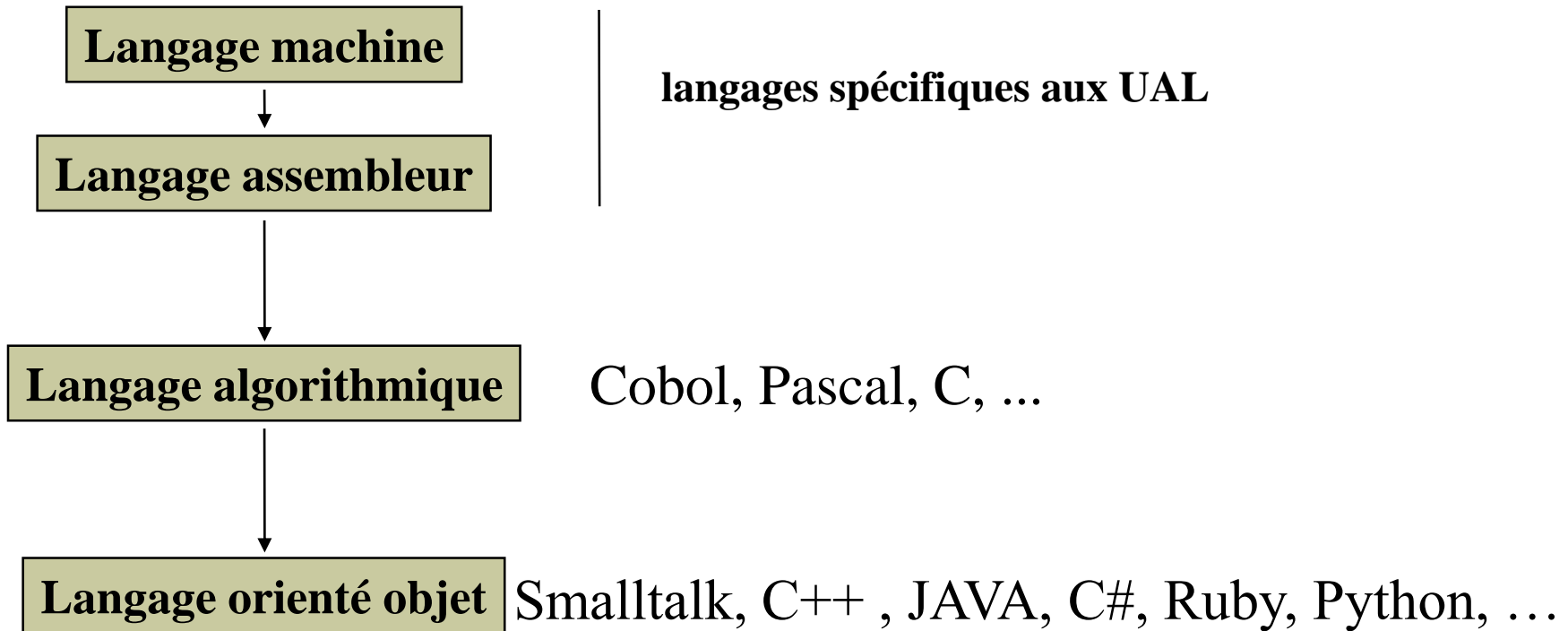

2015-2016

1. Algorithmique avec Java

Différentes générations de langages de programmation



OBTENIR DES PROGRAMMES
FIABLES
PORTABLES
POUVANT EVOLUER
REUTILISABLES

JAVA

- un bon langage orienté objet
- influencé par C++ en évitant ses défauts
- gestion automatisée de la mémoire (garbage collector)
- sérialisation
- gestion d'exceptions

- une librairie standard importante
graphique, réseau, base de données, XML ...

- une portabilité excellente

- un langage très utilisé

Yannick.Parchemal@parisdescartes.fr

Interprétation et compilation

Interprétation

Compilation

**Interprétation
et compilation**

Programme écrit en X



INTERPRETEUR de X

Programme écrit en X



**COMPILATEUR de X
en langage machine**



**Programme
en langage machine**

Programme écrit en X



**COMPILATEUR de X
en Y
(plus proche
du langage machine)**

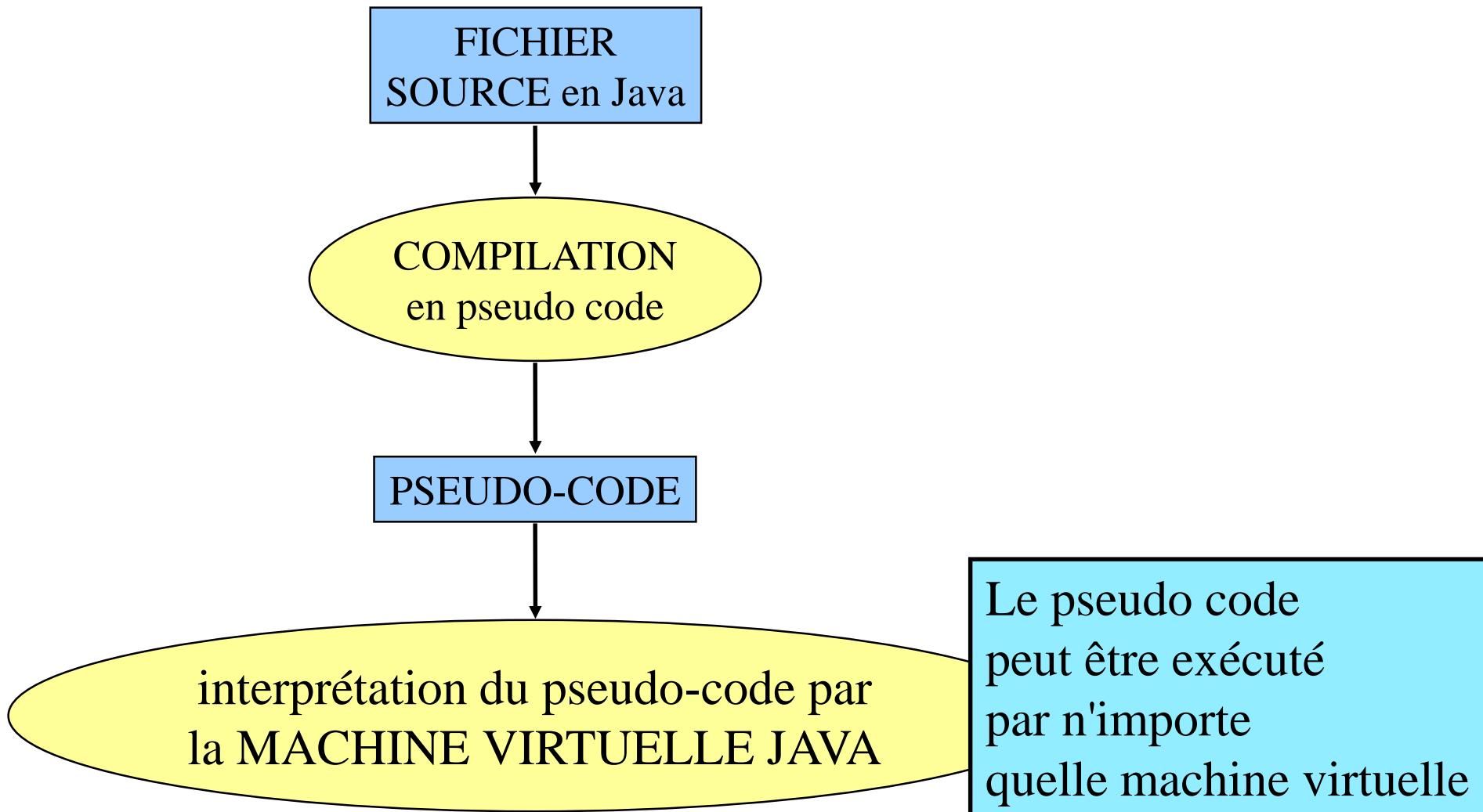


**Programme
en langage Y**



**INTERPRETEUR
de Y**

La machine virtuelle JAVA



JSE : Java Platform Standard Edition

version actuelle : JSE 8

JRE : Java Runtime Environment

machine virtuelle + bibliothèques de base

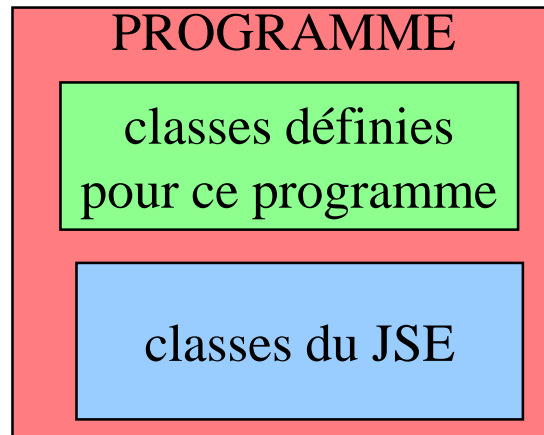
JDK : Java Development Kit

JRE + compilateur

Programme : un ensemble de classes

Un programme JAVA est constitué d'un certain nombre de classes

- de classes **définies** par l'utilisateur
- des classes du JSE



Qu'est-ce qu'une classe ?

Pour une toute première approche, on peut dire que :
classe : type de données + fonctions pour le manipuler

Distribuer un programme JAVA

Il existe des machines virtuelles pour (presque) tous les ordinateurs

La machine virtuelle est toujours accompagnée des bibliothèques du JSE

Les bibliothèques de base du JSE étant présentes sur toutes les machines, il suffit de communiquer à un ordinateur les classes spécifiques à un programme pour que ce programme tourne sur cet ordinateur.

Les classes spécifiques d'un programme sont packagées dans un fichier JAR (Java ARchive)

Premières applications autonomes

```
package up5.mi.pary.jt.hello;  
// un premier programme  
/* la version JAVA du classique  
   Hello World  
*/
```

// Ceci est un commentaire finissant en fin de ligne

/* ceci est un commentaires pouvant encadrer
un nombre quelconques de caractères
sur un nombre quelconque de lignes */

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

Hello World !

L'instruction d'affichage

```
package up5.mi.pary.jt.hello;  
// un premier programme  
/* la version JAVA du classique  
   Hello World  
*/  
  
public class HelloWorld {  
    public static void main(String [ ] args) {  
        System.out.println("Hello World !"); }  
    }  
}
```

System.out.println permet l'impression sur le flux de sortie standard
"Hello World!" : la chaîne de caractères à afficher

Hello World : la fonction "main"

```
package up5.mi.pary.jt.hello;
```

```
public class HelloWorld {
```

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

```
        System.out.println("Hello World !");
```

```
    }
```

```
}
```

l'en-tête de la fonction

l'unique instruction

Nous verrons ceci bientôt, mais sachez dès à présent que :

void : la fonction ne retourne pas de résultat

String : une classe de l'API standard pour les chaînes de caractères

String [] **args** : désigne un tableau de **String** de nom **args**

qui correspond aux arguments fournis lors de l'appel du programme

Hello World : en-tête de la classe

```
package up5.mi.pary.jt.hello;
```

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


en-tête de la classe

définition de la classe
(ici, une seule fonction,
la fonction "main")

L'exécution de ce programme consiste en l'exécution
de la fonction "main" de la classe HelloWorld.
La classe HelloWorld est appelée classe principale du programme.

Le fichier où est définie la classe doit porter le même nom que la classe,
ici HelloWorld.java

Le packaging de la classe HelloWorld



```
package up5.mi.pary.jt.hello;  
public class HelloWorld {  
    public static void main(String [] tArg) {  
        System.out.println("Hello World !");  
    }  
}
```

Les **paquetages** sont des ensembles de classe.

La classe définie ici appartient au **paquetage** nommé **up5.mi.pary.jt.hello**

Le nom simple de la classe est **HelloWorld**.

Le nom complet de la classe est **up5.mi.pary.jt.hello.HelloWorld**.

