

# M2103 – Bases de la Programmation Orientée Objets



Java – Cours 1

Introduction & Bases du Java

# Objectifs du cours

---

- Apprendre un langage orienté objet : Java
- Découvrir, par la pratique, les notions fondamentales de la Programmation Orientée Objet (POO)

# Modalités

---

- Répartition
  - ~1h-2h de Cours/TD
    - Présentation des concepts
    - Illustration avec du code
  - ~2h-3h de TP
    - Application
- Enseignant : Dr Mohammed Belkhatir
  - Bureau 2ème étage
  - E-mail : [mohammed.belkhatir@univ-lyon1.fr](mailto:mohammed.belkhatir@univ-lyon1.fr)
- Evaluation :
  - Notes TP
    - En binôme sauf TP 2, 3 et 4
    - Evaluation lettrée
    - Attention aux absences !
    - Poids : 1/3 de la note globale
  - Note Examen Papier
    - Sous la forme d'un sujet de TP
    - Poids : 2/3 de la note globale

# Enjeux de la programmation

---

- La production d'un logiciel doit remplir les critères suivants :
  - Exactitude (fonctionnement correspondant aux spécifications)
  - Robustesse (vis-à-vis des conditions anormales)
  - Evolutivité (mises à jour du logiciel)
  - Généricité, Réutilisabilité (possibilité de réutiliser des modules dans d'autres logiciels)
  - Portabilité (aptitude à fonctionner sur différentes plateformes)
  - Performances (temps d'exécution, ressources consommées)

# Programmation structurée

---

- Le langage C, étudié au S1, permet une programmation structurée, selon l'équation de Wirth :

Programme = Algorithmes + Structures de Données

- Cette approche a permis d'augmenter *exactitude* et *robustesse*...
- ... Mais n'a pas permis d'atteindre les objectifs souhaités en termes d'évolutivité et de réutilisabilité
- Origine du problème : le découplage entre structures de données et traitements qui oblige à « casser » un module dès que l'on remet en cause une structure de données.

# Apports de la POO : objet

---

- La notion d'objet autorise **l'encapsulation** d'**attributs** (données) et des **traitements** sur ces données

Objet = Méthodes + Attributs

- Exemple d'un objet **Etudiant**
- Impose de manipuler les données *via* les méthodes
- Conséquence : l'objet est uniquement caractérisé par la spécification de ces méthodes... peu importe la structure des données
- On réalise ainsi une **abstraction des données** qui facilite l'évolutivité et la réutilisabilité

# Apports de la POO : classe

---

- La notion de **classe d'objets** est une généralisation de la notion de **type** déjà rencontrée
- **Classe** : description d'un ensemble d'objets ayant les mêmes attributs et les mêmes méthodes
- Un **objet** est **une instance** (une "réalisation") de la classe à laquelle il appartient.

# Apports de la POO : héritage

---

- **L'héritage** permet de construire une nouvelle classe à partir d'une classe déjà existante.
- La nouvelle classe **hérite** de toutes les propriétés de la classe existante et lui en ajoute de nouvelles (on parle de **spécialisation**)
  - Exemple de la classe `EtudiantIUT`
- La notion **d'héritage** facilite la **réutilisabilité**
- Java permet **l'héritage multiple** : une nouvelle classe peut hériter de **plusieurs** classes existantes



# Bases de Java

---

- **Partie 1 : Éléments d'un programme Java**
  - **Code Source**
  - Données
  - Variables et constantes
  - Chaînes de caractères
  - Affichage
  - Opérateurs
- **Partie 2 : Logique d'un programme Java**
  - Sélection
  - Répétition

# Éléments d'un programme Java

## Code Source

---

- Code source dans un fichier **.java**
- Compilé pour produire un fichier **.class** pour chaque classe définie
- Un fichier **.class** doit contenir une méthode **main** appelée pour que le programme soit exécuté

# Éléments d'un programme Java

## Code Source

---

- **Commentaires**

- `//` `/*...*/` `/**...*/`

**Symboles**

- **Déclaration de classe**

- `class ClassName`

**Identificateurs**

- **Déclaration méthode principe**

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

- **Délimitations**

- `{ }` balise les blocs de code
  - `;` termine les autres instructions

**Mots-clé**

**Symboles**

# Éléments d'un programme Java

## Structure

---

### Un programme Java

- est une collection d'1 ou plusieurs classes
- chaque classe contient 1 ou plusieurs méthodes
- Une classe doit contenir une méthode nommée **main**
  - Java recherche la méthode **main** pour débiter l'exécution

