appel de sous-programme I

Salade verte : (*recette courante*)

1. Séparez les feuilles de la salade
2. Lavez les feuilles
3. **Faites une vinaigrette**
égouttez les feuilles
...

Vinaigrette : (*recette externe*)

4. versez l'huile dans un bol
5. versez le vinaigre
6. versez la moutarde
battre jusqu'à émulsion
salez poivrez

- arrête l'exécution de la recette actuelle
- démarre l'autre recette

Appel de sous-programme I

Salade verte : *(recette courante)*

1. Séparez les feuilles de la salade
2. Lavez les feuilles
3. **Faites une vinaigrette**
égouttez les feuilles
...

Vinaigrette : *(recette externe)*

4. versez l'huile dans un bol
5. versez le vinaigre
6. versez la moutarde
7. battre jusqu'à émulsion
salez poivrez

- arrête l'exécution de la recette actuelle
- démarre l'autre recette

Appel de sous-programme I

Salade verte : (*recette courante*)

1. Séparez les feuilles de la salade
2. Lavez les feuilles
3. **Faites une vinaigrette**
égouttez les feuilles
...

Vinaigrette : (*recette externe*)

4. versez l'huile dans un bol
5. versez le vinaigre
6. versez la moutarde
7. battre jusqu'à émulsion
8. salez poivrez

- arrête l'exécution de la recette actuelle
- démarre l'autre recette
- autre recette finie ⇒ redémarre l'actuelle

Appel de sous-programme I

Salade verte : (*recette courante*)

1. Séparez les feuilles de la salade
2. Lavez les feuilles
3. **Faites une vinaigrette**
9. égouttez les feuilles
- ...

Vinaigrette : (*recette externe*)

4. versez l'huile dans un bol
5. versez le vinaigre
6. versez la moutarde
7. battre jusqu'à émulsion
8. salez poivrez

- arrête l'exécution de la recette actuelle
- démarre l'autre recette
- autre recette finie \Rightarrow redémarre l'actuelle

Appel de sous-programme I

Salade verte : *(recette courante)*

1. Séparez les feuilles de la salade
2. Lavez les feuilles
3. **Faites une vinaigrette**
9. égouttez les feuilles
10. ...

Vinaigrette : *(recette externe)*

4. versez l'huile dans un bol
5. versez le vinaigre
6. versez la moutarde
7. battre jusqu'à émulsion
8. salez poivrez

- arrête l'exécution de la recette actuelle
- démarre l'autre recette
- autre recette finie ⇒ redémarre l'actuelle

Sous-programmes (méthodes)

En suivant notre analogie :

- Une recette peut faire appel à des « sous-recettes » :
 - ▶ on les exécute puis on revient pour continuer la « recette initiale ».
- Un programme peut appeler d'autres programmes : ce sont ses **sous-programmes** :
 - ▶ on les exécute puis on revient au point où on était restés dans le « code appelant ».

Sous-programme (méthode)

Instructions regroupée sous un nom, exécutées chaque fois que ce nom apparaît.

- **invoqué**, ou **appelé** (pour exécuter ses instructions),
- on peut lui fournir des entrées (**arguments**, **paramètres**),
- il peut **retourner** une valeur résultat, qu'on pourra utiliser après son exécution.
- 2 sortes de sous-programmes :
 - ▶ **fonctions** : retournent un résultat.
 - ▶ **procédures** : ne retournent pas de résultat.

Omelette pour 1 personne(s)

- 1 Pelez et hachez finement 1/4 oignon(s)
- 2 Dans un bol :
 - 1 battez 2 oeufs entiers
 - 2 ajoutez 1 branche(s) de basilic ciselé
 - 3 ajoutez 1/4 cube(s) de bouillon de volaille.
 - 4 ajoutez sel et poivre à votre goût
 - 5 mélangez bien au fouet.
- 3 Dans une poêle :
 - 1 faites chauffer l'huile d'olive
 - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage

Paramètres de sous-programmes

Une seule personne !

Omelette pour 1 personne(s)

1 Pelez et hachez finement 1/4 oignon(s)

2 Dans un bol :

1 battez 2 oeufs entiers

2 ajoutez 1 branche(s) de basilic ciselé

3 ajoutez 1/4 cube(s) de bouillon de volaille.

4 ajoutez sel et poivre à votre goût

5 mélangez bien au fouet.

