Ministère de l'Education Nationale,
de l'Enseignement Supérieur, de la Formation
des Cadres et de la Recherche Scientifique

Fondation ACADEMIA

Les Journées Informatiques des Classes Préparatoires aux Grandes Ecoles
Rabat, 29 Octobre-2 Novembre 2009

Premiers pas vers java

Mme A. BENOMAR
Professeur à l'Institut National des Postes et Télécommunications

SOMMAIRE

Chapitre 1 **GENERALITES SUR LE LANGAGE JAVA**

1- CARACTERISTIQUES DE BASES DE LA P.O.O

1 - 1 Notion de classe et d'objet

1 - 2 L'encapsulation des données

1 - 3 L'Héritage

1 - 4 Le Polymorphisme

1 - 5 Le Multithreading

2- NAISSANCE ET DEVELOPPEMENT DE JAVA

2- 1 Caractéristiques de JAVA

2- 2 Historique de JAVA

3- ENVIRONNEMENT DE JAVA

3- 1 Java Developer's Kit (JDK)

3- 2 Versions de JAVA

3- 3 Editions de la plate-forme Java 2

3- 4 Le code source

3- 5 La compilation

3- 6 La machine virtuelle java

4- LES PACKAGES

4- 1 Construction de package

4- 2 Utilisation des classes d'un package

4- 3 API de JAVA

5- LES CLASSES EN JAVA

5 - 1 Architecture d'une classe JAVA

5 - 2 Les modificateurs de classe

5 - 3 Conventions d'écriture en JAVA

5 - 4 Un premier exemple

6- UTILISATION DES OBJETS

6 - 1 Déclaration d'un objet

6- 2 Création d'un objet

6 - 3 Un exemple d'utilisation d'objets

7- LES CONSTRUCTEURS

7 - 1 Définition

7 - 2 La variable this

7 - 3 Exemple

8- UTILISATION DES ARGUMENTS DE MAIN

Chapitre 1 **GENERALITES SUR LE LANGAGE JAVA**

1- CARACTERISTIQUES DE BASES DE LA P.O.O

1 - 1 Notion de classe et d'objet

Objet = Données + Traitements

La classe : Abstraction d'un ensemble d'objets qui ont les mêmes caractéristiques (**attributs**) et les mêmes comportements (**méthodes**)

objet est une **instance d'une classe**.

Exemple :

Une classe de nom **Employe** pour modéliser le personnel d'une entreprise

Attributs : numeroCin, nom, prenom, dateNaissance, adresse, …

Méthodes : calcul_salaire(), date_retraite(),…

Objets : tout employé travaillant dans cette entreprise

Employé1 : nom : Benali, pprénom : Mohammed, …

1 – 2 L'encapsulation des données :

- Les données d'une classe peuvent être protégées à l'aide de modificateurs (notion de visiblité et d'accès)

1- 3 L'Héritage

Une classe dérivée hérite des propriétés d'une classe de base.

- Economie de code
- Amélioration de la fiabilité
- Amélioration de la lisibilité

Exemple : Un rectangle peut hériter d'un polygone

1- 4 Le Polymorphisme

Le polymorphisme permet la redéfinition d'une méthode héritée.

1 - 5 Le Multithreading

THREADS : processus qui s'exécutent simultanément à l'intérieur d'un unique programme.

2- NAISSANCE ET DEVELOPPEMENT DE JAVA

2- 1 Caractéristiques de JAVA

- JAVA est un langage de programmation orientée objets proche de C++.

- JAVA fonctionne comme une machine virtuelle
  (indépendant de toute plate forme)

-JAVA est simple (pas de pointeur, pas d'héritage multiple)

- JAVA autorise le multitâche (multithreading)

- JAVA peut être utilisé sur INTERNET

- JAVA est gratuit

JAVA contient une très riche bibliothèque de classes (Packages) qui permettent de :

- Créer des interfaces graphiques
- Utiliser les données multimédia
- Communiquer à travers les réseaux

2- 2 Historique de JAVA

-Projet de recherche en 1991 chez Sun Microsystems

(langage de pilotage des appareils électroniques) a aboutit à la création du l.o.o OAK

- En 1994 conversion du langage Oak pour l'adapter aux technologies des réseaux informatiques tels que INTERNET. C'est la naissance de **JAVA**

Des éditeurs de grande réputation ont rapidement conclu des accords de licence pour diffuser JAVA

- Netscape Communications (23 mai 1995)
- Oracle (30 octobre 1995)
- Borland (8 novembre 1995)
- IBM (6 décembre 1995)
- Adobe Systems (6 décembre 1995)
- Symantec (13 decembre 1995)
- Microsoft (12 mars 1996)
- Novell (21 mars 1996)

3- ENVIRONNEMENT DE JAVA

3- 1 Java Developer's Kit (JDK)

Sun fourni gratuitement un ensemble d'outils et d'API pour permettre le développement de programmes avec Java. Ce kit, nommé JDK, est librement téléchargeable sur le site web de Sun **http://java.sun.com**

Le kit java contient :

***** le **compilateur** (**javac**),

***** l'**interpréteur** (**java**),

***** l'**appletviewer** ou inspecteur d'applet

- Le JDK contient par ailleurs l'**API** de java (Application Programmers Interface) qui correspond à la bibliothèque de java

- Le kit JDK ne comporte pas d'éditeur.

3- 2 Versions de Java

Java évolue au fil des années. En plus de faire évoluer les numéros de versions, le nom complet de chaque version a parfois été changé. Les principaux événements de la vie de Java sont les suivants :

Année	Evénements
1995	mai : premier lancement commercial
1996	janvier : JDK 1.0.1
1997	février : JDK 1.1
1998	décembre : lancement de J2SE 1.2
1999	décembre : lancement J2EE
2000	mai : J2SE 1.3
2002	février : J2SE 1.4
2004	septembre : J2SE 5.0
2006	mai : Java EE 5 décembre : **Java SE 6.0**

Depuis sa version 1.2, Java a été renommé **Java 2**. Les numéros de version 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, à tel point que la dénomination JDK est reprise dans la version 5.0.

En outre, **Java 2** désigne les versions de **J2SE 1.2** à **J2SE 1.4**

La version actuelle est Java **SE 6** alors que la version **Java SE 7** est en cours de développement (sera fini au cours de 2010)

Compatibilité de version

Java assure une compatibilité binaire ascendante (les programmes compilés avec une ancienne version peuvent s'exécuter avec une plate-forme plus récente).

Mise à part l'introduction de nouveaux mots-clés (assert dans 1.4 et enum dans 5.0), la compatibilité binaire ascendante est presque respectée (les sources d'un programme écrit pour être compilées avec une ancienne version peuvent être compilées avec une nouvelle version du compilateur).

3-3 Editions de la plate-forme Java 2

La plate-forme Java 2 est disponible dans 3 éditions différentes qui regroupent des APIs par domaine d'application.

- **java 2 Standard Edition (J2SE),** contient les classes qui forment le cœur du langage Java qui sont nécessaires pour développer des applications et des applets. Cette édition reprend le JDK 1.0 et 1.1.

- **Java 2 Entreprise Edition (J2EE)** pour le développement d'applications d'entreprise. Elle contient les classes J2SE plus d'autres classes pour le développement d'applications d'entreprise tel que JDBC pour l'accès aux bases de données, EJB pour développer des composants orientés métiers, Servlet / JSP pour générer des pages HTML dynamiques, ... Cette édition nécessite le J2SE pour fonctionner.
- **Java 2 Micro Edition (J2ME)** : contient le nécessaire pour développer des applications capables de fonctionner dans des environnements limités tels que les assistants personnels (PDA), les téléphones portables,... Dans ce cas seule une petite partie du langage est utilisée.

Chaque édition contient un kit de développement Java 2 (**SDK**) pour développer des applications et un environnement d'exécution Java (**JRE**) pour les exécuter.

3- 4 Le code source

- Une application java est généralement composée de plusieurs fichiers source. Les fichiers source de java ont l'extension **.java.**

- Un fichier **.java** peut contenir plusieurs classes**.**

- Le nom du fichier source java doit être le même que celui de la classe publique que ce fichier renferme (Un fichier java ne peut pas contenir plus d'une classe publique)

3- 5 La compilation

*** Le byte code**

- La compilation va transformer le code source en J-code **ou byte-code Java** indépendant de toute plate forme

- La commande qui permet la compilation est la suivante :

Javac nom_fich.java

Création d'un ou plusieurs fichiers **.class.**

3- 6 La machine virtuelle java

Le byte code produit par le compilateur java ne contient pas de code spécifique à une plate-forme particulière. Il peut être exécuté et on obtient quasiment les mêmes résultats sur toutes les machines disposant d'une JVM. (**JVM** : Java virtuel Machine) Ce concept est à la base du slogan de Sun pour Java : **WORA** (Write Once, Run Anywhere : écrire une fois, exécuter partout)

- un programme java est exécuté sur tout ordinateur pour lequel une machine virtuelle est disponible.

- Une application sera exécutée par l'interpréteur de java

Java nom_fich

- Les fichiers jar

Les fichiers **.jar** sont des fichiers **java compressés** comme les fichiers **.zip** selon un algorithme particulier devenu un standard de fait dans le monde des PC.

Ils sont parfois appelés **fichiers d'archives** ou, plus simplement, **archives**.

Ces fichiers sont produits par des outils comme **jar.exe**. (qui fait partie du jdk) .

Les fichiers **.jar** peuvent contenir une multitude de fichiers compressés avec l'indication de leur chemin d'accès.

Un fichier .jar peut ainsi être exécuté directement dans tout environnement possédant une machine virtuelle

4- LES PACKAGES

Les packages java permettent :

* L'organisation hiérarchique des classes

*Le regroupement de plusieurs classes qui collaborent entre elles dans une application

* La définition d'un certain niveau de protection

4- 1 Construction de package

- Pour ranger une classe dans un package, il faut insérer la ligne suivante au début du fichier source:

package nomDuPaquetage ;

- Un package (ou un paquetage) qui doit contenir des classes et donc des fichiers correspond à un répertoire. Le package a le même nom que le répertoire qui le contient.