Le packaging auquel appartient une classe est précisé en début de fichier par l'instruction **package**

AFFICHAGE D'INFORMATIONS

System.out.print

System.out.println

**Permet, au cours de l'exécution du programme,
d'afficher des informations sur la sortie standard**

System.out.print** (val)**

**affiche à l'écran la valeur de val
sans passer à la ligne**

System.out.println** (val)**

**affiche à l'écran la valeur de val
puis passe à la ligne**

exemples

```
System.out.println("Fevrier");  
System.out.println (2016-1);  
System.out.print(2);  
System.out.print(" fevrier ");  
System.out.println(2016+1);  
System.out.println("11H");  
System.out.println (2+" fevrier");
```



```
Fevrier  
2015  
2 fevrier 2017  
11H  
2 fevrier
```

**Pour afficher plusieurs informations avec une seule instruction,
on utilise la concaténation de chaînes**

L'opérateur +

Si les deux opérandes sont des nombres:
c'est une somme de nombres

$$5.3 + 4$$

$$8 + 2.1 + 4 \text{ équivalent à } (8+2.1)+4$$

9.3

14.1

Si l'une des deux opérandes est une String:
c'est une concaténation de chaînes

"a"+"bc"

"a"+3

"a" + 8 + 2.1 + 4 équivalent à (("a"+8)+2.1)+4

""+8+2.1+4 équivalent à ((""+8)+2.1)+4

8+"a"

1+1+" fevrier"

abc

a3

a82.14

82.14

8a

2 fevrier

LES VARIABLES

```
package up5.mi.pary.jt.hello;  
public class TestCarreDeLaSomme {  
    public static void main(String [] args) {  
        int a; // déclaration de la variable entière a  
        a = 5 ; // initialisation de la variable a  
  
        int b = 7 ; // déclaration et initialisation de b  
  
        System.out.print("Le carré de la somme de " + a + " et de " + b);  
        System.out.println( "est égal à " + (a+b) * (a+b) );  
  
    }  
}
```

le carré de la somme de 5 et de 7 est égal à 144

La déclaration des variables est obligatoire

L'opérateur d'affectation est =

L'initialisation peut se faire en même temps que la déclaration

Nombres, caractères et booléens en Java

Les 4 types d'entiers : byte short int long

Exemple de constantes entières : 123 -12 0 30000

Opérateurs sur les entiers : + - *

/ (quotient de la division entière)

% (modulo = reste de la division entière)

Les 2 types de flottant : float double

Exemple de constantes flottantes : 3.14159 -13.3 10E-12 -12.0

Opérateurs sur les flottants : + - * /

Le type pour les caractères : char

Exemple de constantes caractères : 'a' 'Z' '\n' (passage à la ligne)

Le type pour les booléens : boolean

2 constantes booléennes : true et false

Opérateurs sur les booléens : && (ET) || (OU) ! (NON)

noms de paquets

**Les paquets dont le nom commence par
"java."
sont réservés à Oracle**

**Le concepteur de ce cours
utilise des paquets
dont le nom
commence par "up5.mi.pary."**

**Le nom de paquetage doit être choisi de telle manière
qu'il identifie sans ambiguïté
la personne ou la société.**

Quelques paquetages du Java Development Kit (JDK)

Le **J**ava **D**evelopment **K**it (**JDK**) désigne un ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé....

Wikipedia

Nom du paquetage	rôle des classes du paquetage
java.lang	les classes de base
java.io	les entrées sorties
java.util	les utilitaires
java.net	communication réseau
javafx.application	application javafx
javafx.event	les événements

Paquetage anonyme (utilisation déconseillée)

Pour
information

L'instruction package est facultative
Toutes les classes définies dans un même répertoire
sans indication explicite de paquetage
appartiennent à un même paquetage anonyme.
Le nom complet d'une telle classe est son nom "simple".

L'utilisation de paquetages anonymes est **déconseillée**
et n'est pas utilisé dans le cadre de ce cours

Organisation des fichiers et commandes de base

Les commandes de base :

Compilation : javac

Exécution : java

Documentation : javadoc

Organisation des fichiers

Il y a 3 types de fichiers.

Les fichiers
source
écrits par le
programmeur

.java

Ils peuvent être écrits à l'aide
d'un éditeur texte quelconque

Les fichiers
de pseudo-code
générés par le
compilateur **javac**

.class

Ce sont les seuls fichiers nécessaires pour
l'exécution. L'exécution des applications
autonomes est effectué avec la commande **java**.
Les fichiers **.class** peuvent être compressés dans
un fichier **jar**

Les fichiers
de documentation
générés par l'utilitaire
javadoc

.html

Ils sont visualisables par un navigateur.
L'utilitaire **javadoc** utilise les commentaires
javadoc présents dans le fichier source

Les commandes java, javac, javadoc

**fichiers
sources
.java**

Ces trois commandes sont fournies avec le JDK

javadoc

javac

**fichiers de pseudo-
codes
.class**

**fichiers de
documentation
.html**

java

**exécution de
l'application
(main)**

les paquets up5.mi.pary....

Les paquets up5.mi.pary.... sont les paquets contenant tous les exemples du cours.

Les classes de ce paquetage peuvent être téléchargées sur le site. Elles sont regroupées dans un fichier compressé coursjava.jar qui contient les fichiers .class nécessaires à l'exécution d'un programme java.

Les fichiers jar (Java ARchive) sont des fichiers compressés (comme zip) utilisés par java .

ATTENTION : nous supposons pour la suite de cet exposé que vous avez mis le fichier coursjava.jar à l'adresse D:\Dupond\coursjava.jar

Exécution des programmes de cet enseignement avec la commande Java

SYNTAXE:

```
java -classpath <adresse des classes nécessaires>  
<nom complet de la classe principale>
```

l'option classpath permet de préciser
où trouver les classes nécessaires à l'exécution du programme.

Soit à exécuter le programme de classe principale **up5.mi.pary.jt.hello.HelloWorld**.
Le fichier up5\mi\pary\jt\hello\HelloWorld.class qui contient le code de cette classe est
inclus dans le fichier jar situé à **D:\Dupond\coursjava.jar**.

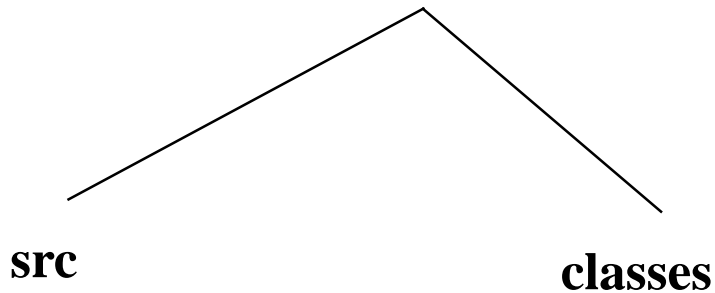
```
java -classpath D:\Dupond\coursjava.jar up5.mi.pary.jt.hello.HelloWorld
```

Hello World !

**ATTENTION : remplacer D:\Dupond\coursjava.jar
par l'emplacement où vous avez mis coursjava.jar.**

Le répertoire du projet

<répertoire du projet>



c'est le répertoire de base des classes pendant le développement

ATTENTION : nous supposons pour la suite de cet exposé que le nom du répertoire de projet est
D:\Dupond\projet1

(sous Linux changer les \ par des /)

Les répertoires src et classes ont la même structure de sous-répertoires.
Le répertoire src contient des fichiers .java
Le répertoire classes contient des fichiers .class générés par javac

Organisation des fichiers .class

**Le nom du paquetage détermine le sous-répertoire
dans lequel sont mis les fichiers**

**exemple: les fichiers correspondant aux classes du paquetage
in400.pierre se trouve à l'adresse relative in400/pierre.**

**Cette organisation est
obligatoire
pour les fichiers .class**

D:\Dupond\projet1\classes

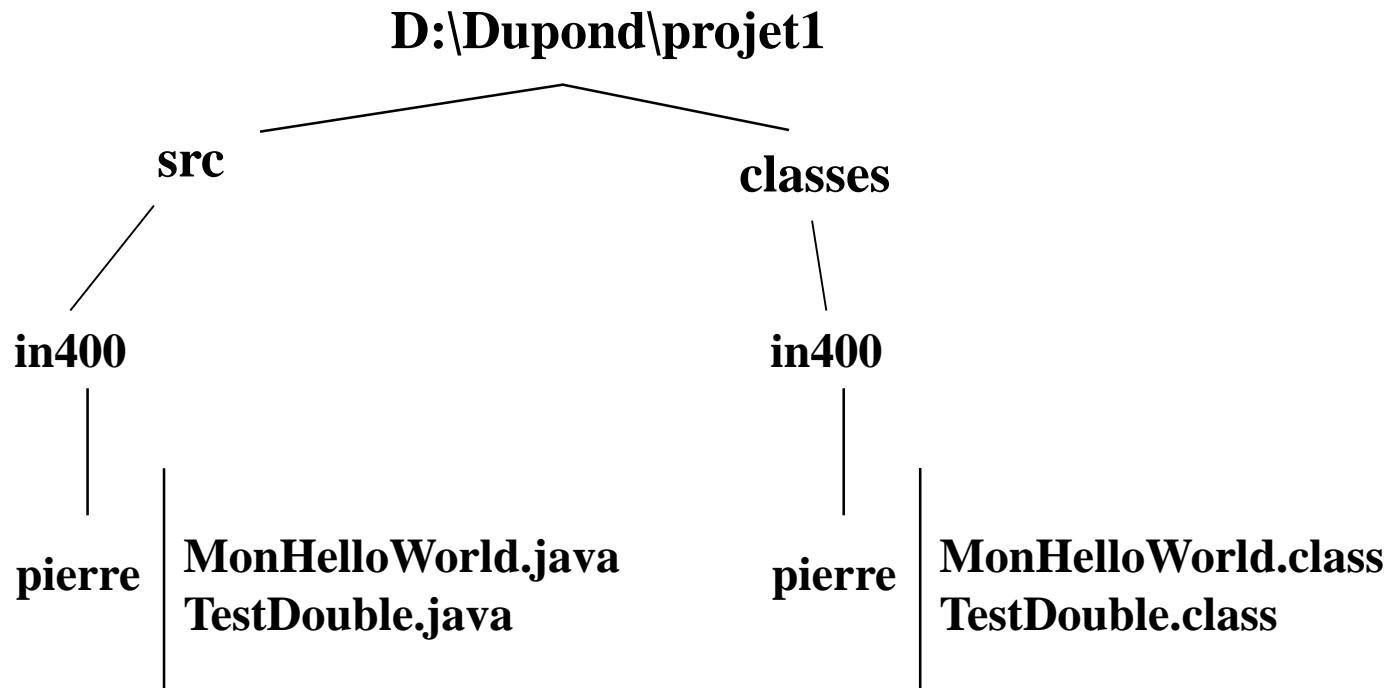
in400

pierre

**MonHelloWorld.class
TestDouble.class**

MonHelloWorld et TestDouble sont deux classes du paquetage in400.pierre

Organisation des fichiers

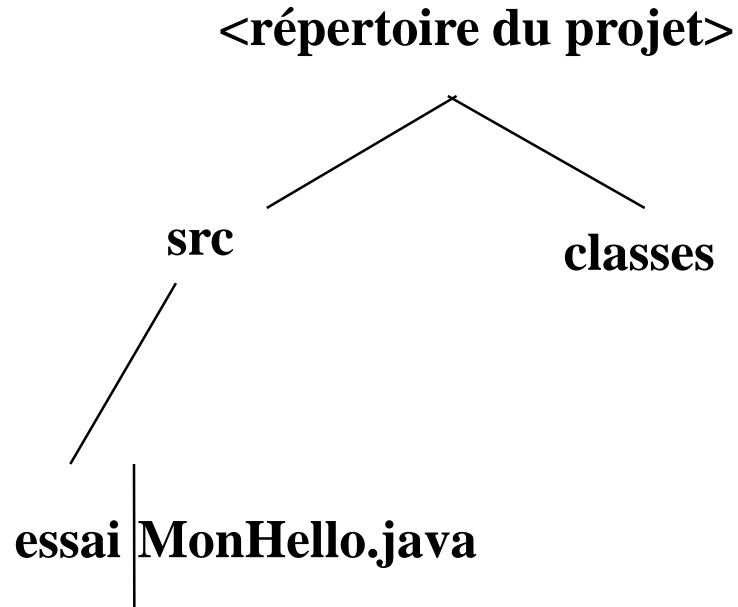


Exemple : la classe `essai.MonHello`

```
package essai;
```

```
public class MonHello{
```

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



1. choisir votre répertoire de projet
2. créer les sous répertoires src et classes
3. créer le répertoire essai dans src
4. Ecrire avec un éditeur de texte le fichier `MonHello.java`

attention 1 faire la sauvegarde en mode texte (et pas RTF ou doc ...))

attention 2 certains éditeurs ajoutent parfois un suffixe .txt
il faut alors le retirer en renommant le fichier.

Exemple 1: compilation et exécution de `essai.MonHello`

SYNTAXE de la commande `javac`(compilation) :

`javac` **`-d`** <répertoire où doivent être mis les fichiers générés>
`-classpath` <adresse des classes déjà compilées nécessaires à la compilation>
<adr. du(des) fichiers à compiler>

l'option `d` permet de préciser le répertoire de base des fichiers `.class` générés par la compilation

Pour ce premier exemple, aucune classe n'est nécessaire lors de la compilation, l'option `classpath` est donc absente.

`javac -d D:\Dupond\projet1\classes D:\Dupond\projet1\src\essai\MonHello.java`

SYNTAXE de la commande `java`(exécution) :

`java` **`-classpath`** <adresse des classes utilisées lors de l'exécution>
<nom complet de la classe principale>

// exécution en précisant le répertoire de base des classes

`java -classpath D:\Dupond\projet1\classes essai.MonHello`

Tout savoir sur l'option classpath

l'option classpath permet de préciser où trouver les classes

- utilisées lors de l'exécution du programme pour la commande java**
- nécessaires à la compilation du programme pour la commande javac**

A cette option peut correspondre une ou plusieurs valeurs, chacune d'elle pouvant être :

- l'adresse (relative ou absolue) d'un fichier jar.**
- l'adresse (relative ou absolue) d'un répertoire de base de classes**

Remarque : les classes de l'A.P.I. ne sont pas concernées par cette option

Si plusieurs valeurs sont associées à une option classpath, elles doivent être séparées par des ; (sous windows) ou des : (sous linux).

La valeur par défaut de cette option est le répertoire courant (désigné par un ".")

ATTENTION : le répertoire de base d'une classe est le répertoire contenant le répertoire racine du paquetage.

