

Module M15

La programmation orientée objet JAVA

Chapitre 1

Programmation classique avec JAVA

Programmation **classique** avec **JAVA**

Plan du cours

1. Généralités sur la programmation
2. Environnement d'exécution de JAVA
3. Premier programme en JAVA
4. Les données en JAVA– les primitives
5. Les tableaux de données
6. Les opérateurs
7. Les structures de contrôle
8. Les méthodes et la surcharge des méthodes

Java

- **Java**: Langage de programmation **orienté objet** créé par James Gosling et Patrick Naughton, employés de Sun Microsystems. Il est présenté au public en 1995.
- Java **reprend en grande partie la syntaxe du langage C++**.
- Au contraire de C++, Java assure plusieurs exigences:
 - **Portabilité totale** quelque soit l'OS à travers l'usage de la **JVM**.
 - **Meilleure gestion de la mémoire** à travers le « ramasse miette ».
 - **Sécurité davantage** à travers l'usage de la **JVM**.
 - Renforcement de **l'orienté objet**.

Java – Environnement

- **JDK**: Java Development Kit (Kit de développement en java). Désigne un ensemble de bibliothèques nécessaires à la programmation avec java.
- **Avec le temps, plusieurs éditions de JDK sont apparues:**
 - **JSDK** : Java Standard Development Kit, pour développer les application DeskTop
 - **JME** : Java Mobile Edition, pour développer les applications pour les téléphones portables
 - **JEE** : Java Entreprise Edition, pour développer les applications qui vont s'exécuter dans un serveur d'application JEE (Web Sphere Web Logic, JBoss).
 - **JCA** : Java Card Editon, pour développer les applications qui vont s'exécuter dans des cartes à puces.

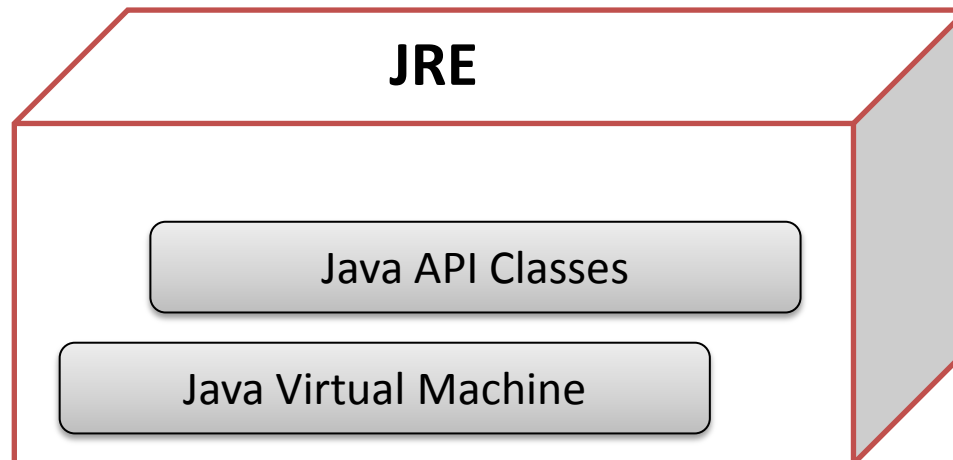
Java – Environnement

- Les programmes nécessaires au développement java sont placés dans le répertoire `C:\Program Files\Java\jdk1.7.0_21\bin\` à savoir:
 - **javac.exe** : Compilateur java.
 - **java.exe** : Interpréteur du bytecode java.
 - **appletviewer.exe** : Pour tester les applets java.
 - **Jdb.exe** : Débogueur java.
 - **Javap.exe** : désassembleur du bytecode.
 - **Javadoc.exe** : Générer la documentation de vos programmes java.
 - **Javah.exe** : Permet de lier des programmes Java avec des méthodes natives, écrites dans un autre langage et dépendant du système.
 - **jar.exe** : Permet de compresser les classes Java ainsi que tous les fichiers nécessaires à l'exécution d'un programme (graphiques, sons, etc.). Il permet en particulier d'optimiser le chargement des applets sur Internet.
 - **jarsigner.exe** : Un utilitaire permettant de signer les fichiers archives produits par **jar.exe**.