4- 2 Utilisation des classes d'un package

On peut importer les packages de deux manières différentes :

- import nomDuPackage.*;

- import nomDuPackage.nomDeLaclasse;

Exemple

Import liste.* ; // importer toutes les classes du package liste

import liste.A ; // importer la classe A du package liste

4- 3 API de JAVA (Application Programmers Interface)

L'API java (bibliothèque de java) se compose de nombreuses classes regroupées selon leur fonctionnalité en packages.

les packages de bases des différentes versions du JDK sont

- java.lang : ce package de base contient les classes les plus fondamentales du langage tel que Object (la super classe primitive de toutes les classes, Class, Math, System, String, StringBuffer, Thread, etc ...

- Ce package est tellement fondamental qu'il est implicitement importé dans tous les fichiers sources par le compilateur (java.lang est importé par défaut).

Exemples de classes de java.lang:

Java.lang.System

Java.lang.Math

Java.lang.String

- java.awt : (awt pour Abstract Window Toolkit) contient les classes pour fabriquer des interfaces graphiques

Exemples :

Java.awt.Graphics

Java.awt.Color

- javax.swing: contient les classes pour fabriquer des interfaces graphiques évolués

- **java.applet** : fondamentale pour faire des applets qui sont des applications utilisables à travers le web

- java.io: contient les classes nécessaires aux E/S

- java.net : accés aux réseaux

- java.util : classes d'utilitaires (Random, Date, ...)

5- LES CLASSES JAVA

- Une classe contient des déclarations d'attributs et des déclarations de méthodes.

5 - 1 Architecture d'une classe JAVA

Une classe se compose de deux parties :

- **l'en-tête**
- **le corps**

Entête d'une classe

[modificateur] **class** <NomClasse>[**extends** <superclass>] [**implements** <Interface>]

[ ] : optionnel

< > : obligatoire

gras : mot clé

Corps d'une classe

Le corps de la classe donne sa définition. Il contient les déclarations des variables et des méthodes.

La syntaxe d'une déclaration de classe est la suivante :

```
entête
{
    Déclarations des attributs et des méthodes
}
```

5 - 2 Les modificateurs de classe

Modificateur	Définition
public	La classe est accessible par toutes les autres classes des autres packages. Elle est visible partout
private	La classe n'est accessible qu'à partir du fichier où elle est définie
final	Les classes finales ne peuvent pas être héritables
friendly	l'accés est garanti à toute classe dans la package.
abstract	Aucun objet ne peut instancier cette classe. Seules les classes abstraites peuvent déclarer des méthodes abstraites. Cette classe sert pour l'héritage

5 - 3 Conventions d'écriture en JAVA

Type	Convention	Exemple
Nom d'une classe	**Débute par une majuscule**	**class Polygone**
Nom d'un objet	**Débute par une minuscule** **Peut s'étaler sur plusieurs mots** **Chaque mot commence par une majuscule sauf le premier**	**premierObjet**
Nom de méthode	**Débute par une minuscule** **Sauf pour le constructeur**	**void ecrire()**
Nom de package	**S'écrit en majuscules**	**Java.lang**
Constante	**S'écrit en m minuscules**	**A_CONSTANT**

5 - 4 Un premier exemple

Il s'agit d'écrire une application qui affiche sur l'écran « **Bravo** »

```
public class Premier
{
  public static void main (String[ ] arg)
  {
       System.out.println( ‘‘Bravo’’) ;
  }
}
```

public : visible de partout , y compris des autres paquetages

static :méthode de classe

System.out.println : La méthode **println** de la classe System écrit sur l'écran (objet out) la chaîne de caractères passée en paramètre à cette méthode

- Cette classe doit se trouver dans le fichier **Premier.java**
- La compilation se fait par la commande **javac Premier.java**
- La compilation crée un fichier **Premier.class**
- Le fichier est exécuté par la commande **java Premier**
- On obtient à l'exécution : **Bravo**

6- UTILISATION DES OBJETS

Pour utiliser une classe (utiliser ses attributs et appeler ses méthodes), elle doit être instanciée . Il y aura alors création d'un objet. Mais avant de créer un objet , il faut le déclarer

6 - 1 Déclaration d'un objet

NomClasse objet ;

Cette instruction déclare un objet de la classe NomClasse mais ne le crée pas.

6- 2 Création d'un objet : opérateur new

Une fois déclaré, objet doit être crée comme suit :

objet = new NomClasse( ) ;

La déclaration et la création d'un objet peuvent être regroupées en une seule instruction :

NomClasse objet = new NomClasse( )

On accède aux méthodes et aux attributs de la classe comme suit :

NomObjet.méthode(arguments de la méthodes)

NomObjet.attribut

6- 3 Exemple d'utilisation d'objet

Il s'agit d'écrire une application en java qui permet de calculer le salaire hebdomadaire d'un employé en fonction du taux horaire et du nombre d'heures

```
public class Employe
 {  String nom;              // le nom de l'employé
    static float taux_h=50;  // le taux horaire en dirhams
    int nb_h;
     float salaireh ()
    {  float s;
       s=(taux_h*nb_h);
       return s;
    }
    public static void main(String[] arg)
    { float s1,s2;        // les salaires de 2 employés
       Employe emp1,emp2; // 2 objets modélisant 2 employés
       emp1=new Employe();
       emp2=new Employe();
       emp1.nom= " Mohammed"; // le nom de l'employé 1
       emp1.nb_h= 35; // le nombre d'heures de l'employé 1
       emp2.nom="Nadia"; // le nom de l'employé 2
       emp2.nb_h=38 ; // le nombre d'heures de l'employé 2
       s1=emp1.salaireh();// le salaire de l'employé 1
       s2=emp2.salaireh();// le salaire de l'employé 2
   System.out.println("Le salaire de "+ emp1.nom+" est"+s1+" dh");
  System.out.println("Le salaire de "+ emp2.nom+" est"+s2+" dh");
    }  // fin de main
 }  // fin de la classe
```

A l'exécution, on aura :

Le salaire de Mohammed est 1750.0 dh

Le salaire de Nadia est 1900.0 dh

7- LES CONSTRUCTEURS

7 - 1 Définition

- Le constructeur est une méthode qui est effectuée au moment de la création d'un objet. Il sert à initialiser les variables contenues dans les objets.

- Le constructeur porte le même nom que sa classe. Il ne retourne pas de valeur et ne mentionne pas void au début de sa déclaration

- Toute classe possède au moins un constructeur. Si une classe ne déclare pas de constructeur, un constructeur vide est crée automatiquement par défaut

- Un constructeur est une fonction qui est appelée par le biais de l'opérateur **new**

monObjet = new Constructeur(arguments) ;

Exemple d'utilisation de constructeur

```
public class Employe_Constructeur
{   String nom;
    static float taux_h=50;
    int nb_h;
    Employe_Constructeur(String identite)  // Constructeur
    {
      nom=identite;
    }
     float salaireh ()
    {  float s;
      s=(taux_h*nb_h);
      return s;
    }
    public static void main(String[] arg)
    { float s1,s2;
      Employe_Constructeur emp1,emp2;
      emp1=new Employe_Constructeur (" Mohammed");
      emp2=new Employe_Constructeur ("Nadia");
    /* Dès la création d'un objet on doit spécifier la valeur du nom */
      emp1.nb_h= 35; // le nombre d'heures de l'employé 1
      emp2.nb_h=38 ; // le nombre d'heures de l'employé 2
      s1=emp1.salaireh();
      s2=emp2.salaireh();
   System.out.println("Le salaire de "+ emp1.nom+" est "+s1+" dh");
  System.out.println("Le salaire de "+ emp2.nom+" est "+s2+" dh");
    }  // fin de main
}  // fin de la classe
```

7– 2 La variable this

La variable **this** sert à référencer dans une méthode l'instance de l'objet en cours d'utilisation. Son emploi peut s'avérer utile dans les constructeurs avec des paramètres lorsque les variables de la classe et celles du constructeur portent exactement le même nom

```
public class Employe_this
{ String nom;
  static float taux_h=50;
  int nb_h;
 Employe_this(String nom)
  {  this.nom=nom; // variable globale de la classe = variable locale
   }
 float salaireh ()
  { float s;
    s=(taux_h*nb_h);
    return s;
  }
  public static void main(String[] arg)
  { float s1,s2;
    Employe_this emp1,emp2;
    emp1=new Employe_this (" Mohammed");
    emp2=new Employe_this ("Nadia");
    emp1.nb_h= 35; // le nombre d'heures de l'employé 1
    emp2.nb_h=38 ; // le nombre d'heures de l'employé 2
   s1=emp1.salaireh();
   s2=emp2.salaireh();
 System.out.println("Le salaire de "+ emp1.nom+" es t"+s1+" dh");
System.out.println("Le salaire de "+ emp2.nom+" est "+s2+" dh");
  } // fin de main
} // fin de la classe
```

8- UTILISATION DES ARGUMENTS DE MAIN

Les arguments de la méthode main permettent de récupérer des chaînes de caractères fournies depuis la ligne de commande

Exemple 1 d'utilisation des arguments de main : Les noms des employés sont considérés des arguments de la méthode main

```
public class Arguments_main
  {
    String nom;
    static float taux_h=50;
    int nb_h;
    float salaireh ()
    {       float s;
      s=(taux_h*nb_h);
      return s;
    }
    public static void main(String[] arg)
    {       float s1,s2;
      Arguments_main emp1,emp2;
      emp1=new Arguments_main();
      emp2=new Arguments_main();
       emp1.nom=arg[0];
       emp2.nom=arg[1];
      emp1.nb_h= 35; // le nombre d'heures de l'employé 1
      emp2.nb_h=38 ; // le nombre d'heures de l'employé 2
      s1=emp1.salaireh();
      s2=emp2.salaireh();
      System.out.println("Le salaire de "+ emp1.nom+" est "+s1+" dh");
      System.out.println("Le salaire de "+ emp2.nom+" est "+s2+" dh");
    } // fin de main
  } // fin de la classe
```

Après compilation du fichier Arguments_main.java , il faut l'exécuter en donnant 2 chaînes de caractères pour les noms des 2 employés.