### Exemple simplifié :

```
class Classe1{  
    main(String args[]) { }  
    méthode a{ }  
}
```

```
class Classe2{  
    méthode b{ }  
}
```

# Éléments d'un programme Java

## Quelques mots-clé

---

- **public**
  - Élément peut être accédé en dehors de la classe
    - À utiliser seulement lorsque nécessaire
- **static**
  - Seulement un par classe et pas un par objet
- **void**
  - Ne retourne pas de valeur

# Éléments d'un programme Java

---

- `main`
  - Il peut y avoir plusieurs classes par programme
  - Une méthode ***main*** par programme
  - Reconnu comme point de départ par l'interpréteur
- `(String[] args)`
  - Permet à l'utilisateur de passer des valeurs au programme (à l'exécution)

# Syntaxe Java

---

- **Partie 1 : Éléments d'un programme Java**
  - Code Source
  - Données
  - Variables et constantes
  - Chaînes de caractères
  - Affichage
  - Opérateurs
- **Partie 2 : Logique d'un programme Java**
  - Sélection
  - Répétition

# Données dans les programmes

## Construction des blocs

---

- Données dans la mémoire
  - Cellules distinctes consécutives
  - Avec une adresse unique
- Bits
  - 0s and 1s
- Octets
  - 8 bits
- Caractères
  - 2 octets
  - Caractères convertis en une valeur numérique  
Eg 'A' = 65, 'a' = 97

01100111



# Données dans les programmes

## Constantes

---

- Nombres
  - 16, -32.3, 0, 1134.954
- Caractères entre quotes
  - `a`, `R`, `?`, `+`, ` `
- Chaînes de caractères entre guillemets
  - "Toto"
- Séquences d'échappement
  - Exemples
    - `\\n` nouvelle ligne
    - `\\t` tabulation
    - `\\r` retour chariot
    - `\\\\` backslash
    - `\\` simple quote
    - `\\` guillemet

***La valeur d'une constante ne peut être modifiée***

# Données dans les programmes

## Types de données Java communs

Mot-clé	Description	Nb. Octets	Intervalle & exemple
int	entier	4	<b>-2,417,483,648 à +2,417,483,647</b> <b>eg 1 0 -11 5462</b>
double	réel/ virgule flottante	8	<b>-1,797,693...E308 à</b> <b>+1,797,693..E308</b> <b>eg 1.0 0.0 -11.4 5462.3789</b>
char	caractère	2	<b>eg `a` `R` `?` `+`</b> <b>` ` `n`</b>

# Variables & constantes

## Déclarations

---

- Déclarations
  - Associe un identificateur à un espace mémoire

- Variables
  - Valeurs affectées pouvant être modifiées

```
int monEntier;  
int monEntier = 5;
```

- Constantes
  - Valeurs affectées ne pouvant être modifiées

```
private static final int ma_constante = 5;
```

Déclarée comme  
variable de  
classe

Valeur  
inchangeable  
(constante)

