



Licence en informatique de gestion

2^{ème} et 3^{ème} Année

Programmation Orientée Objet



Par

Edouard Ngor SARR

Enseignant-chercheur à UCAO-Saint Michel

ED2DS Université de Thiès

edouard.sarr@ucao.edu.sn

Année académique

2017-2018



Sommaire :

CHAPITRE 1 : Introduction à Java.....	6
1. Le Java	6
2. Exécution d'un programme Java	7
3. La compilation et l'interprétation	8
3.1. La compilation	8
3.2. Interprétation.....	9
4. Utilisation du JDK (Kit de développement Java)	11
5. Source de téléchargement du JDK.....	12
6. L'API de java	12
1. La fin d'une instruction.....	13
2. Les identificateurs	13
3. Type de variables	14
4. La déclaration.....	14
5. Les Opérateurs	16
5.1. Opérateurs arithmétiques	16
5.2. Opérateurs d'affectation	16
5.3. Opérateurs de comparaison.....	16
5.4. Opérateurs logiques	17
6. Les commentaires	17
7. L'Affichage	17
8. Les instructions conditionnelles et les boucles	17



8.1. Les instructions conditionnelles.....	18
8.1.1. L'instruction IF	18
8.1.2. L'instruction switch.....	19
8.2. Les boucles.....	20
8.2.1. L'instruction do... while.....	20
8.2.2. L'instruction While.....	21
9. L'instruction For	22
10. Les instructions de branchement.....	23
10.1. L'instruction break.....	23
10.2. L'instruction continue.....	27
 Chapitre 2: Le langage Java	29
1. L'Environnement de développement: ECLIPSE.....	29
2. Création d'un nouveau projet.....	31
3. Créer un nouveau package :.....	32
4. Créer une nouvelle classe.....	33
5. Mon premier programme en Java	34
6. Affichage.....	35
7. Classe et objet en Java	36
8. Destruction des objets: destructeur	39
9. Portée d'une classe dans un package	40
10. Portée des membres d'une classe ; attributs et methodes.....	40



11.1. Première méthode : utiliser un Bufferedreader	41
11.2. Deuxième méthode : utiliser un Scanner	43
12. Notions de variables, classe et méthodes final	45
12.1. Variable final	45
12.2. Méthodes final	45
12.3. Classes final	45
13. Notions d'attributs et méthodes static.....	45
13.1. Attribut static	45
13.2. Méthode static.....	46
14. Le mot clé this.....	47
15. Les Getter/ Setter	47
16. Les tableaux	52
16.1. Déclaration.....	52
16.2. Les tableaux multi-dimensionnels	54
17. Remplissage d'un tableau :	54
18. Affichage d'un tableau : Avec for	55
19. Cumul d'un tableau : Avec for	56
20. Minimum et Maximum d'un tableau : Avec for.....	56
21. Les chaînes de caractères	58
21.1. La classe "StringBuffer"	58
21.2. Quelques méthodes relatives à la classe "String"	59
22. Rappels.....	60



23. Travaux dirigés : N°1	60
---------------------------------	----

CHAPITRE 4 : L'HERITAGE	62
-------------------------------	----

1. Propriétés de l'héritage en Java	62
2. Destruction d'un objet	64
3. Méthodes et classes abstraites	64
3.1. Méthodes abstraites	64
3.2. Classes abstraites	64
4. Les interfaces	65
5. Exercice d'application :	68
6. Travaux dirigés N°2 :	70

CHAPITRE 5 : GESTION DES EXCEPTIONS	72
---	----

1. Mécanisme des exceptions	72
2. Le bloc try {...} catch {...}	72
3. Les exceptions personnalisées	76
4. TP 3 :	79

CHAPITRE 6 : LES ENTREES / SORTIES	80
--	----

1. Généralités sur les flots de données	80
---	----



CHAPITRE 1 : Introduction à Java

1. Le Java

Java est un langage de programmation très utilisé, notamment par un grand nombre de développeurs professionnels. Le langage Java a été développé par Sun Microsystems en 1995, aujourd'hui racheté par Oracle.. Il possède de nombreuses caractéristiques :

*C'est un langage **orienté objet***

Une de ses plus grandes forces est sa **portabilité** : une fois votre programme créé, il fonctionnera automatiquement sous Windows, Mac, Linux, etc. C'est un langage **compilé** : avant d'être exécuté, il doit être traduit dans le langage de la machine sur laquelle il doit fonctionner.

Il emprunte sa syntaxe en grande partie du langage C. On peut faire de nombreux types de programmes avec Java :

- Les programmes Java peuvent être exécutés sous forme d'applications indépendantes ou distribuées à travers le réseau et exécutées par un navigateur Internet sous forme d'applets.
- des applications, sous forme de fenêtre ou de console ;
- des applets, qui sont des programmes Java incorporés à des pages Web ;
- des applications pour appareils mobiles, comme les smartphones, avec J2ME (Java 2 Micro Edition) ;
- des sites web dynamiques, avec J2EE (Java 2 Enterprise Edition, maintenant JEE) ;
- d'autres : JMF (Java Media Framework), J3D pour la 3D. . .

Avec les langages évolués courant (C++, C, etc.) nous avons pris l'habitude de coder sur une machine identique à celle qui exécutera nos applications ; la raison est fort simple : à de rares exceptions près les compilateurs ne sont pas multiplateformes et le code généré est spécifique à la machine qui doit accueillir. Nous devons alors utiliser n compilateurs différents sur n machines.

Aujourd'hui, la généralisation des interfaces graphiques et l'usage de langage plus évolués compliquent encore d'avantage le problème. Ainsi pour développer une application destinée à plusieurs systèmes d'exploitation avec ses différentes couches de bibliothèques et d'interfaces ; les API de ces interfaces étant toutes différentes. Ainsi nos applications sont fortement dépendantes des ressources (y compris graphique) du système hôte, dépendantes des API des interfaces utilisées, et le code produit ne peut s'exécuter que sur le système pour lequel il a été initialement produit. Tout d'abord, Java simplifie le processus de

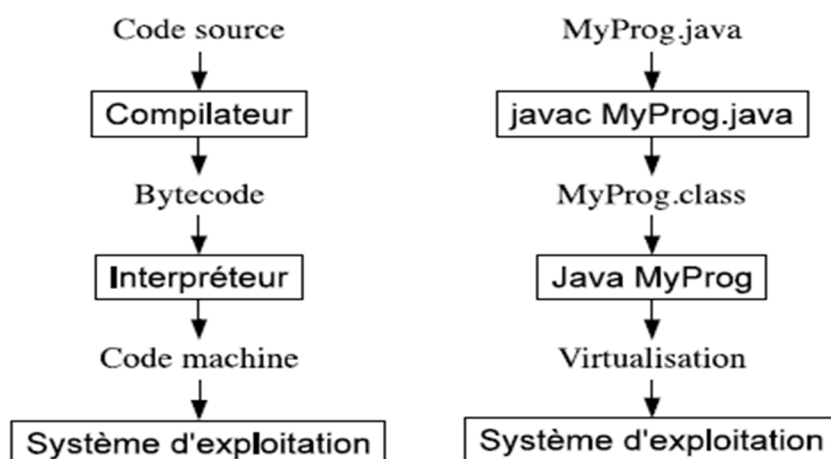
développement : quelle que soit la machine sur laquelle on code, le compilateur fournit le même code (LA PORTABILITE). Ensuite, quel que soit le système utilisé cet unique code est directement opérationnel: En effet, la compilation d'une source Java produit du **pseudo-code** (byte code) Java qui sera exécuté par tout **interpréteur** Java sans aucune modification ou recompilation. Cet interpréteur est couramment dénommé **machine virtuelle Java** ou JVM. Tous les types de matériels disposant d'une JVM : Téléphone, Ordinateur, Tablette...

2. Exécution d'un programme Java

Java est donc un langage de programmation, un langage dit compilé : il faut comprendre par là que ce que vous allez écrire n'est pas directement compréhensible et utilisable par votre ordinateur. Nous devons donc passer par une étape de compilation (étape obscure où votre code source est entièrement transformé). En fait, on peut distinguer trois grandes phases dans la vie d'un code Java :

- la phase d'édition du code source, en langage Java ;
- la phase de compilation de votre code ;
- la phase d'exécution.

Ces phases sont les mêmes pour la plupart des langages compilés (C, C++. . .). Par contre, ce qui fait la particularité de Java, c'est que le résultat de la compilation n'est pas directement utilisable par votre ordinateur.



Un programmeur Java écrit son code source, sous la forme de classes, dans des fichiers dont l'extension est .java. Ce code source est alors compilé par le compilateur javac en un langage appelé bytecode et enregistre le résultat dans un fichier dont l'extension est .class. Le bytecode ainsi obtenu n'est pas



directement utilisable. Il doit être interprété par la machine virtuelle de Java qui transforme alors le code compilé en code machine compréhensible par le système d'exploitation. C'est la raison pour laquelle Java est un langage portable : le bytecode reste le même quel que soit l'environnement d'exécution.

Nous avons deux types de développements sont possibles :

- les Applets : Un programme s'exécutant au sein d'un navigateur web. Téléchargeable en même temps qu'une page web donc Pas besoin d'installation. Ils ont des fonctionnalités limitées (par mesure de sécurité)
- les applications : Un programme standard s'exécutant sur la JVM.

3. La compilation et l'interprétation

3.1.La compilation

La compilation s'effectue par la commande `javac` suivie d'un ou plusieurs nom de fichiers contenant le code source de classes Java. Par exemple :

```
javac TesteSARR.java
```

Compile la classe TestSARR dont le code source est situé dans le fichier TestSARR.java. La compilation nécessite souvent la précision de certains paramètres pour s'effectuer correctement, notamment lorsque le code source fait référence à certaines classes situées dans d'autres répertoires que celui du code compilé. Il faut alors ajouter l'option **-classpath** suivie des répertoires (séparés par un ; sous Windows et : sous Unix) des classes référencées.

Par exemple :

```
javac -classpath /prog/exos1:/cours TestSARR.java
```

Compilera le fichier TestSARR.java si celui-ci fait référence à d'autres classes situées dans les répertoires /prog/exos1 et /cours. Le résultat de cette compilation est un fichier nommé TestSARR.class contenant le bytecode correspondant à la source compilée. Ce fichier est créé par défaut dans le répertoire où la compilation s'est produite. Il est cependant fortement souhaitable de ne pas mélanger les fichiers contenant le code source et ceux contenant le bytecode. Un répertoire de destination où sera créé le fichier TestSARR.class peut être précisé par l'option **-d**.

Exemple :

```
javac -d /prog/exos1 -classpath /cours MyProg.java
```

3.2.Interprétation

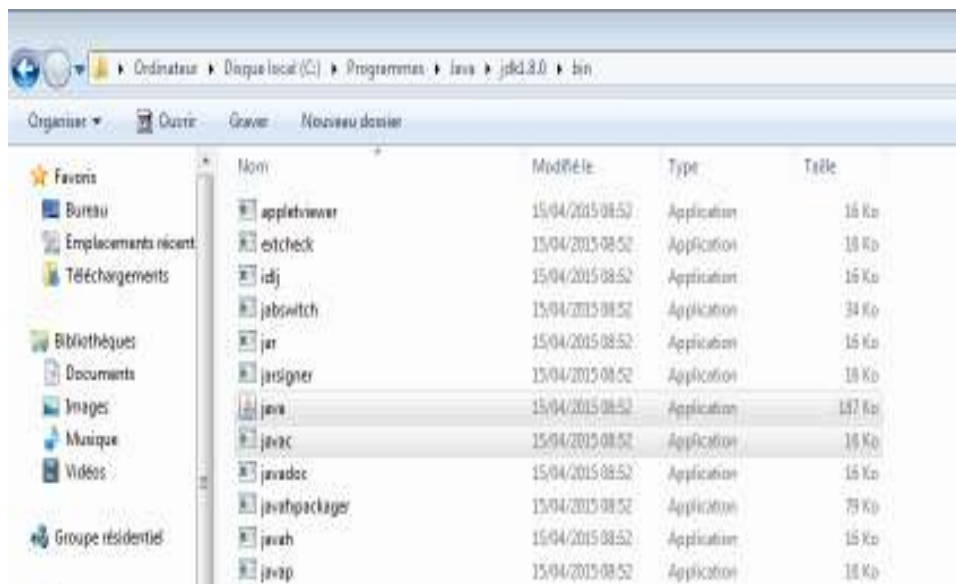
Le bytecode obtenu par compilation ne peut être exécuté qu'à l'aide de l'interpréteur. L'exécution s'effectue par la commande java suivie du nom de la classe à exécuter (sans l'extension .class). Comme lors de la compilation, il se peut que des classes d'autres répertoires soient nécessaires. Il faut alors utiliser l'option **-classpath** comme dans l'exemple qui suit :

```
java -classpath /prog/exos1:/cours MyProg
```

Exemple :



Le JDK seul contient les outils nécessaires pour compiler avec **JAVAC** et exécuter avec **JAVA**. Pour voir ces programmes, installer le JDk et vérifier dans : `C:\Program Files\Java\jdk1.8.0\bin`





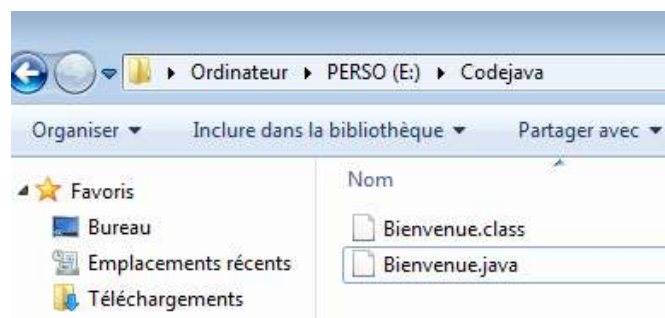
TP : Soit le programme suivant :

```
package COURS;  
public class Bienvenue {  
    public static void main(String[] args) {  
        String Nom="Edouard Ngor SARR";  
        System.out.println("Bienvenue à Ucao :"+Nom);  
    }  
}
```

Essayer de le compiler sous DOS: Se positionner dans le dossier bin de votre JDK et taper la commande Javac suivit du chemin et du nom du programme Java

```
C:\>cd C:\Program Files\Java\jdk1.8.0\bin  
C:\Program Files\Java\jdk1.8.0\bin>javac E:\Codejava\Bienvenue.java  
C:\Program Files\Java\jdk1.8.0\bin>
```

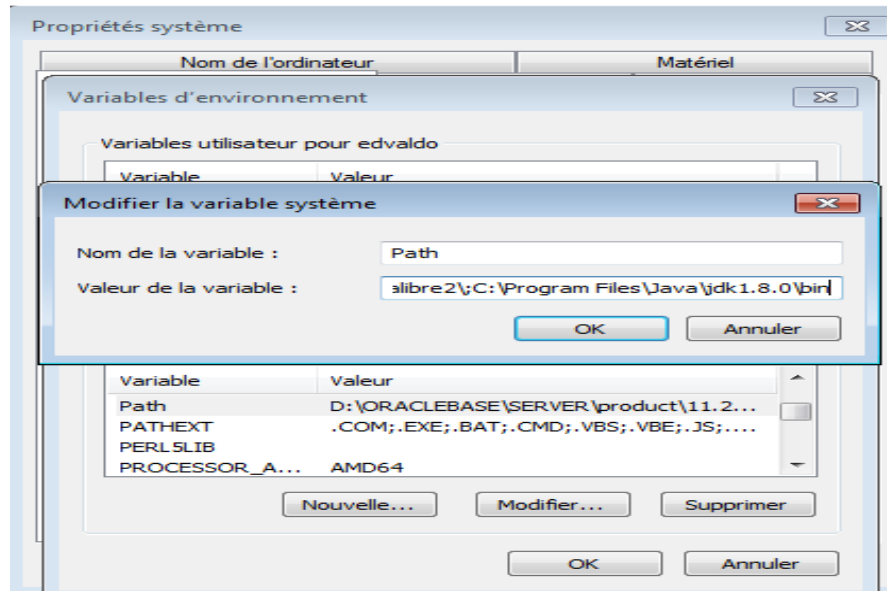
Si vous vérifiez maintenant, vous devez constater la création d'un nouveau fichier avec l'extension .class.