Exemple : le répertoire de base de la classe up5.mi.pary.jh.hello.HelloWorld est le répertoire contenant le dossier up5 (qui lui même contient le dossier mi etc...)

Complément : compilation simultanée de plusieurs fichiers

Compilation de deux fichiers:

```
javac -d D:\Dupond\javadev\classes  
-classpath D:\Dupond\coursjava.jar  
D:\Dupond\projet1\src\essai\TestTerminal.java  
D:\Dupond\projet1\src\essai\MonHello.java
```

Compilation de tous les fichiers d'un répertoire:

```
javac -d D:\Dupond\javadev\classes  
-classpath D:\Dupond\coursjava.jar  
D:\Dupond\projet1\src\essai\*.java
```


PREMIERS PAS AVEC JAVA

les entrées-sorties avec la classe up5.mi.pary.term.Terminal

Java est un langage conçu pour des programmes avec interface graphique.

**La classe Terminal permet de réaliser des programmes
sans programmer d'interfaces graphiques.**

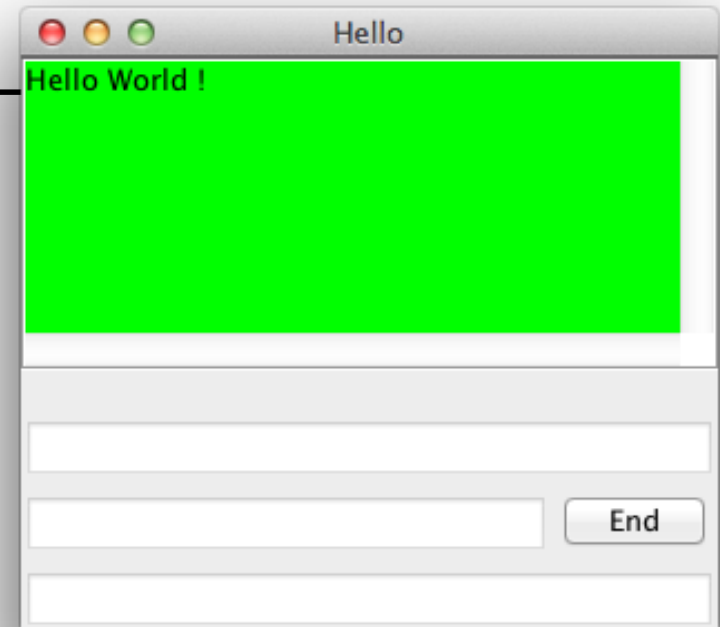
Yannick.Parchemal@parisdescartes.fr

Hello World avec la classe Terminal

```
package up5.mi.pary.jt.term;

import up5.mi.pary.term.Terminal;

public class HelloWorld {
    public static void main(String [] tArg) {
        Terminal term = new Terminal("Hello",300,200);
        term.println("Hello World !");
        term.end( );
    }
}
```



`new Terminal("Hello",400,400)` permet de créer un terminal de taille 400x400

La fonction **println** de Terminal s'utilise comme `System.out.println`

La fonction **end()** permet de signaler visuellement la fin de l'exécution du programme.

SAISIE D'INFORMATIONS au clavier

Plusieurs méthodes de la classe `Terminal` permettent de recevoir des informations de l'utilisateur du programme

EXEMPLE

```
Terminal term = new Terminal("calcul du carré de la somme",300,300);
```

```
int a = term.readInt("donner un entier:") ;
```

```
.....
```

autres méthodes
analogues à `readInt`:

```
readShort readByte readLong  
readFloat readDouble readChar readBoolean  
readString
```

exemple : calcul du carré de la somme de deux nombres

Le programme demande un entier **1**
puis un autre entier **2** puis affiche le
carré de la somme **3**

calcul du carré de la somme

donner un entier:

1

donner un entier:

5

OK

calcul du carré de la somme

donner un entier: 5

donner un autre entier:

2

donner un autre entier:

7

OK

calcul du carré de la somme

3

donner un entier: 5

donner un autre entier: 7

le carré de la somme de 5 et de 7 vaut 144

End

calcul du carré de la somme de deux nombres

```
package up5.mi.pary.jt.term;
import up5.mi.pary.term.Terminal;
public class TestTerminal1 {

    public static void main(String [] tArg) {

        Terminal term = new Terminal("calcul du carré de la somme",300,300);

        int a = term.readInt("donner un entier:");

        int b= term.readInt("donner un autre entier:");

        term.println("le carré de la somme de "+a+
                    " et de "+b+" vaut "+(a+b)*(a+b));

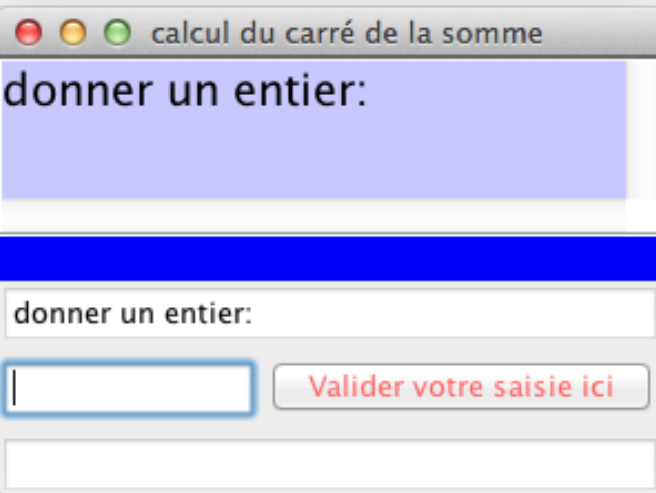
        term.end();
    }
}
```

donner un entier: 5

donner un autre entier: 7

le carré de la somme de 5 et de 7 vaut 144

paramétrage des terminaux



```
package up5.mi.pary.jt.term;  
import up5.mi.pary.term.Terminal;  
public class TestPersonnalisationTerminal {
```

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

```
        Terminal term = new Terminal("calcul du carré de la somme",300,300);
```

```
        term.setBackground(java.awt.Color.BLUE);  
term.setTextAreaColor(new java.awt.Color(200,200,255));  
term.setButtonTextColor(new java.awt.Color(255,100,100));  
term.setButtonLabel("Valider votre saisie ici");  
term.setTextAreaFontSize(20);
```

```
        int a = term.readInt("donner un entier:") ;
```

```
        int b= term.readInt("donner un autre entier:") ;
```

```
        term.println("le carré de la somme de "+a+  
                " et de "+b+" vaut "+(a+b)*(a+b));
```

```
        term.end();  
}  
}
```

la classe Terminal : gestion des erreurs de saisie

calcul du carré de la somme

donner un entier:

donner un entier:

A

Valider votre saisie ici

entier compris entre -2^{31} et $2^{31}-1$ attendu

En cas d'erreur de saisie, un message indique le résultat attendu et une nouvelle réponse est alors attendue

La documentation en ligne de la classe Terminal

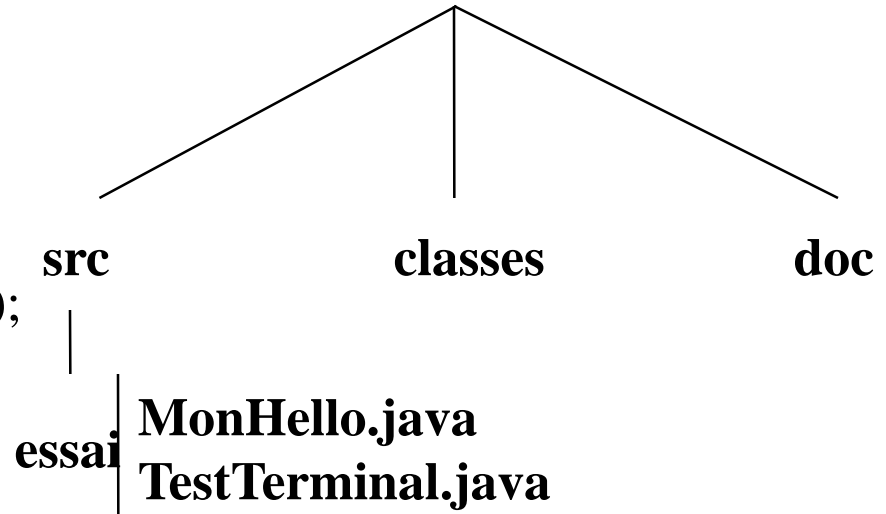
La documentation en ligne de la classe Terminal permet d'avoir connaissance des diverses fonctionnalités offertes afin de pouvoir les utiliser dans les programmes que nous écrivons.

Exemple 2: la classe `essai.Terminal`

Soit à compiler la classe `essai.Terminal` :

```
package essai;  
import up5.mi.pary.term.Terminal;  
public class Terminal{  
  
    public static void main(String[] args){  
        Terminal term = new Terminal  
            ("essai de Terminal",400,400);  
        term.println("Hello");  
    }  
}
```

<nom du répertoire de développement>



Ici, la classe `up5.mi.pary.term.Terminal` est nécessaire lors de la compilation et lors de l'exécution : le fichier jar (ou le répertoire de base des classes) la contenant doit donc être précisée avec l'option `classpath` pour les commandes `java` et `javac`.

Compilation de la classe `essai.TestTerminal`

SYNTAXE:

```
javac -d <répertoire où doivent être mis les fichiers générés>  
-classpath <adresses des classes (répertoires ou fichier jar)>  
<adr. du(des) fichiers à compiler>
```

```
javac -d D:\Dupond\projet1\classes  
-classpath D:\Dupond\coursjava.jar (contient up5.mi.pary.term.Terminal)  
D:\Dupond\projet1\src\essai\TestTerminal.java
```

// exécution

```
java -classpath D:\Dupond\coursjava.jar;D:\Dupond\projet1\classes  
essai.TestTerminal
```

Lorsqu'il y a plusieurs adresses à préciser pour une même option,
séparer les adresses
par des **;** (sous windows) ou **:** (sous linux))

Création et utilisation de fichiers jar

On peut aussi créer un fichier jar correspondant au répertoire
D:\Dupond\javadev\classes.

Pour compresser le répertoire courant

1. Placez vous dans le répertoire de base des classes

2. Exécutez la commande

jar cf <adr. du fichier jar à créer> . (n'oubliez pas le point)

Exemple :

cd D:\Dupond\projet1\classes

jar cf D:\Dupond\allDupond.jar .

et utiliser ensuite ce fichier jar dans les commandes java et javac:

```
java -classpath D:\Dupond\coursjava.jar;D:\Dupond\allDupond.jar  
essai.TestTerminal
```

Le fichier jar doit être recréé après chaque recompilation d'une classe.

rôle de import

```
package up5.mi.pary.jt.term;
import up5.mi.pary.term.Terminal;
public class HelloWorld {
    public static void main(String [] args) {
        Terminal term = new Terminal("Hello",400,400);
        term.println("Hello World !");
        term.end( );
    }
}
```

La déclaration

`import up5.mi.pary.term.Terminal`
permet de préciser le nom complet
de la classe Terminal

Ces deux définitions sont équivalentes

```
package up5.mi.pary.jt.term;
public class HelloWorld {
    public static void main(String [] args) {
        up5.mi.pary.term.Terminal term = new up5.mi.pary.term.Terminal("Hello",400,400);
        term.println("Hello World !");
        term.end( );
    }
}
```

**Sans import, il faut préciser à chaque fois
le nom complet de la classe**

IMPORT de toutes les classes d'un paquetage

// permet d'importer toutes les classes du paquetage **javafx.event**
import **javafx.event.***;

L'instruction **import** **<paquet>.*** permet d'importer toutes les classes d'un même paquetage.

Elle était utile dans le cas où de nombreuses classes devaient être importées d'un même paquetage.

Il est préférable de préciser le nom complet des classes importées : cela constitue un renseignement intéressant pour le lecteur de la classe

Les IDE (Eclipse, NetBeans ...) permettent une génération quasi automatisée des instructions **import**.

IMPORT

les déclarations **import**
permettent, dans le programme,
d'écrire le nom simple des classe (exemple : Terminal)
au lieu du nom complet (exemple : up5.mi.pary.term.Terminal)

C'est une facilité syntaxique
(il n'y a pas d'inclusion de texte comme avec l'instruction include en C).

Syntaxes :

import *nomCompletDeClasse*; // pour l'import d'une classe

import *nomDePaquetage.**; // pour l'import de toutes les classes d'un paquetage

Remarque :

impossible de dire par exemple :

import up5.*.*;

Seuls les deux formes mentionnées ci-dessus sont possibles !

