

COURS DE PROGRAMMATION ORIENTÉE OBJET JAVA



2014-2015

SUP' INFO



Module 1: Introduction

- Introduction
- Les Générations des langages de programmation
- Java un langage Objet
- La Programmation Objet (POO)
- Concept de Classe et Objet
- L'encapsulation
- Java en 4 Editions



Module2: Techniques de base

- Premier Programme java
- Structure du Programme
- Exécution du Programme
- Les structures de contrôle
- Règle d'écriture en java
- Les opérateurs relationnels
- Les opérateurs logiques
- Les opérateurs conditionnels
- La classe Scanner



Module 1

Introduction à Java

Historique : Origines de Java

• 1990

- Internet très peu connu, World Wide Web inexistant .
- boom des PCs (puissance)
- Projet Oak de SUN Microsystems
 - Langage pour la communication des appareils électroniques .

• 1993

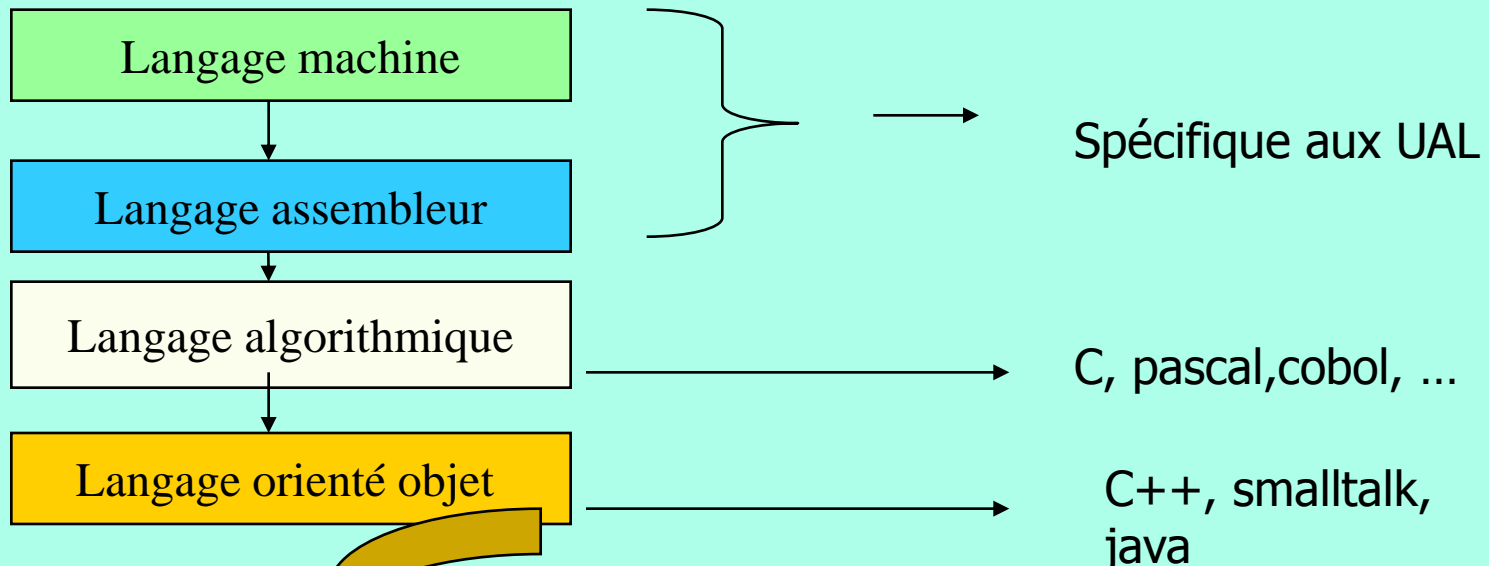
- mars : le NCSA lance MOSAIC, le premier navigateur internet (protocole http, langage html), le web décolle...
- été : Oak change d'orientation et s'adapte à la technologie internet

• 1995

- mai 1995 : projet *HotJava*., navigateur WEB, écrit par SUN en Java .