NB : Afin d'éviter de se positionner dans le dossier bin de votre JDK pour taper **javac**, il est possible de l'ajouter comme variable d'environnement ou PATH. Aller faire un clic droit sur Ordinateur, puis Paramètres système avancés, variables d'environnement, dans variables système, choisir PATH et cliquer sur Modifier.

Ajouter un point virgule ; puis le chemin du Bin du jdk :

C:\Program Files\Java\jdk1.8.0\bin



Fermer CMD et relancer le. Vous n'aurez plus l'erreur disant que JAVAC n'est pas reconnue comme commande interne.

```
C:\Users\edvaldo>javac D:\code\Bienvenue.java
C:\Users\edvaldo>
```

On peut exécuter en tapant la commande **JAVA** suivit du nom du fichier.class. le .class n'est pas nécessaire.

```
C:\Users\edvaldo>javac D:\code\Bienvenue.java -d D:\code\class\
C:\Users\edvaldo>
```

4. Utilisation du JDK (Kit de développement Java)

C'est une boîte à outils qui permet le développement de programmes en Java. Il est constitué de plusieurs outils tel que :

javac.exe : compilateur
java.exe : interpréteur ou JVM
jdb.exe : debugger...

Et d'une importante librairie de classe (API). Le **JRE** (Java Runtime Environment) contient uniquement l'environnement d'exécution de programmes Java. Le JDK contient lui même le JRE. Le JRE seul doit être installé sur les machines ou des applications java doivent être exécutées.



Depuis sa version 1.2, Java a été renommé en Java 2. Les numéros de versions 1.2 et 2 désignent donc la même version. Le JDK a été renommé en J2SDK (Java 2 Software Development Kit) mais la dénomination JDK reste encore largement utilisée. Le JRE lui aussi a été renommé en J2RE (Java 2 Runtime Édition). Quatre éditions de Java existent :

- *J2ME :*

- *Java 2 Micro Édition ·*
- *Applications sur environnement limité*
- *systèmes portables tel MOBILE*
- *systèmes de navigation embarqués*

- *Application lourde Java*

- *Applet*

- *J2EE :*

- *Java 2 Entreprise Édition ·*
- *API pour applications d'entreprise (accès bases de données)*
- *EJB(composants métiers)*
- *JSP(Java Server Pages)*
- *Servlet (HTML dynamique)*

5. Source de téléchargement du JDK

Pour telecharger le JDK de Java : <http://www.sun.com/products/index.html>

6. L'API de java

L'A.P.I. (Application Programming Interface) est un ensemble de classes utilisables par le programmeur. Un programme JAVA est constitué d'un certain nombre de classes :

- *Des classes prédéfinies de l'API (environ un millier de classes) (Application Programming Interface).*
- *Des classes définies par l'utilisateur*

TP : Telecharger et installer le JDK 8



Chapitre 2 : Le langage du Java

1. La fin d'une instruction

Le langage C a servi de base pour la syntaxe du langage Java. Le caractère de fin d'une instruction est le point virgule “;”. Toute instructions en java se termine par un ; sinon le compilateur ne pourra pas l'interpreter.

2. Les identificateurs

Un identificateur est un nom d'une variable, d'une constante ou d'une methode (fonctions ou procédures). Les identificateurs en java acceptent les caractères de : {a..z}, {A..Z}, {0..9} avec un \$ ou _. Le chiffre {0..9} ne doivent pas être le premier caractère de l'identificateur.

Exemple :

<i>identificateurs valides</i>	<i>identificateurs non valides</i>
<i>X</i> <i>Prenom1</i> <i>NOM</i> <i>\$Adresse</i> <i>mon_entier</i>	<i>mon-entier</i> <i>4prenom</i>

Nb : Il faut évidemment que l'identificateur ne soit pas un mot réservé du langage Java :

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



Attention : Java distingue minuscules et majuscules. On dira que le Java prend compte de la CASSE. Ainsi Valeur diffère de VALEUR qui est lui aussi différent de VALEUr.

Exemple : La variable TOTO est différente de la variable toTO

3. Type de variables

En Java on dispose des mêmes types qu'en langage C (int, float, char...). On dispose d'en plus du type *boolean* (1 bit) et *String* pour chaîne de caractères. Ce type a deux valeurs possibles *false* et *true* (initialisation à false). Chaque variable ou constante est typée.

Type	Classe eq.	Valeurs	Portée	Défaut
boolean	Boolean	true ou false	N/A	false
byte	Byte	entier signé	{-128..128}	0
char	Character	caractère	{\u0000..\uFFFF}	\u0000
short	Short	entier signé	{-32768..32767}	0
int	Integer	entier signé	{-2147483648..2147483647}	0
long	Long	entier signé	$\{-2^{31}..2^{31}-1\}$	0
float	Float	réel signé	$\{-3,4028234^{38}..3,4028234^{38}\}$ $\{-1,40239846^{-45}..1,40239846^{-45}\}$	0.0
double	Double	réel signé	$\{-1,797693134^{308}..1,797693134^{308}\}$ $\{-4,94065645^{-324}..4,94065645^{-324}\}$	0.0

4. La déclaration

Déclarer une variable c'est lui affecté un type afin de permettre au compilateur de lui réserver une zone en mémoire en équivalence à la taille destinée a ce type.

Syntaxe :

Type Nom_Variable ;

Exemple :

```
int x ;  
Double resultat ;  
String Prenom ;
```



Nb : la déclaration multiple est possible :

```
int x, y, res ;
```

A l'inverse du langage C, Java est un langage très rigoureux sur le typage des données. Il est interdit d'affecter à une variable la valeur d'une variable d'un type différent si cette seconde variable n'est pas explicitement transformée.

Exemple :

```
int a ;  
Double b = 5.3 ;  
a = b ;
```

Est interdit. Faudra faire un **transtypage(ou Casting)** c'est-à-dire changer le type de b avant de mettre sa valeur dans a. Pour faire cela c'est très simple :

```
(Int) b ;
```

Cette instruction transforme le contenu de b 5.3 en entier, il est alors égal à 5. Le programme devient :

```
int a ;  
Double b = 5.4 ;  
a = (int) b ;
```

Autre Exemple :

```
int a ;  
Double b  
a = (int)b ;
```

Nous venons de convertir un Double en entier. La Conversion automatique est seulement possible lorsque le compilateur sait que la variable destination est assez grande pour contenir la valeur source.

Exemple : Allant de l'entier au double.



```
Int  a
Double b
b = a ;
```

Est correcte car on peut mettre un entier dans un Double.

5. Les Opérateurs

5.1.Opérateurs arithmétiques

Ce sont :

```
+ Addition
- soustraction
* Multiplication
/ Division
% Modulo
++ incrementation
-- Decrementatio
```

Exemple :

```
int X, Y, W,A,Z ;
Z= X +Y ;
W= Z /A ;
```

5.2.Opérateurs d'affectation

```
= Egale
+= Plus égale
-= Moins égale
*= multiplication égale
/= diviser égale
```

5.3.Opérateurs de comparaison

```
< Inferieur
> Supérieur
<= Inferieur ou égale
>= Supérieur ou égale
```




5.4. Opérateurs logiques

! Différent
&& Et
|| Ou

6. Les commentaires

Les commentaires (instruction non traités par le compilateur) se situent entre les symboles `/*` et `*/` pour plusieurs lignes commencent par le symbole `//` pour une seule ligne

Exemple :

```
int a ;  
// ce commentaire tient sur une ligne  
int b ;  
/*Ce commentaire nécessite  
lignes*/
```

7. L’Affichage

Pour afficher dans la console nous allons utiliser **System.out.println**. Pour afficher une expression. Exemple :

```
System.out.println ("Bienvenue ") ;  
System.out.println("donnez le prix hors taxes : ") ;
```

Pour Afficher plusieurs éléments on fait un concaténation avec +

```
System.out.println ("Le prix ttc " + ttc) ;  
System.out.println ("Le prix ttc " + ttc + "FCFA ") ;  
System.out.println ("Le double du prix ttc " + 2*ttc +);
```

8. Les instructions conditionnelles et les boucles

A priori, dans un programme, les instructions sont exécutées séquentiellement, c’est-à-dire dans l’ordre où elles apparaissent. Or la puissance et le comportement intelligent d’un programme proviennent essentiellement de la possibilité de s’affranchir de cet ordre pour effectuer des choix, des boucles (répétitions).



Java (comme C) dispose d'instructions structurées permettant de réaliser :

- *des choix : instructions if...else et switch,*
- *des boucles (répétitions) : instructions do... while, while et for.*

Toutefois, la notion de branchement n'est pas totalement absente de Java puisque, comme nous le verrons il dispose d'instructions de branchement inconditionnel :

break Pour arrêter
continue Pour continuer

8.1. Les instructions conditionnelles

8.1.1. L'instruction **IF**

If est égal à Si en français. C'est une instruction conditionnelle. Un bloc est une suite d'instructions placées entre accolades {et}. Dans l'If, Le mot ELSE définit l'instruction facultative donc le cas par défaut. Lorsque nous avons plus de deux cas (gérable avec IF et ELSE), nous introduisons le ELSEIF. Nous aurons autant de ELSEIF que de nombre de cas – 2.

```
If (Condition)
{ ..... }
ElseIf (Condition)
{ ..... }
Else
{ ..... }
```

Si nous avons une seule instruction il n'est pas nécessaire de mettre les { }

Exemple :

```
Int X, Y ;
X=12 ;
Y=10
If (X<=Y)
    System.out.println ("OK") ;
Else
    System.out.println ("No") ;
```



8.1.2. L'instruction **switch**

Switch nous permet de gérer les choix des utilisateurs. C'est le SELON en Algorithmme. Il fait la même chose que le IF.

La Syntaxe

```
switch (variable)
{
    case valeur0 : instructions;
        break ;
    case valeur1 : instructions;
        break ;
    case valeur2 : instructions;
        break ;
}
```

Exemple :

```
public class ClasseSwitch1
{
    public static void main (String[] args)
    {
        int n ;
        System.out.print ("donnez un nombre entier : ") ;
        n = Clavier.lireInt() ;
        switch (n)
        {
            case 0 : System.out.println ("nul") ;
                break ;
            case 1 : System.out.println ("un") ;
                break ;
            case 2 : System.out.println ("deux") ;
                break ;
        }
        System.out.println ("Au revoir");
    }
}
```



Il est possible d'utiliser le mot-clé **default** comme étiquette à laquelle le programme se branchera si aucune valeur satisfaisante n'a été rencontrée auparavant. C'est comme le ELSE de IF. En voici un exemple :

```
public class Default
{
    public static void main (String[] args)
    {
        int n ;
        System.out.print ("donnez un nombre entier : ") ;
        n = Clavier.lireInt() ;
        switch (n)
        {
            case 0 : System.out.println ("nul") ;
                break ;
            case 1 : System.out.println ("un") ;
                break ;
            default : System.out.println ("deux") ;
        }
        System.out.println ("Au revoir");
    }
}
```

8.2.Les boucles

8.2.1. L'instruction **do... while**

La boucle Do While permet de réaliser des boucles. Une boucle est une ou plusieurs instructions qui se réalisent de manière réplétive. L'instruction : do while (condition) ; répète l'instruction qu'elle contient (ici un bloc) tant que la condition mentionnée (condition) est vraie. A priori, on ne sait pas combien de fois une telle boucle sera répétée. Toutefois, par sa nature même, elle est toujours parcourue au moins une fois. En effet, la condition qui régit cette boucle n'est examinée qu'à la fin de chaque répétition (comme le suggère d'ailleurs le fait que la partie while). Notez bien que la sortie de boucle ne se fait qu'après un parcours complet de ses instructions et non dès que la condition mentionnée devient fausse.

```
Do
{
} While(Condition) ;
```



Notez bien le point virgule après While(Condition)

Exemple :

```
public class DoI
{
    public static void main (String args[])
    {
        int n ;
        do
        {
            System.out.print ("donnez un nombre >0 : ") ;
            n = Clavier.lireInt() ;
            System.out.println ("vous avez fourni " + n) ;
        }
        while (n <= 0) ;
        System.out.println ("reponse correcte") ;
    }
}
```

8.2.2. L'instruction **While**

Le WHILE permet aussi de réaliser des boucles. Elle répète l'instruction qui suit (ici un bloc) tant que la condition mentionnée est vraie, comme le ferait do... while. Mais cette fois, la condition de poursuite est examinée avant chaque parcours de la boucle et non après. Ainsi, contrairement à ce qui se passait avec do... while, une telle boucle peut très bien n'être parcourue aucune fois si la condition est fausse dès qu'on l'aborde.

Syntaxe :

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

Exemple :

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



```
int n, som ;  
som = 0 ;  
while (som < 100)  
{  
    System.out.print ("donnez un nombre : " ) ;  
    n = Clavier.lireInt() ;  
    som += n ;  
}  
System.out.println ("Somme obtenue : " + som) ;  
}  
}
```

Là encore, notez bien la présence de parenthèses pour délimiter la condition de poursuite. En revanche, la syntaxe n'impose aucun point-virgule de fin. La condition de poursuite est évaluée avant le premier tour de boucle. Il est donc nécessaire que sa valeur soit définie à ce moment-là. Si tel n'est pas le cas, le compilateur vous le signalera.

9. L'instruction **For**

Le For Permet de réaliser des boucles. Mais ici nous connaissons d'avance le nombre fois à boucler.

Syntaxe :

```
For (i=Valinitial ; i<= ValFinale ; incrementation)  
{  
    .....  
}
```

Exemple:

```
for (i=1 ; i<=5 ; i++)  
{.....  
}
```

La ligne **for (i=1 ; i<=5 ; i++)** comporte en fait trois expressions :

- La première **i=1** ; correspond à l'initialisation d'un compteur est évaluée (une seule fois) avant d'entrer dans la boucle.



- La deuxième conditionne $i \leq 5$; correspond à la condition d'arrêt celle de la poursuite de la boucle. Elle est évaluée avant chaque parcours.
- La troisième $i++$ correspond à l'incrément du compteur est évaluée à la fin de chaque parcours.

Exemple :

```
public class For1
{
    public static void main (String args[])
    {
        int i ;
        for (i=1 ; i<=5 ; i++)
        {
            System.out.print ("bonjour ") ;
            System.out.println (i + " fois") ;
        }
    }
}
```

Comme dans tous les langages, il faut prendre des précautions avec les compteurs qui ne sont pas de type entier.

10. Les instructions de branchement

10.1. L'instruction **break**

Nous avons déjà vu le rôle de break au sein du bloc régi par une instruction switch. Java vous autorise également à employer cette instruction dans une boucle (while, do... while ou for).

Dans ce cas, elle sert à interrompre le déroulement de la boucle, en passant à l'instruction suivant la boucle.