IMPORT implicites

Sont importées implicitement :

- toutes les classes du paquetage java.lang
- toutes les classes du paquetage courant

```
package dupond.essai;  
// les deux imports suivants sont implicites  
//et il n'est donc pas utile de les mentionner
```

```
import java.lang.*; // java.lang est un package contenant des  
                    // classes très souvent utilisées (System Math String ...)
```

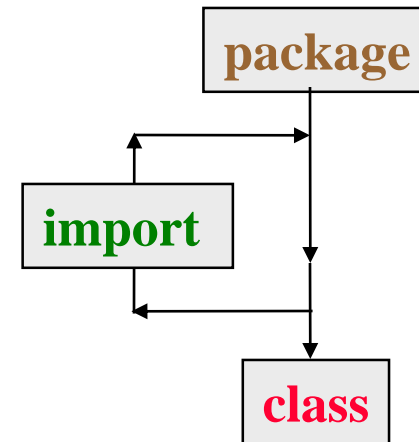
```
import dupond.essai.*; //les classes du paquetage courant sont aussi importées  
implicitement  
// (le paquetage courant est celui de la classe définie dans le fichier)  
public class ...
```

Structure d'un fichier .java

En premier l'instruction **package**,
Ensuite, éventuellement, les instructions **import** ,
Enfin la définition de la **classe**.

```
package dupond.essai;  
  
import up5.mi.pary.term.Terminal;  
  
import java.util.Date;  
  
public class Test {  
  
    ...  
  
}
```

Fichier Test.java



Programme Java

Un programme Java est un ensemble de classes
Certaines sont des classes de l'A.P.I. standard
D'autres sont définies par le programmeur

Une de ces classes est la classe principale du programme.
C'est elle qui est utilisée au lancement du programme.

La classe principale
possède une fonction particulière :
la fonction main qui est exécutée au lancement du programme.

Classes Java

Les classes Java peuvent être classées en deux catégories :

1. les "vraies" classes au sens de la Programmation Orientée Objet

grâce auxquelles on peut créer des objets et qui définissent des fonctionnalités pour manipuler ces objets

exemple :

la classe Terminal qui permet de créer et d'utiliser des terminaux

la classe String qui permet de créer et d'utiliser des chaînes de caractères

Nous commencerons à vraiment utiliser de telles classes au module suivant.

2. les classes (que nous appellerons classes d'utilitaires) dont la raison d'être est de regrouper des fonctions (appelées fonctions statiques)

similaires aux fonctions que l'on rencontre dans les langages non orienté objet et des constantes

exemples :

- la classe Math définit des fonctions comme abs, sqrt, min, max et des constantes comme PI

- la classe HelloWorld avec sa fonction main

Nous utilisons et définissons de telles classes dans ce module.

La classe Math : un exemple de classe d'utilitaires

```
package java.lang;
```

```
/** The class Math contains methods for performing basic numeric operations  
such as the elementary exponential, logarithm, square root, and trigonometric  
functions.
```

```
*/
```

```
public final class Math {
```

```
// Static fields
```

```
/** The double value that is closer than any other to e, the base of the natural  
logarithms */
```

```
public static final double E = 2.7182818284590452354;
```

```
/** Returns the absolute value of an int value.*/
```

```
public static int abs(int a){ return (a < 0) ? -a : a;}
```

```
/** Returns the arc cosine of an angle, in the range of 0.0 through pi.*/
```

```
public static double acos(double a){...}
```

```
/** Returns the square root of a double value.*/
```

```
public static double sqrt(double a){...}
```

```
...}
```

un autre exemple de classe d'utilitaires

```
package up5.mi.pary.jc.statique;
```

```
public class Exemple {
```

```
/** @return le carré de la somme de deux entiers*/
```

```
public static int carreDeLaSomme(int a,int b){
```

```
    int somme=a+b;
```

```
    int carre = somme * somme;
```

```
    return carre;
```

```
}
```

```
/** @return la factorielle d'un entier */
```

```
public static int fact(int n){
```

```
    int res;
```

```
    if (n == 0)
```

```
        res = 1;
```

```
    else res = n* fact(n-1);
```

```
    return res;
```

```
}
```

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

```
    System.out.println("Le carré de 5 et de 7 vaut "+carreDeLaSomme(5,7));
```

```
    System.out.println("fact(5)="+fact(5));
```

```
}}
```

En-tête de la fonction

**Bloc
d'instructions
définissant
la fonction**

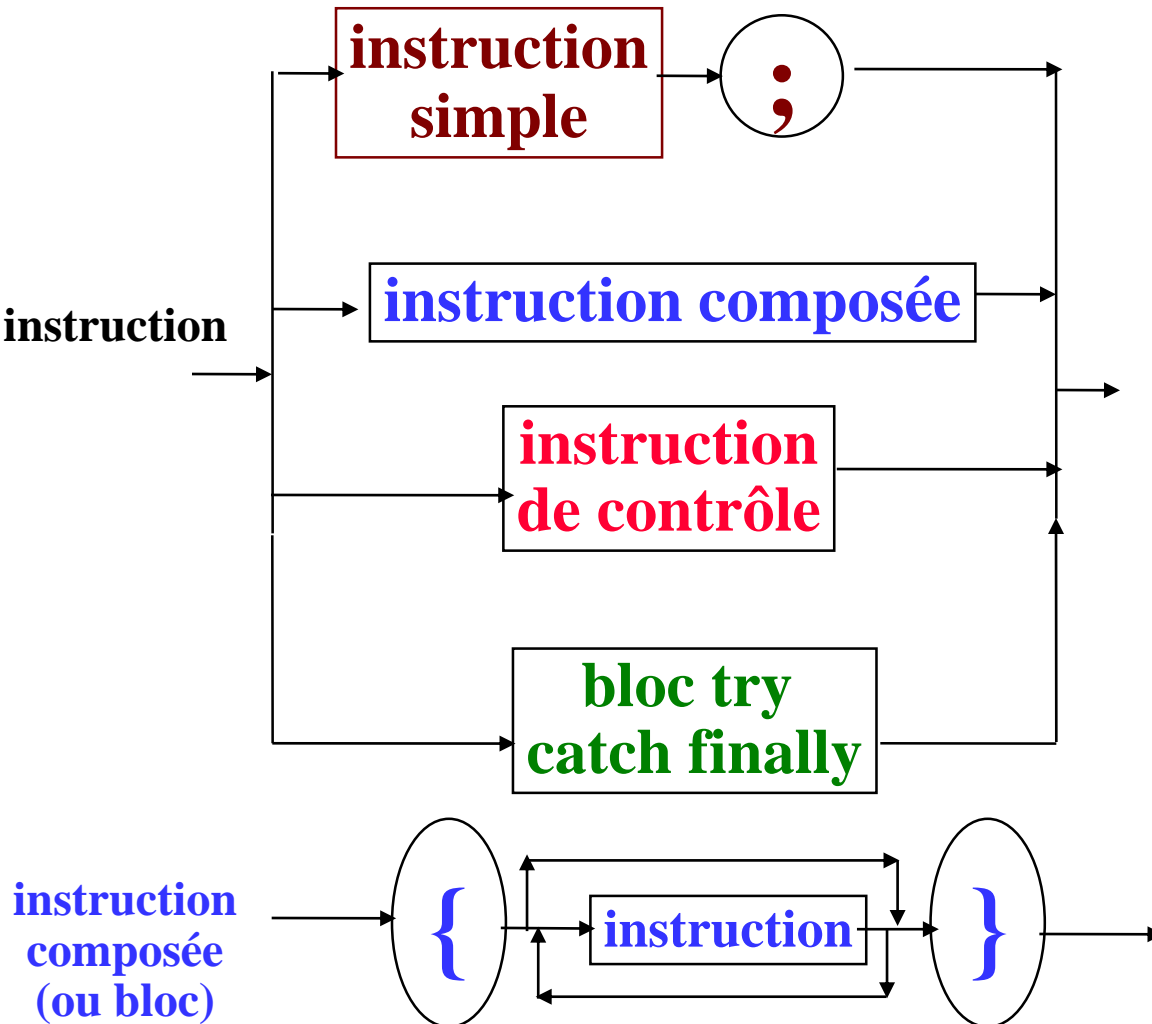
Les instructions

Une fonction est définie par une suite d'instructions

Une instruction peut-être:

- **une instruction simple :**
 - `int somme=a+b;`
 - `System.out.println(fact(5));`
- **une instruction composée**
 - `{int somme=a+b;int carre=somme*somme;}`
- **une instruction de contrôle**
 - **if** (`n == 0`)
 - `res = 1;`
 - else** `res = n* fact(n-1);`
- **une instruction de gestion d'exceptions**
 - try** { `int x = calcul(y,z);` }
 - catch** (`ArithmeticException e`) {
 - `System.out.println("Erreur pendant le calcul");` }

Différentes catégories d'instructions Java



déclaration de variables,
affectations,
appels de fonctions,
instruction vide

suite d'instructions
entre accolades

sélection : if, switch
itération : while , do , for

gestion des exceptions

DECLARATION DE VARIABLES

La déclaration d'une variable doit être effectuée avant son utilisation

déclaration simple

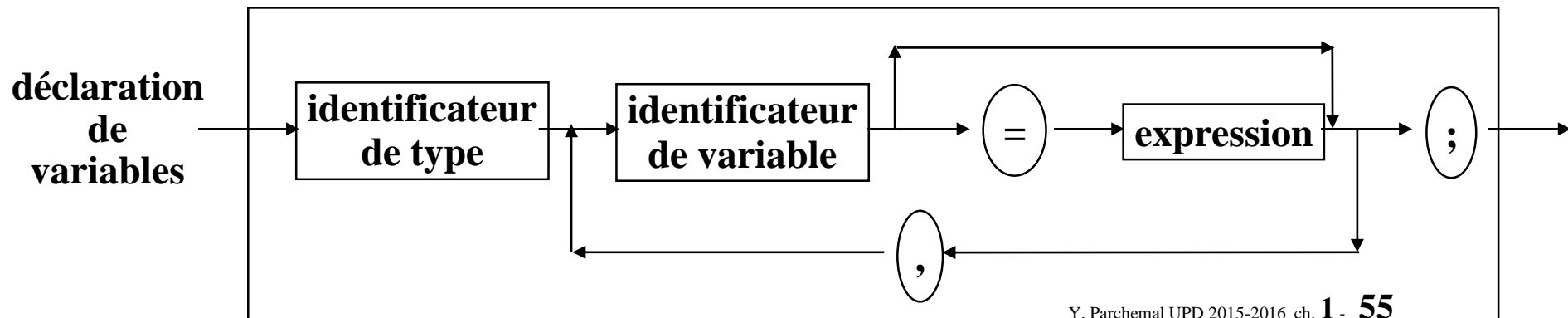
```
char c; /* c est une variable de type char */
String nom;
int w,x; /* w,x sont des variables de type int */
```

Affectation

```
c = 'e';
nom = "Dupond";
w = 5;
```

déclaration avec initialisation

```
char c='e'; /*équivalent à : char c; c='e'; */
int w=5,x;
String nom = "Dupond";
```



Opérateurs d'affectation

=

est l'opérateur général d'affectation
Les autres opérateurs d'affectation
permettent d'alléger l'écriture
pour certains cas particuliers

+= -=
***= /= %=**
et avec tous les
opérateurs binaires

+=

a += exp ;équivaut à **a = a + exp;**

-=

a -= exp ;équivaut à **a = a - exp;**

--
++

--

a--; équivaut à **a= a-1;**

++

a++; équivaut à **a= a+1;**

Les types simples

2 catégories de types de données en JAVA :

**LES 8 TYPES SIMPLES
(tous prédéfinis)**

byte short int long // les entiers
float double // les réels
boolean // les booléens
char // les caractères

LES CLASSES

exemples de classes:

java.lang.**String**

java.util.**Date**

java.util.**List**

**Les éléments représentables par un type simple ne dépendent
ni de la machine, ni du compilateur**

Les entiers en JAVA

Nom	De	à	Taille (en octets)
byte	$-2^7 =$ -128	$2^7 - 1 =$ +127	1
short	$-2^{15} =$ -32768	$2^{15} - 1 =$ +32767	2
int	$-2^{31} =$ -2147483648	$2^{31} - 1 =$ +2147483647	4
long	$-2^{63} =$ -9 223 372 036 854 775 808	$2^{63} - 1 =$ +9 223 372 036 854 775 807	8

Les flottants en JAVA

2 types: **float** (simple précision)

double (double précision)

de la forme $s * m * 2^e$ où

s est le signe , m est la mantisse et e l'exposant

Nom	m	e	Taille (en octets)
float	23 bits	8 bits	4
double	52 bits	11 bits	8

Java suit la
norme IEEE754

valeurs spéciales:

0+ 0- +inf -inf et NaN (indéterminé)

Attention

Pas d'erreurs à l'exécution avec les opérations
sur les float et double mais une gestion de valeurs spéciales

Les constantes numériques littérales


nombre	type	
523	int	
3 000 000 000	long	(> 0x7FFFFFFF)
523L	long	fini par L ou l
7.45	double	double par défaut
7.45F	float	fini par F ou f
1D	double	fini par D ou d
6.023E23	double	

nombre	base	
377	décimal	
0377	octal	commence par 0
0xAF5	hexadécimal	commence par 0x

OPERATEURS DE COMPARAISON

==	"égal"
!=	"différent"
>	"strictement supérieur"
>=	"supérieur ou égal"
<	"strictement inférieur"
<=	"inférieur ou égal"

Le résultat est de type booléen

 **Attention**

=	affectation	a=5; /*donne à 'a' la valeur 5 */
==	égalité	(a==5)/*teste si la valeur de a est 5 */

Les booléens

2 valeurs booléennes

true

false

Le résultat d'un test
est de type boolean

```
int x = 9;  
boolean b = true;  
boolean c = (x>5);
```

Attention

Pas de conversions automatiques entre entiers et booléens

```
boolean b = 1;  
if (1) {...};
```