Les générations des langages de programmation

Plusieurs langages de programmation ont vu le jour :



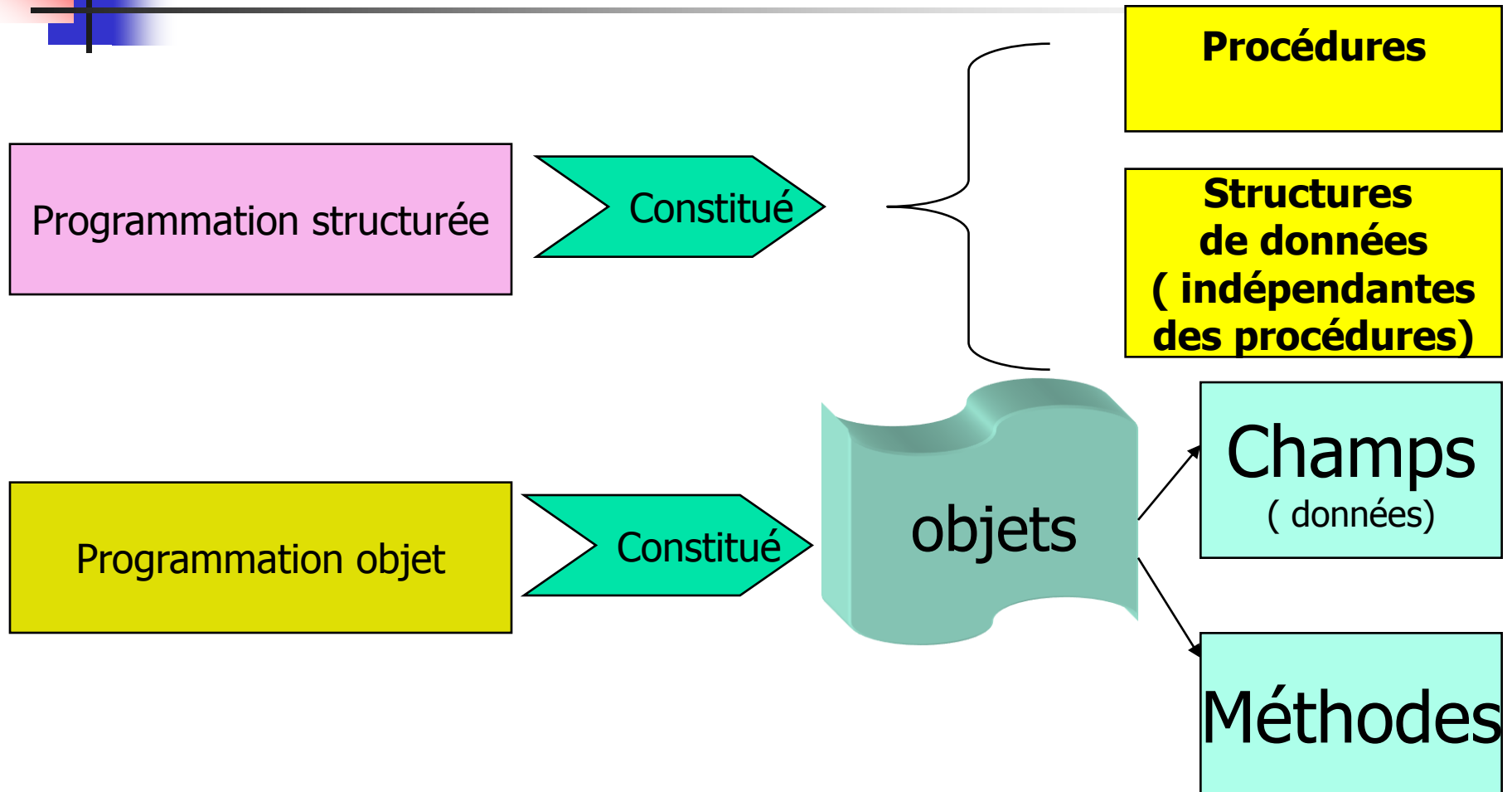
-PORTABILITÉ **EVOLUTIBITE**
- FIABILITÉ **REUTISABILITE**