Bien entendu, cette instruction n'a d'intérêt que si son exécution est conditionnée par un choix ; dans le cas contraire, en effet, elle serait exécutée dès le premier tour de boucle, ce qui rendrait la boucle inutile. Voici un exemple illustrant le fonctionnement de break :



```
public class Break
{
    public static void main (String args[])
    {
        int i ;
        for (i=1 ; i<=10 ; i++)
        {
            System.out.println ("debut tour " + i) ;
            System.out.println ("bonjour") ;
            if (i==3) break ;
            System.out.println ("fin tour " + i) ;
        }
        System.out.println ("apres la boucle") ;
    }
}
```

L'instruction break ordinaire a l'inconvénient de ne sortir que du niveau (boucle ou switch) le plus interne. Dans certains cas, cela s'avère insuffisant. Par exemple, on peut souhaiter sortir de deux boucles imbriquées. Un tel break ; ne convient pas alors. En fait, Java permet de faire suivre ce mot-clé break d'une étiquette **repet** qui doit alors figurer devant la structure dont on souhaite sortir :

```
repet : while
{ .....
    for (.....)
    {
        .....
        break repet ; // cette fois, ce break nous branche
    }
}
```

Exemple avec deux boucles imbriquées

```
package chapitre2;
public class Cfor {
    public static void main(String[] args) {
        int i;
        int j;
```




```
repet: for (i=1; i<=3; i++)  
{  
    System.out.println(i + " éme");  
    for (j=1; j<=4; j++)  
    {  
        System.out.println(" " + j + " éme");  
    }  
}  
}
```

Resultat normal :

```
1 éme  
  1 éme  
  2 éme  
  3 éme  
  4 éme  
2 éme  
  1 éme  
  2 éme  
  3 éme  
  4 éme  
3 éme  
  1 éme  
  2 éme  
  3 éme  
  4 éme
```

Essayons maintenant de sortir des deux boucles avec un Break simple :

```
package chapitre2;  
public class Cfor {  
    public static void main(String[] args) {  
        int i;  
        int j;  
        repet: for (i=1; i<=3; i++)  
        {  
            System.out.println(i + " éme");  
            for (j=1; j<=4; j++)
```



```
        {  
            System.out.println(" " + j + " éme");  
            if ((i==2) && (j==3)) break repet;  
        }  
    }  
}
```

Resultat :

```
1 éme  
  1 éme  
  2 éme  
  3 éme  
  4 éme  
2 éme  
  1 éme  
  2 éme  
  3 éme  
3 éme  
  1 éme  
  2 éme  
  3 éme  
  4 éme
```

Utilisons un break repet :

```
package chapitre2;  
  
public class Cfor {  
  
    public static void main(String[] args) {  
        int i;  
        int j;  
        repet: for (i=1; i<=3; i++)  
        {  
            System.out.println(i + " éme");  
            for (j=1; j<=4; j++)
```

```
        {  
            System.out.println(" " + j + " éme");  
            if ((i==2) && (j==3)) break repet;  
        }  
    }  
}
```

Resultat :

```
1 éme  
 1 éme  
2 éme  
3 éme  
4 éme  
2 éme  
 1 éme  
 2 éme  
3 éme
```

10.2. L'instruction **continue**

L'instruction continue permet de passer prématurément au tour de boucle suivant. En voici un premier exemple avec for :

```
public class Contin1  
{  
    public static void main (String args[])  
    {  
        int i ;  
        for (i=1 ; i<=5 ; i++)  
        {  
            System.out.println ("debut tour " + i) ;  
            if (i<4) continue ;  
            System.out.println ("fin tour " + i) ;  
        }  
        System.out.println ("apres la boucle") ;  
    }  
}
```



En fait, comme pour break, Java permet de faire suivre ce mot-clé continue d'une étiquette qui doit alors figurer devant la structure sur laquelle on souhaite boucler :

```
repet : while (...)  
{ .....  
    for (...)  
    { .....  
        continue repet ; // ce continue nous fait poursuivre la boucle  
while  
    .....  
    }  
}
```

Chapitre 2: Le langage Java

1. L'Environnement de développement: ECLIPSE

Rappel : le JDK est une boîte à outils qui permet le développement de programmes en Java. Il est constitué de plusieurs outils tel que :

*javac.exe : compilateur
java.exe : interpréteur ou JVM
jdb.exe : debugger...
Et d'une importante librairie de classe (API).*

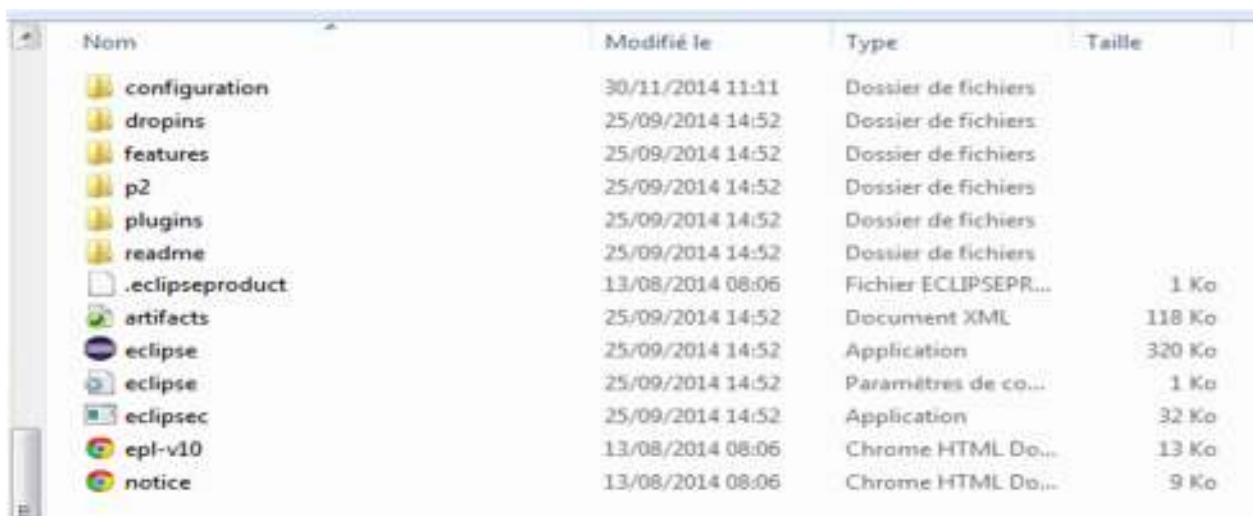
Le **JRE** (Java Runtime Environment) contient uniquement l'environnement d'exécution de programmes Java. Le JDK contient lui même le JRE. Le JRE seul doit être installé sur les machines où des applications java doivent être exécutées. Pour commencer nous allons télécharger un JDK sur

<http://www.sun.com/products/index.html>

Téléchargement de Eclipse sur :

- <http://www.commentcamarche.net/download/telecharger-34075627-eclipse-ide-pour-developpeurs-java>
- <http://www.jetelecharge.com/Developpement/1738p.php>

Dezipper le fichier et vous trouverez le dossier eclipse. Ouvrez-le.



Nom	Modifié le	Type	Taille
configuration	30/11/2014 11:11	Dossier de fichiers	
dropins	25/09/2014 14:52	Dossier de fichiers	
features	25/09/2014 14:52	Dossier de fichiers	
p2	25/09/2014 14:52	Dossier de fichiers	
plugins	25/09/2014 14:52	Dossier de fichiers	
readme	25/09/2014 14:52	Dossier de fichiers	
.eclipseproduct	13/08/2014 08:06	Fichier ECLIPSEPR...	1 Ko
artifacts	25/09/2014 14:52	Document XML	118 Ko
eclipse	25/09/2014 14:52	Application	320 Ko
eclipse	25/09/2014 14:52	Paramètres de co...	1 Ko
eclipsec	25/09/2014 14:52	Application	32 Ko
epl-v10	13/08/2014 08:06	Chrome HTML Do...	13 Ko
notice	13/08/2014 08:06	Chrome HTML Do...	9 Ko

Attention : Eclipse ne s'installe pas. Créer simplement un raccourci de eclipse sur le bureau. Puis aller sur le bureau et double cliquez y.

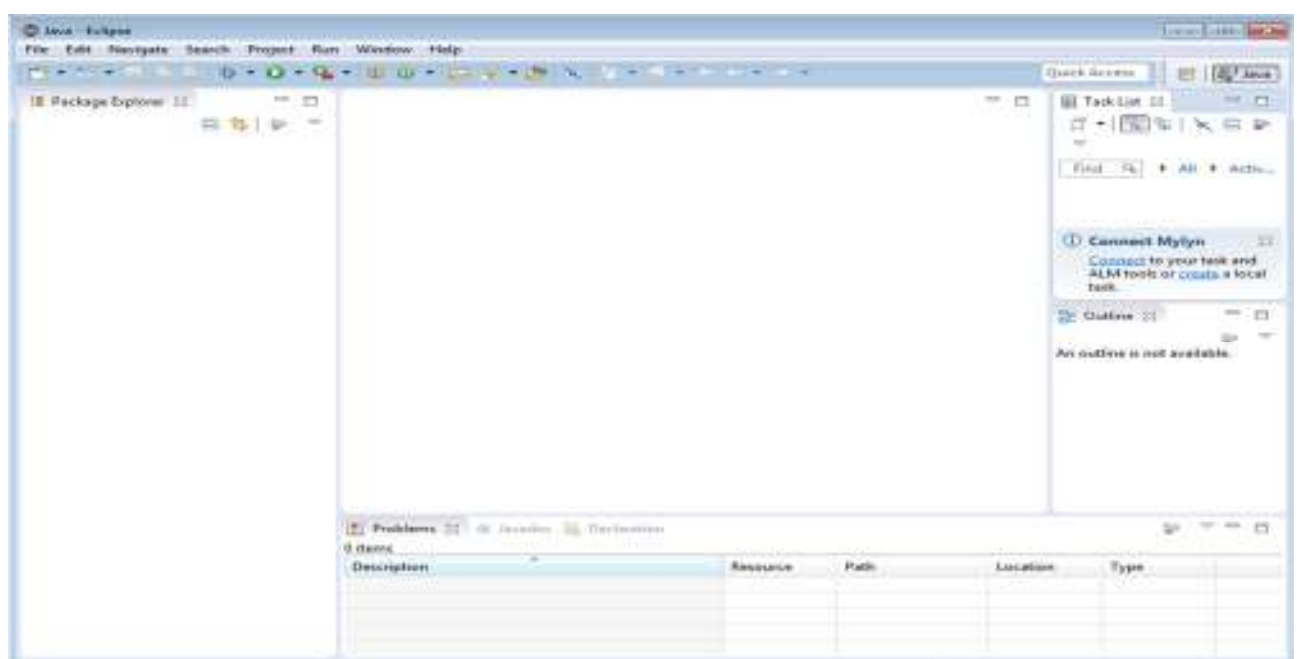




Pour le premier lancement vous devez choisir le **workspace** c'est-à-dire l'emplacement (le repertoire) ou vous allez enregistrer vos projets. Voici ensuite la page d'accueil :

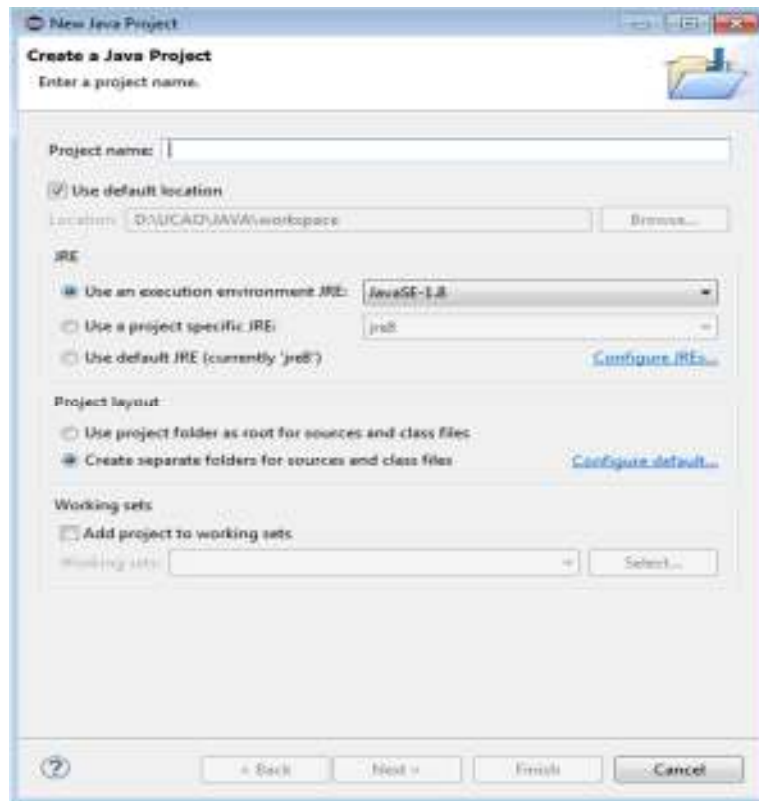


Nous pouvons la fermer.

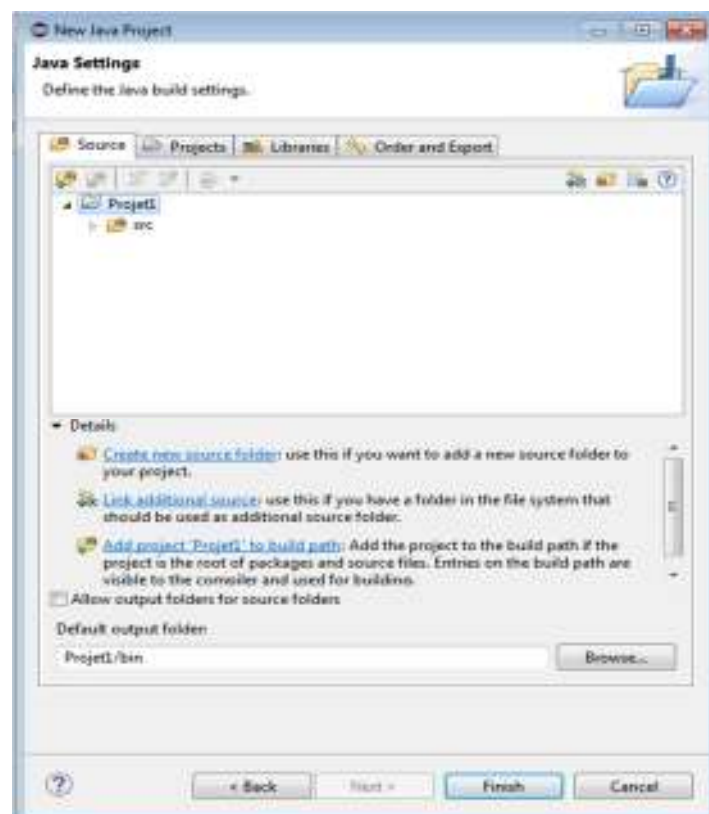


2. Création d'un nouveau projet.

Pour créer un nouveau projet, aller dans File puis New et ensuite java Project



Donnez un nom a notre projet,



C'est fini. Vous verrez le projet créer dans l'explorateur de solution juste à gauche en haut.