Exemple d'exécution : java Arguments_main Ali Sanae

Exemple 2 : Utilisation d'un nombre entier comme argument

Il s'agit de considérer le nombre d'heures travaillées comme argument de main aussi pour pouvoir donner leurs valeurs à l'exécution

```
public class Arguments_entiers
  {
    String nom;
    static float taux_h=50;
    int nb_h;
    float salaireh ()
    {       float s;
      s=(taux_h*nb_h);
      return s;
    }
    public static void main(String[] arg)
    {       float s1,s2;
      Arguments_entiers emp1,emp2;
      emp1=new Arguments_entiers();
      emp2=new Arguments_entiers();
      emp1.nom=arg[0];
       emp1.nb_h= Integer.parseInt(arg[1]);
      emp2.nom=arg[2];
       emp2.nb_h= Integer.parseInt(arg[3]);
      s1=emp1.salaireh();
      s2=emp2.salaireh();
      System.out.println("Le salaire de "+ emp1.nom+" est "+s1+" dh");
      System.out.println("Le salaire de "+ emp2.nom+" est "+s2+" dh");
    } // fin de main
  } // fin de la classe
```

il faut exécuter cette classe en donnant 2 chaînes de caractères pour les noms des 2 employés et leurs nombres d'heures travaillées

Exemple d'exécution : java Arguments_entier Ali 30 Sanae 28

Chapitre 2

OUTILS ELEMENTAIRES POUR LA PROGRAMMATION JAVA

Chapitre 2

OUTILS ELEMENTAIRES POUR LA PROGRAMMATION JAVA

1 – LES COMMENTAIRES EN JAVA

En java, il existe deux types de commentaires :

// commentaire simple sur une ligne

/* */ commentaire de plusieurs lignes

2 – LES TYPES ELEMNTAIRES EN JAVA

JAVA dispose des 8 types élémentaires suivants, ces types sont appelés **primitives** en java :

boolean	Destiné à une valeur logique (true, false)
byte	Octet signé 8 bits (-128 et 127)
short	Entier court signé 16 bits (-32768 – 32767)
char	Caractère 16 bits
int	Entier signé 32 bits
float	Virgule flottante 32 bits
double	Double précision 64 bits
long	Entier long 64 bits

Java dispose également des classes suivantes pour gérer ces primitives :

Classe	Primitive
Character	char
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Boolean	boolean

3- LES PROPRIETES DES ATTRIBUTS

3 - 1 Déclaration

Un attribut correspond à une variable ou une constante pouvant prendre une valeur différente pour chaque objet instance de la classe

Tous les objets d'une même classe ont les mêmes attributs, mais chaque objet associe à ces attributs des valeurs qui lui sont propres.

Avant d'utiliser un attribut, il doit être déclarée comme suit :

<modificateur> type_elementaire identificateur

Modificateur	Définition
final	Un attribut constant.
private	- Un attribut privée ne peut être utilisée que dans les méthodes de la même classe
public	Attribut visible par toutes les autres méthodes
static	attribut de classe indépendant des objets

Par défaut, si on n'indique aucun modificateur, la classe, la méthode, ou l'attribut est visible par toutes les classes se trouvant dans le même package.

3-2 Les identificateurs

A chaque objet , classe, méthode ou attribut est associé un **identificateur**. Les identificateurs Java sont formés d'un nombre quelconque de caractères. Ils doivent respecter les règles suivantes :

- Les majuscules et les minuscules sont distinguées.
- Le premier caractère ne doit pas être un chiffre
- L' identificateur ne doit pas être un mot clé de Java.

En plus des règles ci-dessus, il est conseillé de respecter les directives suivantes :

- Evitez les caractères accentués dans les noms de classes.
- Evitez de donner le même nom à des identificateurs d'éléments différents

3-3 Les mots clé de java

Les mots suivants sont les mots clé de java :

abstract	**assert**	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	**enum**
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	while

3 - 4 Les attributs statiques

Ce sont les attributs qui ont le modificateur **static.** Un **attribut statique** (ou **de classe**) appartient à une classe et non à ses instances. Il est indépendant des objets

- Un **attribut statique** existe dès que sa classe est chargée en mémoire alors qu'un **attribut d'instance** existe lors de l'instanciation de la classe

Pour utiliser un attribut ou une méthode statique, on écrit :

NomClasse.nomAttributStatique Cependant JAVA permet également d'utiliser : **nomObjet.nomAttributStatique** où nomObjet étant une instanciation de la classe qui contient l'attribut ou la méthode statique

Remarque : Si on modifie la valeur d'une variable statique , elle est modifiée pour toutes les instances, mêmes celles créées avant la modification. Cela est impliqué par le fait que la variable n'existe qu'à un seul exemplaire, partagé par la classe et toutes les instances.

Exemple1 : attribut statique

```
class Statique
  {
    int nons;
    static int s;
  }
  public class UtiliseStatic
  {
    public static void main (String[] arg)
    {
      Statique.s=10;
      Statique obj1= new Statique();
      obj1.nons=20;
      Statique obj2= new Statique();
      obj2.nons=30;
      obj2.s=15;//Pour tous ,les objets s=15
```

```
        System.out.println(Statique.s);        //15
        System.out.println(obj1.s);            //15
        System.out.println(obj1.nons);         //20
        System.out.println(obj2.nons);         //30
    }
  }
```

Exemple2 : Calcul des moyennes des élèves ainsi que la moyenne de la classe

```
class   Moyenne
{   static int nb_eleves ;  // nombre d'élèves
   float note1,note2,note3;     // notes des élèves
   int num;                     // numéro de l'élève
   String nom;                  // nom de l'élève
   static float moyenne_classe;
  Moyenne(int num,String nom)          //constructeur
     {  this.num=num;           this.nom=nom ;      }
     float moyenne_eleve()
     {        return ((note1+note2+note3) /3);}
  }
public class Utilise_Moyenne
{  public static void main(String[] arg)
  { Moyenne.nb_eleves=2;
    Moyenne eleve1= new Moyenne(1,"Adnane");
    Moyenne eleve2= new Moyenne(2,"Amal");
    eleve1.note1=15;
    eleve1.note2=11;
    eleve1.note3=17;
     float moy1=eleve1.moyenne_eleve();
     eleve2.note1=12;
    eleve2.note2=18;
     eleve2.note3=16;
     float moy2=eleve2.moyenne_eleve();
   System.out.println("Eleve : numéro : "+eleve1.num);
   System.out.println("Nom : "+eleve1.nom+" Moyenne : "+moy1);
   System.out.println("Eleve : numéro : "+eleve2.num);
   System.out.println("Nom : "+eleve2.nom+" Moyenne : "+moy2);
```

```
  Moyenne.moyenne_classe=(moy1+moy2)/ Moyenne.nb_eleves;
  System.out.println("La moyenne de la classe est "+
Moyenne.moyenne_classe);
    }
  }
```

A l'éxecution on aura :
Eleve 1 Nom : Adnane Moyenne : 14.3333
Eleve 2 Nom : Amal Moyenne : 15.3333
La moyenne de la classe est : 14.833

3-5 – Les opérateurs de java

*****	Multiplication
/	division
%	modulo
+	addition
-	soustraction
=	affectation
==	identité
!=	différent de
&&	et logique
||	ou logique
++	incrémentation

+= - = *= /=

3-6 – Les initialiseurs

Les initialiseurs d'attributs permettent d'affecter à un attribut une valeur initiale. Si on déclare un attribut non static dans une méthode, cet attribut n'a pas de valeur. Toute tentative d'y faire référence produit une erreur de compilation.

En revanche, s'il s'agit d'une variable globale, càd si cette déclaration se trouve en dehors de toute méthode, JAVA l'initialise automatiquement avec une valeur par défaut .

Exemple :

```
class Initialiseur
  {
    int a1; static int a2;
    void m1()
    {   int a3;
      System.out.println(a3) ;        //erreur car a3  non initialisée
    }
    public static void main(String[] arg)
    {         System.out.println("a2="+a2);
      Initialiseur obj= new Initialiseur();
      System.out.println("obj.a1="+obj.a1);
     obj.m1();
    }
  }
```

Pour des initialisations plus complexes, on peut utiliser un bloc placé à l'extérieur de toute méthode ou constructeur

Par exemple, **float e**

```
{
        if (b != 0 )
        { e = (float) a /b ; }
}
```

3-7 Les attributs constants

- Une constante est un attribut qui ne peut plus être modifié une fois qu'il a été initialisé.

- Une constante est initialisée une seule fois soit au moment de sa déclaration, soit dans le constructeur de la classe

- Pour déclarer un attribut constant , on le muni du modificateur **final**

Si on déclare un attribut par **final static int MAX = 10** MAX est une constante de la classe qui vaut toujours 10 pour toutes les instances de la classe.

Par contre si on déclare un attribut final dans le constructeur . Il y en aura un exemplaire par instance créée.