Java : un langage objet

- Imprégné du C++ mais améliorant ses insuffisances
 - > gestion automatique de la mémoire (Garbage Collector)
 - > facilité de stockage des fichiers sur disque (sérialisation)
- Une gigantesque API (Application Programming Interface)
 - une librairie de classes très importante (interface graphique, réseau, web, base de données, ...)
 - portabilité sans mesure
 - langage de plus en plus utilisé et évoluant rapidement.

La Programmation Orientée Objet (P.O.O)





Concept de classe

Le concept de classe correspond simplement à la **généralisation de type** que l'on rencontre dans les langages classiques. En effet, une classe n'est rien d'autre que **la description d'un ensemble d'objets ayant une structure de données commune et disposant des mêmes méthodes.**

Les objets apparaissent alors comme **des variables d'un tel type classe** (en P.O.O, on dit aussi qu'un objet est une **instance** de sa classe). Bien entendu, seule la structure est commune , **les valeurs des champs étant propres à chaque objet.** En revanche, les méthodes sont communes à l'ensemble des objets d'une même classe.



Structure d'un Projet JAVA

Projet

Package1

Classe1

Champs ou variables , constantes

Méthodes

Classe2

.

.

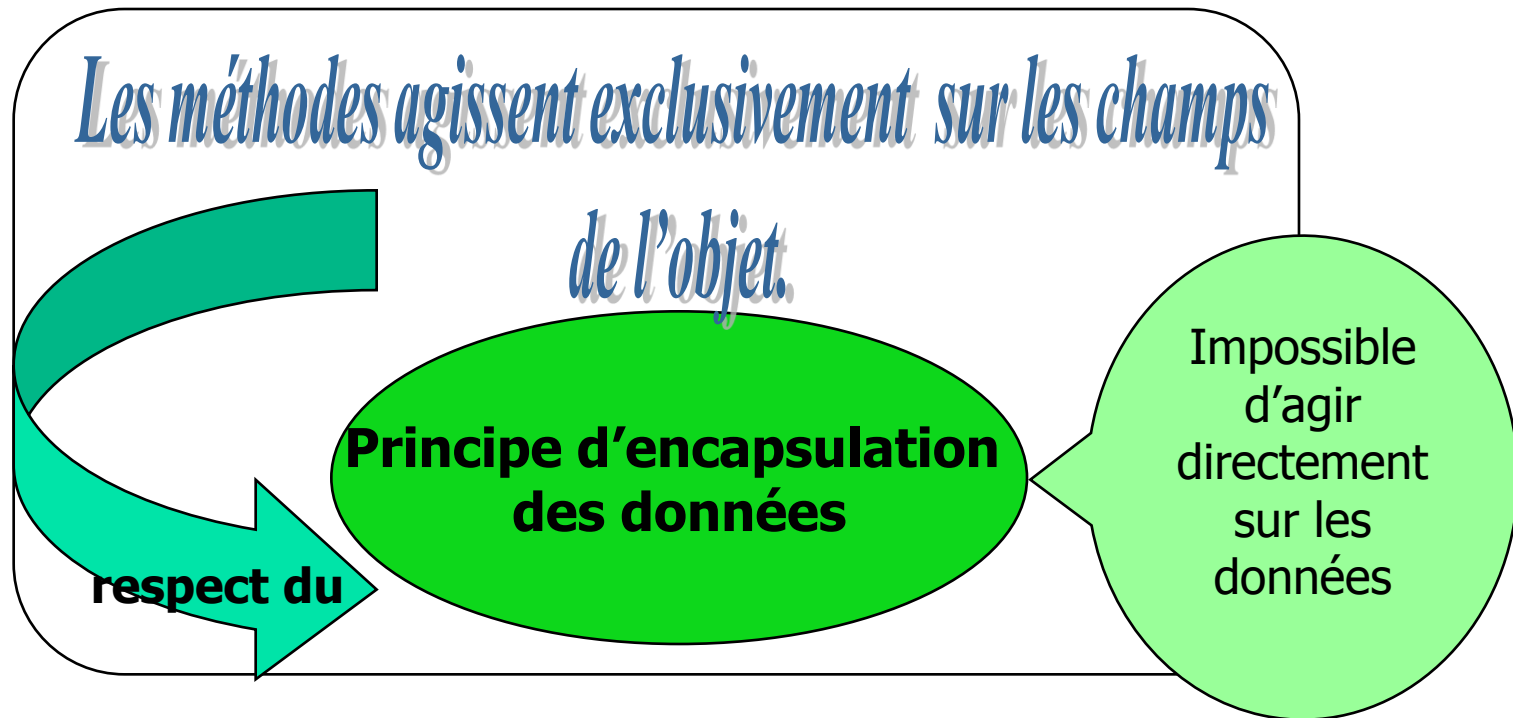
Package2

.

.



La P.O.O : l'encapsulation





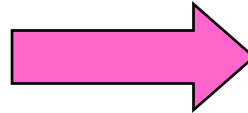
JRE (Java Runtime Environment)

- Le **JRE** contient uniquement *l'environnement d'exécution* de programmes Java. Le JDK contient lui même le JRE. Le
- **JRE** seul doit être installé sur les machines ou des applications java doivent être exécutées.