Valeur  
affectée

# Variables & constantes

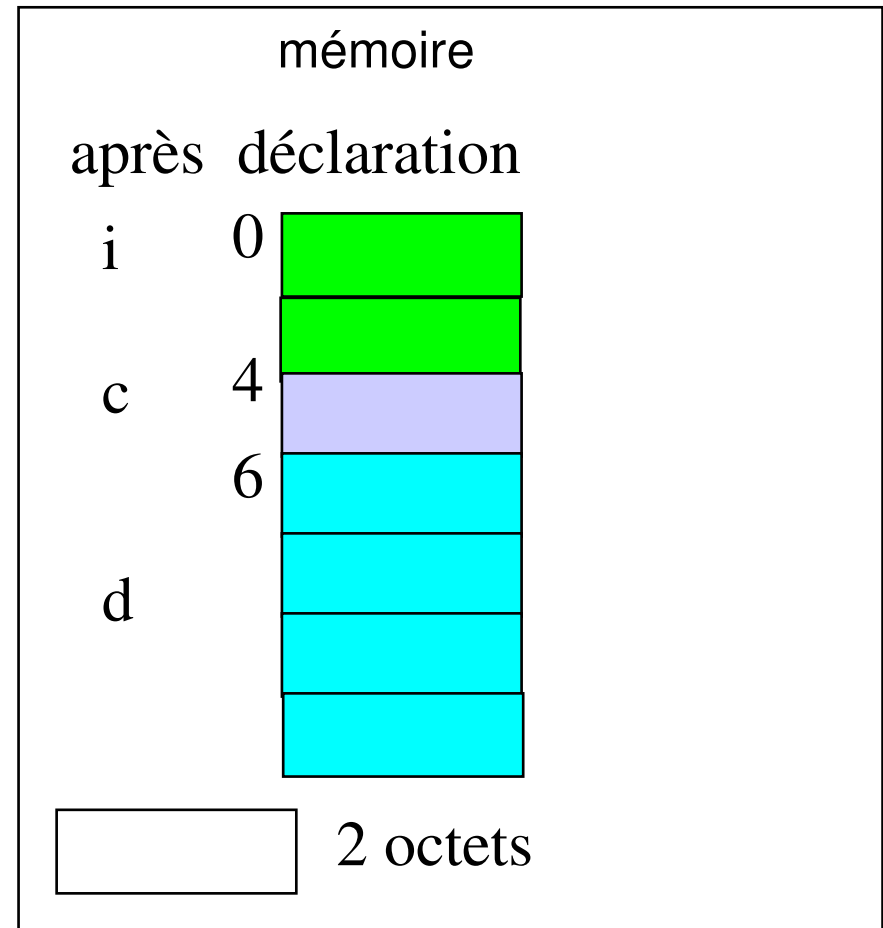
Affecter des valeurs

## *Déclarations*

int i;

char c;

double d;



# Variables & constantes

Affecter des valeurs

## *Déclarations*

int i;

char c;

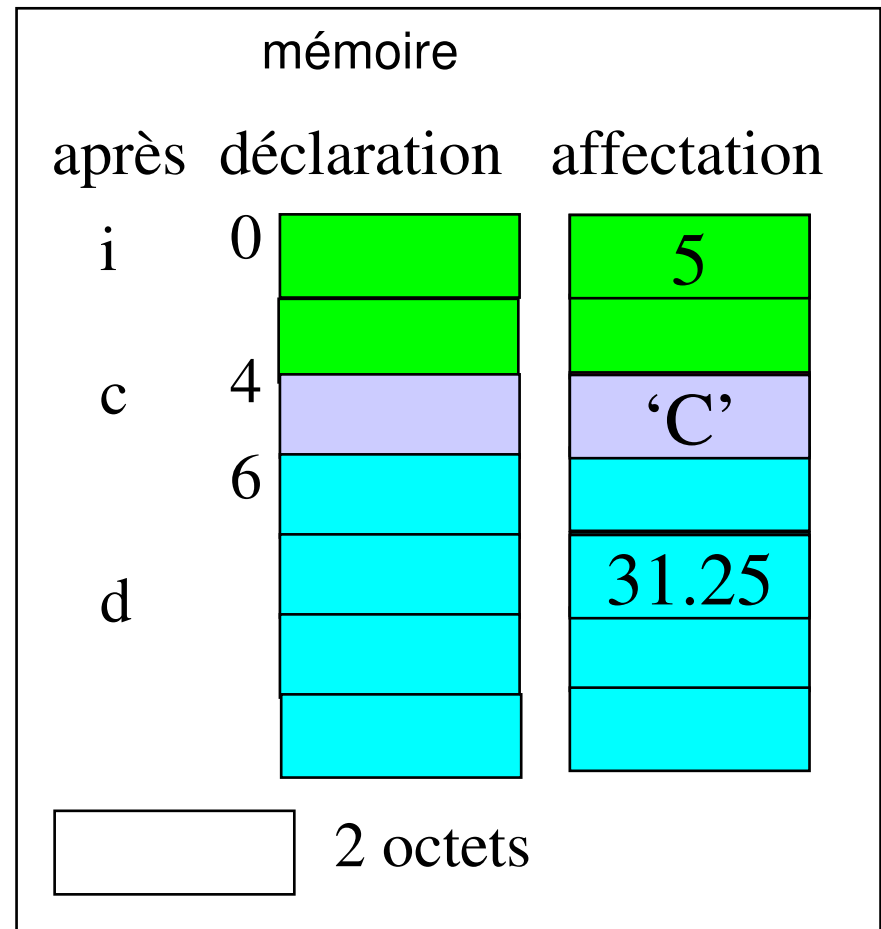
double d;

## *Affecter des valeurs*

i = 5;

c = 'C';

d = 31.25;