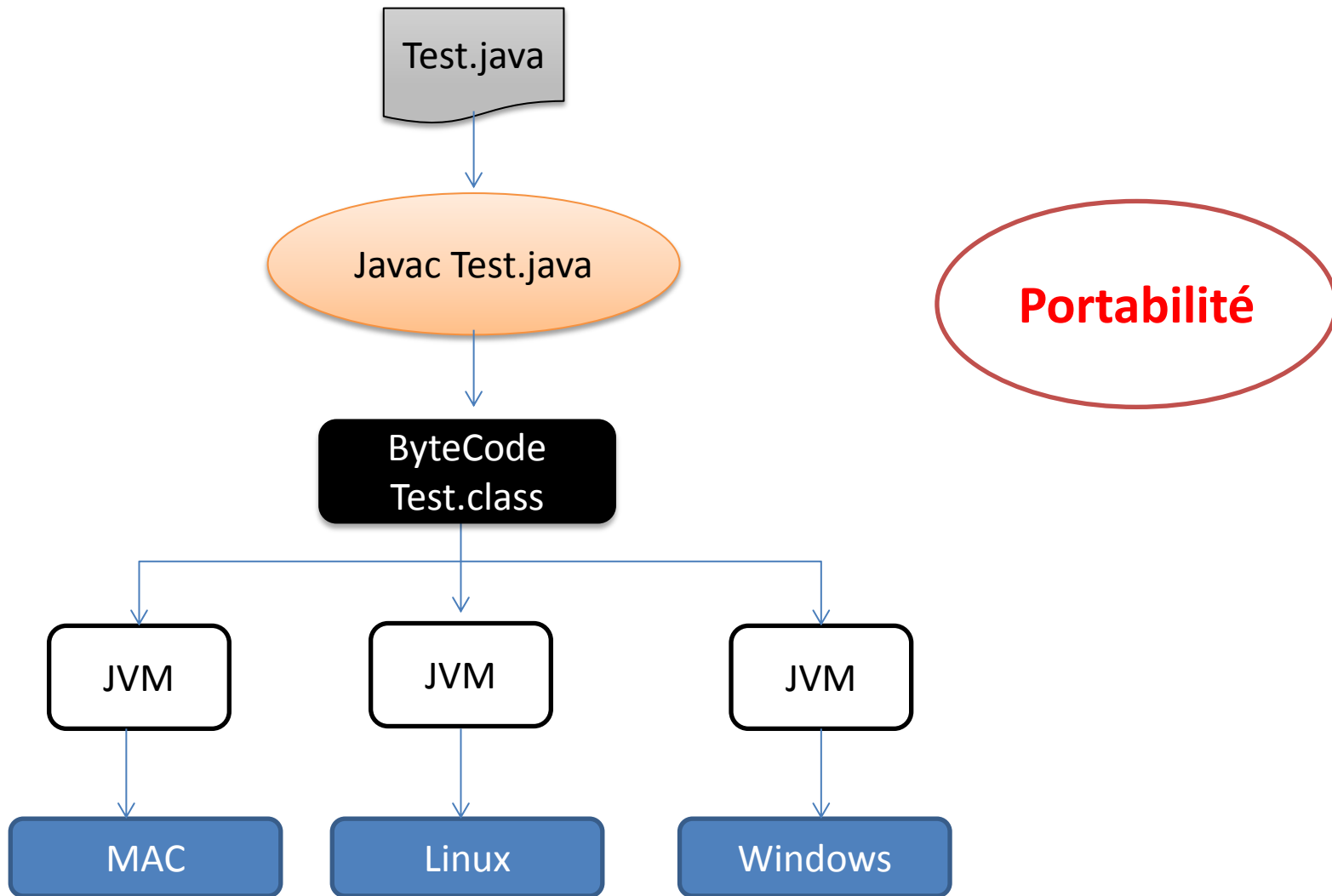
Java > jdk1.7.0_21 > bin					Rechercher dans : bin
Ouvrir Graver Nouveau dossier					
Nom	Modifié le	Type	Taille		
appletviewer.exe	21/04/2013 14:27	Application	15 Ko		
apt.exe	21/04/2013 14:27	Application	15 Ko		
extcheck.exe	21/04/2013 14:27	Application	15 Ko		
idlj.exe	21/04/2013 14:27	Application	15 Ko		
jabswitch.exe	21/04/2013 14:27	Application	54 Ko		
jar.exe	21/04/2013 14:27	Application	15 Ko		
jarsigner.exe	21/04/2013 14:27	Application	15 Ko		
java.exe	21/04/2013 14:27	Application	184 Ko		
javac.exe	21/04/2013 14:27	Application	15 Ko		
javadoc.exe	21/04/2013 14:27	Application	15 Ko		
javafxpackager.exe	21/04/2013 14:27	Application	79 Ko		
javah.exe	21/04/2013 14:27	Application	15 Ko		
javap.exe	21/04/2013 14:27	Application	15 Ko		
java-rmi.exe	21/04/2013 14:27	Application	15 Ko		
javaw.exe	21/04/2013 14:27	Application	185 Ko		
javaws.exe	21/04/2013 14:27	Application	304 Ko		
jcmd.exe	21/04/2013 14:27	Application	15 Ko		
jconsole.exe	21/04/2013 14:27	Application	16 Ko		
jdb.exe	21/04/2013 14:27	Application	15 Ko		
jhat.exe	21/04/2013 14:27	Application	15 Ko		
jinfo.exe	21/04/2013 14:27	Application	16 Ko		
jli.dll	21/04/2013 14:27	Extension de l'app...	154 Ko		
jmap.exe	21/04/2013 14:27	Application	16 Ko		
jps.exe	21/04/2013 14:27	Application	15 Ko		
jrunscript.exe	21/04/2013 14:27	Application	15 Ko		