Exemple de définition et d'utilisation de constantes

```
class   Constantes
{
        final int C ;          /* déclare une constante d'instance qui peut être
                                  initialisée dans le constructeur */
        final static int D= 100 ; /* déclare une constante de classe
                          qui doit être initialisée au moment de sa définition */
        Constantes (int i )   // Constructeur de la classe Constantes
        {
                C = i ;    // initialisation de la constante d'instance C
                // C= D ;     interdit car C doit être définit une seule fois
                // D = 10 ;      interdit car D est une constante de classe
        }
}
public class   EssaiConstantes
{
        public static void main (String[] arg)
        {
                System.out.println ( ( new Constantes (2)).C ) ;
                System.out.println (Constantes.D ) ;
        }
}
```

On obtient à l'exécution :

```
2
100
```

4- Les structures de contrôle

Les structures de contrôle de java sont similaires à celles de C

Les instructions conditionnelles

a – L'instruction if

```
if    (expression)
      InstructionSiVrai ;
else
      InstructionSiFaux
```

b- L'opérateur ?

```
condition      ?    siVrai      :    siFaux
Exemple :      int a=1, b=5 ;
               int min = a<b  ?    a    :    b
```

c- L'instruction switch

```
switch (variable)
{  case cas1 : instruc11 ;
               instruc12 ;
                  .
               break ;
   case cas2 : instruc21 ;
               instruc22 ;
                  .
               break ;
         .
   default : instrDefault ;
}
```

Les instructions de répétition

a – L'instruction while

```
while  (condition)
            instruction ;
```

b – L'instruction do

```
do
    instr ;
while  (condition)
```

c – L'instruction for

for (initialisation, **;** test **;** incrementation**)**

La boucle for La boucle for en java autorise plusieurs expressions séparées par une virgule à apparaître dans la partie **initialisation** et **incrémentation** mais pas dans la partie test

Exemple : for (i=1, j=10; (k<15);i++,j--)

5- Les propriétés des méthodes

5 - 1 Signature d'une méthode

les méthodes sont les fonctions qu'on peut appliquer aux objets. Elles permettent de changer l'état des objets ou de calculer des valeurs

Plusieurs méthodes d'une même classe peuvent avoir le même nom, mais des paramètres différents (surcharge) .

Le nom et le type des paramètres constituent la **signature** de la méthode. Une méthode peut elle aussi posséder des modificateurs.

Modificateur	Définition
final	Une méthode qui ne peut pas être redéfinie.
protected	Méthode ne peut être invoquée que par des sous classes.
private	une méthode privée ne peut être utilisée que dans les méthodes de la même classe
public	méthode visible par toutes les autres méthodes
static	Méthode de classe indépendant des objets

5 -2 Les méthodes statiques

Comme pour les attributs statiques , les méthodes statiques ont le modificateur **static** et sont indépendantes des instances

- Une **méthode statique** ne peut pas être redéfinie. Elle est automatiquement finale

Pour utiliser une méthode statique, on écrit :

NomClasse.nomMethodeStatique(arguments)

Cependant JAVA permet également d'utiliser :

nomObjet.nomMethodeStatique(arguments)

nomObjet étant une instanciation de la classe qui contient l'attribut ou la méthode statique

5-3 – La surcharge

Si plusieurs méthodes possèdent le même nom mais différent par l'ensemble des types de leurs arguments, ou par l'ordre des types de leurs arguments, on dit qu'il y a surcharge. Lorsqu'une méthode surchargée est invoquée , la bonne méthode est choisie pour qu'il y ait correspondance sur les paramètres de l'appel et les arguments de la méthode

EXEMPLE

```
class   Surcharge
{       int n ;
        double  x ;
        Surcharge( )
        {
                n = 1 ;
                x = 3.5 ;     }
```

```
        Surcharge(int  n, double   x)
        {
                this.n = n ;
                this.x = x ;
        }
        int operation(int  p)
        {               return 10*p +n  ; }
        double operation (double y , int  p)
        {               return x*p + y  ;  }
        double operation(int  p, double y )
        {               return (double) n/p + y  ;      }
}
public class EssaiSurcharge
{
        public static void main (String[] arg)
        {
                Surcharge    surcharge   ;
                surcharge = new Surcharge( )  ;
                System.out.println (surcharge.operation (2))  ;
                System.out.println (surcharge.operation (1.5 , 4)) ;
                System.out.println (surcharge.operation (4 , 1.5)) ;
                surcharge = new Surcharge(7 , 2.0)  ;
                System.out.println (surcharge.operation (2))  ;
        }
}
```

Remarque : si on ajoute l'instruction **:** (surcharge.operation (4 , 5)) ; il y'aura erreur de compilation car ambiguité .Par contre s'il n'y avait pas de surcharge (une seule méthode double operation (double y , int p)) cette même instruction ne génerera pas d'erreur car transtypage automatique

6 – LES TABLEAUX

Les tableaux en java sont des objets pouvant contenir un nombre fixe d'éléments de même nature. Chaque élément est accessible grâce à un indice correspondant à sa position dans le tableau.

6-1 La déclaration d'un tableau

Les tableaux doivent être déclarés comme tous les objets. Java dispose de deux syntaxes équivalentes pour la déclaration des tableaux,

type_elements[ ] tableau ; ou **type_elements tableau[ ] ;**

Exemple : int [ ] tab1

int tab2[ ]

6-2 La création d'un tableau

Comme tout objet java, un tableau doit être crée après sa déclaration. Les tableaux Java sont de taille fixe. Leur création devra donc indiquer leur taille. La syntaxe à utiliser est la suivante :

x = **new** type[dimension]; x étant déjà déclarée

- dimension est le nombre d'éléments du tableau. les tableaux peuvent être déclarés et crées sur la même ligne : int[ ] x = new int[5];

Exemple :

```
class TableauA
{ public static void main(String[] arg)
{
    int[]  tableau ;
    tableau = new int[2];
    tableau[0]=-5;
    tableau[1]= 8
    for(int i=0 ;i<2 ;i++)
  System.out.println(tableau[i]);
}
}
```

Autre façon de définir un tableau : On peut définir un tableau avec un tableau littéral au moment de sa déclaration.

Exemple: **boolean tableau[ ] = {true,false,true}; int t[ ]={0, 5, 9, -45}**

6- 3 La longueur d'un tableau

On peut connaître la longueur d 'un tableau en utilisant l 'attribut **length.**

Tableau.length est la longueur du tableau Tableau.

```
class TableauB
{public static void main(String[ ] argv)
 {   int tableau[ ]=new int[3];
    System.out.println(''Taille du tableau : ''+ tableau.length) ;
 }  // affichera 3
}
```

Remarque : Pour parcourir un tableau, il ne faut pas dépasser sa longueur (Tableau.length) sinon il y'aura une erreur à l'exécution.

Exemple : for (int i= 0 ; i <= Tableau.length; i++) générera une erreur

6- 4 Les tableaux d'objets

Avec java, on peut aussi manipuler des tableaux d'objets

Exemple:

```
class TableauD
{       public static void main(String[] argv)
        {       Integer tableau[] = new Integer[4] ; //un tableau de 4 objets
                int  somme = 0 ;
                for (int i=0 ; i<tableau.length ; i++)
```

```
                tableau[i]= new Integer(i) ;
            for (int i=0 ; i<tableau.length ; i++)
                somme+=tableau[i].intValue() ;  //intValue retourne l'attribut i
        System.out.println('' La somme des entiers est  '' +somme) ;
    }
}
```

6-5 Les tableaux multidimensionnels :

Les tableaux de java peuvent comporter plusieurs dimensions. Pour déclarer un tableau multidimensionnel, on doit utiliser la syntaxe suivante : **type [ ] [ ] [ ] tableau**

Exemple : int[ ] [ ] tableau; // 2 dimensions

- La création du tableau peut être effectuée
  - à l'aide de L'opérateur **new** : **int [][] x = new int[2][4];** ou
  - de tableaux littéraux : **int [ ][ ] tableau = {{1, 2, 3, 4},{5, 6, 7, 8}}**;

- Pour accéder à tous les éléments du tableau, on utilise des boucles imbriquées Exemple : int [ ][ ] x = new int[2][4];

```
for (int i = 0; i < x.length; i++)  // x.length : longueur de la 1ère dimension
  for (int j = 0; j < x[i].length; j++) //x[i].length : longueur de la 2dimension
```

7- SAISIE AU CLAVIER

- En java, le clavier est un fichier représenté par un objet particulier de La classe **java.io.BufferedReader** qui admet le constructeur :

BufferedReader (InputStreamReader objet)

Pour représenter le clavier, on doit instancier la classe **BufferedReader** comme suit : **BufferedReader (new InputStreamReader(System.in))**

-la méthode readLine de BufferedReader permet de lire une ligne

Exemple : Ecrire une application qui permet de lire 2 entiers à partir du clavier et d'en faire la somme.

```
import java.io.*;
public class SaisieClavier
{
  static int somme = 0;
  static int a,b;
  static int lire_entier()throws IOException
  {
    BufferedReader entree = new BufferedReader
      (new InputStreamReader(System.in)); //entree représente le clavier
    int i=0;
    i=Integer.parseInt(entree.readLine()); // conversion en entier
    return i;
  }
  public static void main(String[] argv)throws IOException
  { System.out.println(« Entrer au clavier 2 entiers : ») ;
    a=lire_entier();
    b=lire_entier();
    c=lire_entier();
    somme=a+b+c;
    System.out.println("La somme vaut : "+somme);
  }
}
```

8- LES CHAINES DE CARACTERES

En Java, les chaînes de caractères sont des objets de la classe **java.lang.String** ou **java.lang.StringBuffer**

8-1 Les chaînes de caractères de type String

- La classe **java.lang.String** est une classes dont les objets sont immuables (ils ne peuvent pas changer de valeur)

- La JVM crée un objet de type **String** à la rencontre d'une série de caractères entre doubles apostrophes.

- Les chaînes de caractères (String) peuvent se concaténer à l'aide de l'opérateur +.

- Les Constructeurs de java.lang.String

String()	Construit la chaîne vide
String(byte[] bytes)	Construit une chaîne de caractères à partir d'un tableau d'octets
String(byte[] bytes, int offset, int length)	Construit une chaîne de caractères à partir d'une partie de tableau d'octets
String(byte[] bytes, int offset, int length, String enc)	Construit une chaîne de caractères à partir d'une partie de tableau d'octets, et d'un encodage
String(byte[] bytes, String enc)	Construit une chaîne de caractères à partir d'un tableau d'octets, et d'un encodage
String(char[] value)	Construit une chaîne de caractères à partir d'un tableau de caractères
String(char[] value, int offset, int count)	Construit une chaîne de caractères à partir d'une partie de tableau de caractères
String(String value)	Construit une chaîne à partir d'une autre chaîne.
String(StringBuffer buffer)	Construit une chaîne à partir d'une autre chaîne de typr *StringBuffer*.