# Éléments d'un programme Java

## Chaînes de caractères

---

- Mots et phrases peuvent être considérés comme des tableaux de caractères
- Dans plusieurs langages, ce sont des chaînes de caractères (ou STRINGS)
- Java a une classe `String` pour traiter les chaînes de caractères
- Classe inclut plusieurs méthodes de traitement

# Éléments d'un programme Java

## Chaînes de caractères

---

```
String monNom = "Mohammed";
```

```
String tonNom = "Toto";
```

```
int longueurNom = monNom.length();
```

```
char initiale = monNom.charAt(0);
```

```
int comp = monNom.compareTo(tonNom);
```

- Explorer les autres méthodes de la classe String

# Éléments d'un programme Java

## Affichage

---

- `System.out.print();`
- `System.out.print("Hi");`

'H'	'i'
-----	-----

Affichage puis saut de ligne

- `System.out.println();`
- `System.out.println("Hi");`

'H'	'i'	'\n'
-----	-----	------



# Syntaxe Java

---

- **Partie 1 : Éléments d'un programme Java**
  - Code Source
  - Données
  - Variables et constantes
  - Chaînes de caractères
  - Affichage
  - **Opérateurs**
- **Partie 2 : Logique d'un programme Java**
  - Sélection
  - Répétition

# Opérateurs Mathématiques

+, -, \*, /, %

- Division par zéro engendre une erreur (parce qu'impossible)
- Division entière fournit un quotient entier
- Modulo entier fournit un reste entier
- Division virgule flottante fournit un résultat réel

- Division

- **entière**

- 11 / 3 = 3**

- 11 % 3 = 2**

- **Virgule flottante**

- 11.0/3.0 = 3.666667**

# Opérateurs Mathématiques

---

- $3 + 6 =$

- $3.4 - 6.1 =$

- $2 * 3 =$

- $8 / 2 =$

- $8.0 / 2.0 =$

- $8 / 8 =$

- $8 / 9 =$

- $8 / 7 =$

- $3.0 + 6.0 =$

- $0 \% 7 =$

- $0 / 7 =$

- $7 / 0 =$

- $5 \% 2.3 =$

- $8 \% 8 =$

- $8 \% 9 =$

- $8 \% 7 =$

# Opérateurs Mathématiques

---

- $3 + 6 = 9$
- $3.4 - 6.1 = 2.7$
- $2 * 3 = 6$
- $8 / 2 = 4$
- $8.0 / 2.0 = 4.0$
- $8 / 8 = 1$
- $8 / 9 = 0$
- $8 / 7 = 1$
- $3.0 + 6.0 = 9.0$
- $0 \% 7 = 0$
- $0 / 7 = 0$
- $7 / 0 = \text{erreur}$
- $5 \% 2.3 = \text{erreur}$
- $8 \% 8 = 0$
- $8 \% 9 = 8$
- $8 \% 7 = 1$

# Opérateurs Mathématiques

$+=$   $-=$   $*=$

---

$\text{total} += 3$  équivalent à  $\text{total} = \text{total} + 3$

$\text{taxe} -= 5$  équivalent à  $\text{taxe} = \text{taxe} - 5$

$\text{paye} *= 1.2$  équivalent à  $\text{paye} = \text{paye} * 1.2$

$a *= b + 3$  équivalent à  $a = a * (b + 3)$

**ATTENTION A NE PAS SACRIFIER LA LISIBILITE!**

# Opérateurs Mathématiques

++ et --

---

- Incrémentation ( ++ ) et Décrémentation ( -- )
- `total++` est équivalent à `total = total + 1`  
ou `total += 1`
- `total--` est équivalent à `total = total - 1`  
ou `total -= 1`
- -- et ++ peuvent être utilisés comme opérateurs préfixé ou infixé
  - `++total` est évalué comme la valeur après incrémentation de *total*
  - `total++` est évalué comme la valeur avant incrémentation de *total*

# Opérateurs Mathématiques

++ et --

---

## **\*Préfixé\***

*incrémente/décromente*

*avant utilisation de la valeur*

a = 6;

b = ++a - 1;

a devient **7**

b devient **6**

## **\*Postfixé\***

*incrémente/décromente*

*après utilisation de la valeur*

a = 6;

b = a++ - 1;

a devient **7**

b devient **5**

# Opérateurs Mathématiques

## Utilisation de l'incrémentation/décrémentation

---

- Plus 'efficace' que l'affectation
- Opérateurs préfixés ou postfixés peuvent être appliqués à des variables mais pas à des expressions