Java – Environnement

- **JRE**: *Java Runtime Environment* (Environnement d'exécution java). Considéré comme une plateforme informatique au même titre qu'un système d'exploitation qui permet aux programmes java de s'exécuter sur différents environnements.
- **Machine virtuelle (JVM)**: Machine informatique fictive. Elle exécute des programmes compilés sous forme de **bytecode** Java.
- **Java API Classes**: contient les bibliothèques standards de java

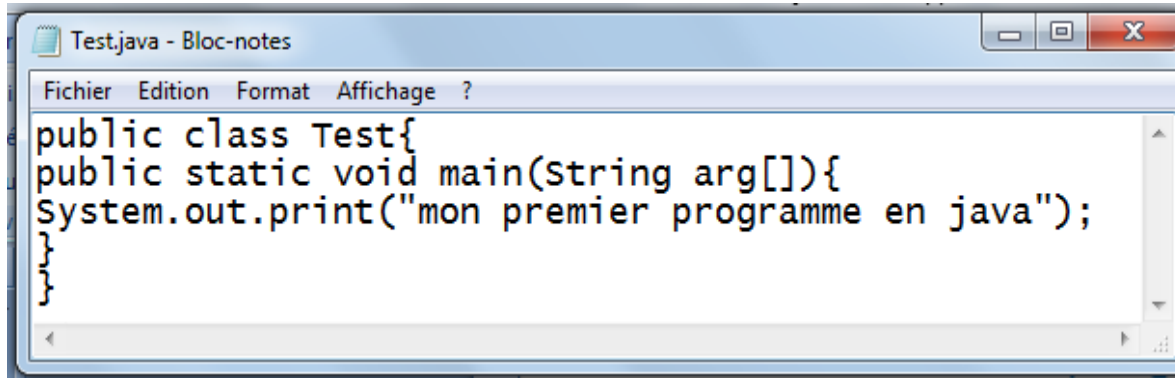


Java – Environnement d'exécution



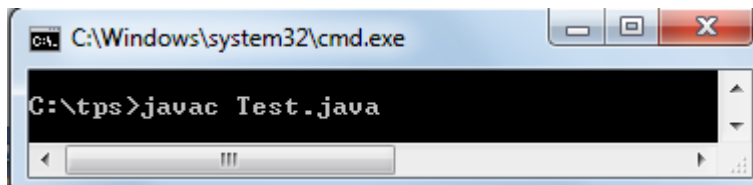
Java – Premier programme (sans IDE)

- Ecrire mon premier programme:



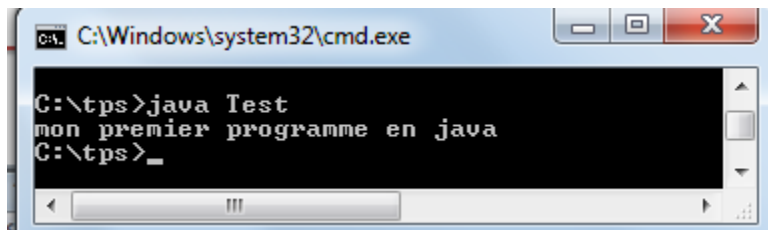
```