3 Dans une poêle :

1 faites chauffer l'huile d'olive

2 faites revenir les oignons hachés pendant 2 min, en remuant

4 Versez dans la poêle les oeufs et le fromage

Paramètres de sous-programmes

⇒ Paramètre n

Omelette pour n personne(s)

- 1 Pelez et hachez finement $n/4$ oignon(s)
- 2 Dans un bol :
 - 1 battez $2 \times n$ oeufs entiers
 - 2 ajoutez n branche(s) de basilic ciselé
 - 3 ajoutez $n/4$ cube(s) de bouillon de volaille.
 - 4 ajoutez sel et poivre à votre goût
 - 5 mélangez bien au fouet.
- 3 Dans une poêle :
 - 1 faites chauffer l'huile d'olive
 - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage

Omelette pour 3

Omelette pour 3 personne(s)

- 1 Pelez et hachez finement 3/4 oignon(s)
- 2 Dans un bol :
 - 1 battez 2×3 oeufs entiers
 - 2 ajoutez 3 branche(s) de basilic ciselé
 - 3 ajoutez 3/4 cube(s) de bouillon de volaille.
 - 4 ajoutez sel et poivre à votre goût
 - 5 mélangez bien au fouet.
- 3 Dans une poêle :
 - 1 faites chauffer l'huile d'olive
 - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage

Omelette pour 4

Omelette pour 4 personne(s)

- 1 Pelez et hachez finement 4/4 oignon(s)
- 2 Dans un bol :
 - 1 battez 2×4 oeufs entiers
 - 2 ajoutez 4 branche(s) de basilic ciselé
 - 3 ajoutez 4/4 cube(s) de bouillon de volaille.
 - 4 ajoutez sel et poivre à votre goût
 - 5 mélangez bien au fouet.
- 3 Dans une poêle :
 - 1 faites chauffer l'huile d'olive
 - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage

Paramètres de sous-programmes

Omelette pour 4 ← paramètre réel

Omelette pour 4 personne(s)

- 1 Pelez et hachez finement 4/4 oignon(s)
- 2 Dans un bol :
 - 1 battez 2/4 oeufs entiers
 - 2 ajoutez 4/4 branche(s) de basilic ciselé
 - 3 ajoutez 4/4 cube(s) de bouillon de volaille.
 - 4 ajoutez sel et poivre à votre goût
 - 5 mélangez bien au fouet.
- 3 Dans une poêle :
 - 1 faites chauffer l'huile d'olive
 - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage

Exemples d'appels de méthodes

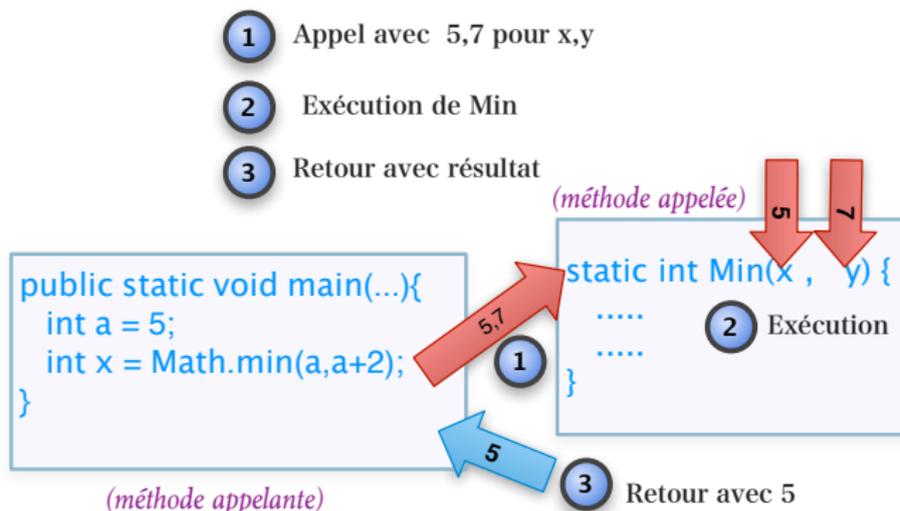
```
Terminal.ecrireStringln("Bonjour")
```

- son argument réel : chaîne "Bonjour",
- son comportement : afficher `Bonjour` à l'écran
- c'est une procédure.

```
Terminal.lireDouble()
```

- arguments : aucun
- comportement : retourner (vers le code appelant) la prochaine valeur de type `double` entrée au clavier.
- c'est une fonction.