### ***Invalide***

- `( a + b)++;`



# Opérateurs Mathématiques

Evaluation de c

---

**int a = 1, b = 3, c = 2;**

**1. c += a;**

**2. c -= 2;**

**3. c = a + b++;**

**// b = ?**

**4. c = a + ++b;**

**// b = ?**

**5. c -= b - a;**

**6. c = --b \* a--;**

**// a = ?, b = ?**

# Opérateurs Mathématiques

Evaluation de c

```
int a = 1, b = 3, c = 2;
```

```
1. c += a;
```

```
2. c -= 2;
```

```
3. c = a + b++;
```

```
// b = ?
```

```
4. c = a + ++b;
```

```
// b = ?
```

```
5. c -= b - a;
```

```
6. c = --b * a--;
```

```
// a = ?, b = ?
```

```
1. c = 2 + 1      = 3
```

```
2. c = 2 - 2      = 0
```

```
3. c = 1 + 3      = 4
```

```
// b = 4
```

```
4. c = 1 + 4      = 5
```

```
// b = 4
```

```
5. c = 2 - (3 - 1) = 0
```

```
6. c = 2 * 1 = 2
```

```
// a = 0, b = 2
```

# Opérateurs Relationnels

---

`==` égal à

`>` supérieur à

`<` inférieur à

`>=` supérieur ou égal

`<=` inférieur ou égal

`!=` différent

- ***Attention à la différence entre '`=`' et '`==`'***

- Si x est égal à 3 et y à 6

`x != y`

`y > x`

`y >= x`

`x < y`

`x <= y`

- ASCII majuscules < minuscules

``T` < `b``

``b` != `B``

# Opérateurs Relationnels

## Expressions

- Les opérateurs relationnels peuvent comparer des constantes, variables et expressions arithmétiques

x	y	Expression	Résultat
3	2	$x + 3 \leq y * 5$	true
20	4	$x + 3 \leq y * 5$	false
7	2	$x + 3 == y * 5$	true
7	1	$x + 3 != y * 5$	true

# Opérateurs Logiques

---

binaires : **&&** (AND) , **||** (OR)

unaire : **!** (NOT)

- utilisés en règle générale sur des variables booléennes et des expressions logiques
- peuvent être utilisés sur n'importe quel type de données ainsi que sur des expressions arithmétiques
- e.g. `conducteur = ((age >= 18) && (aPermis));`

# Opérateurs Logiques

Table de vérité

---

x	y	$x \& y$	$x \parallel y$	$\neg x$
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

# Opérateurs Logiques

Evaluer les expressions

---

X=true, Y=false, Z=true

- `X&&Y||X&&Z`
- `(X||Y)&&(!X||Z)`
- `X||Y&&Z`
- `!(X||Y)&&Z`

# Opérateurs Logiques

Evaluer les expressions

---

X=true, Y=false, Z=true

- |                   |         |
|-------------------|---------|
| ■ X&&Y  X&&Z      | ■ true  |
| ■ (X  Y)&&(!X  Z) | ■ true  |
| ■ X  Y&&Z         | ■ true  |
| ■ !(X  Y)&&Z      | ■ false |



# Opérateurs Logiques

Convertir les phrases suivantes en expressions logiques

---

- *maLettre* n'est pas le S
- *maLettre* et *taLettre* sont le S
- *tonAge* est 19 ou 20
- *monAge* est entre 15 et 40

# Opérateurs Logiques

Convertir les phrases suivantes en expressions logiques

---

- *maLettre* n'est pas le S
- *maLettre* et *taLettre* sont le S
- *tonAge* est 19 ou 20
- *monAge* est entre 15 et 40
- $(\text{maLettre} \neq \text{'S'})$
- $(\text{maLettre} == \text{'S'}) \ \&\& \ (\text{taLettre} == \text{'S'})$
- $(\text{tonAge} == 19) \ || \ (\text{tonAge} == 20)$
- $(15 \leq \text{monAge}) \ \&\& \ (\text{monAge} \leq 40)$