- Depuis sa version 1.2, Java a été renommé Java 2. Les numéros de versions 1.2 et 2 désignent donc la même version.
- Le JDK a été renommé J2SDK (Java 2 Software Development Kit) mais la dénomination JDK reste encore largement
- utilisée.
- Le JRE a été renommé J2RE (Java 2 Runtime Édition).
- Sun fourni le JDK, à partir de la version 1.2, sous les plate-formes Windows, Solaris et Linux.



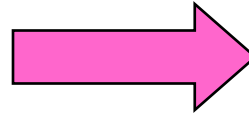
Java en 4 éditions différentes

J2ME(Java Micro Édition)



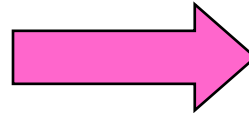
- Applications sur environnement limité
- systèmes portables
- systèmes de navigation embarqués

J2SE(Java Standard Édition)



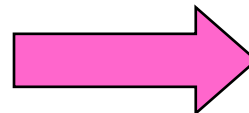
- Applications
- Applet

J2EE(Java Entreprise Édition)



- API pour applications d'entreprise (accès bases de données)
- EJB(composants métiers)
- JSP(Java Server Pages)
- Servlet (HTML dynamique)

Java Card



- Carte à puces

Applications : la console vs G.U.I

**Programme à interface
console**

L'utilisateur fournit des infos au clavier sous forme de lignes de texte. Le programme décide du séquençement des opérations. L'opérateur est sollicité au moment voulu.

**Programme à interface
graphique : GUI
(Graphical User Interface)**

L'interaction programme-opérateur se fait essentiellement via des *composants* graphiques. C'est la **programmation évènementielle** : le programme réagit à des événements provoqués par l'utilisateur.

FIN DU CHAPITRE

Module 2

Techniques de base du langage

Premier programma Java.

```
package home.user.java.essai;  
// 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 !

Structure du programme (1/2)