- La concaténation de chaîne peut également se faire à l'aide de la méthode **concat(String s).**
- la méthode **length()** renvoie la longueur (nombre de caractères) de la chaîne.

- Les Comparaisons de chaînes de caractères

int compareTo(Object o)	Compare une chaîne de caractère à un autre objet. Renvoie une valeur <0 ==0 ou > 0
int compareTo(String anotherString)	Compare une chaîne de caractère à un autre objet. Renvoie une valeur <0 ==0 ou > 0. La comparaison est une comparaison lexicographique.
int compareToIgnoreCase(String str)	Compare une chaîne de caractère à un autre objet. Renvoie une valeur <0 ==0 ou > 0. La comparaison est une comparaison lexicographique, ignorant la casse.
boolean equals(Object anObject)	Compare la chaîne a un objet et retourne *true* en cas d'égalité et *false* sinon
boolean equalsIgnoreCase(Object anObject)	Compare la chaîne a un objet et retourne *true* en cas d'égalité et *false* sinon (on ignore la casse)
boolean regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len)	Teste l'égalité de deux parties de chaînes
boolean regionMatches(int toffset, String other, int ooffset, int len)	Teste l'égalité de deux parties de chaînes

- Les Caractère et sous-chaîne.

char charAt(int i)	Retourne le caractère à l'indice spécifié en paramètre.
String substring(int d)	Sous-chaîne depuis *d* jusqu'à la fin
String substring(int d, int f)	Sous-chaîne depuis *d* jusqu'au caractère d'indice *f* non inclus.
boolean startsWith(String prefix)	Renvoie *true* si *prefix* est un préfixe de la chaîne
boolean startsWith(String prefix, int i)	Renvoie *true* si *prefix* est un préfixe de la chaîne à partir de *i*.
boolean endsWith(String suffix)	Retourne true si suffix est un suffixe de la chaîne

- Les Conversions

String toLowerCase()	Retourne une chaîne égale à la chaîne convertie en minuscules.
String toLowerCase(Locale locale)	Retourne une chaîne égale à la chaîne convertie en minuscules.
String toString()	Retourne une chaîne égale à la chaîne
String toUpperCase()	Retourne une chaîne égale à la chaîne convertie en majuscules.
String toUpperCase(Locale locale)	Retourne une chaîne égale à la chaîne convertie en majuscules.
String trim()	Retourne une chaîne égale à la chaîne sans les espaces de début et de fin.
String replace(char ac, char nc)	Retourne une chaîne ou tous les *ac* ont été remplacé par des *nc*. S'il n'y a pas de remplacement, la chaîne elle-même est retournée.

static String copyValueOf(char[] data)	Construit une chaîne de caractères à partir d'un tableau de caractères
static String copyValueOf(char[] data, int offset, int count)	Construit une chaîne de caractères à partir d'une partie de tableau de caractères
byte[]getBytes(String enc)	Convertit la chaîne en tableau de byte.
byte[]getBytes()	Convertit la chaîne en tableau de byte.
void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)	Copie les caractères de la chaîne dans le tableau en paramètres.
char[]toCharArray()	Convertit la chaîne de caractères en un tableau de caractères.

static String valueOf(boolean b)	Retourne la représentation en chaîne du booléen
static String valueOf(char c)	Retourne la représentation en chaîne du caractère
static StringvalueOf(char[] data)	Retourne la représentation en chaîne du tableau de caractères
static String valueOf(char[] data, int offset, int count)	Retourne la représentation en chaîne du tableau de caractères (partie)

static String valueOf(double d)	Retourne la représentation en chaîne du double
static String valueOf(float f)	Retourne la représentation en chaîne du float
static String valueOf(int i)	Retourne la représentation en chaîne du int
static String valueOf(long l)	Retourne la représentation en chaîne du long
static String valueOf(Object obj)	Retourne la représentation en chaîne de l'objet

- La recherchr dans une chaîne de caractère

int indexOf(int ch)	Retourne l'indice de la première occurrence du caractère
int indexOf(int ch, int fromIndex)	Retourne l'indice de la première occurrence du caractère à partir de *fromIndex*
int indexOf(String str)	Retourne l'indice de la première occurrence de la chaîne
int indexOf(String str, int fromIndex)	Retourne l'indice de la première occurrence de la chaîne à partir de *fromIndex*
int lastIndexOf(int ch)	Retourne l'indice de la dernière occurrence du caractère
int lastIndexOf(int ch, int fromIndex)	Retourne l'indice de la dernière occurrence du caractère à partir de *fromIndex*
int lastIndexOf(String str)	Retourne l'indice de la dernière occurrence de la chaîne
int lastIndexOf(String str, int fromIndex)	Retourne l'indice de la dernière occurrence de la chaîne à partir de *fromIndex*

Exemple :

```
public class Chaine1
  {
    String ch1="langage" ;
    String ch2= " java" ;
    String ch3=ch1+ch2;
    void affiche_longueur()
    { System.out.println(ch1+" a pour longueur " + ch1.length());
      System.out.println(ch2+" a pour longueur " + ch2.length());
      System.out.println(ch3+" a pour longueur " + ch3.length());
    }
    void affiche_caractere(String ch)
    {
      for(int i = 0;i<ch.length();i++)
        System.out.println(ch+" à l'indice "+i+" = "+ch.charAt(i));
    }
    public static void main(String[] arg)
    {   Chaine1 obj = new Chaine1();
      obj.affiche_longueur();
      obj.affiche_caractere(obj.ch3);
    }
  }
```

A l'exécution :

```
langage a pour longueur 7
 java a pour longueur 5
langage java a pour longueur 12
langage java à l'indice 0 = l
langage java à l'indice 1 = a
langage java à l'indice 2 = n
langage java à l'indice 3 = g
langage java à l'indice 4 = a
langage java à l'indice 5 = g
langage java à l'indice 6 = e
langage java à l'indice 7 =
langage java à l'indice 8 = j
langage java à l'indice 9 = a
langage java à l'indice 10 = v
langage java à l'indice 11 = a
```

8-2 Les chaînes de caractères de type StringBuffer

Pour une utilisation plus évoluée d'une chaîne de caractères on peut utiliser des objets de la classe java.lang.**StringBuffer** qui contient des méthodes qui ne sont pas définies dans la classe String (append,.setLength,…)

Les objets de cette classe contiennent des chaînes de caractères variables, ce qui permet de les agrandir ou de les réduire.

La classe StringBuffer dispose de nombreuses méthodes qui permettent de modifier le contenu de la chaîne de caractère

- Les Constructeurs de la classe StringBuffer

StringBuffer()	Construit une chaîne vide de capacité initiale de 16 caractères.
StringBuffer (int l)	Construit une chaîne vide de capacité initiale de l caractères.
StringBuffer (String s)	Construit une chaîne de caractères à partir de la chaîne *s*

int capacity() la capacité de la chaîne de caractères : longueur max

int length() la longueur de la chaîne de caractères

void setLength (int n) la longueur de la chaîne est *n* ; les caractères éventuellement ajoutés ont une valeur indéterminée.

- Concaténation

StringBuffer append(boolean b)	Concatène la représentation en chaîne du booléen.
StringBuffer append(char c)	Concatène la représentation en chaîne du caractère.
StringBuffer append(char[] str)	Concatène la représentation en chaîne du tableau de caractères.
StringBuffer append(char[] str, int offset, int len)	Concatène la représentation en chaîne du tableau de caractères.
StringBuffer append(double d)	Concatène la représentation en chaîne du double.
StringBuffer append(float f)	Concatène la représentation en chaîne du float.
StringBuffer append(int i)	Concatène la représentation en chaîne du int.

StringBuffer append(long l)	Concatène la représentation en chaîne du long.
StringBuffer append(Object obj)	Concatène la représentation en chaîne de l'objet
StringBuffer append(String str)	Concatène la représentation en chaîne de la chaîne.

- Caractère et sous-chaîne.

char charAt(int i)	Retourne le caractère à l'indice spécifié en paramètre.
String substring(int i)	Sous chaîne depuis *i* jusqu'à la fin
String substring(int i, int l)	Sous chaîne depuis *i* et de longueur *l*.

- Conversion

String toString()	Retourne une chaîne égale à la chaîne

- Modifications

StringBuffer delete(int start, int end)	Enlève tous les caractères compris entre *start* et *end*.
StringBuffer deleteCharAt(int index)	Enlève le caractère à l'indice *index*

Exemple : utilisation de la longueur et de la capacité

```
public class Chaine2
  {static StringBuffer stb1,stb2;
    public static void main(String[ ] arg)
    {  stb1=new StringBuffer();
       stb2=new StringBuffer(10);
       System.out.println("stb1 a pour longueur "+stb1.length());  //0
       System.out.println("stb1 a pour capacité "+stb1.capacity()); //16
       System.out.println("stb2 a pour longueur "+stb2.length()); //0
       System.out.println("stb2 a pour capacité "+stb2.capacity()); //10
```

```
        stb1=new StringBuffer("Chaine");
        System.out.println("stb1 a pour longueur "+stb1.length()); //6
        System.out.println(stb1+ " pour capacité "+stb1.capacity()); //22=16+6
        stb1.append(" StringBuffer");
        System.out.println(stb1+" a pour longueur "+stb1.length());//19
        System.out.println(stb1+" a pour capacité"+stb1.capacity()); //22
        stb1.setLength(3);
        System.out.println(stb1+" a pour longueur "+stb1.length());  //3
        System.out.println(stb1+" a pour capacité"+stb1.capacity());//22
    }
  }
```

9 – OPERATIONS SUR LES OBJETS

9 – 1 L'affectation =

ObjetA = ObjetB ObjetA désigne le même objet que ObjetB, c'est **la copie de référence**

Toute modification effectuée sur l'objet référencée par objetA sera répercutée sur l'objet référencé par objetB et vice versa.