Test.java - Bloc-notes
Fichier  Edition  Format  Affichage  ?
public class Test{
public static void main(String arg[]){
System.out.print("mon premier programme en java");
}
```

- Compiler mon premier programme:



```
C:\Windows\system32\cmd.exe
C:\tps>javac Test.java
```

- Exécuter mon premier programme:



```
C:\Windows\system32\cmd.exe
C:\tps>java Test
mon premier programme en java
C:\tps>_
```

Création de **Test.java**
Avec **Bloc-notes**
(Code source)



Compilation de Test.java
Avec **javac Test.java**
Crée **Test.class**
(ByteCode)



Exécution de Test.class
Avec **java Test**
Exécute le programme

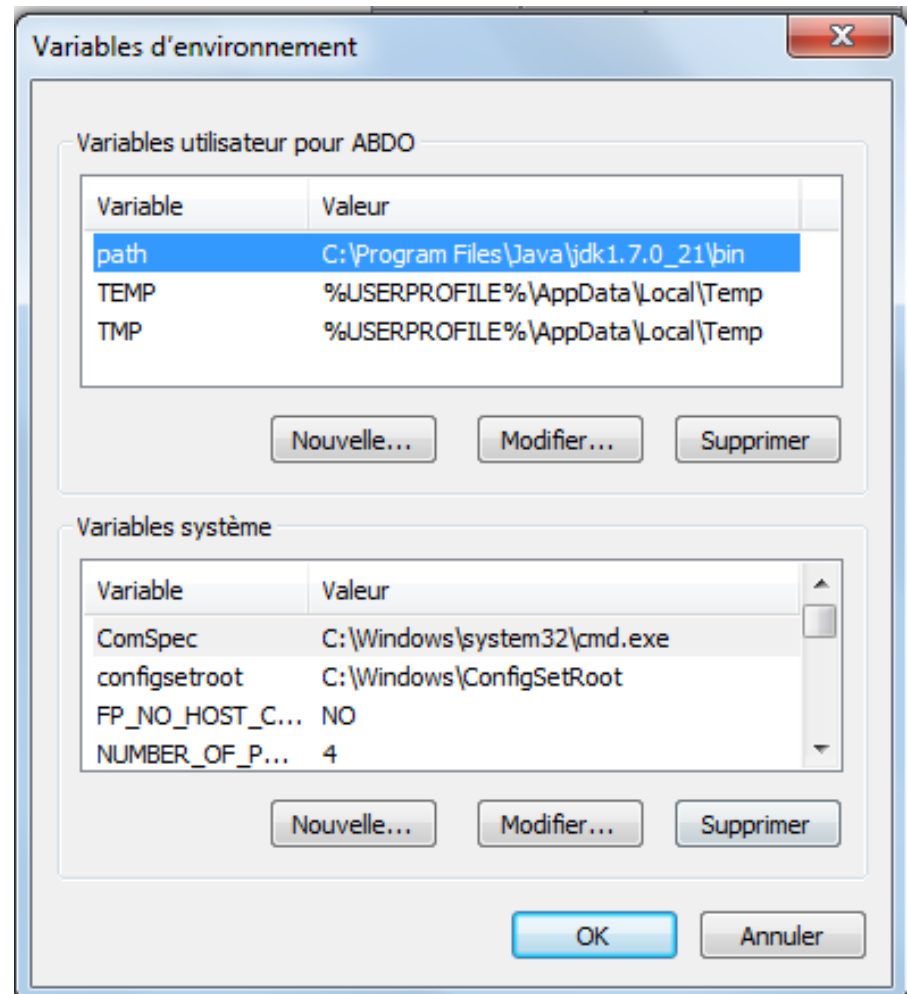
Java – Premier programme (sans IDE)