Les opérateurs booléens

Symbole	Correspond à	Exemple	Évaluation en court circuit
!	Négation logique	! (x > 5)	
&	Et logique	(x>1)&(x<8)	non
&&	Et logique	(x>1)&&(x<8)	oui
	Ou logique	(x>10) (x<8)	non
	Ou logique	(x>10) (x<8)	oui
^	Ou exclusif	(x>1)^(y<8)	

Les opérateurs && et || réalisent les mêmes opérations que & et | mais avec une évaluation en court-circuit:
le second argument n'est alors évalué que lorsque sa valeur est susceptible de modifier le résultat

Les caractères en JAVA

UTILISE LE STANDARD UNICODE
et permet de coder 65536 caractères (2 octets)

Caractères "Unicode" de '\u0000' to '\uffff' (sur 2 octets)

De '\u0000' à '\u007f' (0 à 127) : identiques aux codes ASCII

De '\u0080' à '\u00ff' (128 à 255): codes Latin-1

codage des caractères de 0 à 127

code ASCII

chiffres	'0': 48 '1': 49 '2':50 '9': 57
majuscules	'A': 65 'B' : 66 'C' : 67 ... 'Z' : 90
minuscules	'a' : 97 'b': 98 'c': 99 ... 'z': 122
autres	' ' : 32 '#': 35 '<' : 60 '{': 123

Les caractères spéciaux

Passage à la ligne	\n
Tabulation	\t
Apostrophe	\'
Double apostrophe	\"
Antislash	\\
Code unicode	\uxxxx

exemples

`char c = \'"; // l'apostrophe`

```
String s = "Hello \n \"World\"";  
System.out.println(s);
```

**Hello
"World"**

Préséance des opérateurs (pour information)

1	[] ()
2	++ -- ! ~ instanceof
3	* / %
4	+ -
5	<< >> >>>
6	< > <= >=
7	== !=
8	&
9	^
10	
11	&&
12	
13	?:
14	= += -= %= *= /=

**En cas d'égales préséances
les opérations sont réalisées
de gauche à droite**

**int a,b=10,c=5,d=2;
a=b/c*d // a=((b/c)*d);**

**En cas de doute,
utiliser les parenthèses**

**boolean b=true;
boolean c=false;
a=!b&& c;//a=((!b)&&c);**

Conversion d'un type simple en un autre type simple

```
int x = 5;  
float f = x; // conversion d'un int en float
```

**La représentation interne d'un float et d'un int n'est pas du tout la même.
L'entier 5 est converti en float 5 au moment de l'affectation.**

On aurait pu utiliser l'opérateur de conversion :

```
int x = 5;  
float f = (float) x; // conversion explicite avec l'opérateur de conversion
```

**mais dans le cas d'une conversion d'un int en float,
cela n'est pas nécessaire**

Conversion d'un type simple en un autre type simple

conversion implicite

byte short int long float double



pas de conversion implicite il faut utiliser les opérateurs de conversion

Exemple de
conversion implicite

```
short s = 6; int i = 8;  
double d = s; // ou double d = (double) s;  
long l = s*i;
```

Cas de conversion
explicite obligatoire

```
double d = -4.3;  
float f = (float) d;  
int i = (int) d; // i vaut -4
```

remarques

Conversion impossible entre les nombres et les booléens

Pour le type char, conversion implicite de char vers int, long, double, float
conversion explicite nécessaire dans les autres cas

Les fonctions statiques

**Les fonctions dont nous parlons ici sont des fonctions comme celle que l'on rencontre dans des langages non orienté objets comme C.
Elles sont utiles pour étudier l'algorithmique en Java.
Nous verrons que d'autres types de fonctions sont utilisées en
Programmation Orientée Objet.**

Définition de fonctions statiques

Les définitions de fonctions sont constituées

- d'une en-tête (ou signature) indiquant en particulier
 - les paramètres de la fonction
 - le type du résultat retourné,
- d'un **bloc d'instructions** exécuté lors de l'appel de la fonction

```
/**@return le cube d'un flottant de type double*/
```

```
public static double cube(double x)    l'en-tête
```

```
{return x*x*x;}
```

le bloc d'instructions



Les fonctions statiques sont caractérisées par le mot clé "static" dans leur en-tête.

Exemple de fonctions statiques élémentaires

1 : fonctions rendant un résultat

```
/** rend le cube d'un double */  
public static double cube(double x){  
    return x*x*x;  
}
```

```
/** rend le carré de la somme des deux double 'x' et 'y' */  
public static double carreDeLaSomme(double x,double y){  
    double somme = x+y;  
    return somme*somme;  
}
```

```
/** rend l'expression de la forme ax2+bx+c connaissant les coeffs 'a', 'b' et 'c'*/  
public static String getExpressionBinome(double a, double b, double c){  
    String res = a+ " x2+ " + b+ "*x" + c;  
    return res;  
}
```

return permet de préciser le résultat rendu par d'une fonction

Exemple de fonctions statiques élémentaires 1 : fonctions rendant un résultat (suite)

```
/** rend le cube de 'x'*/
```

```
public static double cube(double x){  
    return x*x*x;  
}
```

```
/** rend 'x' puissance 6 */
```

```
public static double puissance6(double x){  
    double x3 = cube(x); // on utilise la fonction définie ci-dessus  
    return x3*x3;  
}
```


Exemple de fonctions statiques élémentaires 2 : fonctions ne rendant pas de résultat

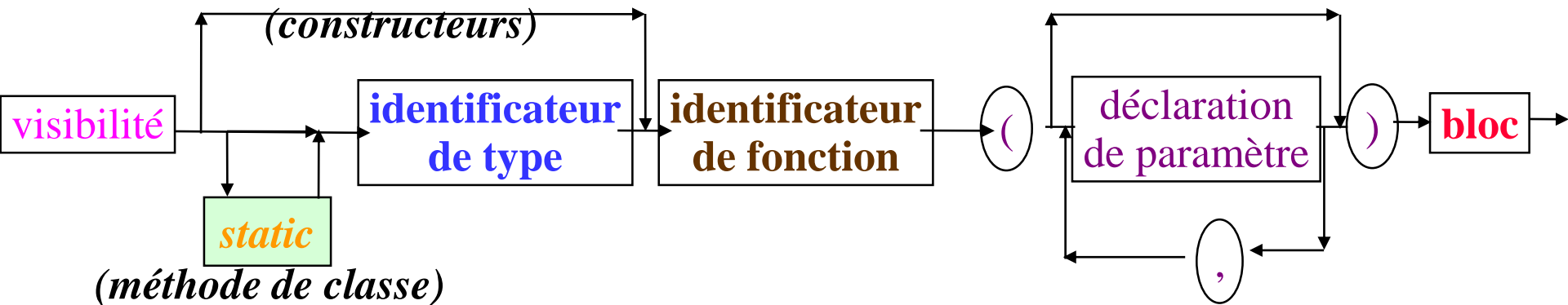
```
/** affiche un message de bienvenue pour une personne
 * connaissant le 'nom' et le 'prenom'*/
public static void afficherBienvenue(String nom,String prenom){
    System.out.println("Bonjour, "+prenom+" "+nom);
}
```

```
/** affiche le menu pour un programme gérant des comptes */
public static void afficheMenu( ){
    System.out.println("1 : consulter le solde");
    System.out.println("2 : enregistrer une nouvelle opération");
    System.out.println("3 : consulter l'historique du compte");
}
```

void Les fonctions ne rendant pas de résultat ont **void** comme type de retour

Syntaxe de la définition de fonctions

```
/** rend le carré de la somme des deux double 'x' et 'y' */  
public static int carreDeLaSomme(double x,double y){  
    double somme = x+y;  
    return somme*somme;  
}
```



L'instruction simple « return »

provoque la terminaison de la fonction
et le retour à la fonction appelante

Obligatoire si la fonction a une valeur de retour

return <expression>;

expression est la valeur de retour de la fonction

```
public static double cube(double a){  
    return(a*a*a);  
}
```

Rarement utilisé si la fonction n'a pas de valeur de retour

return;

le return est souvent implicite

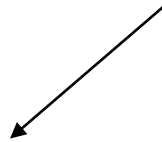
La fonction se termine "naturellement"

```
public static void afficherBienvenue(String nom,String prenom){  
    System.out.println("Bonjour, "+prenom+" "+nom);  
    return; // facultatif}
```

choix des identificateurs

le nom des identificateurs (paramètres, variables, fonctions, ...) doit pouvoir être **justifié**

```
public static void saluerUtilisateur(String nomUtilisateur)
```



Salue l'utilisateur du programme

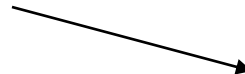


le nom de l'utilisateur
du programme

```
//class up5.mi.pary.term.Terminal  
public String readString(String message)
```



pour lire une chaîne de caractères



le message à afficher

CONVENTIONS SUR LES IDENTIFICATEURS

Types d'identificateurs	convention	exemple
classe	le premier symbole est une majuscule H elloWorld	S ystem
variable, fonction	le premier symbole est une minuscule p rintln	n om
constante	tout en majuscule	PI

Pour séparer les mots composant un identificateur, on met des majuscules.
exemple : read**S**tring Hello**W**orld

A propos de la définition de fonctions

**De façon générale, lorsque c'est possible,
il est préférable
de définir plusieurs "petites" fonctions
plutôt que
moins de fonctions mais plus longues**

**Ainsi, la définition de la fonction main doit être courte
(quelques lignes pour les programmes que nous écrirons)**

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

```
<entrée des données>
```

```
<appel de la fonction réalisant la tâche>
```

```
<affichage des résultats>
```

```
}
```

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

```
Terminal term = new Terminal("exemple",400,400);  
int x=term.readInt("Donner un entier");
```

```
int res = TestCube.cube(x);
```

```
term.println(res);  
}
```

APPEL DE FONCTIONS statiques

L'appel d'une fonction statique est effectué comme suit :
<nom de la classe> . <nom de la fonction>(<arguments>)

```
package up5.mi.pary.jt.algo;  
public class MathUtil {  
    public static double cube(double x){  
        return x*x*x;  
    }  
}
```

MathUtil.java

```
package up5.mi.pary.jt.puis6;  
public class TestPuis6 {  
    public static void main(String[] args){  
        Terminal term = new Terminal("puissance 6",400,400);  
        double x = term.readInt("donner un nombre");  
        double y = up5.mi.pary.jt.algo.MathUtil.cube(x) ;  
        term.println("la puissance sixième de "+x+" est "+y*y);  
    }  
}
```

TestPuis6.java

**Rq: Si la classe est importée (explicitement ou implicitement),
le nom simple de la classe suffit.**

APPEL DE FONCTIONS statiques

Si la classe est importée (explicitement ou implicitement),
le nom simple de la classe suffit.

```
package up5.mi.pary.jt.puis6;  
import up5.mi.pary.jt.algo.MathUtil;  
public class TestPuis6 {  
    public static void main(String[ ] args){  
        Terminal term = new Terminal("puissance 6",400,400);  
        double x = term.readInt("donner un nombre");  
        double y = MathUtil.cube(x) ;  
        term.println("la puissance sixième de "+x+" est "+y*y);  
    }  
}
```

TestPuis6.java

Appel à une fonction statique de la même classe

```
package up5.mi.pary.jt.algo;
public class MathUtil{

    public static int cube(int x){
        return x*x*x;
    }

    public static void main(String[] args){
        Terminal term = new Terminal("cube",400,400);
        int x = term.readInt("donner un entier");
        int y = cube(x); // ou MathUtil.cube(x)
        term.println("le cube de "+x+" est "+y);
    }
}
```

L'appel à une fonction statique de la même classe peut se faire en mentionnant simplement le nom de la fonction

Affectations et passage de paramètres pour les types simples

Affectation avec recopie

`int i = 7;` i

7

`int j = i;` i

7

`/* copie`
`de i dans j*/` j

7

`i=i+1;` i

8

 j

7

Passage par valeur

```
public class TestPassage {  
    public static int f(int j){  
        j=j+1;  
        return(j);  
    }  
    public static void main(String[] tArg) {  
        int i = 9;  
        System.out.println("i="+i);  
        int r = f(i);  
        System.out. println("i="+i+" r="+r);  
    }  
}
```

i=9
i=9 r=10

**Remarque : c'est le seul mode d'affectation et
de passage de paramètres pour les types simples**

Les conditionnelles

if (<condition>) <instruction>

if (<condition>) <instruction> else <instruction>

switch (<expression>){

 case <valeur> : <instructions>

 ...