Exemple :

```
public class Copie
{  public static void main(String[] argv)
        {
        int[ ] tableau = {1,2}
        int[ ] table ;
        table = tableau;
        table[0] =7;
        System.out.println(‘‘tableau contient :’’ + tableau[0] +tableau [1]);}
        /* tableau[0]=7  et tableau[1]=2                              */
        }
```

9–2 Egalité entre objets

- Si obj1 et obj2 sont deux références d'objets de la même classe. **(obj1== obj2)** est vraie si ces deux références obj1 et obj2 désignent le même objet

Exemple :

```
class C1
{  }
public class Egalite
{      public static void main( String[] arg)
   { C1 obj1,obj2,obj3;
       obj1=new C1();
       obj2=new C1();
      obj3=obj1;
      System.out.println(obj1==obj2);    // false
      System.out.println(obj1==obj3);    //true
   }
}
```

9– 3 L'objet null

L'objet **null** n'appartient pas à une classe, mais il peut être utilisé en lieu et place d'un objet de toute classe (il peut être retourné en cas d'erreur d'ouverture de fichiers, …)

```
class   Case
{       int val
        Case (int val )   // Constructeur de la classe Case
        {       this.val=val ;       }
        boolean estEgal (Case c)
        {       return ( val = = c.val );          }
}
public class   DemoNull
{       public static void main (String[] arg)
        {       Case c1 = new Case (10) ;
                Case c2 = new Case (15) ;
                Case c3 = null ;
                System.out.println (" c1 = =c2 :"+c1.estEgal(c2))  ;
                System.out.println (" c1= =c3 :"+ c1.estEgal(c3))  ;
                System.out.println ("c1= =null:"+c1.estEgal(null));     }
}
/* Le programme se compile correctement, en revanche son exécution est
interrompue  à la ligne c1.estEgal(c3) puisque c3 = null et on ne peut pas accéder à
c3.val (dans la méthode estEgal )                  */
```

Chapitre 3 **CLASSES ET HERITAGE EN JAVA**

1 – GENERALITES SUR L'HERITGE EN JAVA

1 – 1 Définition de l'héritage

1 –2 Déclaration de l'héritage

1 –3 Propriétés de l'héritage

1 – 4 Opérateurs : super et instanceof

1 –5 Un exemple utilisant l'héritage

2 – LA CLASSE java.lang.Object

2 - 1. La méthode public final Class getClass()

2 - 2. La méthode public String toString()

2 - 3. La méthode public boolean equals(Object obj)

2 - 4. La méthode protected void finalize()

3 – MASQUAGE DES ATTRIBUTS ET REDEFINITION DES METHODES

4 – CASTING OU TRANSTYPAGE

5 – LES INTERFACES

5 – 1 Définition d'une interface

5 –2 Présentation d'une interface

5 –3 Utilisation d'une interface

5 – 4 Exemple d'utilisation d'une interface

6 - LES CLASSES ABSTRAITES

Chapitre 3 **HERITAGE ET INTERFACE EN JAVA**

1 – GENERALITES SUR L'HERITGE EN JAVA

1-1 Définition de l'héritage

L'héritage est l'un des aspects les plus utiles de la programmation orientée objet. Il permet à une classe de transmettre ses attributs et ses méthodes à des sous classes. Ainsi, si une classe B hérite d'une classe A, alors la classe B contient implicitement l'ensemble des attributs non privés de A et peut invoquer toutes les méthodes non privées de la classe A.

1- 2 – Déclaration de l'héritage

En java, pour déclarer l'héritage, on utilise le mot clé **extends**

Syntaxe : **Class B extends A** :

- B **hérite** de A
- B est **dérivée** de A
- B est une **sous classe** de A
- A est la **super classe** de B (ou classe **de base**)

1-3 – Propriétés de l'héritage

- En java, l'héritage est simple , toute classe ne peut hériter que d'une et **une seule** classe. Si une classe n'hérite d'aucune autre classe, elle hérite par défaut de la super classe **Object** définie dans la package **java.lang**

- Une référence à une classe de base peut être affectée d'une référence à une classe dérivée, l'inverse doit faire l'objet d'une opération de conversion de type ou *cast* :

class A {}

class B extends A {}

Soient les instructions : A a= new A();
B b= new B();
a=b; // possible
b= a; // faux
b= (B) a; // possible

-**Exemple :**

```
class Pere
{     final String nom_famille; // nom de famille est constant par objet Pere
     Pere (String nom_famille) // constructeur
     {     this.nom_famille=nom_famille;     } }
class Fils extends Pere
{String prenom;
     Fils(String nom)   // Constructeur
     {super(nom);}    // Appel du constructeur de la super classe (Pere)
     public static void main(String[] arg)
     {  Pere sonPere= new Pere("SEDRATI");
     Fils monFils = new Fils(sonPere.nom_famille);
     // monFils est considéré de type Pere
     // Par consequent, il peut acceder au champ nom_famille
       System.out.println(" Nom de famille de mon fils  " +
        monFils.nom_famille);
     monFils.prenom="Anass";
     System.out.println(" Prénom de mon fils "+monFils.prenom);
     // sonPere.prenom="Driss"; Faux
     sonPere= monFils; // est possible car monFils est de même type //
que son père et par conséquent peut devenir Pere
     // monFils=sonPere; est faux ( la réciproque)     }
}
```

- Si la déclaration d'une classe est précédée de **final**, alors cette classe ne peut pas être une super classe

```
final class Mere
{}
class Fille extends Mere { }
    // erreur de compilation car la classe Mere ne peut avoir d'enfants
```

- Si le constructeur par défaut d'une classe dérivée est défini mais ne commence pas par un appel de constructeur de la classe mère, Java insère un appel du constructeur par défaut de la classe mère.

1-4- Opérateurs : super et instanceof

- L'opérateur super :

Pour faire référence à la classe parente (super classe) , il suffit d'utiliser le mot clé **super.** Ce mot permet d'invoquer un attribut ou une méthode de la super classe

On utilise **super(arguments)** pour invoquer le constructeur de la super classe ayant les arguments correspondants

- L'opérateur instanceof :

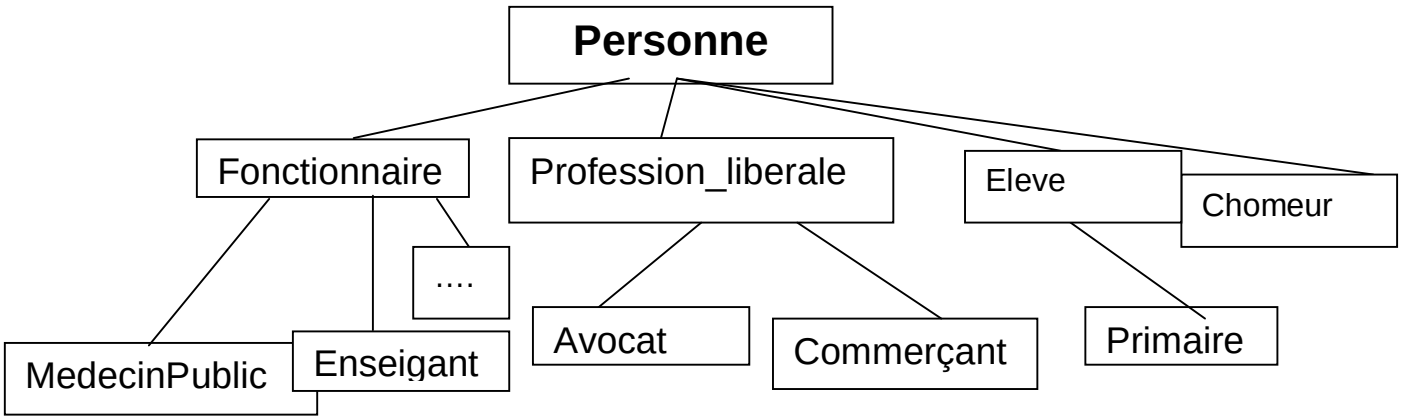
- L'opérateur ***instanceof*** permet de savoir à quelle classe appartient une instance :

```
class A
  { }
  class B extends A
  {  }
  public class C
  {
    public static void main(String[] arg)
```

```
    {
       A a= new A();
       B b = new B();
       C c = new C();
       System.out.println( a instanceof A  ); // true
       System.out.println( a instanceof B  );//false
       System.out.println( b instanceof A  );//true
       System.out.println( b instanceof B  );//true
       a=b;
       System.out.println( a instanceof A  );//true
       System.out.println( a instanceof B  );//true
       //System.out.println( c instanceof A  ); :erreur de compilation
        //(c et A ne sont pas de même type)
    }
}
```

1-5- Un exemple de l'héritage

Prenons la hiérarchie suivante :



- Les classes **Fonctionnaire, Profession_liberale**, **Eleve** et **Chomeur** sont dérivés de la classe **Personne**

```
class Personne     // la super classe
  {
    final String nom;
    final int annee_naissance;
    String adresse; // sauf pour les SDF!!
    Personne(String nom,int annee_naissance)
    {this.nom=nom;
      this.annee_naissance= annee_naissance;}
    String getNom()
    {       return nom;}
    int getNaissance()
    {       return annee_naissance;}
    public   String toString()
    {
     return("Je suis "+getNom()+" et je suis "+getClass()).getNom();}
  }
/*********************************************************/
  class Fonctionnaire extends Personne
  {int numeroSomme;
    double salaire;
    String fonction;
    Fonctionnaire(String nom,int annee_naissance,int numeroSomme)
    {super(nom,annee_naissance);
      this.numeroSomme=numeroSomme;
    }
    double getSalaire()
    {        return salaire;}
  }
/*****************************************************************/
  class MedecinPublic extends Fonctionnaire
  { String specialite;
    MedecinPublic(String nom,int annee_naissance,int numeroSomme,
                  String specialite)
    {
      super(nom,annee_naissance,numeroSomme);
      this.specialite=specialite;     }
  }
/*****************************************************************/
  class Profession_liberale extends Personne
  {int numeroCNSS;
    double salaire;
    String fonction;
    double patente; // impot
```

```
    Profession_liberale(String nom,int annee_naissance,
                        int numeroCNSS)
    {super(nom,annee_naissance);
      this.numeroCNSS=numeroCNSS;     }
  }
/*************************************************************/
  class Chomeur extends Personne
  {String diplome;
    Chomeur(String nom,int annee_naissance)
    {super(nom,annee_naissance);}
  }
/*********************************************************/
  class Eleve extends Personne
  {String classe;
    int numeroEleve;
    Eleve(String nom,int annee_naissance,String classe,
          int numeroEleve)
    {super(nom,annee_naissance);
      this.numeroEleve=numeroEleve;
      this.classe=classe;          }
  }
/*************************************************************/
  public class Heritage
  {
     public static void main(String[] arg)
    {  Personne[] desPersonnes= new Personne[4];
     desPersonnes[0]= new Fonctionnaire("F",1980,10);
     desPersonnes[1]= new MedecinPublic("M",1970,100," cardiologue");
    desPersonnes[2]= new Eleve("E",1990,"INE1",20);
    desPersonnes[3]= new Chomeur("C",1980);
    for(int i=0;i<desPersonnes.length;i++)
       System.out.println(desPersonnes[i]);
    }
  }
```