Exemple d'un appel



En mémoire, au retour de l'appel : $x \leftarrow 5$

Appels imbriqués

Appel dont un ou plusieurs arguments correspondent à d'autres appels de méthodes.

```
int x = 4;  
int y = 10;  
Terminal.ecrireInt(Math.min(x, y));
```

L'appel à `Terminal.ecrireInt` a pour argument un appel à `Math.min` (méthode qui retourne le plus petit parmi ses arguments).

Comprendre les appels imbriqués

Règle : Avant d'exécuter un appel, on doit calculer la valeur de tous ses arguments.

```
int x = 4;
int y = 10;
Terminal.ecrireInt(Math.min(x, y));
```

Conclusion : dans un appel imbriqué, on commence par exécuter l'appel le plus interne.

Comprendre les appels imbriqués

```
int x = 4;  
int y = 10;  
Terminal.ecrireInt (Math.min(x, y) );
```

Exécution :

- 1 Calcul des arguments de `Math.min` : on récupère leurs valeurs en mémoire \Rightarrow 4,5
- 2 Appel à `Math.min(4, 5)` \Rightarrow retourne 4
- 3 4 est l'argument pour l'appel à `Terminal.ecrireInt`
- 4 Appel à `Terminal.ecrireInt(4)` \Rightarrow affichage de 4

Écrire les appels des méthodes

Si la méthode ...

- retourne un résultat \Rightarrow celui-ci doit être utilisé, et l'appel doit être une expression ;
- ne retourne pas de résultat \Rightarrow l'appel est une instruction

```
// On utilise le résultat  
x = Terminal.lireDouble();  
// L'appel est une instruction  
Terminal.ecrireInt(6);  
// Un appel de fonction et de procédure  
Terminal.ecrireInt(Math.min(2, 7));
```

Exemples d'écriture d'appels

Quelles sont les lignes erronées ?

Qu'affichent les lignes correctes ?

```
int x; int y = 10;
x = Terminal.lireInt();
Terminal.lireInt();
Terminal.ecrireInt(Terminal.lireInt());
Terminal.ecrireInt(Terminal.lireInt() + 4);
x = Math.min(Terminal.lireInt(), y);
Math.min(Terminal.lireInt(), y);
Terminal.ecrireInt(Math.min(Terminal.lireInt(), y));
y = Terminal.ecrireInt(5);
Terminal.ecrireInt(5);
```

- Les méthodes que l'on peut appeler dans un programme sont toujours **définies dans une classe**.
- Certaines méthodes sont **statiques** : on les invoque avec la syntaxe

```
<classe-de-définition>.<méthode> (<arguments>);
```

- ▶ on fait précéder le nom de la méthode (p.e : lireDouble),
- ▶ par le nom de la classe où elle est définie (p.e : Terminal) suivie d'un «.»
- ▶ et on donne les arguments entre parenthèses.

Exemple : `Terminal.ecrireInt (5);`

Méthodes statiques et non statiques

- Les méthodes statiques sont déclarées avec le mot clef **static**. Ex : `main` est une méthode statique.
- elles sont toujours appelées avec le nom de leur classe. Ex : méthode `ecrireInt` de la classe `Terminal` : `Terminal.ecrireInt(5)`,
- Les méthodes non statiques sont appelées sur des objets.
- Dans `nfa031`, nous ne verrons que les méthodes statiques.

6. Entrées/sorties standard (clavier/écran).

Les actions d'entrées/sorties

C'est quoi ? Méthodes pour **acquérir** ou **enregistrer** des données à partir/sur des dispositifs périphériques : clavier, écran, disque, etc.

entrées (**lecture**) : on *acquiert* des données. **Ces méthodes retournent la valeur lue.**

sorties (**écriture**) : on *enregistre* des données. **Ces méthodes ne retournent aucune de valeur.**

dispositif dans ce cours nous lirons au clavier et écrirons « à l'écran »

Méthodes d'entrées/sorties en Java

`System` : classe prédéfinie dans la bibliothèque Java avec méthodes d'entrées/sorties.

`Terminal` : classe écrite par les enseignants, avec méthodes d'entrée/sortie sur le clavier/écran pour tous les types primitifs utilisés dans ce cours.

Le fichier `Terminal.java`, doit se trouver dans le même répertoire que vos programmes.