# Opérateur de cast

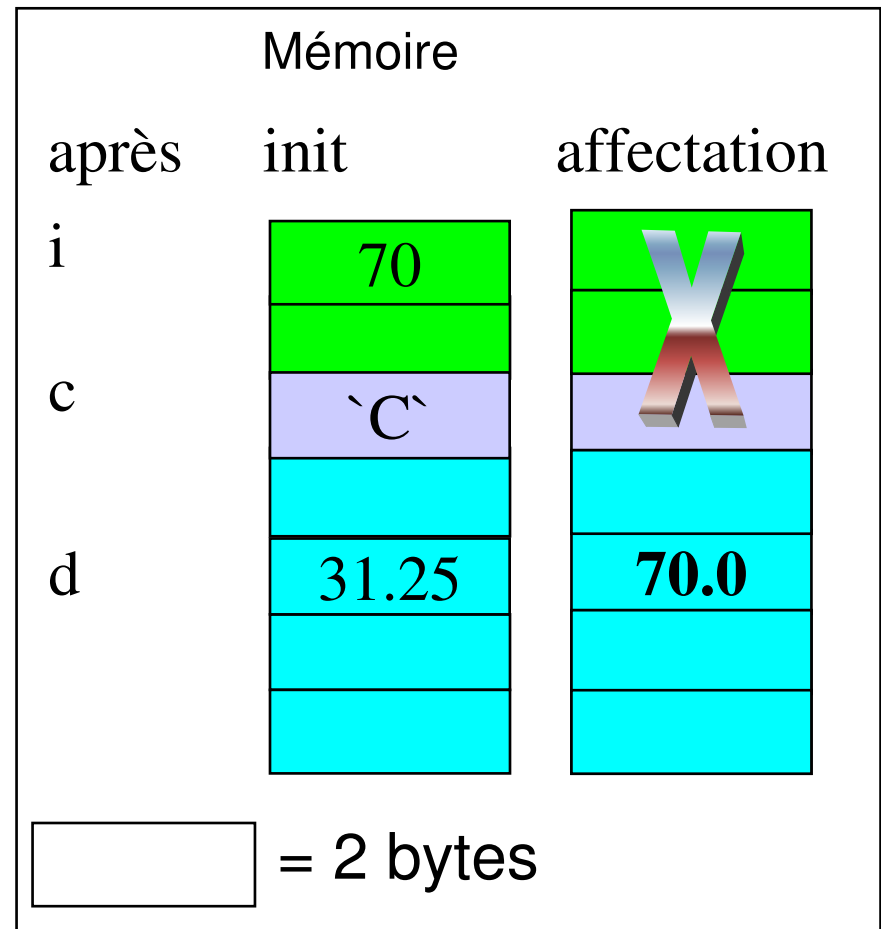
```
int i = 70;
```

```
char c = 'C';
```

```
double d = 31.25;
```

Qu'est-ce qu'il se passe si nous faisons :

- `i = d;`
- `c = i;`
- `d = i;`



# Opérateur de cast (*type*)

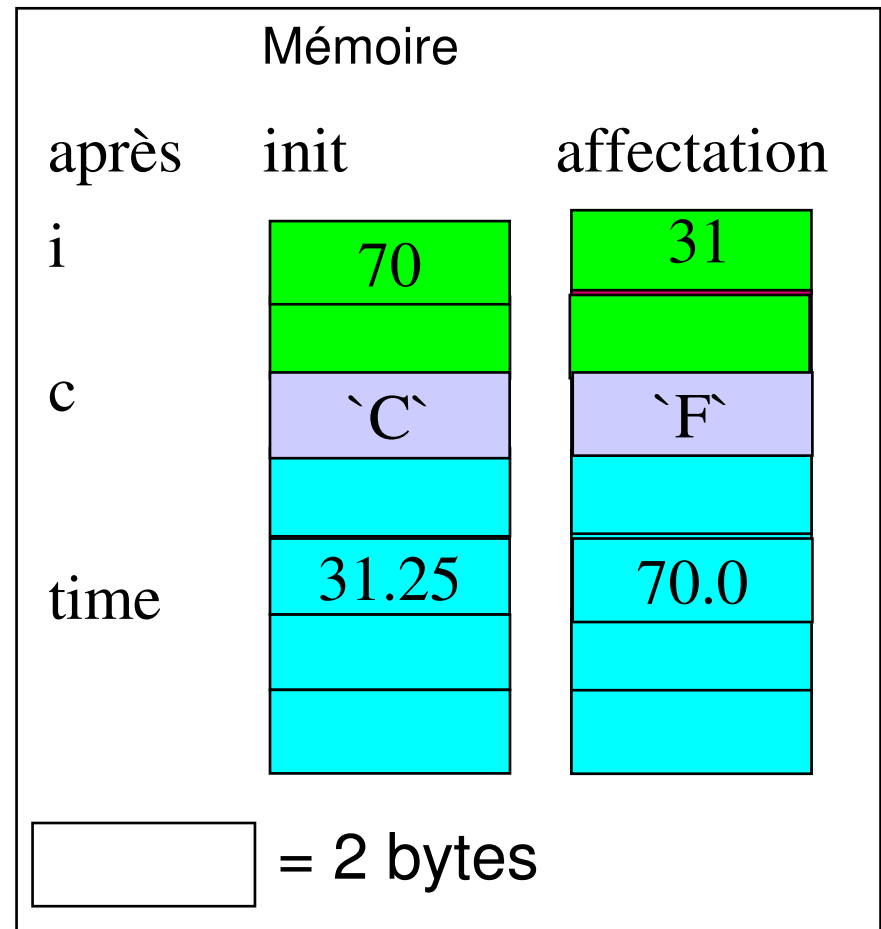
```
int i = 70;
```

```
char c = 'C';
```

```
double d = 31.25;
```