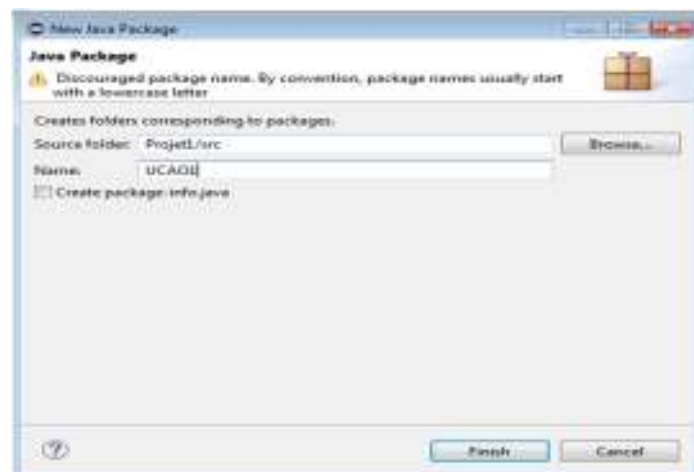


3. Créer un nouveau package :

Un paquetage ou package est un regroupement de classes. On regroupe dans un même paquetage des classes ayant une thématique et des fonctionnalités communes. Un paquetage contient des classes et des sous-paquetages. Les classes de l'API appartiennent à plusieurs packages (Exemple : java.util, java.awt...). Pour utiliser une classe qui appartient à un package de l'API, on utilise l'instruction **import** suivi du nom du package. Exemple :

- Pour utiliser la classe *Vector* du package *java.util* on écrit : `import java.util.Vector ;` au tout début du programme.
- Pour utiliser toutes les classes du package *java.util* on écrit : `import java.util.* ;`

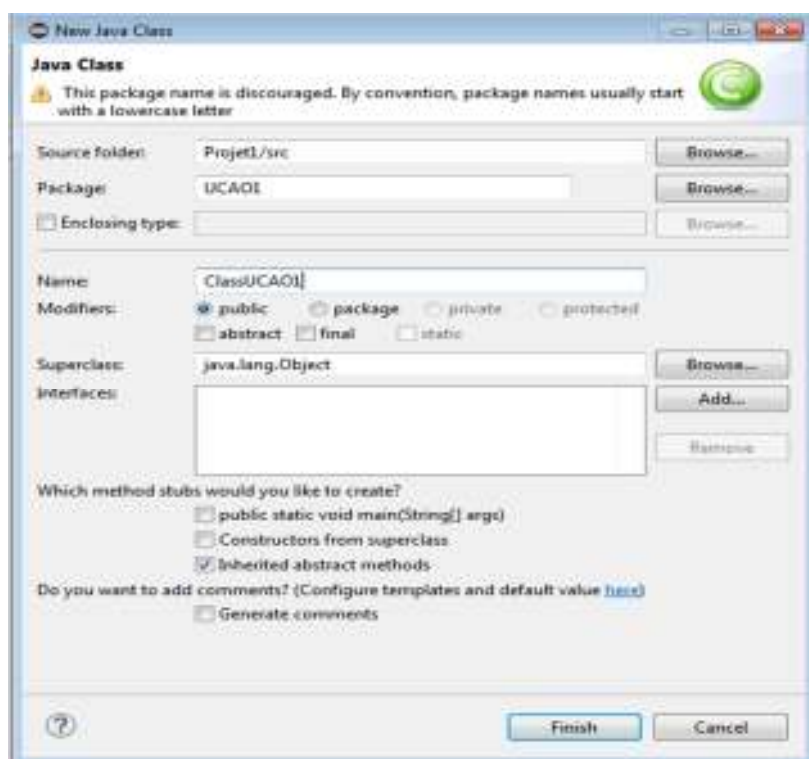
Si vous souhaitez qu'une classe que vous avez créé appartienne à un package particulier, vous devez le spécifier explicitement au moyen de l'instruction `package`. Il faudra aussi créer un répertoire au nom du package et sauvegarder les classes appartenant au package dans ce répertoire.



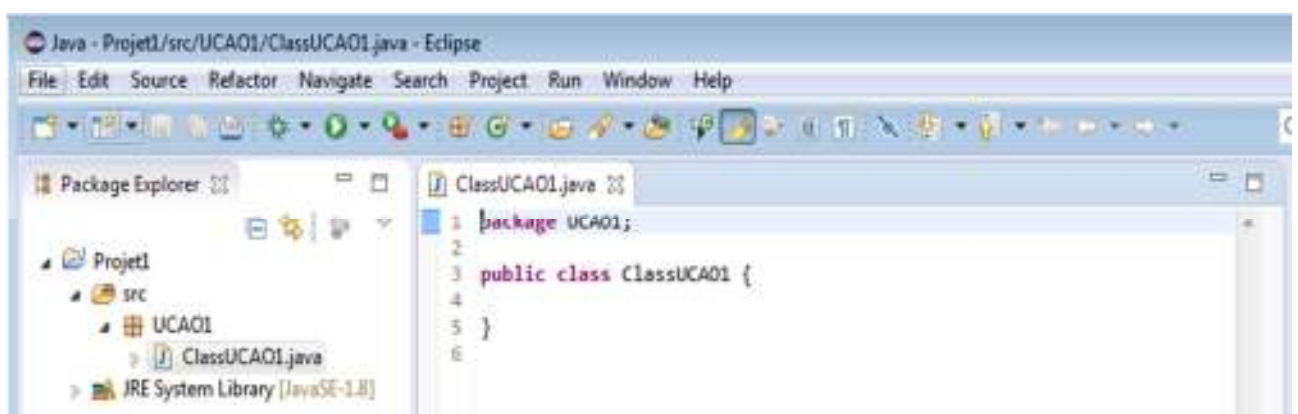


4. Créer une nouvelle classe.

Aller dans File, new puis Class



Une fois la classe créée, vous verrez le début du code montrant le paquage et la définition de la classe.





Remarque : lorsqu'on écrit un programme en Java il faut éviter de donner au fichier .java un nom différent de la classe. Par exemple, si pour sauvegarder la classe Etudiants nous choisissons le nom Profs.java à la fin de la compilation le fichier Profs.class sera créé. Il y aura ensuite des erreurs au niveau de l'exécution car la classe profs n'existe pas.

5. Mon premier programme en Java

Pour pouvoir exécuter un programme en Java, il faut définir une fonction particulière appelée : **main**. Cette méthode doit être contenue dans une classe (le tout objet), elle possède un en-tête obligatoire fixé :

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

- *static* : signifie qu'elle ne pourra faire appel qu'aux champs statiques de la classe
- *String[] arg* : tableau d'arguments du programme

Exemple: Bonjour UCAO

```
package UCAO1;  
public class PremierClass {  
    public static void main(String[] arg)  
    {  
        System.out.println("Bonjour UCAO" );  
    }  
}
```

Exercice d'application Sous Eclipse Créer :

```
Un projet nommé Projet1  
Un package nommé UCAO1  
Une Classe pour faire l'addition entre 1000 et 3426
```

Solution :

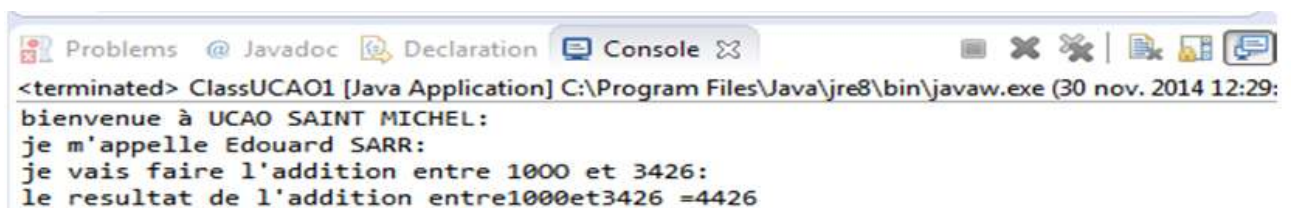
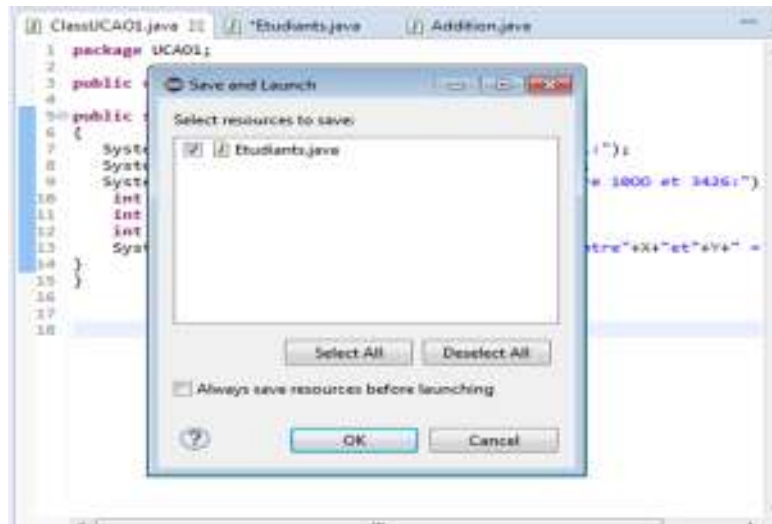
```
package UCAO1;  
public class ClassUCAO1 {  
    public static void main(String[] arg)  
    {
```

```

System.out.println("bienvenue à UCAO SAINT MICHEL:");
System.out.println("je m'appelle Edouard SARR:");
System.out.println("je vais faire l'addition entre 1000 et 3426:");
int X = 1000;
int Y = 3426;
int Res = X+Y;
System.out.println("le resultat de l'addition entre"+X+"et"+Y+" =" +
Res);
}
}

```

Exécution: Avec éclipse cliquez sur



6. Affichage



Pour afficher sur écran, on utilise la méthode **println**. La méthode *println* appartient à la classe **System.out**.



Exemple :

```
System.out.println("toto")
System.out.println("le resultat est :"+ RES)
System.out.println("l'addition entre :"+X+" et "+ Y+" est "+ RES)
```

NB : /n permet d'aller à la ligne.

```
System.out.println("toto  /n")
```

System.Out se trouve dans le package java.util on peut donc l'importer avec :
Import java.util. ;*

7. Classe et objet en Java

Une classe permet de définir un type d'objet, éventuellement compliqué, associant des données et des procédures qui accèdent à ces données. Les données se présentent sous forme de champ désignés par des identificateurs et dotés d'un type. A une donnée, on associe un mot clé qui désigne l'accessibilité de cette donnée (la portée). Un Objet une instance de la classe.

Les méthodes définissent les opérations possibles sur ces composants. A une méthode aussi, on associe un mot clé désignant l'accessibilité de cette méthode.

TP : Créer un projet nommé Gestion des Cours, dans la quelle nous avons le packages PETudiant avec les classes :

Etudiants

- *Propriétés :*

- *Prenom*
- *Nom*
- *Matricuel*
- *Adresse*
- *Age*
- *Sexe*

- *Methodes :*

- *listerEtudiants()*
- *ModifierEtudiants(.....)*



Une classe GestionETudiants avec la methode : Static void main dans la quelle :

- nous intancions deux objets Toto et Tata
- nous listons les infos des deux objets
- nous modifions les infos de toto
- nous listons encore les infos de toto
- nous modifions l'adresse et le nom de l'objet tata pour lui donner ceux de toto
- nous listons encore toto et tata

Correction : La classe Etudiants :

```
package PEtudiants;
public class Etudiants {
    String matricule, prenom, nom, sexe, adresse;
    int age;
    /*******methode pour lister les infos d'un etudidiant*****/
    public void listerEtudiants()
    {
        String titre;
        if(this.sexe == "M")
            titre = "Mr";
        else
            titre = "Mme";

        System.out.println(titre + " " + this.prenom + " " + this.nom + " Adresse
        =" + this.adresse + " Age =" + this.age);
    }
    public void ModifierEtudiants(String matricule, String prenom, String
    nom, String adresse, String sexe, int age )
    {
        this.prenom = prenom;
        this.nom = nom;
        this.matricule = matricule;
        this.adresse = adresse;
        this.sexe = sexe;
        this.age = age;
        System.out.println( "Etudiant bien modifié");
    }
}
```



```
/**construteur***/
public Etudiants(String matricule, String prenom, String nom, String
adresse, String sexe, int age )
{
    this.prenom=prenom;
    this.nom=nom;
    this.matricule=matricule;
    this.adresse=adresse;
    this.sexe=sexe;
    this.age=age;
    System.out.println( "Etudiant bien créé");
}
}
```

La classe Etudiants possède cinq attributs. La methode ModiferEtudiants(String matricule, String prenom, String nom, String adresse, String sexe, int age)

Le mot clé *private* indique que l'attribut est privé il ne sera accessible que par les méthodes appartenant à la classe. Un attribut public est accessible par une méthode appartenant à une autre classe.

Remarque : Le corps (contenu) des méthodes d'une classe sera défini dans la classe elle-même.

Initialisation des objets : constructeur

Une classe peut définir une ou plusieurs procédures d'initialisation appelées *constructeurs*.

Un constructeur est une méthode avec aucune valeur de retour et qui porte le même nom que la classe. S'il existe plusieurs méthodes dans une classe, elles se distinguent par le type ou le nombre de paramètres: le Polymorphisme.

Exemple de constructeurs :

```
public Etudiants()
public Etudiants(Pe String, Nm String, Ag int)
```



Peuvent être des constructeurs de la classe Etudiants. Le constructeur permet de créer une instance de la classe. Une instance de classe est dite *objet*. Pour créer une instance de la classe Etudiants (on dit objet de la classe Etudiants), il faut exécuter **new** en citant un constructeur de la classe Etudiants, avec ses paramètres. La primitive new rend en résultat la référence sur l'objet créé, qu'il suffit alors d'affecter à la variable déclarée :

Exemple :

```
Etudiants Toto = new Etudiants () ;
```

Est un Objet de la classe Etudiants instancier via le constructeur Etudiants ()

```
Etudiants Toto = new Etudiants(P, N, A);
```

Est un Objet de la classe Etudiants instancier via le constructeur Etudiants ()

8. Destruction des objets: destructeur

Avec Java, plus besoin d'écrire une fonction destructeur. En effet, il existe un programme appelé 'garbage collector' (ramasseur d'ordures) qui est exécuté automatiquement dès que la mémoire disponible devient inférieure à un certain seuil. La classe GEtudiants :

```
package PEtudiants;
public class GEtudiants {
    public static void main(String[] args) {
        Etudiants toto = new Etudiants("M01", "Samba", "SENE", "Dakar", "M", 72);
        Etudiants tata = new Etudiants("M02", "Elisa", "Diaham", "Dakar", "F", 22);
        toto.listerEtudiants();
        tata.listerEtudiants();
        toto.ModiferEtudiants("M03", "Malick", "SARR", "Mbour", "M", 28);
        toto.listerEtudiants();
        tata.nom=toto.nom;
        tata.adresse=toto.adresse;
        toto.listerEtudiants();
        tata.listerEtudiants()
    }
}
```




Resultat :

Etudiant bien créé
Etudiant bien créé
Mr Samba SENE Adresse =Dakar Age=72
Mme Elisa Diaham Adresse =Dakar Age=22
Etudiant bien modifié
Mr Malick SARR Adresse =Mbour Age=28
Mr Malick SARR Adresse =Mbour Age=28
Mme Elisa SARR Adresse =Mbour Age=22

9. Portée d'une classe dans un package

La portée d'une classe est définie comme la zone dans laquelle elle est visible. Seules les classes publiques sont accessibles depuis l'extérieur du package.

<i>Modificateur de portée</i>	<i>Portée</i>
<i>Aucun</i>	<i>Le paquetage</i>
<i>Public</i>	<i>Partout</i>

10. Portée des membres d'une classe ; attributs et methodes

- **protected** : Les membres d'une classe peuvent être déclarés *protected*. Dans ce cas, l'accès en est réservé aux méthodes des classes dérivées, des classes appartenant au même package ainsi qu'aux classes appartenant au même package que les classes dérivées.
- **private** : Accès au niveau de la classe
- **public** : accès partout

Remarque : Un attribut sans modificateur (Getter/Setter) n'est pas accessible depuis l'extérieur du paquet de la classe.

11. saisir des donnees envoyees par le clavier

Si vous voulez lire des données envoyées tout simplement par le clavier, nous vous indiquons deux méthodes illustrées ci-dessous. Les deux programmes d'illustration ont pour objectif de lire des entiers envoyés par l'intermédiaire du clavier, et d'en faire la somme. Nous notons en rouge les instructions significatives sur le thème de la saisie des données au clavier :



11.1. Première méthode : utiliser un **BufferedReader**

On utilise deux classes de java.io, la classe **InputStreamReader** et la classe **BufferedReader**.

La classe **InputStreamReader** admet un constructeur **InputStreamReader (InputStream)**, c'est-à-dire un constructeur qui admet en paramètre un flot d'entrée. `System.in` est une instance de la classe **InputStream**. Avec une instance de **InputStreamReader**, on ne peut grosso modo que lire des caractères un à un.

La classe **BufferedReader** a un constructeur qui prend en argument une instance de **Reader** dont hérite la classe **InputStreamReader**. Cette classe permet en particulier de lire une ligne d'un texte, mais en revanche, on ne peut pas lui demander de lire un entier, un double etc...