Affichage avec System et Terminal

- `System.out.print(v)` : affiche la valeur primitive ou chaîne `v`.

```
System.out.print("bonjour");  
System.out.print(5 + 2);
```

- `Terminal.ecrireType(v)` (ou `ecrireTypeLn(v)`) où `Type` est `int`, `double`, `boolean`, `char` ou `String`. `v` doit être du type `Type`

```
Terminal.ecrireInt(5);  
Terminal.ecrireChar('a');  
Terminal.ecrireIntLn(5);
```

Lecture avec Terminal

```
Terminal.lireType()
```

où `Type` est `int`, `double`, `boolean`, `char` ou `String`.

- La lecture se fait après un retour chariot.
- Si la valeur saisie n'est pas du type `Type`, produit une **erreur à l'exécution**, (`TerminalException`) qui arrête l'exécution du programme.
- Si la valeur lue est du bon type, **retourne cette valeur en résultat**. L'appelant doit récupérer ce résultat.

Exemples de lecture avec Terminal

```
int x;  
double y;  
char c;  
x = Terminal.lireInt() + 4;  
y = Terminal.lireDouble();  
c = Terminal.lireChar();  
Terminal.ecrireInt(Terminal.lireInt());
```

Conversions entre types.

Expressions qui mélangent des types : autorisées (dans certains cas)

Java autorise certaines expressions ou affectations qui *mélangent des valeurs et variables de types différents*.

- *focus sur types prédéfinis* : primitifs + `String`;

Exemples valides :

- (expression) `3 + 4.3 = 7.8` (`int + double`) \Rightarrow `double`
- (expression) `3 + "abc" = "3abc"`, (`int + String`) \Rightarrow `String`
- (affectation) `double z = 5;` (une variable de type `double` reçoit une valeur de type `int`).

Opérateurs et types des opérandes (types prédéfinis)

- $-$, $*$, $\%$ \Rightarrow applicables sur n'importe quel type primitif numérique.
- $+$ \Rightarrow applicable sur types numériques *et aussi* sur `String`.
- Selon le type des opérandes, l'opération réellement exécutée diffère.
 - ▶ $(+)$: on exécute une concaténation entre chaînes, ou une addition entre nombres ;
 - ▶ $(*)$: on exécute une multiplication différente selon que les nombres multipliés ont ou non une partie décimale.
- En coulisses, les opérations réellement exécutées sont applicables **uniquement** sur des opérandes de même type.

Nécessaire

Mettre toutes les opérandes au même format avant d'effectuer les opérations.

Qu'est-ce qu'une conversion entre types ?

Conversion entre types

Changement du type d'une valeur, et en particulier de sa représentation en mémoire. Il y a deux sortes de conversions :

- **Implicite** : c'est le compilateur qui décide, sans que ce soit demandé par le programmeur. Uniquement dans certains cas.
 - ▶ `double m = 3;` Le type (int) de 3, et son format (32 bits) est changé en type double et format 64 bits (nombre à virgule) ;
 - ▶ le changement est opéré à l'exécution, *avant* d'effectuer l'affectation en mémoire.
- **Explicite** : c'est le programmeur qui le demande, par une syntaxe dédiée.

Attention : on ne modifie jamais *le type des variables*. Les conversion s'effectuent **toujours sur des valeurs**.

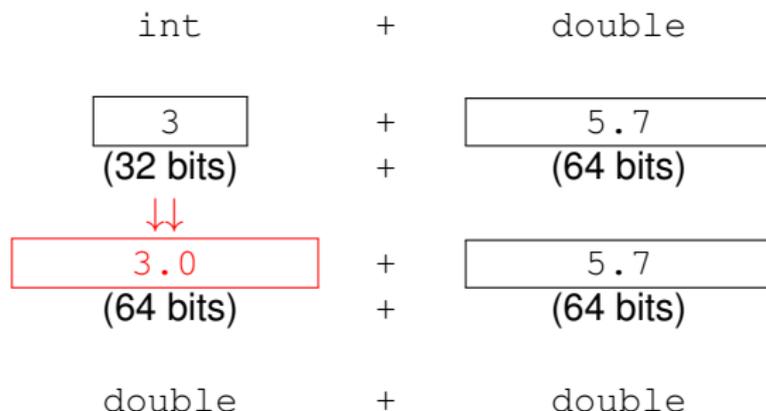
Conversions implicites entre types (primitifs)

- si opérandes de même type :
 - ▶ pas besoin de conversion ;
 - ▶ le résultat est du type commun ;
 - ▶ ex : $3+4 = 7$, $(\text{int} + \text{int}) \Rightarrow \text{int}$

- si opérandes des types différents :
 - ▶ Le compilateur réalise une conversion implicite (automatique) des opérandes de manière à les rendre uniformes.
 - ▶ vers le type le «plus grand» si expression arithmétique ;
 - ▶ ou vers `String`, si concaténation avec `String`
 - ▶ si la conversion est impossible : erreur à la compilation.