```
package home.user.java.essai ;
```

```
public class HelloWorld
```

```
{
```

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

```
    {
```

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

```
    }
```

```
}
```

→ en-tête de la classe

} Définition de la classe
avec une seule méthode
(la méthode *main*)



Structure du programme (2/2)

- Le mot clé *static* précise que la méthode *main* n'est pas liée à une instance (objet) particulière de la classe.
- Le paramètre *String[] args* est un tableau de chaînes de caractères qui permet de récupérer des arguments transmis au programme au moment de son lancement. Ce paramètre est **OBLIGATOIRE** en Java.
- Le mot clé *public* dans *public class* sert à définir les droits d'accès des autres Classes (en fait de leurs méthodes) à la classe . [A voir].
- Le mot clé *public* dans *public static void main* est **obligatoire** pour que votre programme s'exécute. Il s'agit d'une convention qui permet à la machine virtuelle d'accéder à la méthode *main* .

Exécution du programme (1/2)

La sauvegarde du programme se fait impérativement dans un fichier qui porte le nom `HelloWorld.java`



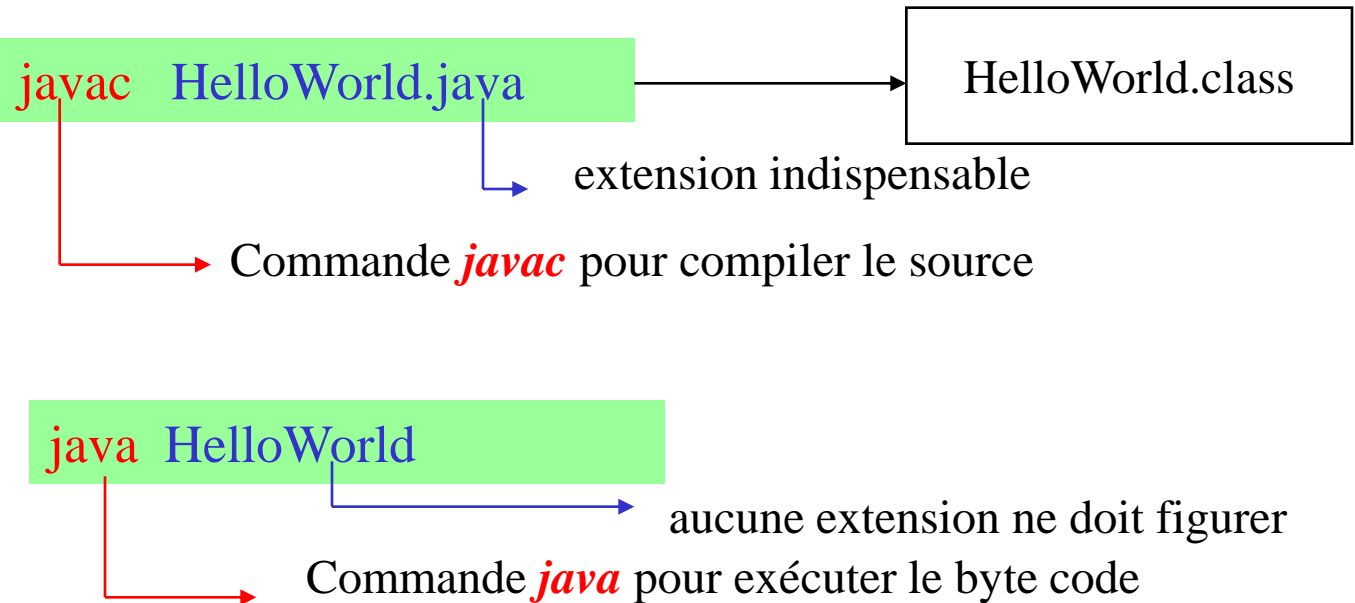
Le code source d'une classe *publique* doit toujours se trouver dans un fichier portant le même nom et possédant l'extension *.java*.

La classe contenant la méthode `main` est appelée la **classe principale** du programme. C'est cette classe qu'il faut exécuter. EN FAIT ON EXECUTE QUE LES INSTRUCTIONS DE LA METHODE `main`.

Exécution du programme (2/2)

On procède à la **COMPILATION** de ce programme pour la génération du *byte code*.

Si elle se passe bien(sans erreurs) on obtient un fichier d'extension **.class** . Ici, il s'agit de **HelloWorld.class** .



ATTENTION: en pratique, on tiendra toujours compte des variables d'environnement **PATH** et **CLASSPATH** (cf. diapos suivantes).

Règles d'écriture en Java (1/3)

Les différentes entités utilisées dans un programme (méthodes, variables, classes, objets,) sont manipulées au travers *d'identificateurs*.

Un *identificateur* est formé de *lettres* ou de *chiffres*, le premier caractère étant obligatoirement une lettre. Les lettres comprennent les majuscules A-Z et les minuscules a-z, ainsi que le caractère « souligné »(_) et le caractère \$

Exemple :

ligne *Clavier* *valeur_5* *_total* *_56* *\$total* ~~*2nombre*~~

Remarque très importante :

Java est très sensible à la casse : *ligne* \neq *Ligne* .



Règles d'écriture en Java (2/3)

Un identificateur ne peut pas appartenir à la liste des mots réservés du langage Java :

abstract	assert	boolean	break	byte
case	catch	char	class	const
continue	default	do	double	else
extends	false	final	finally	float
for	if	implements	import	instanceof
int	interface	long	native	new
null	package	private	protected	public
return	short	static	super	switch
synchronized	this	throw	throws	transient
true	try	void	volatile	

Règles d'écriture en Java (3/3)

Voici quelques conventions de codage en java