- A l'exécution :

```
Je suis F et je suis Fonctionnaire
Je suis M et je suis MedecinPublic
Je suis E et je suis Eleve
Je suis C et je suis Chomeur
```

2- LA CLASSE java.lang.Object

La classe d'entête **public class Object** est la super classe de toutes les classes Java : toutes ses méthodes sont donc héritées par toutes les classes.Voici quelques méthodes élémentaires de la classe Object

2.1. La méthode public final Class getClass()

La méthode **getClass()** renvoie un objet de la classe Class qui représente la classe de l'objet.

Le code suivant permet de connaitre le nom de la classe de l'objet

Exemple : String nomClasse = monObject.getClass().getName();

2-2 La méthode public String toString()

La méthode **toString()** permet d'obtenir la représentation d'un objet quelconque par une chaîne de caractères. Ainsi lorsque une référence r vers un objet de classe quelconque est dans une expression en lieu et place d'une chaîne de caractères c'est en fait la chaîne retournée par le message **r.toString()** qui est utilisée dans l'expression.

La méthode toString de la classe Object retourne une chaîne correspondant au **nom de la classe de l'objet** suivi de @ suivi de l'adresse mémoire de la structure de données définissant les caractéristiques de l'objet (cette adresse est la valeur contenue dans la variable référencant l'objet).

La méthode toString() est souvent redéfinie

Exemple 1: Utilisation de la méthode toString() héritée

Soit la classe Livre définie comme suit

```
public class Livre
{   double prix;
    String titre;
    String auteur;
    public Livre(double p, String t, String a)
    {   prix=p;
       titre = t;
       auteur = a;
    }
  public static void main(String[] arg)
  {   Livre monLivre= new Livre(50.00,"Programmation en langage
                          Java","Mme A. BENOMAR");
       System.out.println(monLivre);
     // La classe Livre ne redéfinissant pas la méthode toSTring
     // c'est celle héritée de la classe Object qui est invoquée.
  }
}
```

// A l'exécution, on aura : **Livre@ba34f2**

Exemple 2 : **Utilisation de la méthode toString() redéfinie**

```
public class Livre
{   double prix;
    String titre;
    String auteur;
```

```
    public Livre2(double p, String t, String a)
    {   prix=p;
        titre = t;
        auteur = a;
    }
```

public String toString()

```
{return ("Titre : " +titre +"\n"+"Auteur : " +  auteur + "\n" + "Prix : " + prix);}
```

```
  public static void main(String[] arg)
  {   Livre2 monLivre= new Livre2(50.00,"Programmation en langage
                          Java","Mme A. BENOMAR");
      System.out.println(monLivre);
    // La classe Livre ne redéfinissant pas la méthode toSTring
    // c'est celle héritée de la classe Object qui est invoquée.
  }
}
```

- A l'exécution, on aura :

:Titre : Programmation en langage Java

Auteur : Mme A. BENOMAR

Prix : 50.0

2-3. La méthode public boolean equals(Object obj)

La méthode equals() implémente une comparaison par défaut. Sa définition dans Object compare les références : donc obj1.equals(obj2) ne renverra true que si obj1 et obj2 désignent le même objet. Dans une

sous classe de Object, pour laquelle on a besoin de pouvoir dire que deux objets distincts peuvent être égaux, il faut redéfinir la méthode equals héritée de Object.

- Dans la classe Object cette méthode est définie comme suit :

```
public boolean equals(Object obj) {
        return (this == obj); }
```

Exemple 1 : Utilisation de la méthode equals héritée

```
public class LivreCompare
{   double prix;
   String titre;
   String auteur;
   public LivreCompare(double p, String t, String a)
   {  prix=p;
     titre = t;
     auteur = a;    }
   public static void main(String[] arg)
   {      LivreCompare l1 = new LivreCompare(50.00,"Programmation en langage Java","Mme A. BENOMAR");
     LivreCompare l2 = new LivreCompare(50.00,"Programmation en langage Java","Mme A. BENOMAR");
     System.out.print("Les livres référencés par l1 et l2 sont ");
     if (l1.equals(l2))  // ici équivalent à l1 == l2 puisque la méthode
                         //equals n'est pas redéfinie
     // On compare le contenu des deux références l1 et l2 (c'est à dire
      //les adresses mémoire des deux Livres
```

```
            System.out.println("identiques");
        else
            System.out.println("différents");
    }
  }
```

- A l'exécution : les livres référencés par l1 et l2 **sont différents**

Remarque : Pour avoir un comportement plus conforme à ce que l'on attend pour la méthode equals il faudrait redéfinir celle-ci

Exemple 2 : Utilisation de la méthode equals redéfinie

```
public class LivreCompare2
  {
    double prix;
    String titre;
    String auteur;
    public LivreCompare2(double p, String t, String a)
    {   prix=p;
       titre = t;
       auteur = a;     }
```

public boolean equals(LivreCompare2 l)

```
 {       return (this.titre.equals(l.titre) && this.auteur.equals(l.auteur) &&
this.prix == l.prix);  }
    public static void main(String[] arg)
    {   LivreCompare2 l1 = new LivreCompare2 (50.00, "Programmation
en langage Java","Mme A. BENOMAR");
```

LivreCompare2 l2 = new LivreCompare2 (50.00,"Programmation en langage Java","Mme A. BENOMAR");

System.out.print("Les livres référencés par l1 et l2 sont ");

if (**l1.equals(l2))** // ici on compare les attributs de l1 et l2

System.out.println("identiques");

else

System.out.println("différents");

}

- A l'exécution : les livres référencés par l1 et l2 **sont identiques**

2-4. La méthode protected void finalize()

A l'inverse de nombreux langages orientés objet tel que le C++ ou Delphi, le programmeur Java n'a pas à se préoccuper de la destruction des objets qu'il instancie. Ceux-ci sont détruits et leur emplacement mémoire est récupéré par le ramasse miette de la machine virtuelle dès qu'il n'y a plus de référence sur l'objet.

La machine virtuelle garantit que toutes les ressources Java sont correctement libérées mais, quand un objet encapsule une ressource indépendante de Java (comme un fichier par exemple), il peut être préférable de s'assurer que la ressource sera libérée quand l'objet sera détruit. Pour cela, la classe Object définit la méthode protected finalize, qui est appelée quand le ramasse miettes doit récupérer l'emplacement de l'objet ou quand la machine virtuelle termine son exécution

3- Masquage des attributs et redéfinition des méthodes

- On dit qu'un attribut d'une classe masque un attribut d'une super classe s'il a le même nom qu'un attribut de la super classe

- On dit qu'une classe redéfinit une méthode d'une super classe si elle définie une méthode ayant même nom, même suite de types d'arguments et même valeur de retour qu'une méthode de la super classe

Il y'a une différence dans la façon dont le masquage d'attributs et la redéfinition des méthodes sont traitées en java

Supposons qu'une classe **A** possède un attribut **attr** et une méthode **meth** et qu'une classe **B** étendant A définisse un attribut de même nom attr et redéfinisse la méthode **meth**.

Considérons alors les déclarations suivantes :

A a // déclaration d' un objet a de la classe A

B b = new B() ; // declaration et creation d'un objet b de la classe B

a = b ; // Tout objet de B est objet de A

Alors :

b.attr est l'attribut attr de la classe **B**

a.attr est l'attribut attr de la classe **A**

Par contre :

b.meth est la méthode redéfinie dans **B**

a.meth est la méthode redéfinie dans **B**

L'exemple suivant illustre le masquage d'un attribut et la redéfinition d'une méthode :

```
class A
  {      int attr=10;
      void meth()
       {      System.out.println("Je suis la méthode de A");      }
  }

  class B extends A
  {
    int attr=20;
     void meth()
     {        System.out.println("Je suis la méthode de B");      }
  }

  public class Masque
  {     public static void main(String[] argv)
     {        A a ;
        B b = new B();
        a=b;  // car b est aussi un objet de la classe A (héritage)
        System.out.println("b.attr= "+b.attr);
        System.out.println("a.attr= "+a.attr);
        b.meth();
        a.meth();
     }
  }
```

A l'exécution , on aura :

```
            b.attr= 20
            a.attr= 10
           Je suis la méthode de B
            Je suis la méthode de B */
```

4- CASTING OU TRANSTYPAGE

Le **casting** (mot anglais qui signifie moulage), également appelé **transtypage**, consiste à effectuer une conversion d'un type vers un autre type. Le casting peut être effectué dans deux conditions :

* Vers un type plus général. On parle alors de **sur-casting**.

* Vers un type plus particulier. On parle alors de **sous- casting**.

- Dans le cas des **primitives,** le **sur-casting** consiste à convertir vers un type dont la précision est supérieure. C'est le cas, par exemple, de la conversion d'un **byte** en **short**, d'un **int** en **long** ou d'un **float** en **double**. On parlera en revanche de **sous-casting** lors de la conversion d'un type vers un autre de précision inférieure, par exemple de **double** en **float**, de **long** en **int** ou de **short** en **byte**.