Conversions implicites entre int et double

Le compilateur effectue une **conversion implicite** vers un type commun (sans perte d'information).



Conversions implicites entre int et double

- implicite = automatique : le compilateur la réalise sans prévenir !
- seules les conversions sans perte d'information se font automatiquement :
 - ▶ si nécessaire, **réalisée** dans le sens : `int` → `double`.
 - ▶ **jamais réalisée** dans le sens : `double` → `int`. On perdait la partie décimale.

Exemple :

3.5 (`double`) ne peut pas être convertit **implicitement** vers un `int`.
On perdrait la partie décimale.

Conversions implicites entre String et primitifs

- réalisée **uniquement** si *concaténation avec opérande String* ;
- le résultat est un String.

Exemples :

- `6+2+"abc" = "62abc"` , `(int + int + String) ⇒ String`
- `(6+2) + "abc" = "8abc"` ,
`((int + int) + String) = int + String ⇒ String`

Affectations (int, double) avec conversions

(type variable)	=	(type valeur)	conversion nécessaire ?
int	=	int	non
double	=	double	non
double	=	int	implicite : int → double
int	=	double	explicite (cast) sinon erreur

Exemple :

```
double m = 6; // conversion int → double  
int x = m; // erreur
```

Division entière

La division de deux entiers donne en résultat un entier

```
5/2  donne en résultat  2  
      et non pas  2.5
```

Pourquoi ?

Dans :

```
double m = 5/2;
```

quelle est la valeur de m ?

Conversions explicites (Cast)

La conversion entre types avec perte d'information est autorisée, lorsqu'elle est réalisée explicitement par le programmeur.

Par exemple, pour convertir `3.67` vers un `int`, il suffit de d'écrire :

```
(int) 3.67 ⇒ 3
```

ce qui donne la valeur `3` de type `int`. Une conversion explicite, appelée cast en Java, prend la forme `(type_cible) v`, où `type_cible` est le type vers lequel on souhaite faire la conversion.

Attention : cette conversion n'est possible que si elle a un sens vis-à-vis des types. Ainsi, par exemple, la conversion

`(boolean) 5` est invalide car `boolean` n'est pas un type numérique.

Conversions explicites pour `char`

Conversions explicites entre types numériques et `char` :

- `(char) 97 ⇒ 'a'`
- `(int) '+' ⇒ 43`

Règles de style en Java

- Java n'impose pas des règles de formatage des programmes.
- On peut écrire tout un programme sur une seule ligne : c'est un entrée valide pour le compilateur. Mais ce n'est pas très lisible pour les humains.
- Certaines conventions d'écriture sont devenues des «standards» : elles visent à améliorer la lisibilité des programmes par les programmeurs eux-mêmes.

Règles de style (2)

- Les noms des classes débutent par une majuscule (`Terminal`, `Conversion`).
- Les noms des variables et méthodes débutent par une minuscule (`main`, `x`, `euros`, `lireInt`).
- Les noms composés se font par adjonction de plusieurs mots, chaque mot débutant par une majuscule (`CompteBancaire`, `lireInt`).
- Chaque variable initialisée est déclarée (toute seule) sur une ligne.

Règles de style (3)

- Les variable essentielles au programme sont déclarées en début de la méthode main. Les variables auxiliaires, juste au moment où elles sont nécessaires.
- Chaque nouvelle structure est décalée de 2 ou 3 caractères à droite par rapport à la structure qui la contient. On parle **d'indentation**.

```
public class Conversion {  
    public static void main (String[] args) {
```

Règles de style (4)

- Les instructions sont indentées à droite par rapport au bloc qui les contient.

```
public static void main (String[] args) {  
    double euros;  
    double francs;  
    Terminal.ecrireStringln("Somme_en_euros?_");  
    euros = Terminal.lireDouble();  
    ...  
}
```

- Chaque instruction est écrite sur une ligne. Toutes les instructions d'un bloc sont alignées sur la même colonne.

Règles de style (5)

- L'accolade fermante d'un bloc est alignée sur la même colonne que le début de la structure qu'elle délimite.

```
public class Conversion {
    public static void main (String[] args) {
        ....
        francs = euros * 6.559;
        Terminal.ecrireStringln("Conversion=_" + francs);
    }
}
```