```
public class MaPremiereClasse
```

```
{ public void affiche( int argument )
```

```
{ int nombreEntier =12 , i =1;
```

```
final float NOMBRE =100;
```

```
while ( i >100)
```

```
{ System.out.println (" i = "+ i );  
  .  
  }
```

```
}
```

Nom de classe commence par une **majuscule**.

Nom de méthode , de variables ou d'attributs commence par une **minuscule**.

Nom de constante écrit tout en **majuscule**.

structures de contrôle: mettre des accolades

Indenter votre programme pour plus de lisibilité



A propos des commentaires

Commenter *toujours* les entêtes de fonctions

Un bon commentaire permet de pouvoir utiliser la fonction sans consulter le code.

- il indique à l'aide d'une phrase le rôle de la fonction en faisant intervenir le nom de tous les paramètres
 - il précise le rôle de chaque paramètre
- il indique la signification du résultat retourné
- il indique les restrictions sur la valeur des paramètres

Commenter *si nécessaire* des fragments de codes difficiles (un bon programme en contient généralement peu)

Éviter les commentaires inutiles

```
A =5; /* a prend la valeur 5 */
```

Types de commentaires en Java

```
package home.user.java.essai;
```

```
/**
```

```
 * @param autor Assane
```

```
 * @since 1.0
```

```
 */
```

```
// un premier programme
```

```
/* la version JAVA du classique  
   Hello World
```

```
 */
```

```
public class HelloWorld {
```

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

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

```
    }
```

```
}
```

/* ceci est un commentaire de documentation automatique javadoc */

// Ceci est un commentaire sur une seule ligne

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

Les opérateurs relationnels(1/2)

Opérateur	signification
<	inférieur à
<=	inférieur ou égal à
>	supérieur à
>=	supérieur ou égal à
= =	égal à
!=	différent de

Les quatre premiers(<, <=, >, >=) sont de même priorité. Les deux derniers(= = et !=) possèdent également la même priorité, mais celle-ci est inférieure à celle des précédents

Ces opérateurs sont moins prioritaires que les opérateurs arithmétiques. Ils soumettent eux aussi leurs opérandes aux promotions numériques et ajustement de type .

Les opérateurs relationnels(2/2)

Exemple :