 }

if (<test>) <instruction>

if (<test>) /* toujours entre parenthèses */
<instruction> /*exécutée si test vrai*/

```
int r=-5;  
System.out.print("|"+r+"|=");  
if (r<0)  
    r = -r;  
System.out.println(r);
```

$|-5|=5$

```
int r=+5;  
System.out.print("|"+r+"|=");  
if (r<0)  
    r = -r;  
System.out.println(r);
```

$|5|=5$

if (<test>) <instruction>

Si plusieurs instructions doivent être exécutées alors qu'une seule instruction est prévue (ce qui est le cas pour l'instruction if), on utilise **un bloc d'instructions**

```
int r=-5;  
if (r<0)  
    {r = -r;r=r*2;}  
System.out.println(r);
```

10

On peut aussi utiliser un bloc lorsqu'il n'y a qu'une instruction (bien que ce ne soit pas nécessaire)

```
int r=-5;  
System.out.print("'" + r + "'|=");  
if (r<0)  
    {r = -r;}  
System.out.println(r);
```

'|-5|=5

if (<test>) <instruction> else <instruction>

if (<test>) /* toujours entre parenthèses */
<instruction> /*exécutée si test vrai*/
else **<instruction>** /*exécutée si test faux*/
partie else facultative

if avec else

```
if (r>0)
    System.out.println("positif");
else
    System.out.println("négatif ou nul");
```

instructions if imbriquées

if (<test>) <instruction> *else* <instruction>

if (moyenne<10)
term.println("Recalé");
else

```
if (moyenne < 12)
    term.println("Passable");
else if (moyenne < 14)
    term.println("Assez Bien");
else if (moyenne < 16)
    term.println("Bien");
else
    term.println("Très Bien");
```

instructions if imbriquées : une autre façon d'indenter

if (<test>) <instruction> *else* <instruction>

```
Terminal term = new Terminal("voici votre mention",400,400);  
int moyenne=term.readInt("Quelle est votre moyenne");
```

if (moyenne<10)	term.println("Recalé");
<i>else if</i> (<i>moyenne<12</i>)	<i>term.println("Passable");</i>
<i>else if</i> (<i>moyenne < 14</i>)	<i>term.println("Assez Bien");</i>
<i>else if</i> (<i>moyenne < 16</i>)	<i>term.println("Bien");</i>
<i>else</i>	<i>term.println("Très Bien");</i>

L'instruction à choix multiples :SWITCH

```
Terminal term = new Terminal("switch",400,400);
char c = term.readChar("Repondre Oui ou Non");
switch (c){
    case 'O': term.println("Vous m'avez repondu Oui" );break;
    case 'N': term.println(" Vous m'avez repondu Non" );break;
    default: term.println(" Je n'ai pas compris" );break;
}
```

Repondre Oui ou Non
O
Vous m'avez repondu Oui

```
Terminal term = new Terminal("switch",400,400);
char c = term.readChar("Repondre Oui ou Non");
switch (c){
    case 'O':
    case 'o' : term.println("Vous m'avez repondu Oui" );break;
    case 'n' :
    case 'N': term.println(" Vous m'avez repondu Non" );break;
    default: term.println(" Je n'ai pas compris" );break;
}
```

Repondre Oui ou Non
O
Vous m'avez repondu Oui

SWITCH : un exemple avec une expression entière

```
Terminal term = new Terminal("switch",400,400);
int n = term.readInt("Repondre 1 ou 2");
switch (n){
    case 1 : term.println("Vous m'avez repondu 1" );break;
    // surtout pas case '1' avec des apostrophes car cela désigne alors un caractère !
    case 2: term.println("Vous m'avez repondu 2" );break;
    default: term.println("Je n'ai pas compris" );break;
}
```

Attention : ne pas confondre 1 et '1' : le compilateur Java ne signale pas l'erreur

traduire une instruction switch en if

```
switch (c){  
  case 'O':  
  case 'o' : term.println("Vous m'avez répondu Oui" );break;  
  case 'n' :  
  case 'N': term.println(" Vous m'avez répondu Non" );break;  
  default: term.println(" Je n'ai pas compris" );break;  
}
```

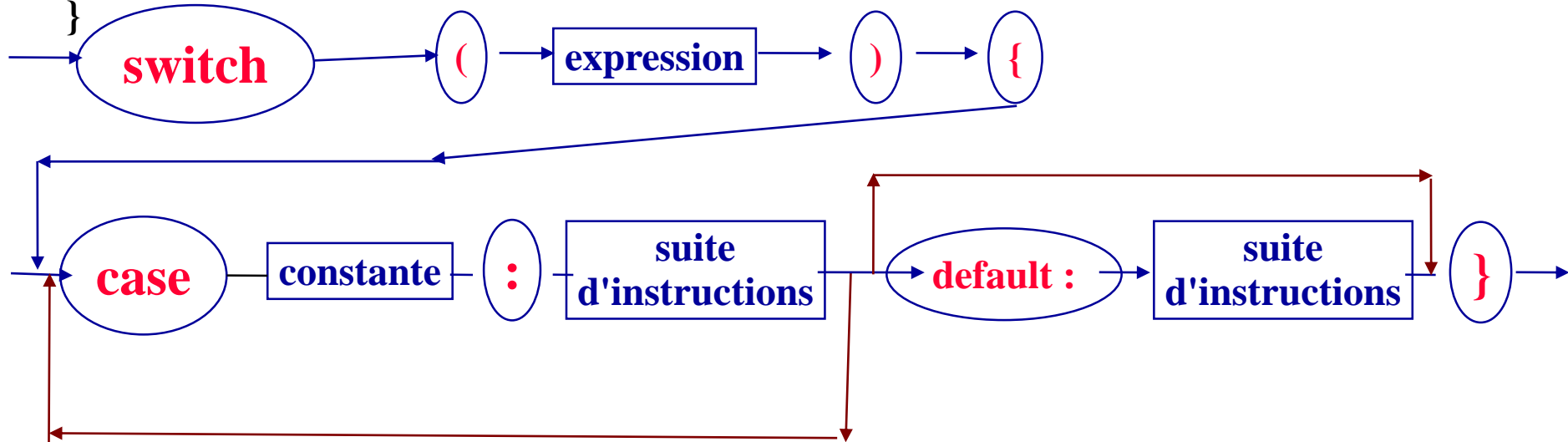
est équivalent à

```
if ((c=='O')||(c=='o'))  
  term.println("Vous m'avez répondu Oui" );  
else if ((c=='N')||(c=='n'))  
  term.println(" Vous m'avez répondu Non" );  
else term.println(" Je n'ai pas compris" );
```

remarques : || est l'opérateur de disjonction et se lit "OU"
== est l'opérateur d'égalité

SWITCH: syntaxe

```
switch (c){  
  case 'O': term.println("Vous m'avez répondu Oui");break;  
  case 'N': term.println(" Vous m'avez répondu Non" );break;  
  default: term.println(" Je n'ai pas compris" );break;  
}
```



restriction

Le type de l'expression testée doit être un type entier ou "char"
Depuis java7, également String

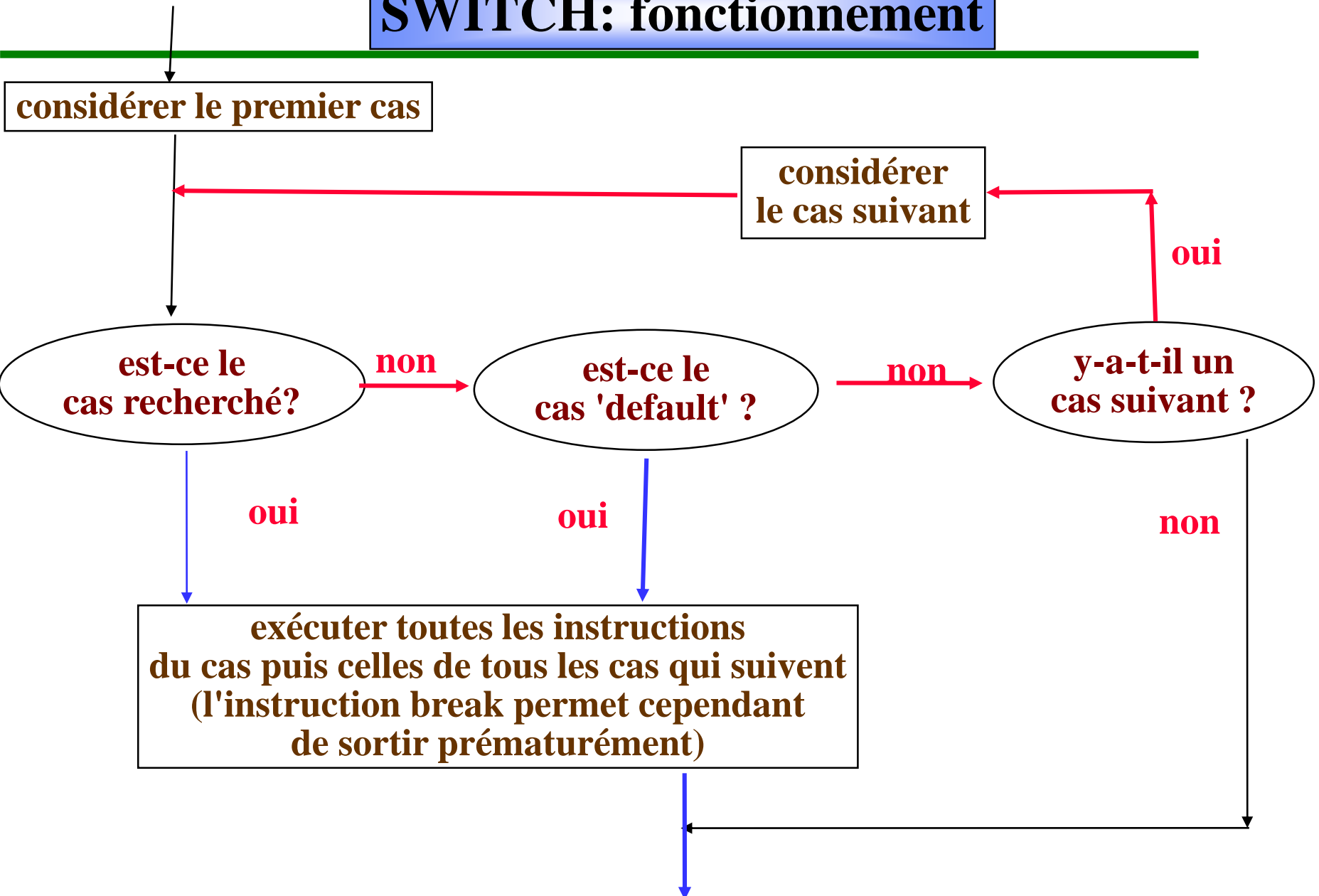
fonctionnement

Lorsqu'un cas est sélectionné, toutes les instructions
qui le suivent sont exécutées,
y compris celles des cas suivants!

break

L'instruction simple 'break' permet
de sortir prématurément de l'instruction switch

SWITCH: fonctionnement



while

```
while (<test>)  
    <instruction>
```



TANT QUE le test est vérifié
exécuter l'instruction

while (<test>) <instruction>

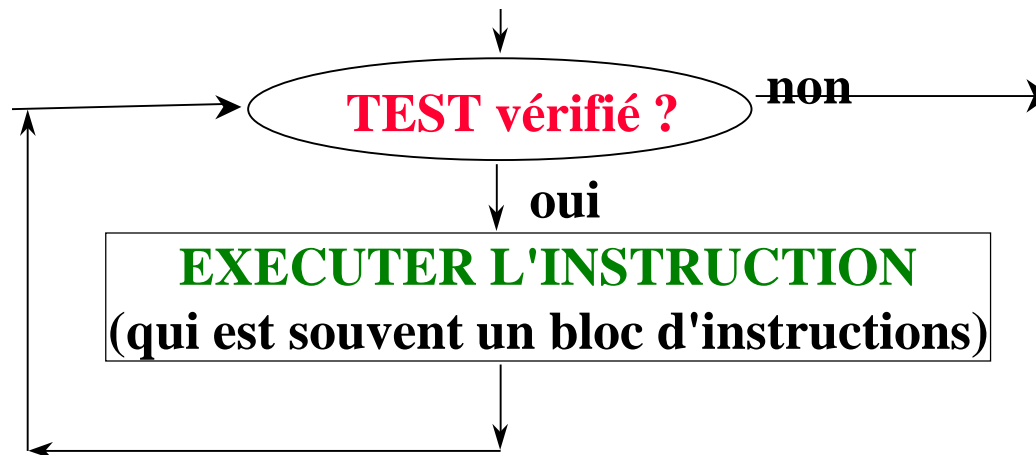
**un premier
exemple**

```
int i = 0;  
while (i<=5){  
    System.out.println(i);  
    i=i+1;  
}
```

0
1
2
3
4
5

fonctionnement

TANT QUE LE TEST EST VERIFIE,
L'INSTRUCTION EST EXECUTEE
(on dit aussi : **une itération supplémentaire est effectuée**)



while (<test>) <instruction>

un deuxième exemple

```
int i=1;
double res=1;
while (i<=4){
    res=res*3;
    i=i+1;
}
System.out.println(res);
System.out.println(i);
```

81
5

i	res	
Avant l'itération n°1	1	1
Avant l'itération n°2	2	3
Avant l'itération n°3	3	9
Avant l'itération n°4	4	27
En sortie de boucle	5	81

while : puissance nième d'un nombre entier

un exemple d'utilisation dans une fonction

```
/** @return la puissance entière d'un double */  
public static double puissance (double n, int p){  
    int i=1;  
    double res=1;  
    while (i<=p){  
        res=res*n;  
        i=i+1;  
    }  
    return res;  
}
```

```
public static void main(String [ ] args){  
    System.out.println("2^4="+puissance (2,4));}
```

2^4=16

while : affichage de cubes d'entiers

```
public static void main(String [] args) {  
    Terminal term = new Terminal("while",400,400);  
    int max=term.readInt("donner un entier");  
    term.println("je vais afficher les cubes inférieurs à "+max+" des premiers entiers");  
    int i=0;  
    while (i*i*i<max){  
        term.println("cube("+i+")="+i*i*i);  
        i = i+1;}  
}
```

donner un entier 100

je vais afficher les cubes inférieurs à 100 des premiers entiers

cube(0)=0

cube(1)=1

cube(2)=8

cube(3)=27

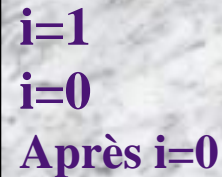
cube(4)=64

WHILE : évaluation du test

Le test est évalué avant chaque itération
et uniquement à ce moment là

```
int i = 2;

while (i !=0){
    i= i - 1;
    System.out.println("i="+i);
}
System.out.println("Après i="+i);
}
```



i=1
i=0
Après i=0

Il n'y a pas *arrêt brutal* du while lorsque i prend la valeur 0 :
toute l'**instruction** du while est exécutée
et ce n'est qu'après la fin de cette **instruction** que le **test** est à
nouveau effectué.