Accès aux exécutables de JDK

Définir la variable d'environnement **path** qui indique le chemin d'accès aux programmes exécutables : Cette variable path devrait contenir le chemin du JDK utilisé:

`C:\Program Files\Java\jdk1.7.0_21\bin`

Quand elle exécute une application java, la JVM consulte la variable d'environnement **classpath** qui contient le chemin d'accès aux classes java utilisées par cette application.



Java – Premier programme (sans IDE)

Remarques

- Le nom du fichier java doit être le même que celui de la classe qui contient la fonction principale main.
- Pour compiler le programme source, il faut faire appel au programme javac.exe qui se trouve dans le dossier `C:\Program Files\Java\jdk1.7.0_21\bin`
- Pour rendre accessible ce programme depuis n'importe quel répertoire, il faut ajouter la commande : path `C:\Program Files\Java\jdk1.7.0_21\bin`

javac Test.java

- Après compilation du programme Test.java, il y a génération du fichier Test.class qui représente le ByteCode du programme
- Pour exécuter ce programme en byte code, il faut faire appel au programme java.exe qui représente l'interpréter du bytecode.

java Test

Outils de développement

Pour la création des programmes en Java, il est préférable de:
Utiliser un éditeur conçu pour la programmation java

- Ultraedit, [JCreator](#),
- Eclipse, environnement de développement java le plus préféré pour les développeurs java. Il est gratuit et ouvert.
- Autres IDE java :
 - JDevlopper de Oracle
 - JBuilder de Borland.
 - NetBeans

Java – Les données primitives

Java dispose des primitives suivantes :

Type	Signification	Taille (en octets)	Plage de valeurs acceptées
char	Caractère Unicode	2	0 à 65535
byte	Entier très court	1	-128 à 127
short	Entier court	2	-32 768 à 32 767
int	Entier	4	-2 147 483 648 à + 2 147 483 647
long	Entier long	8	$-9,223 \times 10^{18}$ à $9,223 \times 10^{18}$
float	Nombre réel simple	4	1.4×10^{-45} à 3.4×10^{38}
double	Nombre réel double	8	$4,9 \times 10^{-324}$ à $1,8 \times 10^{308}$
boolean	Valeur logique (booléen)	1	true (vrai), ou false (faux)

Java – Les données primitives

Utilisation des primitives

Les primitives sont utilisées de façon très simple. Elles doivent être déclarées avec une syntaxe similaire au langage C, par exemple :

```
int i;
```

```
char c;
```

```
boolean fini;
```

Les primitives peuvent être initialisées en même temps que la déclaration.

```
int i = 12;
```

```
char c = 'a';
```

```
boolean fini = true;
```

Java – Les données primitives

Utilisation des primitives

- **Comment choisir le nom d'une variable:**

Pour respecter la typologie de java, les nom des variables commencent toujours par un caractère en minuscule et pour indiquer un séparateur de mots, on utilise les majuscules.