Qu'est-ce qu'il se passe si nous faisons :

- `i = (int)d;`
- `c = (char)i;`
- `d = i;`
- Toujours possible d'affecter une valeur pour un type utilisant plus d'octets sans avoir besoin d'utiliser un cast.



# Opérateur conditionnel

? :

---

- Peut être utilisé à la place d'un 'if...else'

**max = (a > b) ? a : b;**

est équivalent à

**if (a>b)**

**max = a;**

**else**

**max = b ;**

# Opérateur virgule

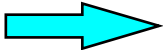
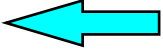
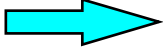

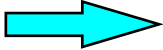

,

---

- `int i = 10, j = 12;`
- Des affectations séparées par une virgule sont évaluées de la gauche vers la droite.
- A éviter dans des situations 'compliquées' (lisibilité).

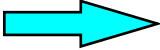
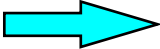
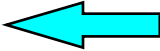
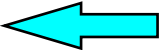
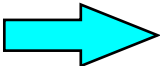
# Priorité des opérateurs (1/2)

---

<u>Opérateur</u>	<u>Type</u>	<u>Associativité</u>
()	parenthèses	
++ -- + - !	unaire (eg -a)	
* / %	multiplicatif	
+ -	additif (eg a-b)	
< <= > >=	relationnel	
== !=	égalité	

# Priorité des opérateurs (2/2)

---

<u>Opérateur</u>	<u>Type</u>	<u>Associativité</u>
&&	AND logique	
	OR logique	
?:	conditionnel	
= += -= *= /= %=	affectation	
,	virgule	



# Exemples

---

**Evaluer c:**

**int a = 1, b = 3, c = 2;**

**1. if (a <= b && b==3)**

**c %=2;**

**2.**

**c = (a>=b) ? a+ 5: a++;**

**3. c = -(b-a);**

**Traduire l'expression suivante à  
l'aide d'une seule affectation :**

**if (a > b)**

**c = 20;**

**else**

**c = 10;**

# Exemples

---

**Evaluer c:**

**int a = 1, b = 3, c = 2;**

**1. if (a <= b && b==3)**

**c %=2;**

**2.**

**c = (a>=b) ? a+ 5: a++;**

**3. c = -(b-a);**

**1.c= 0    2. c=1 (a = 2)    3. c= -2**

**Traduire l'expression suivante à l'aide d'une seule affectation :**

**if (a > b)**

**c = 20;**

**else**

**c = 10;**

**c = (a>b) ? 20 : 10;**

# Éléments d'un programme Java

---

- Partie 1 : Éléments d'un programme Java
  - Code Source
  - Données
  - Variables et constantes
  - Chaînes de caractères
  - Affichage
  - Opérateurs
- Partie 2 : Logique d'un programme Java
  - Sélection
  - Répétition

# Sélection

## IF..THEN

---

IF condition, THEN action; sinon ne fait rien.

```
if (noteExam < SEUIL && noteTPs > 18) {  
    //bloc exécuté si la condition est TRUE  
    System.out.println("Plagiat possible!");  
    noteTPs *= 0.75;  
}  
//instruction exécutée dans les deux cas  
noteTotale= (2*noteExam + noteTP)/3;
```

# Sélection

## IF..THEN..ELSE

IF condition, THEN action; ELSE action alternative

```
if (noteExam < SEUIL && noteTPs > 18) {  
    //bloc exécuté si la condition est TRUE  
    System.out.println("Plagiat possible!");  
    noteTotale = 0;  
}  
else {  
    //bloc exécuté si la condition est FALSE  
    noteTotale = (2*noteExam + noteTP)/3;  
}
```