```
public class Oper_Relat {  
    public static void main(String args [])  
    { int n = 10 ;  
      short s =10 ;  
      float x = 100;  
      double d= 200;  
      System.out.println("Affichage 1 :"+(n == s) );  
      System.out.println("Affichage 2 :"+(d <= x) );  
    }  
}
```

Affichage 1 : true
Affichage 2 : false



Les opérateurs logiques(1/3)

Java dispose d'opérateurs logiques classées par ordre de priorités décroissantes (il n'existe pas deux opérateurs de même priorité).

**Le résultat est
toujours un
booléen.**

Opérateur	Signification
!	négation
&	et
^	ou exclusif
	ou inclusif
&&	et(avec court-circuit)
	Ou inclusif(avec court-circuit)

Les opérateurs logiques (2/3)

$(a < b) \&\& (c < d)$ ou $(a < b) \& (c < d)$

prend la valeur true (vrai) si les deux expressions $a < b$ et $c < d$ sont toutes les deux vraies (true), la valeur false (faux) dans le cas contraire.

$(a < b) \parallel (c < d)$ ou $(a < b) \mid (c < d)$

prend la valeur true si l'une **au moins** des deux conditions $a < b$ et $c < d$ est vraie, la valeur false dans le cas contraire.

$(a < b) \wedge (c < d)$

prend la valeur true si **une et une seule** des deux conditions $a < b$ et $c < d$ est vraie, la valeur false dans le cas contraire.

$!(a < b)$

prend la valeur true si la condition $a < b$ est fausse, la valeur false dans le cas contraire. Cette expression possède la même valeur que $a \geq b$.



Les opérateurs logiques (3/3)

Les opérateurs de court-circuit && et || .

Ces deux opérateurs recèlent une propriété très importante: leur second opérande (figurant à droite de l'opérateur) n'est évalué que si la connaissance de sa valeur est indispensable pour décider si l'expression correspondante est vraie ou fausse.

Exemple :

```
( 15 <10 ) && ( 10 > 4) //on évalue 15 <10 , le résultat  
                        // est faux donc on n'évalue pas  
                        // 10 > 4
```

Opérateurs d'incrémentation et de décrémentation(1/2)

incrémentation

```
int i = 10 ;
```

→ post incrémentation

```
i++ ; // cette expression vaut 10  
      //mais i vaut 11
```

```
int j = 10 ;
```

```
++j ; // cette expression vaut 11  
      //et j vaut aussi 11
```

→ pré incrémentation

En fait en écrivant :

```
int n= i++ ;
```

on a :

```
n= i ;
```

```
i = i+1 ;
```

Et en écrivant :

```
int p= ++j ;
```

on a:

```
j = j+ 1 ;
```

```
p =j ;
```

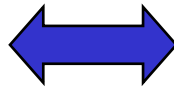
Il existe un opérateur de décrémentation notée - -



Opérateur Conditionnel

? :

```
if(ciel == bleu)
    temps = "beau"
else
    temps = "mauvais"
```



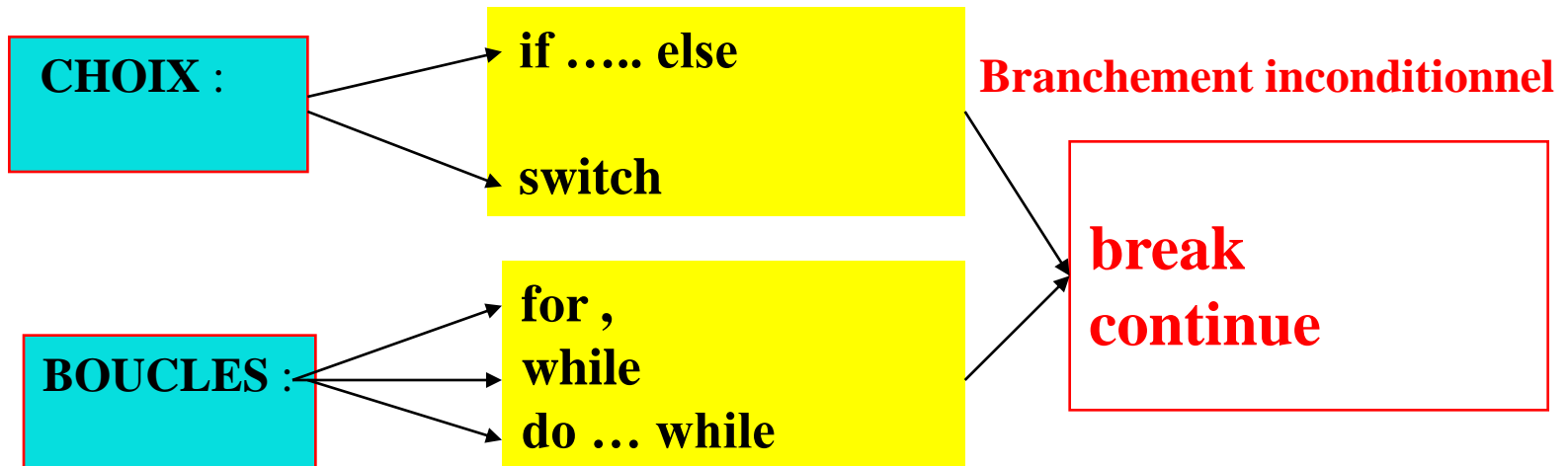
```
temps = ciel == bleu ? "beau" : "mauvais"
```


Module 3

Les structures de contrôle

Les instructions d'un programme (main) sont à priori exécutées de façon séquentielle.

Les instructions de contrôle permettent de s'affranchir de cet ordre pour effectuer des choix et des boucles.





Choix : if ...else switch