Exemples:

```
int nbPersonnes;  
boolean estFini;
```

- **Valeurs par défaut des primitives:**

Toutes les primitives de type numérique utilisées comme membres d'un objet sont initialisées à la valeur 0. Le type boolean est initialisé à la valeur **false**.

- **Définir une constante:**

```
final double pi=3.14159 // impossible de la changer par la suite
```


Java – Les données primitives

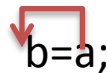
Casting des primitives

Sur-casting : Le sur-casting peut se faire implicitement ou explicitement.

Exemples :

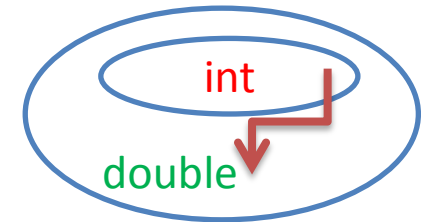
```
int a=6; // le type int est codé sur 32 bits  
long b; // le type long est codé sur 64 bits
```

Casting implicite :



b=a;

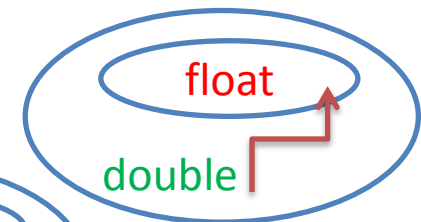
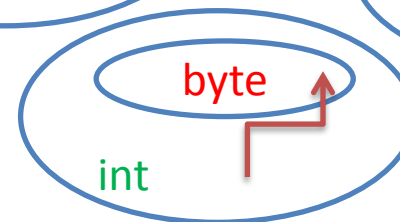
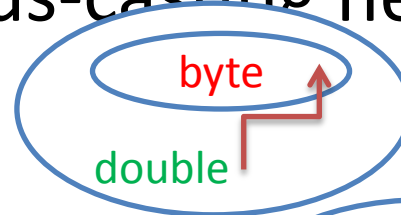
Casting explicite

b=(long)a;



Sous-Casting : Le sous-casting ne peut se faire qu'explicitement.


1 : float a = (float)5.5;
2 : double b = 8;
3 : byte c = (byte)b;
3. int d=4;
4 : byte e=(byte)d;



Java – package

Package (cf Chapitre POO)

- Un package est un ensemble de dossiers et de sous-dossiers.
- C'est un mécanisme d'organiser les classes java de l'ensemble projet (un projet peut avoir plusieurs classes)
- Il est utilisé pour organiser les classes par catégorie ou offrant la même nature de fonctionnalités
- Il est possible de le compresser dans un dossier jar
- Son mode d'emploi:
 - instruction écrite en début de la classe ayant la forme
`package pack1;` // le fichier .class doit appartenir au dossier pack1
`package com.gl.java;` // le fichier .class doit appartenir au dossier com\gl\java
 - Le nom du package est soumis à une convention de nommage.
 - Si vous voulez utiliser un mot clé Java dans le nom de votre package, vous devez le faire suivre d'un underscore (« _ »).

Java – package

Utiliser les classes (cf Chapitre POO) d'un package

- Il est possible d'utiliser les fonctionnalités offertes par une autre classe qui existe dans un autre package du même projet
- Pour ce faire, on utilise l'instruction `pour utiliser une classe précise (nom_classe)`
`import chemin_package.nom_classe;`
- Ou bien, `pour utiliser la plus part ou la totalité des classes d'un package`
`import chemin_package`
- Exemple (la classe Scanner permet de lire les données au clavier)