# Sélection

IFs

---

## Ifs multiples

```
noteTotale = 0;
if ((noteExam < SEUIL) && (noteTPs > 18)) {
    System.out.println("Plagiat possible!");
}
else if ((noteExam > 18) && (noteTPs < SEUIL)) {
    System.out.println("criteres non remplis!");
}
else {
    noteTotale = (2*noteExam + noteTPs)/3;
}
```

# Sélection

## Case

---

```
switch (chiffreRomain) {  
    case 'I' : valeur = 1;  
        break;  
    case 'V' : valeur = 5;  
        break;  
    case 'X' : valeur = 10 ;  
        break;  
    case 'L' : valeur = 50 ;  
        break;  
    case 'C' : valeur = 100 ;  
        break;  
    case 'D' : valeur = 500 ;  
        break;  
    case 'M' : valeur = 1000 ;  
        break;  
    default :  valeur = 0;  
}
```

En Java, la variable testée:

- Doit être évaluée en tant qu'entier unique (les chars sont OK)
- Ne peut consister en un intervalle de valeurs

# Éléments d'un programme Java

---

- Partie 1 : Éléments d'un programme Java
  - Code Source
  - Données
  - Variables et constantes
  - Chaînes de caractères
  - Affichage
  - Opérateurs
- Partie 2 : Logique d'un programme Java
  - Sélection
  - Répétition



# Répétition

Boucles for (1/2)

---

**for** (début ; test ; pas)

*où début, test et pas sont des expressions.*

E.g.

```
for (ctr = 1; ctr <= 10; ctr ++){  
    System.out.println(ctr * ctr);  
}
```

Aussi longtemps que la condition de test reste vérifiée, les instructions à l'intérieur de la boucle s'exécutent.

# Répétition

## Boucles for (2/2)

---

- Utilisées lorsque l'on connaît le nombre de répétitions requises
- Les boucles imbriquées sont courantes

```
for (ligne=0; ligne<4; ligne++) {  
    for (col=0; col<8; col++) {  
        System.out.print("\t" + col * ligne);  
    }  
    System.out.println();  
}
```

# Répétition

## Exercices sur les boucles for

---

1. Ecrire des boucles **for** permettant d'imprimer à l'écran

a) 1 2 3 4

b) 1

2

3

4

2. Fournir la trace de la portion de code suivante

```
for (a=4; a>=1; a--) {  
    for(b=a; b>=1; b--)  
        System.out.print(b + " ");  
    System.out.println(a);  
}
```

# Répétition

## Exercices sur les boucles for

---

1.

```
for ( i = 1; i < 5; i++)
```

```
    System.out.print(i + " ");
```

```
for (i = 1; i < 5; i++)
```

```
    System.out.println(" " + i);
```

2.

4 3 2 1 4

3 2 1 3

2 1 2

1 1

# Répétition

## Boucles while

---

- si *Expression* est TRUE, alors le corps est exécuté
- si *Expression* est FALSE, le corps est ignoré
- si *Expression* est FALSE dès le départ, la boucle n'est jamais exécutée

```
while (Expression)
```

```
{
```

```
    instruction 1
```

```
    ....
```

```
    instruction n
```

```
}
```

corps

# Répétition

## Boucles do while

---

- Exécution des instructions entre le 'do' et le 'while' aussi longtemps que *Expression* est TRUE lorsque la fin de la boucle est atteinte
- Attention à la syntaxe : se termine par un ';

```
do {  
    instruction 1;  
    ....  
    instruction n;  
} while (Expression);
```

# Répétition

## while vs do while

---

- ***Solution while***

***Evalue la variable de test avant d'entrer dans la boucle***

***Met à jour la variable de test avant de sortir de la boucle***

```
inCh = User.readChar();  
while (inCh != 'n') {  
    .....  
    inCh = User.readChar();  
}
```

- ***Solution do while***

***Met à jour la variable de test en entrant dans la boucle***

***Test sur la fin de la condition à l'intérieur la boucle***

```
do {  
    inCh = User.readChar();  
    if(inCh != 'n'){  
        .....  
    }  
} while (inCh != 'n');
```

# Attention

---

- Travail à la maison
  - Transparents du cours
  - TP
- Minimum 2h+ (plus pour les TPs exigeants)