Le **sous-casting** présente un risque de perte d'informations. C'est pourquoi Java est autorisé à effectuer de façon implicite un sur-casting, alors qu'un sous-casting doit être demandé explicitement en faisant précéder la valeur par l'indication du nouveau type entre parenthèses

Exemple :

```
class Casting
{     static float f1,f2;
      static double d1,d2;
      public static void main (String[] arg)
     {  // f1=10.5 ;       // erreur car 10.5 est considéré de type double
        f1=(float)10.5;   // sous casting 10.5 est considérée double
        d1=f1; // pas de problème sur casting implicite : float en double
         System.out.println("d1 est égal à  :"+d1); //d1 est égal à  :10.5
        d2=12.5;
       // f2=d2 est interdit sous-casting doit être demandé explicitement.
    f2= (float) d2;// conversion d'un double en float : sous casting explicite
     System.out.println("f2 est égal à  :"+f2);// f2 est égal à  :12.5    }  }
```

Casting des objets

Il est possible d'utiliser le casting d'objets dans le cas de l'héritage. Si une classe B hérite d'une classe A . Alors un objet de la classe A peut être convertti en objet de la classe B s'il est créé avec le constructeur de la classe B

- Soient A et B deux classes telles que : class B extends A
- Soit objA uin objet de la classe A A objA ;
- On peut convertir objA en type B ((B)objA) si objA est crée avec le constructeur de la classe B càd : objA=new B() ;

Exemple :

```
class Animal
{ }

class Chien extends Animal
{     int  taille = 80;       }

class Conversion
{        public static void main (String[] argv)
  {
        Chien chien; // déclaration d'un objet de type Chien
        Animal animal = new Chien (); //Chien étend Animal
       chien = (Chien)animal; // Conversion de l'objet animal en
        //objet Chien car animal est crée comme un chien
        System.out.println(" Ce chien mesure : "+ chien.taille +" cm");

         animal = new Animal(); // création d'un objet de type Animal
        // chien = (Chien)animal; erreur  d'execution car on tente de
   //convertir un  animal qui n'est pas crée comme un chien en chien
  System.out.println(" Tout animal ne peut pas être converti en chien");
      }
    }
```

5 - LES INTERFACES

5-1 – Définition d'une interface

Une interface est un type purement abstrait au sens où il ne définit aucune implémentation et ne comporte pas de constructeur : une interface déclare uniquement des méthodes publiques

Une interface ne contient que des prototypes de méthodes et des constantes ayant les modificateurs static et final

Une interface sert à regrouper des constantes et à traiter des objets qui ne sont pas tous nécessairement de la même classe mais qui ont en commun une partie . Cette partie sera gérée par l'interface

Les interfaces portent souvent des noms se terminant en able. Par exemple (Comparable ,Runnable, …)

5-2- Présentation d'une interface

Une interface se compose de deux parties :

- **l'en-tête**
- **le corps**

Entête d'une interface

[modificateur] interface **<Nom>** **[**extends **<interface>]**

[ ] : optionnel

< > : obligatoire

gras : mot clé

Remarques :

- Une interface peut avoir le modificateur public .
- Une interface est toujours abstract sans qu'il soit nécessaire de l'indiquer
- Elle peut être attribuée à un package . Si elle ne l'est pas, elle fera partie du package par défaut
- Une interface peut dériver de plusieurs autres interfaces (héritage multiple pour les interfaces et non pas pour les classes)

```
interface I1
{ }

abstract interface I2     //abstract  n'est pas nécessaire
{ }
public interface I3 extends I1,I2 //héritage multiple
{ }
```

Corps d'une interface

Le corps d'une interface contient une liste de constantes et de prototypes de méthodes.

La syntaxe d'une déclaration d'une interface est la suivante :

entête

{

déclarations des constantes et des prototypes de méthodes

}

Exemple **:**

```
interface Ecriture
 {      static final int LIGNE_MAX = 50 ;
        void  ecrire() ;
}
```

5- 3 Utilisation d'une interface

- Pour utiliser une interface, il faut avoir une classe qui l'implémente. Pour cela on emploie le mot clé **implements** suivi de la liste des interfaces dont la classe dérive , séparées par des virgules

exemple **: class A implemets I1, I2**

- Une interface I peut être implémentée par une classe A signifie :

  * Toutes les méthodes figurant dans l'interface I doivent être définies par la classe A avec le modificateur public

- Si une classe A implémente une interface I, les sous classes de A implémentent aussi automatiquement I

Une interface peut être utilisée comme une classe.

Si I est une interface, on pourra référencer par objI de type I n'importe quel objet d'une classe implémentant l'interface I et on pourra appliquer **à cet objet une méthode déclarée dans l'interface**

```
Interface  I
{……}
class A implements I
{.....}

class B implements I
{ .....}

public class C
{ public static void main(String[] arg)
{ I  objI;   // objI peut être aussi bien un objet de A que de B
 objI= new A() ;   // obji est considéré un objet de la classe A
objI = new B() ; // objI est considéré un objet de la classe B
/* Donc le type I de l'interface définit un type général pouvant représenter n'importe quel objet d'une classe implémentant I */
}
}
```

5- 4 Exemple d'utilisation d'une interface

```
   /* Fichier UtiliseInterface.java     */
  interface Demeure
  {
    void affiche_commentaire() ;            }
```

```
  class Appartement implements Demeure
  {
    public void affiche_commentaire()
   {  System.out.println (" La demeure est un joli appartement ! ") ;  }
  }
class Villa implements Demeure
{
 public void affiche_commentaire()
  {  System.out.println (" La demeure est une Villa splendide ! ") ; }
}
class Baraque implements Demeure
{
 public void affiche_commentaire()
 { System.out.println (" La demeure est une baraque seul abri des
                                pauvres") ;      }
}
public class UtiliseInterface
{
  public static void main(String[] argv)
 {
      Demeure[ ] table_demeures = new Demeure[3];
                              // tableau de 3  Demeures
       table_demeures[0]= new Appartement();
       table_demeures[1]= new Villa() ;
       table_demeures[2]= new Baraque() ;
       for (int i=0; i<table_demeures.length ; i++)
         table_demeures[i].affiche_commentaire();
    }
  }
```

Remarque : le fait d'utiliser un tableau a permis d'accéder à ses éléments par la boucle for (d'où l'intérêt d'avoir un type commun)

6 – LES CLASSES ABSTRAITES

Définition

- Une classe abstraite est une classe qui contient au moins une méthode abstraite, elle doit être déclarée avec le modificateur abstract

- Dans une classe, une méthode est dite abstraite si seul son prototype figure. Elle n'a pas de définition explicite

- Une classe abstraite ne peut pas être instanciée , il faut l'étendre pour pouvoir l'utiliser

Exemple : Une classe pour modéliser des formes géométriques planes

La classe **Forme** modélise des formes géométriques planes. Elle contient deux méthodes abstraites (**perimetre** et **surface**) et deux méthodes non abstraites (**estPlutotRond** et **description**)

```
abstract class Forme
{
        abstract float perimetre();
        abstract float surface();
        boolean estPlutotRond()
        {
                float lePerimetre=perimetre();
                return surface() <= lePerimetre*lePerimetre/16;

         }
        void description()
         {
          if (estPlutotRond())
              System.out.println (this + " s'etale au moins comme  un carre");
       else
            System.out.println (this + " s'etale  moins q'un carre");          }
}
```

La classe **Disque** étend la classe **Forme** et définit les deux méthodes abstraites de celle ci . Elle définit aussi la méthode **toString** de java.lang.Object

```
class Disque extends Forme
/* La classe Disque hérite de la classe Forme, elle doit définir toutes ses méthodes abstraites */
{
 private   int   rayon;

Disque( int rayon)
       { this.rayon=rayon ; }

float perimetre()
{
  return  (float)(2*Math.PI*rayon);  }
 float surface()
  {
   return  (float)Math.PI*rayon*rayon;   }

  public String toString()   // redéfinition de la méthode toString
 {
return ("Le disque de rayon "+ rayon);
 }
/ * lorsqu'on applique la méthode toString à un objet de la classe Disque, il y aure retour du message et avec println affichage de ce message    */
}    //fin de la classe
```

La classe **Rectangle** étend la classe **Forme** et définit les deux méthodes abstraites de celle ci . Elle définit aussi la méthode **toString** de java.lang.Object

```
class Rectangle extends Forme
 {
  private int longueur,largeur;

  Rectangle (int longueur, int largeur)  // Constructeur
  {
```

```
   this.longueur=longueur;
   this.largeur=largeur;
   }
  float perimetre()
  {
     return 2* (longueur+largeur) ;
  }
  float surface()
  {
    return longueur*largeur;
  }
  public String toString()
  {
  return ("Le rectangle de longueur "+longueur+ "et de largeur
"+largeur);
  }
}
```

La class **EssaiForme** montre qu'on peut référencer avec une variable de type Forme un objet Disque ou bien un objet Rectangle puisque ces deux dernières classes héritent de la classe Forme

```
class EssaiFormes
{
 public static void main(String[] argv )
  {
  Forme[ ] desFormes = new Forme[5];
      desFormes[0]=new Disque(1);
       desFormes[1]=new Disque(2);
       desFormes[2]=new Disque(3);
       desFormes[3]=new Rectangle(1,1);
       desFormes[4]=new Rectangle(2,1);
  for(int i = 0 ;  i < desFormes.length  ;  i++)
       desFormes[i].description();
  }    // fin de la méthode main ()
}      // fin de la classe EssaiFormes
```

BIBLIOGRAPHIE

http://www.infres.enst.fr/~charon/coursJava/

http://www.ensta.fr/java/java_index.html

http://www.developpez.com/java/cours.htm

http://www.larcher.com/eric/guides/javactivex/ :

http://cedric.cnam.fr/~farinone/Java/