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        System.out.print("Donner un nombre:");
        Scanner clavier=new Scanner(System.in);
        int nb=clavier.nextInt();
        System.out.println("le nombre lu au clavier est: " +nb);
    }
}
```

**Programme permettant
de lire des entrées de
type entier**

Java – Les tableaux de données primitives

- En java, un tableau est une structure de données contenant un groupe d'éléments ayant le même type.
- Lors de la définition d'un tableau, les `[]` précisent qu'il s'agit d'un tableau.
- Les `[]` peuvent être placés avant ou après le nom du tableau.

`type_primitif T[]` précise qu'il s'agit d'un tableau à une dimension

`type_primitif T[] []` précise qu'il s'agit d'un tableau à deux dimensions

Les déclarations suivantes: **`int T[] = {1,5,7}`** et **`int [] T = {1,5,7}`** sont les mêmes.

Java – Les opérateurs

Opérateur d'affectation:

- **x=3;** // x reçoit 3
- **x=y=z=w+5;** // z reçoit w+5, y reçoit z et x reçoit y

Les opérateurs arithmétiques à deux opérandes:

- **+** : addition
- **-** : soustraction
- ***** : multiplication
- **/** : division
- **%** : modulo (reste de la division euclidienne)

Java – Les opérateurs

Les opérateurs arithmétiques à deux opérandes (Les raccourcis)

`x = x + 4;` ou `x+=4;`

`z = z * y;` ou `z*=y;`

`v = v % w;` ou `v%=w;`

Les opérateurs relationnels:

- `==` : équivalent
- `<` : plus petit que
- `>` : plus grand que
- `<=` : plus petit ou égal
- `>=` : plus grand ou égal
- `!=` : non équivalent

Les opérateurs d'incrémentations et de décrémentation:

`++` : Pour incrémenter (`i++` ou `++i`)

`--` : Pour décrémentation (`i--` ou `--i`)

Java – Les opérateurs

Les opérateurs logiques

&& Et (deux opérandes)

|| Ou (deux opérandes)

! Non (un seul opérande)

L'opérateur à trois opérandes ?:

Affectation **condition** ? **expression_si_vrai** : **expression_si_faux**

exemple : **x** = (**y** < 5) ? **4** * **y** : **2** * **y**;

Equivalent à :

if (**y** < 5)

x = **4** * **y**;

else

x = **2** * **y**;

Java – Les structures de contrôle

Les structures de choix simple

If (condition)

Bloc d'instructions 1;

Else

Bloc d'instruction 2;

NB :La partie de Else peut être absente

Exemple

Ecrire un programme qui lit au clavier la variable moyenne et qui permet d'afficher à l'écran « Validé (é) » si la moyenne est supérieure ou égale à 12 et « non validé (e) » si non.

Java – Les structures de contrôle

Les structures de choix imbriquées

Java permet d'écrire ce type de structure sous la forme :

```
if (Condition 1) {  
    bloc1;  
}  
else if (Condition 2) {  
    bloc2;  
}  
else if (Condition 3) {  
    bloc3;  
}  
else {  
    bloc4;  
}
```

Exemple

Compléter le programme précédent de manière à afficher le résultat accompagné de la mention

Java – Les structures de contrôle

Les structures de cas

Syntaxe :

```
switch( variable) {  
    case valeur1: instr1;break;  
    case valeur2: instr2;break;  
    case valeurN: instrN;break;  
    default: instr;break;  
}
```

Exemple

Ecrire un programme permettant d'afficher le jour correspondant à un nombre lu au clavier.
Si le nombre entré ne se trouve pas dans l'intervalle [1-7] afficher un message d'erreur.

1 → Lundi

...

7 → Dimanche

Java – Les structures de contrôle

Les structures de contrôle répétitives (Boucles)

- Une **boucle** est une structure de contrôle destinée à **exécuter une portion de code plusieurs fois**.
- La structure de contrôle branchant le pointeur ordinal au début de cette portion **tant qu'une condition de continuation est remplie** ou, selon les boucles, qu'une condition de sortie n'est pas remplie.
- Normalement, une boucle **s'exécute** selon le cas, soit un **nombre de fois connu à l'avance**, soit jusqu'à ce qu'une condition permette de sortir de la boucle.
- Il arrive toutefois qu'une erreur de programmation fasse que la **condition ne devienne jamais vraie**. Le programme s'exécute alors indéfiniment à l'intérieur de cette **boucle infinie**.

Java – Les structures de contrôle

Boucle *for*