Attention, ça boucle !

```
/** @return la puissance entière d'un nombre */  
public static double puissance (double n, int p){  
    int i=1;  
    int res=1;  
    while (i<=p){  
        res=res*n;  
    }  
    return res;  
}
```

```
public static void main(String [ ] args){  
    System.out.println("2^4="+puissance (2,4));  
}
```

Attention, ça boucle car

do ... while

do <instruction>

while (<test>) ; // point virgule obligatoire



exécuter l'instruction

TANT QUE le test est vérifié

différence avec l'instruction while

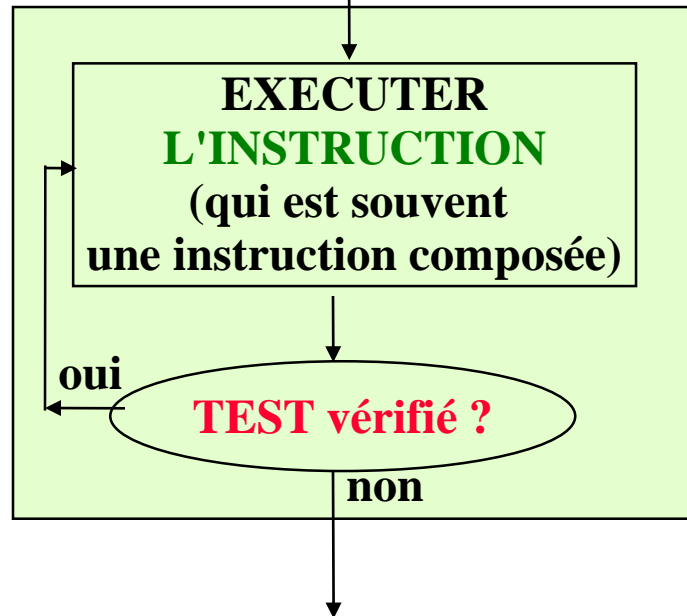
**LE TEST EST REALISE
EN FIN DE BOUCLE**

do <instruction> while (<test>) ;

exemple

```
int annee;  
do {  
    annee = term.readInt('Donner une année >= 1880');  
}  
while (annee < 1880);
```

fonctionnement

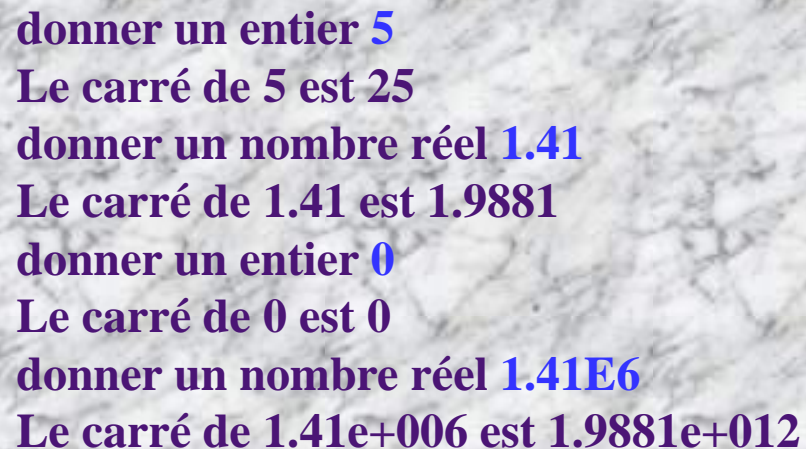


utilisation

dans les cas (assez peu fréquents)
où au moins une itération doit être effectuée

do ... while : exemple

```
public class Test {  
    public static void main(String[ ] args) {  
        Terminal term = new Terminal("dowhile",400,400);  
        int n;double x;  
        do {n = term.readInt("donner un entier");  
            term.println("Le carre de '"+n+"' est '"+n*n);  
            x = term.readDouble("donner un nombre réel");  
            term.println("Le carré de '"+x+"' est '"+x*x);}  
        while (x*n!=0); // continuer sauf si l'un des deux nombres est nul  
        term.end();  
    }  
}
```



```
donner un entier 5  
Le carré de 5 est 25  
donner un nombre réel 1.41  
Le carré de 1.41 est 1.9881  
donner un entier 0  
Le carré de 0 est 0  
donner un nombre réel 1.41E6  
Le carré de 1.41e+006 est 1.9881e+012
```

for

```
for (<expInit>;<expTest>;<expIncr>)  
    <instruction>
```

Permet de traduire des énoncés comme :

Pour i variant de i_{\min} à i_{\max} faire <instruction>

Répéter n fois <instruction>

Pour i variant de i_{\min} à i_{\max} et tant que test vérifié faire <instruction>

for (<expInit>;<expTest>;<expIncr>) <instruction>

Pour i variant de i_{\min} à i_{\max} faire <instruction>

p=1;

Pour i variant de 1 à n faire p=p*i;

```
int p = 1;  
for (int i=1 ; i<= n ; i++)  
    p=p*i;
```

Répéter n fois <instruction>

répéter 40 fois écrire "-"

```
for (int i=1 ; i<= 40 ; i++)  
    System.out.println('-');
```

traduction de for en while

exemple

```
for (int i=1 ; i<= n ; i++)  
    p=p*i;
```

produit le même résultat que:

```
{ int i=1 ;  
  while (i<=n){  
    p=p*i;  
    i++;  
  }  
}
```

équivalence avec while

```
for (<expInit>;<expTest>;<expIncr>) <instruction>
```

```
{<expInit>;  
while (<expTest>){  
    <instruction>  
    <expIncr>;  
}  
}
```

Autre utilisation de la boucle for

Pour i variant de i_{\min} à i_{\max} et tant que test vérifié
faire **<instruction>**

$p=1$;

Pour i variant de 1 à n et tant que $p < 1000$ faire $p=p*i$;


```
int p = 1;  
for (int i=1 ; ( i<= n)&& (p<1000) ; i++)  
    p=p*i;
```

syntaxe

for (**<expInit>**;**<expTest>**;**<expIncr>**) **<instruction>**

for : la variable de boucle

i est souvent déclarée à l'intérieur de l'instruction for



```
for (int i=1;i<=5;i++)  
    System.out.println(""+i+" "+i*i*i );
```

La variable i est locale à la boucle for

1	1
2	8
3	27
4	64
5	125



remarque

on aurait pu aussi écrire:

```
int i;  
for (i=1;i<=5;i++) // la variable i n'est plus locale  
    System.out.println(""+i+" "+i*i*i );  
System.out.println(i); // on peut l'utiliser après la boucle !
```

conventions:

Sauf exception, la variable de boucle est déclarée dans l'instruction for.

for ou while ?

Nous avons étudié quelques cas d'utilisation de la boucle for
et nous nous limiterons à ces cas.
Dans les autres cas, nous utiliserons while (ou do while)

exemple:

```
for (int i=0; i<5; i++){  
    System.out.println(i);  
    i=i+2;  
}
```

Non !

conventions: La valeur de la variable de boucle ne doit pas être
modifiée dans l'instruction.
(dans de tels cas, on utilise une boucle while)

Portée des variables

Déclarations de variables

Elle peuvent être faites à tous les endroits d'un programme où il est possible de mettre une instruction

Portée des variables: principe

De l'endroit du bloc où elles sont déclarées jusqu'à la fin de ce bloc

```
a=5;  
{// b n'est pas encore défini  
a=a+7;  
int b=8;  
b=b*2;  
System.out.println(b);  
}  
// b n'est plus défini
```

Portée des variables et des paramètres

```
public static int pgcd(int a, int b){  
    int x = a;  
    int y = b;  
    while (x!=y){  
        if (x>y){  
            int i= x;  
            x= y;  
            y= i;}  
        else y= y-x;  
    }  
    return(x);}
```

portée de y

```
public static int pgcd(int a,int b){  
    int x = a;  
    int y = b;  
    while (x!=y){  
        if (x>y){  
            int i= x;  
            x= y;  
            y= i;}  
        else y= y-x;  
    }  
    return(x);}
```

portée de i

```
public static int pgcd(int a, int b){  
    int x = a;  
    int y = b;  
    while (x!=y){  
        if (x>y){  
            int i= x;  
            x= y;  
            y= i;}  
        else y= y-x;  
    }  
    return(x);  
    }
```

portée de a

Portée des variables et boucle for

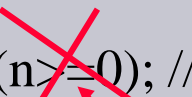
Dans les instructions for,
la portée de la variable de boucle inclut
l'expression de test et d'incrément

```
int n=3;  
for (int i = 1;i<=n ;i++ ) {  
    System.out.print(i);  
    System.out.println(n);  
    }  
System.out.print("fini");  
}
```

portée de i

Portée des variables et do-while

```
// un programme refusé par le compilateur
do {
    int n=term.readInt("donner un nombre positif ou nul");
}
while (n>=0); // n n'est pas défini en dehors du bloc interne au do-while
```



Message d'erreur du compilateur : n : variable indéfinie

Les variables utilisées dans le test d'une boucle do-while doivent être déclarées avant la boucle.

```
// la bonne solution
int n;
do {
    n=term.readInt("donner un nombre positif ou nul");
}
while (n>=0);
```

continue et break

continue

permet, dans une boucle while, do ou for de finir prématurément l'itération en cours

`continue;`

`continue <etiquette>;`

break

permet de sortir prématurément d'un bloc d'instructions

`break;`

`break <etiquette>;`

Attention

Exceptée l'utilisation du break dans le switch, il faut éviter d'utiliser break et continue qui peuvent rendre difficile la compréhension des algorithmes.

continue et break: exemple avec une boucle for

```
int n=3;  
for (int i = 1;i<=n;i++){  
    System.out.print(" "+i);  
    System.out.print(":"+i);} 
```

1:1 2:2 3:3

```
int n=3;  
for (int i = 1;i<=n;i++){  
    System.out.print(" "+i);  
    if (i==2) continue;  
    System.out.print(":"+i);} 
```

1:1 2 3:3

```
int n=3;  
for (int i = 1;i<=n;i++){  
    System.out.print(" "+i);  
    if (i==2) break;  
    System.out.print(":"+i);} 
```

1:1 2

continue et break: exemple avec 2 for imbriqués

```
int n=3;
for (int x = 9;x<=11;x++){
  for (int i = 1;i<=n;i++){
    System.out.print(" "+x+'/' +i);
    // if ((i==2)||(x==10)) continue ;
    // if ((i==2)||(x==10)) break;
    System.out.print(": "+i);
  }
  System.out.print("&" +x);
}
```

9/1:1 9/2:2 9/3:3&9 10/1:1 10/2:2 10/3:3&10 11/1:1 11/2:2 11/3:3&11

continue;

9/1:1 9/2 9/3:3&9 10/1 10/2 10/3&10 11/1:1 11/2 11/3:3&11

break;

9/1:1 9/2&9 10/1&10 11/1:1 11/2&11

continue et break avec étiquettes

```
int n=3;
ici:
  for (int x = 9;x<=11;x++){
    for (int i = 1;i<=n;i++){
      System.out.print(" "+x+'/' +i);
      // if ((i==2)||(x==10)) continue ici;
      // if ((i==2)||(x==10)) break ici;
      System.out.print(": "+i);
    }
    System.out.print("& "+x);
  }
```

9/1:1 9/2:2 9/3:3&9 10/1:1 10/2:2 10/3:3&10 11/1:1 11/2:2 11/3:3&11

continue ici;

9/1:1 9/2 10/1 11/1:1 11/2

break ici;

9/1:1 9/2

Les tableaux

**Un tableau est une collection ordonnée
de variables du même type**

exemple :

tableau de 8 float

tableau de 5 String

tableau de 10 Terminal

**Chacune de ces variables est repérée par son
indice (qui est sa position dans le tableau):
si un tableau comporte n éléments,
les indices vont de 0 à $n-1$**

Les tableaux : déclaration, initialisation

1 déclaration **float [] tabNote;**

// tabNote est un tableau de "float"

2 initialisation
du tableau

int n = term.readInt("taille du tableau ?");

4

tabNote = new float[n];

// tabNote est un tableau de n float d'indice 0 à n-1)

3 initialisation
des éléments
du tableau

tabNote[0] tabNote[1] tabNote[2] tabNote[3]