NB : On lit ici ce qui est envoyé par le clavier ligne par ligne et on découpe le contenu de chaque ligne avec un `StringTokenizer` pour récupérer les entiers attendus. L'utilisateur devra taper return deux fois de suite pour interrompre la saisie.

Voici le programme que nous proposons.

```
package TESTE;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
class LireClavier
{
    public static void main (String[] argv) throws IOException
    {
        BufferedReader entree = new BufferedReader (new
        InputStreamReader(System.in));
        System.out.println("Donner votre nom") ;
        String NOM= entree.readLine();
        System.out.println("Votre nom est : "+ NOM);
    }
}
```



Exécution

Donner votre nom

SARR

Votre nom est : SARR

NB : pour récupérer des données de type numérique (int ou Double), il faut impérativement faire le transtypage avec les méthodes :

- *parseInt* pour les entiers
- *parseDouble* pour les réels

Exemple :

- *System.out.println ("Donner votre age");*
- *lage= Integer.parseInt(Clavier.readLine());*
- *System.out.println ("Donner votre note");*
- *lage= Double.parseDouble(Clavier.readLine());*

Exo 1 : Ecrire un programme nommé Afficher nom qui demande à l'utilisateur de lui donner le prénom, le nom et l'adresse. Il affiche à la fin le nom complet et l'adresse des utilisateurs

Correction Exo 1 :

```
package UCAOI;
import java.io.*;
import java.util.*;
public class AFFICHERNOM {
    public static void main (String[] argv) throws IOException,
    NumberFormatException
    {
        String prenom, nom, adresse;
        StringTokenizer st;
        BufferedReader entree = new BufferedReader(new
        InputStreamReader(System.in));
        System.out.println("Donner le prenom : ");
        prenom = entree.readLine();
        System.out.println("Donner le Nom : ");
```



```
nom = entree.readLine();
System.out.println("Donner l'Adresse : ");
adresse = entree.readLine();
System.out.println("Vous etes: "+prenom + " " +nom+" et vous habitez
au "+adresse );
}
}
```

Exo 2 : Ecrire un programme qui reçoit deux valeurs et revoie le double de leur somme.

Correction Exo 2 :

```
package UCAOI;
import java.io.*;
import java.util.*;
public class SOMME {
    public static void main (String[] argv) throws IOException,
    NumberFormatException
    {
        int x, y, somme;
        String Rep;
        BufferedReader entree = new BufferedReader(new
        InputStreamReader(System.in));
        System.out.println("Donner x : ");
        Rep = entree.readLine();
        x= Integer.parseInt(Rep);
        System.out.println("Donner y : ");
        y= Integer.parseInt(entree.readLine());
        somme=x+y;
        System.out.println("Le resultat est : "+2*somme );
    }
}
```

NB : pour la tranformation en Double, il y'a la methode `Double.parseDouble ()` ;

11.2. Deuxième méthode : utiliser un **Scanner**

Depuis le JDK 5.0 , la saisie à partir de la fenêtre d'exécution est plus simple. La classe **java.util.Scanner**, qui possède différents constructeurs dont un



de type `InputStream` que nous utilisons dans notre exemple, permet de saisir différents types de données.

Exemple d'importation :

```
//Ceci importe la classe Scanner du package java.util
import java.util.Scanner;
//Ceci importe toutes les classes du package java.util
import java.util.*;
```

La deuxième est plus simple mais très coûteuse en ressource.

Exemple : Formulaire de contacts

```
import java.util.*;
class SaisieClavierBis {
    public static void main (String[] arg) {
        java.util.Scanner entree = new java.util.Scanner\(System.in\);
        System.out.println("Donnez votre prénom et votre nom");
        String prenom = entree.next\(\);
        String nom = entree.next\(\);
        System.out.println("Donnez votre âge");
        int age = entree.nextInt\(\);
        entree.nextLine();
        System.out.println("Ecrire votre phrase");
        String phrase = entree.nextLine\(\);
        System.out.println(prenom + " " + nom + ", " + age + " ans, dit : " +
phrase);
    }
}
```

TP : Reprenez l'exo 1 et l'exo 2.

NB : En voulant recevoir des entiers, nous pouvons utiliser `entree.nextInt()` à la place de `entree.next()` ;. De même on peut utiliser `entree.nextDouble()` pour les doubles.

Attention, lorsque vous tapez un double par exemple 10,34 vous devez mettre la virgule à la place du point.



Donner votre taille

1.78 va générer une erreur

1,78 sera correcte

12. Notions de variables, classe et méthodes final

12.1. Variable final

Une variable déclarée *final* ne peut plus voir sa valeur modifiée

Exemple :

final int X=1 ;

X ne pourra plus être modifier.

12.2. Méthodes final

Les méthodes *final* ne peuvent pas être redéfinies dans les classes dérivées.

12.3. Classes final

Une classe final ne peut être étendue pour créer des sous-classes.

13. Notions d'attributs et méthodes static

13.1. Attribut static

Un attribut statique est un attribut partagé par tous les objets de la classe (tous les objets ont la même valeur pour cet attribut).

Exemple : *static int x=3* tous les objets instanciés via cette classe auront la même valeur pour x.

Exemple :

```
package cours;
public class employes {
    final int idservice=1010;
    static int salaire=200000;
    public employes(int id,int sal) {
```



```
        this.salaire=sal;
    }
    public void afficher()
    {
        System.out.println("ID          ="+"this.idservice+"
        SALAIRE="+this.salaire);
    }
    public static void main(String[] args) {
        employes E1=new employes(1011,300000);
        employes E2=new employes(1011,500000);
        employes E3=new employes(1011,900000);
        E1.afficher();
        E2.afficher();
        E3.afficher();
    }
}
```

Resultat :

```
ID =1010 SALAIRE=900000
ID =1010 SALAIRE=900000
ID =1010 SALAIRE=900000
```

Explication : Une variable finale se comporte comme une constante. Quelque soit l'objet qui le manipule, elle n'accepte pas une modification de valeur. Une variable STATIC retourne à tous les objets son contenu c'est-à-dire la dernière valeur prise.

13.2. Méthode static

Une méthode statique (déclarée avec le mot clé static : *public static void f()*) est une méthode qui n'a accès qu'aux membres static de la classe.

Exemple :

```
package cours;
public class Clients {
    int GlobV1;
```



```
static int GlobV2;  
public void test ()  
{  
    GlobV2=10; // une variable STATIC peut etre utiliser dans une methode non STATIC  
}  
public static void main(String[] args) {  
    int Loc;  
    GlobV2=13;  
    GlobV1=1; // genere une erreur car la variable n'est pas static  
}  
}
```

14.Le mot clé this

Le this désigne une référence sur l'objet courant. Dans un constructeur, Il permet de distinguer le paramètre du constructeur à l'attribut de la classe.

Exemple :

```
public class lespoints  
{  
    int x ;  
    int b ;  
    Point (int a, int b)  
    {  
        this.x = a ;  
        this.b = b ;  
    }  
}
```

Ici le This nous permet de signifier que nous parlons des variables courantes et non des parametres du constructeur.

15.Les Getter/ Setter

Par défaut une variable definit dans une classe en Java est de visibilité private. Si dans une classe nous souhaitons donner accès à une variable "maVariable" depuis l'extérieur de la classe, nous pouvons soit :



- Mettre la variable en lecture seule en définissant une fonction `GetMavvariable()` avec une visibilité public.
- Mettre la variable en écriture seule en définissant une fonction `SetMavvariable()` avec une visibilité public.
- définir des méthodes publiques `getMaVariable` et `setMaVariable` pour donner accès total à la variable depuis l'extérieur et donc possibilité pour les tierces parties de faire n'importe quoi (ex : mettre la variable dans un état "incohérent" par rapport au traitement).

NB ; Par défaut tous les attributs sont privés

Exemple : créer la classe Etudiants :

- le prenom en lecture et écriture pour tous
- le nom est en lecture pour tous mais en écriture que pour les objets fils
- l'âge n'est consultable que pour ses objets
- l'adresse n'est lisible que pour les objets fils

```
Package PEtudiants;
public class Etudiants {
    private String matricule, prenom,nom,sexe, adresse;
    int age;
    //*****constructeur*****
    public Etudiants(String matricule, String prenom, String nom, String
adresse, String sexe, int age )
    {
        this.prenom=prenom;
        this.nom=nom;
        this.matricule=matricule;
        this.adresse=adresse;
        this.sexe=sexe;
        this.age=age;
        System.out.println( "Etudiant bien créé");
    }
    //*****methode pour lister les infos d'un etudidiant***
    public void listerEtudiants()
    {
        String titre;
        if(this.sexe == "M")
            titre = "Mr";
    }
}
```




```
        else
            titre="Mme";
        System.out.println(titre+" "+this.prenom+" "+this.nom+" Adresse
="+this.adresse + " Age="+this.age);
    }
    //*****methode modifier etudiants*****
    public void ModifierEtudiants(String matricule, String prenom, String
nom, String adresse, String sexe, int age )
    {
        this.prenom=prenom;
        this.nom=nom;
        this.matricule=matricule;
        this.adresse=adresse;
        this.sexe=sexe;
        this.age=age;
        System.out.println( "Etudiant bien modifié");
    }
    //*****les getter*****
    public String getprenom()
    {
        return this.prenom;
    }
    public String getnom()
    {
        return this.nom;
    }
    private int getage()
    {
        return this.age;
    }
    protected String getadresse()
    {
        return this.adresse;
    }
    //*****setter*****
    public void setprenom(String newPrenom)
    {
        this.prenom=newPrenom;
    }
    protected void setnom(String newnom)
```

```
{  
    this.prenom=newnom;  
}  
}
```

Exemple : Créer une classe pour la gestion des clients d'une entreprise. Nous aurons à manipuler le Prénom, le Nom et le Matricule. Créer la méthode listerétudiants et la méthode Modifierétudiants. Faire des getters setters pour les propriétés de la classe. Créer la methode Gestion_Etudiants pour manipuler les etudiants.

```
package TESTE;  
public class Clients {  
    String Prenom, Nom;  
    int Matricule;  
    *****constructeur*****  
    public Clients (String P, String Nom, int M)  
    {  
        this.Prenom= P;  
        this.Nom=Nom;  
        this.Matricule=M;  
    }  
    *****Getter /setter pour prenom*****  
    public String get_Prenom()  
    {  
        return this.Prenom;  
    }  
    Public void set_Prenom(String NewPrenom)  
    {  
        this.Prenom=NewPrenom;  
    }  
    *****Getter /setter pour nom*****  
    public String get_nom()  
    {  
        return this.Nom;  
    }  
    public void set_nom(String Newnom)  
    {  
        this.Nom=Newnom;  
    }  
}
```



```
/**get set Matricule*/
public int get_Matricule()
{
    return this.Matricule;
}
public void set_Matricule(int NewMA)
{
    this.Matricule=NewMA; }
/**
public void ModifierEtudiant(String newP,String newN, int NewM)
{
    this.Prenom=newP;
    this.Nom=newN;
    this.Matricule=NewM;
}
/**lister*/
public void ListerEtudiant()
{
    System.out.println("le prenom est "+this.Prenom+ " le nom est
"+ this.Nom+ " le matricule est "+this.Matricule);
}
}
```

La classe main :

```
package TESTE;
import java.util.Scanner;
public class Gestionetudiant {
    public static void main(String[] args) {
        Scanner cl = new Scanner(System.in);
        System.out.println("Donner un prenom");
        String P=cl.next();
        System.out.println("Donner un nom");
        String N=cl.next();
        System.out.println("Donner un matricule");
        int M=cl.nextInt();
        /** intancier un objet de la classe etudiants */
        Etudiants toto =new Etudiants(P,N,M);
        /** Modifions les parametres de toto */
        toto.set_Prenom ("edouard");
    }
}
```



```
toto.set_Nom ("SARR");
toto.set_Matricule (1212);
// *****recuperons les infos de toto
System.out.println("Prenom"+toto.get_Prenom());
System.out.println("Prenom"+toto.get_Nom());
System.out.println("Prenom"+toto.get_Matricule());
//*****listons les infos de toto*****
toto.ListerEtudiant();
}
}
```

16. Les tableaux

16.1. Déclaration

Un tableau est une structure de données contenant des elements de meme type.

```
Type [ ] NomTableau ;
Ou
Type NomTableau[ ] ;
```

- Type : est le type de la variable.
- NomTableau : est le nom du tableau

Exemple:

```
Int [ ] x ;
où
int x [ ] ;
```

x est une référence correspondant à un tableau qui contient des entiers.

Initialisation

Il est aussi possible de declarer une variable tout en lui affectant des valeurs de depart.



Exemple :

```
int[] x;  
x = new int[3];
```

x est une référence sur un tableau de taille égale à 3

```
int  tableauEntier [ ] = {0,1,2,3,4,5,6,7,8,9};  
double tableauDouble [ ] = {0.0,1.0,2.0,3.0,4.0,5.0,6.0,7.0,8.0,9.0};  
char  tableauCaractere [ ] = {'a','b','c','d','e','f','g'};  
String tableauChaine [ ] = {"chaine1", "chaine2", "chaine3", "chaine4"};
```

Remarque :

- Si on ne fait pas d'initialisation, une initialisation automatique est alors faite :
- pour les tableaux de numerique à 0
- pour les tableaux de caractere à null
- Tab[0] indique le contenu de la premiere case du tableau.
- Tab[1] indique le contenu de la deuxieme case du tableau.
- Tab[3] indique le contenu de la quatrieme case du tableau.
- La taille d'un tableau est fixe. Néanmoins, il n'est pas nécessaire qu'elle soit connue au moment de la compilation (la declaration). Dans ce cas, on peut connaître la taille d'un tableau avec le champs **length** (System.out.println(y.length)).

Exemple

```
Import java.util.*;  
class Test2 {  
    public static void main(String[] arg) {  
        int x = 5;  
        int [ ] y;  
        y = new int[ x ] ;  
        System.out.println( x );  
    }  
}
```

Exemple : Nous allons afficher la taille du tableau.



```
Import java.util.*;
class Test2 {
    public static void main(String[] arg){
        int x = 5;
        int[] y;
        y = new int[x] ;
        System.out.println(x);
        System.out.println(y.length)
    }
}
```

16.2. Les tableaux multi-dimensionnels

Syntaxe de declaration d'un tableau à deux dimensions

```
int[][] x;
x = new int[2][4];
```

Initialisation d'un tableau à deux dimensions

```
int[][] x = {{1,2,3,4}, {5,6,7,8}};
int premiersNombres[][] = { {0,2,4,6,8},{1,3,5,7,9} };
```

NB : un tableau débute toujours à l'indice 0 ! Mais lorsque vous la declarer, vous donnez toujours le nombre d'elements du tableau.