```
package home.user.java.essai ;
public class Exemple_If_Else{
int final MIN = 100;
int final Max = 1000;
int solde ;
public static void main(String args [ ])
{ if ( solde < MIN)
    System.out.println("solde insuffisant" ) ;
  else
    if (solde == MAX)
      System.out.println("solde suffisant" ) ;
}
```

```
package home.user.java.essai ;
public class Exemple_Switch{
int final MIN = 100;
int final Max = 1000;
int choix , solde;
public static void main(String args [ ])
{ switch(choix)
  { case 1: solde = MIN;
    System.out.println("solde insuffisant" )
    break;
    case 2: solde = MAX ;
    System.out.println("solde suffisant" ) ;
    break;
    default : break
  } }}
```


Syntaxes :

if ...else

switch

if (condition)

instruction_1

[else

instruction_2]

Condition booléenne (true / false)

Expressions quelconques

Les **crochets** renferment des instructions facultatives.

switch (expression)

```
{ case constante_1 : [suite_d'instruction_1 ]  
  case constante_2 : [suite_d'instruction_2 ]  
  .....  
  case constante_n : [suite_d'instruction_n ]  
  [ default : suite_d'instructions ]  
}
```

Expression de type **byte**, **char**, **short** ou **int** .

Expression **constante** d'un type compatible par affectation avec le type de **expression**

L'instruction do while

```
package home.user.java.essai ;  
import java.util.Scanner ; // importation de classe de l'API  
public class Exemple_Do_While{  
    public static void main (String args [ ])  
    { Scanner clavier = new Scanner (System.in) ;  
do  
    { System.out.println ("saisir un entier strictement positif " ) ;  
      n = clavier.nextInt ( ) ;    // saisir à partir du clavier  
      if ( n < 0) System.out.println ("la saisie est invalidée: recommencez" ) ;  
    }  
while ( (n < 0) || (n == 0) ) ;  
}}
```

do instruction

Expression quelconque

while (condition) ;

Condition booléenne

L'instruction while

```
package Assane.cours.java ;
public class Exemple_While{
public static void main(String args [ ])
{ while ( n <= 0)
  { System.out.println ( "saisir un entier strictement positif " ) ;
    n = clavier.nextInt( ) ;    // saisir à partir du clavier
    if ( n < 0) System.out.println ( "la saisie est invalidée: recommencez" ) ;
  }
}
}
```

while (condition) ;

Condition booléenne

instruction

Expression quelconque

L'instruction for

```
package cours.java ;  
public class Exemple_For{  
public static void main (String args [ ])  
{ int tab [ ] = new int [ 100] ; // tableau d'entiers de taille 100  
  for( int i = 0 ; i < 100 ; i ++ )  
  {  
  
    tab [ i ] = i + 1;  
  
  }  
}  
}
```

for ([initialisation] ; [condition] ; [incrémentation])

instruction



Saisie

```
Scanner clavier=new Scanner(System.in);  
System.out.println("Saisir la valeur de a");  
int a=clavier.nextInt();  
System.out.println("Valeur de a="+a);
```