```
for (initialisation; test; incrémentation) {  
    instructions;  
}
```

Exemple :

```
for (int i = 2; i < 10; i++) {  
    System.out.println("I="+i);  
}
```

Java – Les structures de contrôle

*Sortie d'une boucle par **return***

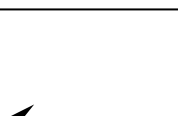
```
int[] tab=new int[]{4,6,5,8};  
for (int i = 0; i < tab.length; i++) {  
    if (tab[i] == 5) {  
        return i;  
    }  
}
```

*Branchement au moyen des instructions **break** et **continue***

break:

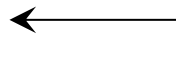
```
int x = 10;  
for (int i = 0; i < 10; i++) {  
    x--;  
    if (x == 5) break;  
}
```

System.out.println(x);



continue:

```
for (int i = 0; i < 10; i++) {  
    if (i == 5) continue;  
    System.out.println(i);  
}
```



Java – Les structures de contrôle

La boucle While

```
while (condition){  
    BlocInstructions;  
}
```

La boucle do .. while

```
do{  
    BlocInstructions;  
}  
while (condition);
```

Exemple :

```
int s=0;int i=0;  
while (i<10){  
    s+=i;  
    i++;  
}  
System.out.println("Somme="+s);
```

Exemple :

```
int s=0;int i=0;  
do{  
    s+=i;  
    i++;  
}while (i<10);  
System.out.println("Somme="+s);
```

Java – Les fonctions

Une fonction en Java:

- fait une tâche
- a un nom
- reçoit des paramètres en cas de besoin
- Retourne un résultat si nécessaire

Exemple

```
public static float appliquer_tva(float somme){  
    return somme+somme*20/100;  
}
```

Float: le type de retour de la fonction

Appliquer_tva: le nom de la fonction

Float somme: le seul paramètre de la fonction

NB: clés **public** et **static** (cf chapitre de la POO)

Java – Les fonctions

La surcharge des fonctions:

- La surcharge d'une fonction consiste à garder le même nom
- Il s'agit d'un changement dans la signature de la fonction (type et nombre de paramètres)

Exemple

```
static void parcourirTableau(String[] tab)  
{  
    for(String str : tab)  
        System.out.println(str);  
}
```

```
static void parcourirTableau(int[] tab)  
{  
    for(int str : tab)  
        System.out.println(str);  
}
```

```
static void parcourirTableau(int[] tab,int i)  
{  
    for(int d : tab)  
        System.out.println(d*i);  
}
```


Java – Les fonctions

La récursivité

La récursivité désigne le concept dans lequel une fonction appelle dans son elle-même.

Exemple

```
public static void r(int n){  
    System.out.println(n*n);  
    if (n>0)  
        r(n-1);  
    System.out.println(n);  
}
```

```
Public static void main(String arg[]){  
    r(5);  
}
```

Le résultat de ce programme est:



0
1
2
3
4
5

Java – Les fonctions

Exercice 1

Ecrit une fonction qui reçoit en paramètre un entier n , et retourne la somme des n premier nombres.

Solution1: en utilisant une fonction classique

Solution 2: en utilisant une fonction récursive

Java – Les fonctions

Exercice 2

Donner le résultat du programme suivant:

```
public class Test {  
    public static void main (String[] args) {  
        int var1 = 1, var2 = 5;  
        System.out.println("Avant l'appel de la méthode : "+var1+" "+var2);  
        modifier (var1, var2);  
        System.out.println("Après l'appel de la méthode : "+var1+" "+var2);  
    }  
    public static void modifier (int var1, int var2) {  
        var1 += 4;  
        var2 += 7;  
    }  
}
```

Résultat:

Avant l'appel de la méthode : 1 5
Après l'appel de la méthode : 1 5

Fin de la première partie

- **Questions et réponses**
- **Truc et astuces**
- **TD à Corriger**
- **Exercices à rendre**