Exemple : en declarant un tableau de 10 element on mettra Tabl[10] ;

17.Remplissage d'un tableau :

Pour remplir un tableau on utilise une Boucle et For de preference. Ecrire un programme qui remplit un tableau de 10 entiers à une dimension

```
package TESTE;
public class RemplirTableau {
    public static void main(String[] args) {
        // initialiser le tableau
        int Tab[];
```



```
Tab=new int[9];
java.util.Scanner Clavier = new java.util.Scanner(System.in);
int i;
for(i=0; i<Tab.length; i++)
{
    System.out.println("Donner une valeur");
    int Valeur= Integer.parseInt(Clavier.next());
    Tab[i]=Valeur;
}
}
```

18. Affichage d'un tableau : Avec for

Ecrire un programme qui remplit un tableau de 20 prenom à une dimension et qui affiche le contenu du tableau

```
package TESTE;
public class RemplirAfficherTableau {
    public static void main(String[] args) {
        // initialiser le tableau
        String Tab[]=new String[19];
        java.util.Scanner Clavier = new java.util.Scanner(System.in);
        int i;
        // remplissage*****
        for(i=0; i<Tab.length; i++) {
            System.out.println("Donner une valeur");
            String Valeur= Clavier.next();
            Tab[i]=Valeur;
        }
        // Affichage*****
        for(i=0; i<Tab.length; i++) {
            System.out.println(Tab[i]);
        }
    }
}
```

Resultat

Maxime



Wade
fatou
fatou

19.Cumul d'un tableau : Avec for

Ecrire un programme qui remplit un tableau d'entiers à une dimension et calcule la somme des elements du tableau.

```
package TESTE;
public class Sommetableau {
    public static void main(String[] args) {
        // initialiser le tableau
        int Tab[]=new int[10];
        // *****
        java.util.Scanner Clavier = new java.util.Scanner(System.in);
        int i;
        int somme = 0;
        // remplissage*****
        for(i=0; i<Tab.length; i++) {
            System.out.println("Donner une valeur");
            int Valeur= Integer.parseInt(Clavier.next());
            Tab[i]=Valeur;
        }
        // somme*****
        for(i=0; i<Tab.length; i++) {
            somme=somme+Tab[i];
        }
        System.out.println("la somme est : " +somme );
    }
}
```

Resultat :

la somme est :330

20.Minimum et Maximum d'un tableau : Avec for

Ecrire un programme qui remplit un tableau d'entiers à une dimension et qui cherche la plus petite valeur et la plus grande valeur.



```
package TESTE;
public class MINMAXTableau {
    public static void main(String[] args) {
        // initialiser le tableau
        int Tab[]=new int[10];
        // *****

        java.util.Scanner Clavier = new java.util.Scanner(System.in);
        int i;
        // remplissage*****
        for(i=0; i<Tab.length; i++) {
            System.out.println("Donner une valeur");
            int Valeur= Integer.parseInt(Clavier.next());
            Tab[i]=Valeur;
        }

        // MINIMUM*****
        int Min = Tab[0];
        for(i=0; i<Tab.length; i++) {
            if(Tab[i]<Min)
                Min=Tab[i];
            else
                Min=Min;
        }
        System.out.println("le minimum est : " +Min );
        // MAXIMUM*****
        int Max = Tab[0];
        for(i=0; i<Tab.length; i++)
        {
            if(Tab[i]>Max)
                Max=Tab[i];
            else
                Max=Max;
        }
        System.out.println("le Maximum est : " +Max );
    }
}
```

Resultat :

```
le minimum est :5
le Maximum est :999999
```



21. Les chaînes de caractères

En Java, les chaînes de caractères sont des objets. Ce sont des instances de la classe ***String***. Java utilise une approche particulière, en effet, les contenus des chaînes de caractères ne peuvent plus être modifiés une fois initialisés.

Exemple

```
public class chaines {  
    public static void main(String[] arg) {  
        String chaineA = new String ("Bonjour");  
        System.out.println(chaineA);  
        String chaineB = chaineA;  
        System.out.println(chaineB);  
        ChaineA = "Au revoir";  
        System.out.println(chaineA);  
        System.out.println(chaineB);  
    }  
}
```

Résultat

```
Bonjour  
Bonjour  
Au revoir  
Bonjour
```

Bien que les chaînes soient des objets, ils se comportent ici comme des primitives. Cela est dû au fait que les chaînes ne peuvent être modifiées.

Remarque : Java dispose d'une autre classe, appelée `StringBuffer`, qui permet de gérer des chaînes dynamiques.

21.1. La classe "StringBuffer"

La classe `StringBuffer` permet de modifier des chaînes de caractères notamment grâce à la méthode `append()` pour la concaténation.

Exemple



```
StringBuffer chaine;  
chaine = new StringBuffer ("Bon");  
chaine.append("jour");  
System.out.println(chaine);
```

Affichage : Bonjour

21.2. Quelques méthodes relatives à la classe "String"

Concaténation : APPEND

Les chaînes de caractères peuvent être concaténées

Exemple

```
String chaine1 = "Bonjour";  
String chaine2 = "tout le monde";  
chaine1 = chaine1 + chaine2; // création d'une chaîne constante "bonjour  
tout le monde et affectation à la variable chaine1
```

La méthode length() : retourne le nombre de caractères de la chaîne.

La méthode equals() : permet de tester l'égalité de deux chaînes de caractères

Remarque : L'emploi de l'opérateur == pour tester l'égalité de 2 chaînes n'est pas correct (== teste l'égalité des références des variables et non celles de leurs contenus).

La méthode substring() : La méthode substring () extrait une sous-chaîne d'une chaîne donnée. La sous-chaîne commence à l'indice donné en premier argument et se termine à l'indice donné en second argument.

Exemple :

```
String chaine;  
String chaine1;  
chaine1.substring(2,6);
```

22. Rappels

- *Un constructeur d'une classe est une méthode particulière. Elle n'a pas de type de retour, et porte le même nom que la classe dans laquelle elle se trouve. Un constructeur peut jeter des exceptions.*
- *Une classe peut avoir autant de constructeurs que l'on a le courage de lui en créer, dès l'instant qu'ils ont des signatures différentes, c'est-à-dire des paramètres différents. Il n'est toutefois pas conseillé de multiplier les constructeurs, ni de créer des constructeurs prenant des listes interminables de paramètres.*
- *Une classe qui ne déclare aucun constructeur explicitement en possède en fait toujours un : le constructeur vide par défaut, qui ne prend aucun paramètre. Si l'on définit un constructeur explicite, alors ce constructeur vide par défaut n'existe plus ; on doit le déclarer explicitement si l'on veut encore l'utiliser. Écrire ce constructeur systématiquement est une bonne habitude de programmation : le bon fonctionnement de nombreux framework repose sur l'existence systématique de ce constructeur. Prenons l'exemple de la classe `Marin` ci-dessous.*

23. Travaux dirigés : N°1

Travaux Pratiques de Programmation Orientée Objet en Java. Environnement de travail : l'IDE Eclipse sera notre principal environnement de travail. Tous le travail sera fait dans int main.

Exercice 1 : Créer un programme qui permet de calculer et d'afficher la somme de deux entiers donnés par l'utilisateur.

Attention : les entiers que vous saisissez seront considérés comme des chaînes de caractères. Donc dans le programme vous penserez à les convertir en nombres entiers en utilisant la méthode statique `parseInt` de la classe `Integer` du package `java.lang`.

Exemple : si on a : `String p = "24"` ; on fait : `int k = Integer.parseInt (p) ; //et on a k=24`

Exercice 2 : Refaire le meme exercice mais cette fois ci c'est à l'utilisateur de définir le nombre de valeur dont il souhaite faire la somme.



Exercice 3 : Faire un programme pour recevoir et afficher les informations suivantes pour un étudiant : Prenom, Nom, Adresse, Age et Tel.

Exercice 4 : Ecrire un programme pour calculer le factoriel d'un nombre donnée par l'utilisateur.

Exercice 5 : Ecrire un programme qui remplit et affiche les 6 notes de devoir de toto en Java et qui à la fin affiche le moyenne de Toto en Java apres avoir reçu sa note de l'examen.

Exercice 6: Ecrire un programme pour concevoir une liste de la classe LPIG2 Etat. Une fois la liste remplie, Afficher le contenu. L'utilisateur donnera des le depart le nombre d'etudiants de la classe.

Exercice 7 : Faire un programme permettant de calculer la somme de la série harmonique suivante : $1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$. La valeur n sera un entier strictement positif demandé à l'utilisateur. L'utilisateur n'a que 5 possibilités pour choisir une valeur comprise nécessairement entre 1 et 100 sinon on met fin au programme.

Exercice 8 : Réaliser un programme qui permet de calculer les racines d'une équation du second degré en x. ($ax^2 + bx + c$) a, b et c seront donnés au clavier. Voir les cas où le discriminant est nul, négatif strictement ou positif strictement.



CHAPITRE 4 : L'HERITAGE

1. Propriétés de l'héritage en Java

Supposons qu'on dispose de 2 classes A et B, si B hérite de A, on appelle B la **sous classe** et A la **super classe**.

- Une sous-classe possède tous les champs et toutes les méthodes de sa super classe à l'exclusion de ceux qui sont **privés**. Une classe fille hérite donc de toutes les propriétés et méthodes de sa classe mère (Super classe) sauf :

- Le constructeur
- Les méthodes et les propriétés déclarées private.

- Chaque classe ne peut avoir qu'une super classe (pas d'héritage multiple) de même chaque personne ne peut avoir deux papa biologique.
- Pour définir une sous-classe, on utilise le mot clé **extends**
- Il est impossible d'appeler le constructeur de la classe mère en dehors du constructeur de la classe fille. L'appel du constructeur d'une classe mère doit toujours se situer dans un constructeur de la sous classe et toujours en tant que première instruction. On utilisera **super** pour spécifier le nom du constructeur de la super classe. Si aucun constructeur vide n'est accessible dans la classe supérieure, une erreur se produit lors de la compilation.
- Le constructeur de la sous classe doit forcément faire appel (implicitement ou explicitement) à celui de sa super classe. Lors qu'un objet instance d'une classe fille qui en étend une autre est construit, au moins un constructeur de cette super classe doit être appelé. Si aucun appel explicite n'est écrit, alors la JVM exécute le constructeur vide par défaut. S'il n'existe pas, elle génère une erreur à la compilation. Il est également possible pour un constructeur d'appeler explicitement un unique constructeur. Cet appel ne peut être que la première instruction de ce constructeur. Voyons cela sur un exemple.
- La classe fille n'a pas besoin d'instancier un objet de sa classe mère pour pouvoir utiliser ses membres.
- Les constructeurs ne sont pas hérités par une sous-classe. Il faut donc écrire un constructeur spécifique pour la sous classe.
- Une méthode de la super classe peut être redéfinie. Ainsi, **la méthode Afficher ()** peut être redéfinie dans la classe Etudiants, grâce au polymorphisme. On dit que la méthode est polymorphe. Elle a



- plusieurs formes, et à chaque appel, la forme à exécuter est choisie en fonction de l'objet associé à l'appel. Mais la méthode `Afficher ()` de la classe `Personnes` est toujours accessible dans la classe `Etudiants` mais en la préfixant du mot clé **super** pour montrer que c'est la méthode `Afficher ()` de la super classe `Animal`.
- Si aucun constructeur n'est défini dans la classe mère alors la classe fille dans son constructeur pourra faire appel au constructeur implicite avec `Super() ;`

Exemple : Soit la classe `Etudiants` qui est une classe fille de la classe `Personnes`

```
package PPersonnes;
public class Personnes {
    String prenom,nom;
    public Personnes(String prenom, String nom) {
        this.prenom=prenom;
        this.nom=nom;
    }
    public void afficher()
    {
        System.out.println("Bonjour Mr SARR");
    }
}
```

La sous classe `Etudiants`

```
package PClients;
public class Etudiants extends Personnes{
    public Etudiants() {
        super("edouard","sarr");
        afficher();
        super.afficher();
    }
    public void afficher()
    {
        System.out.println("Bonjour Monsieur SARR");
    }
}
```

2. Destruction d'un objet

Rappelons que l'utilisateur n'a pas à se préoccuper de la destruction de ses objets : la notion de destructeur n'existe pas en Java. Il existe une méthode `finalize()` dans la classe `Object`, qui joue le rôle de callback avant que le garbage collector ne détruise un objet.

3. Méthodes et classes abstraites

3.1. Méthodes abstraites

Une méthode est abstraite si seul son en-tête (sa signature) est donné, le corps est remplacé par un point virgule, la méthode est alors de type `abstract`. Une méthode `abstract` correspond à une propriété dont on veut forcer l'implémentation dans chaque sous-classe. Si une sous-classe ne redéfinit pas une méthode `abstract`, et si cette méthode n'est pas aussi `abstract` dans cette sous-classe on aura automatiquement une erreur.

Exemple :

```
public abstract void afficher();
```

3.2. Classes abstraites

*Une classe qui possède au moins une méthode abstraite est abstraite et doit être déclarée avec le modificateur : `abstract`. **Aucun objet ne peut être instancié via une classe abstraite.***

Exemple erroné : car la classe doit forcément être abstraite vue qu'elle dispose d'une méthode abstraite.

```
1 package PClients;
2
3 public class Personnes {
4     String prenom,nom;
5
6     public Personnes(String prenom, String nom) {
7         this.prenom=prenom;
8         this.nom=nom;
9     }
10
11     public abstract void afficher();
12
13
14 }
```




Exemple Correcte : Ajoutons le mot clé `abstract`

```
package PClients;  
public abstract class Personnes {  
  
    String prenom,nom;  
  
    public Personnes(String prenom, String nom) {  
        this.prenom=prenom;  
        this.nom=nom;  
    }  
  
    public abstract void afficher();  
}
```

4. Les interfaces

Java n'admet pas d'héritage multiple (héritage à partir de plusieurs classes). Pour répondre à cette situation, Java permet une forme d'héritage multiple avec des classes très particulières que l'on appelle *interface*.

*Les interfaces possèdent un niveau d'abstraction supérieur à celui des classes abstraites. Contrairement aux classes abstraites, les interfaces ne possèdent pas de champs, à l'exception des constantes (champs de classe constants) mais ont des méthodes qui sont toutes implicitement abstraites (elles n'ont pas de corps). De même que la classe abstraite une interface ne peut être instanciée. Une interface est déclarée grâce au mot clé **interface***

Exemple :

```
package PClients;  
public interface Clients  
{  
    public abstract void test1();  
    public abstract void test2();  
    public abstract void test3();  
    public abstract void test4();  
}
```



Une classe peut 'hériter' des caractéristiques d'une interface, on dira dans ce cas que *la classe implémente l'interface* car elle devra obligatoirement redéfinir toutes les méthodes de l'interface car elles sont toutes abstraites. Pour spécifier le fait qu'une classe implémente une interface, on utilise le mot clé **implements**.