		8	
--	--	----------	--

tabNote[2] = 8; // initialisation du 3ème élément

remarque

En JAVA,
l'indice est un nombre entier positif
l'indice minimum d'un tableau est **toujours zéro**

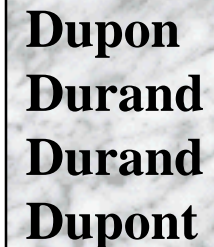
LES TABLEAUX : exemple

```
String [ ] tab = new String [3];
```

```
tab[0] = "Dupond";  
tab[1] = "Durand";  
tab[2] = "Dupon";
```

```
System.out.println(tab[2]);  
System.out.println(tab[1]);
```

```
int i = 1;  
System.out.println(tab[i]);  
tab[i+1]=tab[i+1]+'t';  
System.out.println(tab[2]);
```



Dupon
Durand
Durand
Dupont

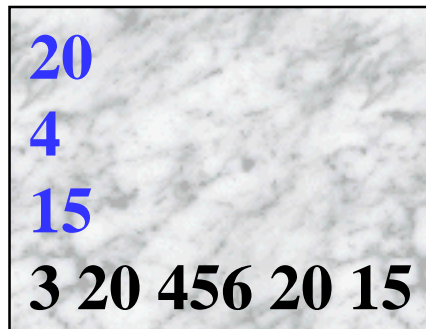
LES TABLEAUX : exemple

un tableau de 5 entiers

```
int [ ] tab; /*declaration d'un tableau d'entiers*/  
tab = new int [5];
```

exemples
d'utilisation

```
int tab[ ]=new int[5];  
tab[2]=456; /* tab[2] est l'élément d'indice 2 du tableau tab */  
tab[2-2]=3;  
tab[3]=term.readInt("");  
tab[1] = tab[tab[0]];  
i= term.readInt(""); tab[i] = term.readInt("")  
term.println(""+ tab[0]+ tab[1]+ tab[2]+  
tab[3]+ tab[4]);
```



```
20  
4  
15  
3 20 456 20 15
```

Tableaux : initialisation

```
int [ ] tab = {7,3,8,9};
```

équivalent à:

```
int [ ] tab = new int[4];  
tab[0]=7;tab[1]=3;tab[2]=8;tab[3]=9;
```

équivalent à:

```
int [ ] tab ; tab = new int[4];  
tab[0]=7;tab[1]=3;tab[2]=8;tab[3]=9;
```

exemple `String[] tMois={"Janvier","Février","Mars","Avril","Mai","Juin","Juillet",
 "Août","Septembre","Octobre","Novembre","Décembre"};`

Les tableaux

nombre d'éléments: l'attribut length

On peut accéder au nombre d'éléments d'un tableau grâce à l'attribut "length"

```
String [ ] args= new String[12];  
System.out.println(args.length);  
// args.length désigne la longueur  
// du tableau args
```

12



On ne peut pas modifier la valeur de "length" :
c'est un attribut constant

~~args.length = 4;~~

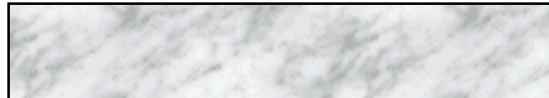
Valeurs par défaut des variables

Lorsqu'un tableau est créé, une valeur par défaut est affectée à chacun des éléments de ce tableau.

type	valeur
boolean	false
char	\u0000
byte,short,int,long	0
float,double	0.0
type «classe »	null

quel affichage ?

```
int [] tabInt = new int[10];  
for (int i=0;i<tabInt.length-2;i++){  
    tabInt[i]=i;  
}  
for (int i=7;i<tabInt.length;i++)  
    System.out.print(tabInt[i]);  
  
System.out.println( );
```



algorithmes de base sur les tableaux

- affichage des éléments d'un tableau
- saisie des éléments d'un tableau
- somme des éléments d'un tableau d'entiers
- test de l'appartenance d'un élément à un tableau

AFFICHAGE DES ELEMENTS D'UN TABLEAU

```
/** affiche les éléments du tableau 'tab' sur le terminal 'term'*/  
public static void afficher(Terminal term,int [] tab){  
    for (int i = 0; i < tab.length ; i++){  
        term.print(tab[i]);  
        if (i!=tab.length-1) term.print(" ");  
    }  
    term.println();  
}  
  
public static void main (String[] args){  
    int [ ] tabInt = {32,12,23,21,5};  
    Terminal term = new Terminal("affichage de tableaux",400,400);  
    afficher(term , tabInt);  
}
```

SAISIE DES ELEMENTS D'UN TABLEAU

```
/* rend un tableau de 'n' entiers demandés
à l'utilisateur à l'aide du terminal 'term'*/
public static int [ ] saisieTabInt(Terminal term,int n){
int[] tab =new int[n];/* le résultat est un tableau de n éléments*/
for (int i = 0; i < n ; i++)
    tab [i] = term.readInt("");/* saisie de l'élément d'indice i*/
return tab;
}

public static void main (String [ ] args){
    Terminal term = new Terminal("Saisie et affichage d'un tableau");
    int [ ] tabInt=saisieTabInt(term,5);
    afficher(term,tabInt);
}
```


SOMME DES ELEMENTS D'UN TABLEAU

```
/* rend la somme des éléments du tableau 'tab' */  
public static int somme (int [ ] tab){  
int res=0;  
for (int i = 0; i < tab.length ; i++){  
    res=res + tab[i] ;  
return res;  
}
```

```
public static void main (String [ ] args){  
    int [ ] tabInt = {32,12,23,21,5};  
    System.out.println(somme (tabInt) );  
}
```

93

TEST DE L'APPARTENANCE D'UN ELEMENT A UN TABLEAU

```
int [ ] t = {1,2,3,4,5};
```

3 est-il un élément de t ?

t [0] == 3 ? non

t [1] == 3 ? non

t [2] == 3 ? oui 3 est un élément de t

12 est-il un élément de t ?

t [0] == 12 ? non

t [1] == 12 ? non

t [2] == 12 ? non

t [3] == 12 ? non

t [4] == 12 ? non

tant que il reste des éléments à tester
et que on n'a pas trouvé l'élément cherché
tester l'élément suivant

```
trouve=false;  
i=0; /*indice de l'élément à tester*/  
tant que (i<t.length)  
    et (! trouve) // (trouve ==false)  
    si t[i] est l'élément cherché  
        alors trouve=true  
    sinon i=i+1;
```

TEST DE L'APPARTENANCE D'UN ELEMENT A UN TABLEAU

```
trouve=false;
i=0; /*indice de l'élément à tester*/
tant que (i<t.length)
    et (! trouve) // (trouve ==false)
    si t[i] est l'élément cherché
        alors trouve=true
        sinon i=i+1;
```

```
/* teste si 'e' est un élément du tableau 't'*/
```

```
public static boolean appartient(int [ ] tab , int elt){
```

```
boolean trouve=false;
```

```
int i=0;
```

```
while ((i<tab.length)&&(!trouve)){
```

```
    if (tab [i] == elt)
```

```
        trouve=true;
```

```
    else i++;
```

```
    }
```

```
return trouve;
```

```
}
```

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

```
    Terminal term = new Terminal("test de l'appartenance à un tableau",400,400);
```

```
    int [ ] tab= saisieTabInt(5);
```

```
    int x=term.readInt("quel nombre dois-je chercher ?");
```

```
    if (appartient(tab,x))
```

```
        term.println( x+" appartient au tableau ");
```

```
}
```

Tri de tableaux

- tri par sélection échange
- tri par insertion

Objectif d'un algorithme de tri

trier les éléments d'un tableau selon un critère d'ordre donné.

Exemple de critère

par ordre croissant (pour des nombres)

par ordre lexicographique (pour des chaînes)

Temps d'exécution

il dépend de l'algorithme choisi et du nombre d'éléments à trier

algorithme de tri élémentaire tri par insertion, tri par sélection-échange
le temps est proportionnel à $n*n$ (n étant le nombre d'éléments à trier)

algorithme de tri rapide tri par fusion, quicksort
le temps est proportionnel à $n * \log(n)$

TRI PAR SELECTION-ECHANGE D'UN TABLEAU par ordre croissant

indices:	0	1	2	3	4	5	6	
tableau à trier	32	12	23	1	21	5	10	
étape n°1:	32	12	23	1	21	5	10	sélection
1	12	23	32	21	5	10		échange
étape n°2:	1	12	23	32	21	5	10	sélection
1	5	23	32	21	12	10		échange
étape n°3:	1	5	23	32	21	12	10	sélection
1	5	10	32	21	12	23		échange
étape n°4:	1	5	10	32	21	12	23	sélection
1	5	10	12	21	32	23		échange
étape n°5:	1	5	10	12	21	32	23	sélection
1	5	10	12	21	32	23		échange
étape n°6:	1	5	10	12	21	32	23	sélection
1	5	10	12	21	23	32		échange
résultat:	1	5	10	12	21	23	32	

TRI PAR SELECTION-ECHANGE

d'un tableau d'entiers par ordre croissant

TRI en $n-1$ étapes

Soit n le nombre d'éléments du tableau

à l'étape n° i ($1 \leq i \leq n-1$):

soit j l'indice du plus petit élément entre les indices $i-1$ et $n-1$
permuter les éléments d'indice $i-1$ et j

```
/* trie des éléments d'un tableau d'entiers par ordre croissant */
public static void trier(int [ ] tab){
for (int etape=1; etape<=tab.length-1; etape++){
    // dep est l'indice à partir duquel le plus petit élément est cherché
    int dep = etape-1;
    // recherche de l'indice du plus petit élément entre les indices dep et tab.length-1
    int j = indiceDuPlusPetit(tab, dep, tab.length-1);
    // permutation des éléments d'indice dep et j
    echanger(tab, dep, j);
}
}
```

RECHERCHE DE L'INDICE DU PLUS PETIT ELEMENT

```
/** recherche l'indice du plus petit élément
 * du tableau 'tab' entre les indices 'imin' et 'imax' */
public static int indiceDuPlusPetit(int [ ] tab,int imin,int imax){
    int res = imin;
    for (int i=imin+1;i<=imax;i++)
        if (tab[i] < tab [res]) res=i;

    return res;
}
```


Echange (permutation) de deux éléments du tableau

```
/* échange les éléments 'tab'['i'] et 'tab'['j'] */  
public static void echanger(int [] tab, int i, int j){  
    int tabj=tab [j];  
    tab [j]=tab [i];  
    tab [i]=tabj;  
}
```

TRI PAR INSERTION D'UN TABLEAU de n éléments

Objectif : trier par ordre croissant un tableau d'entiers de n éléments

Exemple avec un tableau de 7 éléments :

tableau initial:	32	12	23	<i>1</i>	<i>21</i>	5	<i>10</i>
tableau après le tri :	1	5	12	12	21	23	32

TRI PAR INSERTION D'UN TABLEAU de n éléments

	indices: 0	1	2	3	4	5	6
étape n°1:	32	<u>12</u>	23	1	21	5	10
12	32	23	1	21	5	10	
étape n°2:	12	32	<u>23</u>	1	21	5	10
12	23	32	1	21	5	10	
étape n°3:	12	23	32	<u>1</u>	21	5	10
1	12	23	32	21	5	10	
étape n°4:	1	12	23	32	<u>21</u>	5	10
1	12	21	23	32	5	10	
étape n°5:	1	12	21	23	32	<u>5</u>	10
1	5	12	21	23	32	10	
étape n°6:	1	5	12	21	23	32	<u>10</u>
1	5	10	12	21	23	32	
résultat:	1	5	12	12	21	23	32

à l'étape n°i ($1 \leq i \leq n-1$):

- les i premiers éléments sont (déjà) par ordre croissant
- on insère l'élément d'indice i dans le "sous-tableau" (indices 0 à i) de telle sorte que les i+1 premiers éléments de t soient rangés par ordre croissant

TRI PAR INSERTION D'UN TABLEAU de n éléments

TRI en n-1 étapes

à l'étape n°i ($1 \leq i \leq n-1$):

- les i premiers éléments sont (déjà) par ordre croissant
- on insère l'élément d'indice i dans le "sous-tableau" (indices 0 à i) de telle sorte que les i+1 premiers éléments de t soient triés par ordre croissant

/ les éléments de 't' sont réorganisés de telle sorte qu'ils soient rangés par ordre croissant*/*

```
public static void triInsertion(int [] tab){  
    for (int i = 1; i < tab.length; i++)  
        insererElementDansTableau(tab,i);  
}
```

TRI PAR INSERTION D'UN TABLEAU de n éléments

/** 'tab' est un tableau d'entiers dont les éléments d'indice 0 à 'i'-1
* sont rangés par ordre croissant
* les éléments d'indice 0 à 'i' de 'tab' sont réorganisés de telle sorte
* qu'ils soient rangés par ordre croissant */

```
public void insererElementDansTableau(int [] tab,int i){  
    int elementAPlacer = tab[i];  
    int placeProposee= i ;  
    boolean placeTrouve=false;  
    while (! placeTrouve) {  
        if (placeProposee==0 || tab[placeProposee-1]< elementAPlacer )  
            placeTrouve=true;  
        else {  
            tab[placeProposee] = tab[placeProposee-1];  
            placeProposee--;  
        }  
    }  
    tab[placeProposee]= elementAPlacer ;  
}
```