Exemple :

```
package PClients;  
public class Clients2 implements Clients  
{  
    @Override  
    public void test1() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test2() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test3() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test4() {  
        // TODO Auto-generated method stub  
    }  
}
```

Remarque : Une interface peut étendre une autre interface

```
package PClients;  
public interface Clients3 extends Clients  
{  
    public abstract void test11();  
    public abstract void test12();  
    public abstract void test13();  
    public abstract void test14();  
}
```



Une classe peut implémenter plusieurs interfaces, mais ne peut étendre qu'une seule classe (abstraite ou non).

```
package PClients;  
public class ClientsFille implements Clients, Clients2 {  
    @Override  
    public void test11() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test12() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test13() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test14() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test1() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test2() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test3() {  
        // TODO Auto-generated method stub  
    }  
    @Override  
    public void test4() {  
        // TODO Auto-generated method stub  
    }  
}
```



5. Exercice d'application :

Créer une classe CompteBancaire qui crée et manipule des comptes en banque. Un compte sera identifié par le solde, le nom et le prénom du titulaire, le sexe et le découvert initial. Imaginer et créer les méthodes nécessaires de manipulation de comptes bancaires. Voici la classe COMPTEBANQUE

```
package TESTE;

public abstract class COMPTEBANQUE {
    static String Prenom;
    static String Nom;
    static String Sexe;
    static int Solde;
    static int DecIni;
    //*****Constructeur de la classe mère*****
    public void COMPTEBANQUE_(String Pre,String Nm, String Se,int
Slde,int DcIni) {
        System.out.println("***Constructeur Classe Mère***");
        Prenom=Pre;
        Nom=Nm;
        Sexe=Se;
        Solde=Slde;
        DecIni=DcIni;
        System.out.println("COMPTE CREER EN RESPECTANT LES REGLES");
    }
    //*****methode deposer est private donc ne pourra pa être héritée
    private static int deposer(int Montant)
    {
        System.out.println("***Deposer de la classe mère***");
        Solde=Solde + Montant;
        return Solde;
    }
    //methode retirer est protected donc peut être hérité elle est finale donc ne peut
être surchargée. Static car elle accede à des attributs statics*****
    protected final static int retrait(int Montant)
    {
        System.out.println("****Retirer de la classe mère ");
        Solde=Solde - Montant;
        return Solde;
    }
}
```



```
/**Methode abstraite AfficherSolde***/  
static void AfficherSolde ();  
}
```

La classe COMPTEBANQUESGBS qui est une classe fille de la classe COMPTEBANQUE

```
package TESTE;  
import java.util.*;  
public abstract class COMPTEBANQUESGBS extends  
COMPTEBANQUE {  
    static String prenom;  
    static String nom;  
    static String sexe;  
    static int Solde;  
    static int Montant;  
    static int decouvert;  
    static Scanner Clavier=new Scanner(System.in);  
  
    /**Creer compte SGBS dans le construteur de la classe fille l'appel du  
    constructeur de la classe mère se fait dans le constructeur de la classe fille  
    comme premiere instruction***/  
  
    public static void COMPTEBANQUESGBS ( String prenom, String  
nom, String sexe, int solde, int decouvert) {  
        COMPTEBANQUE CB= new COMPTEBANQUE (prenom, nom,  
sexe, solde, dec);  
    }  
  
    /**test de Abstrait  
  
    public static void AfficherSolde ()  
    {  
        System.out.println("ca marche car la classe est aussi abstraite et  
que la methode est redefinie");  
    }  
  
    // ***** la classe main *****
```

```
public static void main(String[] args) {
    System.out.println("Donner votre Nom");
    nom=Clavier.next();
    System.out.println("Donner votre Prenom");
    prenom=Clavier.next();
    System.out.println("Donner votre sexe");
    sexe=Clavier.next();
    System.out.println("Donner votre Solde");
    Solde=Clavier.nextInt();
    System.out.println("Donner votre Decouvert Initial");
    dec=Clavier.nextInt();
    // * appel du constructeur de la classe mille *****
    COMPTEBANQUESGBS (prenom, nom, sexe, solde, dec);

    System.out.println("Donner le montant ");
    int Mnt=Clavier.nextInt();
    // *** on peut appeler la methode retirer mais la redefinir
    car elle est Final

        retirer(Mnt) ;
    //**** la methode AfficherSolde
        AfficherSolde ();
    // ****l instruction qui suit va générer une erreur car la
    methode déposer de la classe mère est private donc non heritable*****
        déposer(Mnt) ;
    }
}
```

6. Travaux dirigés N°2 :

Travaux Pratiques de Programmation Orientée Objet en Java

Exercice 1 : Créer une classe permettant de gérer le personnel administratif d'une entreprise (employersFixe et journaliers). On supposera q'un employé Fixe ou journalier sera identifié par son nom, son prénom, son age, son sexe, son numéro de matricule, sa fonction au sein de l'entreprise. Créer une classe mère nommé Employes avec ses classes fille EmployersFixe et Journaliers.

Exercice 2 : Créer un programme pour gérer les informations pour les étudiants, les professeurs. Nous considérons que les professeurs et les étudiants partages certaines informations telles que : Prénom, Nom, Age, Adresse et Tel



Mais le professeur a un salaire tant dis que l'étudiant est inscrit dans une classe déterminer. Mais Tous sont des personnes. On ne peut modifier le prenom, le nom d'une personne. On peut par contre les lister. On ne peut pas visualiser le salaire d'un professeur. La methode Afficher est à redefinir.

- Implémenter la classe Personne
- Implémenter la classe Professeur
- Implémenter la classe Etudiant
- Implémenter la classe Gestion_Personnel qui manipule l'ensemble du personnel.

CHAPITRE 5 : GESTION DES EXCEPTIONS

1. Mécanisme des exceptions

Une exception est une erreur se produisant dans un programme qui conduit le plus souvent à l'arrêt de celui-ci. On dit qu'une exception est levée. Il vous est sûrement déjà arrivé d'obtenir un gros message affiché en rouge dans la console d'Eclipse : eh bien, cela a été généré par une exception. . . qui n'a pas été capturée.

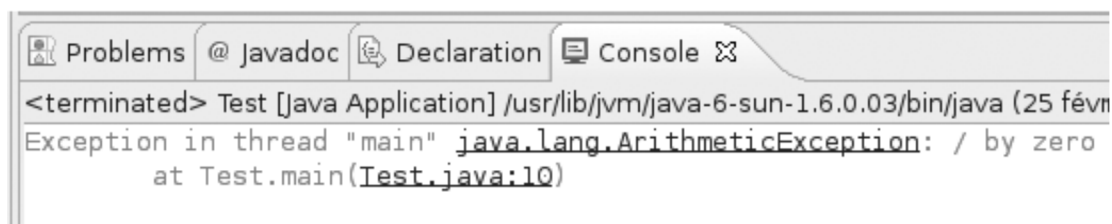
Le fait de gérer les exceptions s'appelle aussi la capture d'exception ! Le principe consiste à repérer un morceau de code (par exemple, une division par zéro) qui pourrait générer une exception, de capturer l'exception correspondante et de traiter celle-ci, c'est-à-dire d'afficher un message personnalisé et de continuer l'exécution. Lorsqu'une exception est levée, le traitement du code de la méthode est interrompu et l'exception est propagée à travers la pile d'exécution de méthode appelée en méthode appelante. Si aucune méthode ne capture l'exception, celle-ci remonte jusqu'à la méthode du fond de la pile d'exécution : l'exécution se termine avec une indication d'erreur.

2. Le bloc try{...} catch{...}

Java contient une classe nommée Exception dans laquelle sont répertoriés différents cas d'erreur. La division par zéro dont nous parlions plus haut en fait partie. Si vous créez un nouveau projet avec seulement la classe main et y mettez le code suivant:

```
int j = 20, i = 0;  
System.out.println(j/i);  
System.out.println("je suis ici !");
```

Vous verrez apparaître un message d'erreur Java (en rouge) comme celui ci.



Mais surtout, vous devez avoir constaté que lorsque l'exception a été levée, le programme s'est arrêté ! D'après le message affiché dans la console, le nom de



l'exception qui a été déclenchée est **ArithmeticException**. Nous savons donc maintenant qu'une division par zéro est une **ArithmeticException**. Nous allons pouvoir la capturer, avec un bloc `try{...} catch{...}`, puis réaliser un traitement en conséquence et afficher un message personnalisé lors d'une division par 0.

Exemple sans erreur :

```
package UCAO1;
public class TestPourExeption {
    public static void main(String[] args)
    {
        int j = 20, i = 2;
        try
        {
            System.out.println(j/i);
        }
        catch (ArithmeticException e)
        {
            System.out.println("Erreur Division par zéro !");
        }
    }
}
```

Résultat : **10** C'est le bloc Try qui est executer.

Exemple avec erreur :

```
package UCAO1;
public class TestPourExeption {
    public static void main(String[] args)
    {
        int j = 20, i = 0;
        try {
            System.out.println(j/i);
        }
        catch (ArithmeticException e) {
            System.out.println("Erreur vous divisez un nombre par zéro !");
        }
    }
}
```



Erreur vous divisez un nombre par zéro !");

Explication : Dans le premier exemple c'est le bloc Try qui est exécuté car une erreur n'a pas été détectée. Dans le deuxième exemple, c'est le bloc Catch qui est exécuté car il y a erreur.

Fonctionnement : Nous isolons le code susceptible de lever une exception de type `ArithmeticException` : `System.out.println(j/i);` afin de lui spécifier un traitement de sortie dans le Catch. Notre bloc catch contient justement un objet de type `ArithmeticException` en paramètre. Nous l'avons appelé `e`. L'exception étant capturée, l'instruction du bloc catch s'exécute ! Notre message d'erreur personnalisé s'affiche alors à l'écran.

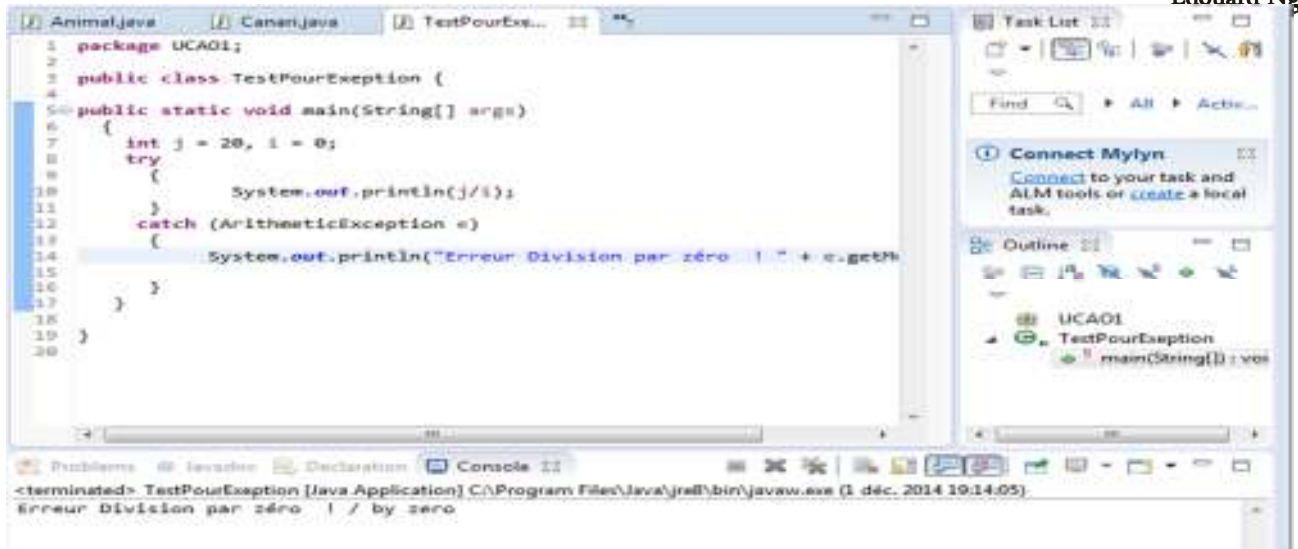
Le paramètre de la clause catch permet de connaître le type d'exception qui doit être capturé. Et l'objet ici, `e` peut servir à préciser notre message grâce à l'appel de la méthode `getMessage()`. Vous pouvez encore faire ce test en remplaçant l'instruction du catch par celle-ci :

```
System.out.println(" Division par zéro ! » + e.getMessage());
```

Pour voir le message que le système génère.

```
package UCAOI;
public class TestPourExeption {
    public static void main(String[] args)
    {
        int j = 20, i = 0;
        try
        {
            System.out.println(j/i);
        }
        catch (ArithmeticException e)
        {
            System.out.println("Erreur Division par zéro ! " + e.getMessage());
        }
    }
}
```

Sur eclipse :



```

1 package UCAO1;
2
3 public class TestPourException {
4
5     public static void main(String[] args)
6     {
7         int j = 20, i = 0;
8         try
9         {
10             System.out.println(j/i);
11         }
12         catch (ArithmeticException e)
13         {
14             System.out.println("Erreur Division par zero : " + e.getMessage());
15         }
16     }
17 }
18
19 }
20

```

Console Output: <terminated> TestPourException [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (1 d c. 2014 10:14:05)
Erreur Division par zero : / by zero

Vous verrez que la fonction getMessage() de notre objet ArithmeticException nous pr cise la nature de l'erreur. Le principe de capture d'exception permettait de ne pas interrompre l'ex cution du programme. En effet, lorsque nous capturons une exception, le code pr sent dans le bloc catch(){...} est ex cuted, mais le programme suit son cours.

Resum  : Donc lorsqu'une ligne de code l ve une exception, l'instruction dans le bloc try est interrompue et le programme se rend dans le bloc catch correspondant   l'exception lev e.

Exemple : gestion de la saisie d'un caractere   la place d'un nombre

```

import java.util.*;
public class GestionSaisie {
    public static void main(String[] args) {
        int x;
        Scanner Cla = new Scanner(System.in);
        System.out.println("Donner un entier");
        try{
            x=Cla.nextInt();
        }
        catch (InputMismatchException e){
            System.out.println("Vous devez entrer un entier mais pas une
chaine de caracteres ");
        }
        System.out.println("SUITE");
    }
}

```



Nous venons de gérer une exception relative à la saisie de chaîne de caractères à la place d'un entier. Le type de l'exception est `InputMismatchException`.

3. Les exceptions personnalisées

Il est possible de ne pas utiliser le `Try Catch` pour gérer les exceptions Java. La procédure pour faire des exceptions personnalisées est un peu particulière. En effet, nous devons apprendre deux mots clés :

- *throws* : ce mot clé permet de signaler à la JVM qu'un morceau de code, une méthode, une classe. . . est potentiellement dangereux et qu'il faut utiliser un bloc `try{...}catch{...}`. Il est suivi du nom de la classe qui va gérer l'exception.
- *throw* : celui-ci permet tout simplement de lever une exception manuellement en instanciant un objet de type `Exception` (ou un objet hérité).

Pour mettre en pratique ce système, commençons par créer une classe `NombreHabitantException` qui va gérer nos exceptions pour le cas où le nombre d'habitants est inférieur ou égal à 0. Celle-ci, je vous le rappelle, doit hériter d'`Exception` :

```
public class NombreHabitantException extends Exception{  
    public NombreHabitantException (Double NbHabitant)  
    {  
        System.out.println("Impossible de creer une ville avec "+NbHabitant+ "  
Habitants");  
        System.out.println("Nombre Habitants doit etre > 0");  
    }  
}
```

En gros, nous devons dire à notre constructeur de `Ville` : si l'utilisateur crée une instance de `Ville` avec un nombre d'habitants négatif ou égal à 0, créer un objet de type `NombreHabitantException`. Voilà à quoi ressemble le constructeur de notre objet `Ville` à présent :

```
public class Ville {  
    public Ville (String NomVille, Double NbHabitant) throws  
NombreHabitantException {
```

```
        If (NbHabitant<=0) throw
                        new NombreHabitantException(NbHabitant);
        else{
            System.out.println("Ville bien creer NOM VILLE :"+NomVille+
" Nombre habitant :"+NbHabitant);
        }
    }
}
```

throws NombreHabitantException nous indique que si une erreur est capturée, celle-ci sera traitée en tant qu'objet de la classe NombreHabitantException, ce qui nous renseigne sur le type de l'erreur en question. Elle indique aussi à la JVM que le constructeur de notre objet Ville est potentiellement dangereux et qu'il faudra gérer les exceptions possibles. Si la condition `if(nbre <=0)` est remplie, `throw new NombreHabitantException();` instancie la classe NombreHabitantException. Par conséquent, si un nombre d'habitants est négatif, l'exception est levée. Apportons une petite modification, retournons dans notre classe main, et ajoutons une classe main pour créer un objet Ville de votre choix.

```
public class Ville {
    public Ville (String NomVille, Double NbHabitant) throws
NombreHabitantException {
        if (NbHabitant<0) throw
            new NombreHabitantException(NbHabitant);
        else {
            System.out.println("Ville bien creer NOM VILLE :"+
+NomVille+ " Nombre habitant :"+NbHabitant);
        }
    }
    public static void main(String[] args) {
        Scanner cl =new Scanner(System.in);
        System.out.println("Donner le nom de la ville");
        String V=cl.next();
        System.out.println("Donner le nombre habitants");
        Double N=cl.nextDouble();
        Ville NouvelleVille = new Ville(V,N);
        System.out.println("Merci");
    }
}
```



Nous devons tomber sur une erreur persistante sur la ligne nous demandant d'utiliser un bloc try{} catch{}. C'est tout à fait normal et dû à l'instruction throws.

```
Ville NouvelleVille = new Ville(V,N);
```

Cela signifie qu'à partir de maintenant, vu les changements dans le constructeur, il vous faudra gérer les exceptions qui pourraient survenir dans cette instruction avec un bloc try{} catch{}. Ainsi, pour que l'erreur disparaisse, il nous faut entourer notre instanciation avec un bloc try{...}catch{...}

Reprenons alors :

```
package TESTE;
import java.util.*;
public class Ville {
    public Ville(String NomVille, Double NbHabitant) throws
NombreHabitantException {
        if(NbHabitant<0) throw
                                new NombreHabitantException(NbHabitant);
        else
        {
            System.out.println("Ville bien creer NOM VILLE :"+
+NomVille+ " Nombre habitant :"+NbHabitant);
        }
    }
    public static void main(String[] args) {
        Scanner cl =new Scanner(System.in);
        System.out.println("Donner le nom de la ville");
        String V=cl.next();
        System.out.println("Donner le nombre habitants");
        Double N=cl.nextDouble();
        try {
            Ville o= new Ville(V,N);
        } catch (NombreHabitantException e) {
            System.out.println(e.getMessage());
        }
        System.out.println("Merci");
    }
}
```

Exécution Normale :



```

1 package TESTE;
2 import java.util.*;
3 public class Ville {
4
5
6     public Ville(String NomVille, Double NbHabitant) throws NombreHabitantException {
7
8         if(NbHabitant<0) throw new NombreHabitantException(NbHabitant);
9     }
10    {
11        System.out.println("Ville bien creer NOM VILLE : " +NomVille+ " Nombre habitant :"+NbHabitant);
12    }
13 }
14
15
16
17 public static void main(String[] args) {
18     Scanner cl =new Scanner(System.in);
19     System.out.println("Donner le nom de la ville");
20     String V=cl.next();
21     System.out.println("Donner le nombre habitants");
22     Double N=cl.nextDouble();
23
24     try {
25         Ville v= new Ville(V,N);
26     } catch (NombreHabitantException e) {
27
28     }
29 }

```

Problems Javadoc Declaration Console

<terminated> Ville [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (26 janv. 2015 06:36:48)

Donner le nom de la ville
Dakar
Donner le nombre habitants
1000
Ville bien creer NOM VILLE :Dakar Nombre habitant :1000.0
Merci

Exécution erronée avec Nbhabitant = -23000



Problems Javadoc Declaration Console

<terminated> Ville [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (26 janv. 2015 06:37:31)

Donner le nom de la ville
Dakar
Donner le nombre habitants
-23000
Impossible de creer une ville avec -23000.0 Habitants
Nombre Habitants doit etre >0
null
Merci

4. TP 3 :

Nous voulons gerer les exceptions sur une classe etudiant. Un Etudiant est créé avec un nom, un Prenom, une Adresse et un Age. L'exception est levée lorsque l'age de l'etudiant est inferieur à 18 ou si son adresse n'est pas Dakar.

- Créer la classe qui gere l'exception sur la creation d'un etudiant
- Créer la classe etudiants



CHAPITRE 6 : LES ENTREES / SORTIES

Une entrée/sortie en Java consiste en un échange de données entre le programme et une autre source, par exemple la mémoire, un fichier, le programme lui-même... Pour réaliser cela, Java emploie ce qu'on appelle un stream (qui signifie « flux »). Celui-ci joue le rôle de médiateur entre la source des données et sa destination.

Toute opération sur les entrées/sorties doit suivre le schéma suivant :

- *Ouverture du flux de Stream*
- *Lecture ou l'écriture des données (I/O)*
- *fermeture du flux de Stream*

Sachez que Java a décomposé les objets traitant des flux en deux catégories :

- *les objets travaillant avec des flux d'entrée (**in**), pour la lecture de flux*
- *les objets travaillant avec des flux de sortie (**out**), pour l'écriture de flux.*

Le package **java.io** fournit un mécanisme d'entrée/sortie au moyen de flux de données, de caractères, d'objets. Le fichier est un émetteur ou récepteur privilégié de ces flux.

1. Généralités sur les flots de données

Les flots de données permettent la gestion des opérations de lecture et d'écriture sur ou vers des sources différentes. Il existe quatre classes principales permettant de traiter les flots. Pour la source ou la destination, on utilise les préfixes suivants:

<i>Source / Destination</i>	<i>Préfixe</i>
<i>Chaîne de caractères</i>	<i>String</i>
<i>Fichier</i>	<i>File</i>
<i>Flux d'octets</i>	<i>InputStream ou OutputStream</i>



1.1.Flux d'octets

Le package **java.io** possède deux classes principales :

- *InputStream* : cette classe abstraite définit les fonctions de lecture (entrée ou input en anglais),
- *OutputStream* : cette classe abstraite définit les fonctions d'écriture (sortie ou output en anglais).

Ces deux classes abstraites définissent des fonctions bas-niveau et sont implémentées dans différentes sous-classes concrètes. Trois types de flux sont possibles, les flux sur fichiers, les flux sur tubes, et les flux sur zones de mémoire. Par exemple on a :

FileInputStream : lecture d'un fichier,
FileOutputStream : écriture d'un fichier.

1.1.1. La classe FileInputStream

Propose les méthodes:

- *int read ()* : lire un octet (retourné en int), -1 si le flux est terminé.
- *int read (byte buf[])* : lire une séquence d'octets et la mettre dans buf, retourner un nombre d'octets lus (entre 0 et buf.length), -1 si le flux est terminé.
- *Close ()* : fermer le flux, libérer des ressources associées.

1.1.2. La classe FileOutputStream

Propose les méthodes:

- *void write (int b)* : écrire un octet (représenté avec int) dans le flux
- *void write (byte buf[])* : écrire une séquence d'octets dans le flux.
- *Flush ()* : Si la mémoire tampon est utilisée, envoyer toutes les données de la mémoire tampon vers la destination.
- *Close ()* : terminer l'écriture sur la source de données.



1.2.Flots de caractères

1.2.1. La classe Reader

Propose les méthodes:

int read () : lire un caractère (retourné en int), -1 si le flux est terminé.
int read (char buf[]) : lire une séquence de caractères et la mettre dans buf, Retourner un nombre de caractères lus (entre 0 et buf.length), -1 si le flux est terminé.
Close () : fermer le flux, libérer des ressources associées.

1.2.2. La classe Writer

Propose les méthodes:

void write (int c) : écrire un caractère (représenté avec int) dans le flux
void write (char buf[]) : écrire une séquence de caractères dans le flux.
Flush () : comme OutputStream
Close () : terminer l'écriture sur la source de données.

2. Ecrire sur un fichier avec la classe FILE

Les classes abstraites InputStream/ OutputStream et Reader/ Writer sont à spécialiser pour représenter des flux concrets.

- *FileInput(Output) Stream : lire/ écrire des fichiers*
- *ByteArrayInput(Output) Stream : lire/ écrire des octets dans un tableau via les méthodes de InputStream/ OutputStream*
- *ObjectInput(Output) Stream : lire et écrire des objets dans un flux quelconque (fichiers, réseaux).*

Avant de commencer, créez un fichier avec l'extension que vous voulez pour le moment, et enregistrez-le à la racine de votre projet Eclipse. Personnellement, je me suis fait un fichier test.txt dont voici le contenu :

Je suis étudiant à UCAO ST MICHEL
Je fais Informatique de gestion
Je me nomme Toto



```
package UCAOI;
import java.io.File;
//Package à importer afin d'utiliser l'objet File
public class SaisieClavierBis {
    public static void main(String[] args) {
        //Création de l'objet File
        File f = new File("D:/UCAO/JAVA/doc/test.txt");
        System.out.println("Chemin absolu du fichier : " +
f.getAbsolutePath());
        System.out.println("Nom du fichier : " + f.getName());
        System.out.println("Est-ce qu'il existe ? " + f.exists());
        System.out.println("Est-ce un répertoire ? " + f.isDirectory());
        System.out.println("Est-ce un fichier ? " + f.isFile());
        System.out.println("Affichage des lecteurs à la racine du PC : ");
        for(File file : f.listRoots())
        {
            System.out.println(file.getAbsolutePath());
            try {
                int i = 1;
                //On parcourt la liste des fichiers et répertoires
                for(File nom : file.listFiles()){
                    //S'il s'agit d'un dossier, on ajoute un "/"
                    System.out.print("\t\t" + ((nom.isDirectory()) ?
nom.getName()+"/" : nom.getName()));
                    if((i%4) == 0){
                        System.out.print("\n");
                    }
                    i++;
                }
                System.out.println("\n");
            }
            catch (NullPointerException e)
            {
                //L'instruction peut générer une NullPointerException
                //s'il n'y a pas de sous-fichier !
            }
        }
    }
}
```



Vous conviendrez que les méthodes de cet objet peuvent s'avérer très utiles !
Nous venons d'en essayer quelques-unes et nous avons même listé les sous-fichiers et sous dossiers de nos lecteurs à la racine du PC.

```
- System.out.println("Chemin absolu du fichier : " +  
    f.getAbsolutePath());  
- System.out.println("Nom du fichier : " + f.getName());  
- System.out.println("Est-ce qu'il existe ? " + f.exists());  
- System.out.println("Est-ce un répertoire ? " + f.isDirectory());  
- System.out.println("Est-ce un fichier ? " + f.isFile());
```

Vous pouvez aussi :

*effacer le fichier grâce la méthode delete()
créer des répertoires avec la méthode mkdir() 1. . .*

Maintenant nous pouvons commencer à travailler avec notre fichier !

3. Creation d'un fichier texte avec la methode `createNewFile()`; de la classe **FILE**

```
package ENTRESORTIE;  
import java.io.*;  
public class CreationFichier {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        //Création de l'objet File  
        File MonFic = new File("Test.txt");  
        boolean Rep = MonFic.exists();  
        if (Rep == true)  
        {  
            System.out.println("Fichier existe deja");  
        }  
        Else {  
            try {  
                MonFic.createNewFile();  
            } catch (IOException e) {  
                // TODO Auto-generated catch block  
                e.printStackTrace();  
                System.out.println(" Erreur de creation du
```



```
fichier");  
  
        }  
    }  
}
```

En executant pour la premiere fois nous aurons :

```
False  
True
```

Pour montrer que le fichier n'existait pas avant et qu'il est maintenant créé. En executant pour la deuxieme fois nous aurons :

```
True  
True
```

Le fichier existe alors il sera ecrasé.

4. Ecriture et lecture dans un fichier texte avec la Classe **FILEWRITER** et **FILEREADER**

Supposons que l'on veuille écrire un programme permettant d'écrire une chaîne de caractères dans un fichier nommé fic1.txt. Nous allons utiliser pour cela les deux classes FileReader et FileWriter.

4.1.L'écriture : **FILEWRITER**

```
package UCAOI;  
import java.io.*;  
class LireEcrireTexte  
{  
    public static void main(String arg[]) throws IOException  
    {  
        FileWriter fichier = new FileWriter("D:/UCAO/JAVA/doc/fic1.txt");  
        fichier.write ("je suis etudiant et je fais du JAVA avec M SARR UCAO ST MICHEL");  
        fichier.close();  
    }  
}
```



4.2.La lecture : **FILEREADER**

Ecrire un programme permettant d'afficher le contenu du fichier nommé fic.txt.

Nous allons utiliser pour cela les deux classes FileReader et BufferedReader.

```
package TESTE;
import java.io.*; // pour utiliser FileReader et BufferedReader
public class AfficherFichier {
    public static void main(String[] args) throws IOException{
        FileReader fichier = new FileReader ("D:/UCAO/JAVA/fic.txt");
        BufferedReader buffer = new BufferedReader (fichier);
        String ligne_lue = buffer.readLine() ;
        while (ligne_lue != null )
        {
            System.out.println (ligne_lue) ;
            ligne_lue = buffer.readLine () ;
        }
        buffer.close() ; // on ferme le filtre
        fichier.close() ; // on ferme le flux
    }
}
```

Resultat :

```
je suis edouard ngor sarr
je suis prof a UCAO St Michel
Merci
```

5. Exemple complé : Ecriture/lecture

Ecrire un programme permettant d'écrire une chaîne de caractères donnée par l'utilisateur dans un fichier nommé fic2.txt. Nous allons utiliser pour cela les deux classes FileReader et FileWriter

```
package ENTRESORTIE;
import java.io.*;
import java.util.Scanner;
public class LireEcrire
{

```



```
public static void main(String[] args) throws IOException
{
    // TODO Auto-generated method stub
    Scanner Clavier= new Scanner(System.in);
    System.out.println("Donner votre prenom ");
    String Prenom= Clavier.nextLine();
    System.out.println("Donner votre nom");
    String Nom= Clavier.nextLine();
    System.out.println("Donner votre adresse");
    String ADresse= Clavier.nextLine();
    String Personne= Prenom+" "+Nom+" "+ADresse;
    //*****écriture*****
    FileWriter Fic = new FileWriter("LireEcrire.txt");
    Fic.write(Personne);
    Fic.close();
    //*****lecture:*****
    FileReader Fichier = new FileReader("LireEcrire.txt");
    BufferedReader Tampon= new BufferedReader(Fichier);
    String Ligne = Tampon.readLine();
    while(Ligne!=null)
    {
        System.out.println(Ligne);
        Ligne=Tampon.readLine();
    }
    Tampon.close();
    Fichier.close();
}
}
```

6. Copier d'un fichier à un autre

Nous disposons d'un simple fichier source.txt sous notpad et mettons ce contenu :

```
bonjour UCAO st Michel
C est M edouard SARR
```

Nous voulons copier le contenu de ce fichier source.txt dans le fichier destination.txt.



Solution1

```
package ENTRESORTIE;
import java.io.*;
public class Copy {
    public static void main(String[] args) throws IOException {
        FileReader Lire = new FileReader("source.txt");
        FileWriter Ecrire = new FileWriter("destinataire.txt");
        BufferedReader Tampon = new BufferedReader(Lire);
        String Ligne = Tampon.readLine();
        while(Ligne!=null) {
            Ecrire.write(Ligne); //****ecrire dans destinataire****
            Ligne=Tampon.readLine();
        }
        Tampon.close();
        Lire.close();
        Ecrire.close();
    }
}
```

Soluton 2

```
package UCAOI;
import java.io.*;
public class LireetEcriredansFichier {
    public static void main(String[] args) throws IOException {
        FileReader in = new FileReader("D:/UCAO/JAVA/doc/source.txt" );
        //objet lecture
        FileWriter out = new FileWriter("D:/UCAO/JAVA/doc/destination.txt");
        //objet ecriture
        int c;
        //copier de source.txt à destination.txt
        while ((c = in.read()) != -1) {
            out.write(c);
        }
        in.close(); // déverrouiller (lecture seule) le fichier
        out.close(); // écrire les données de la mémoire tampon vers le fichier et
        le fermer
    }
}
```




7. Ecrire à la fin d'un fichier

Le constructeur de `FileWriter` peut prendre un paramètre booléen qui passait à `true` permet d'écrire à la suite du fichier.

`FileWriter(String fileName, boolean append)`

Le booléen sert à dire écrire à la fin du fichier...
donc nous aurons :

```
FileWriter MyFile= new FileWriter("D:/Sarr/Etudiants.txt",true)
```

Alors dans ce cas il crée le fichier si celui ci n'existe sinon il écrit à la suite si il est déjà créé. Exemple :

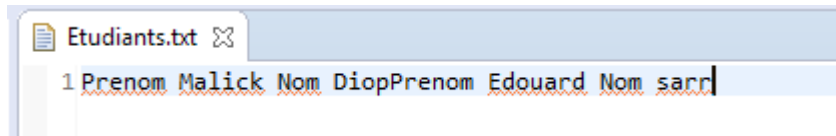
```
package IO;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
public class ClassFile {
    public static void main(String[] args) throws IOException {
        Scanner Cl=new Scanner(System.in);
        System.out.println("Donner votre prenom et votre nom");
        String p =Cl.next();
        String n =Cl.next();
        String info="Prenom "+p+ " Nom "+n;
        FileWriter FET=new FileWriter("Fichiers/Etudiants.txt",true);
        FET.write(info);
        FET.close();
    }
}
```

Première exécution :

```
Donner votre prenom et votre nom
Edouard
sarr
```



Le fichier :



```
Etudiants.txt
1 Prenom Malick Nom Diop Prenom Edouard Nom sann
```

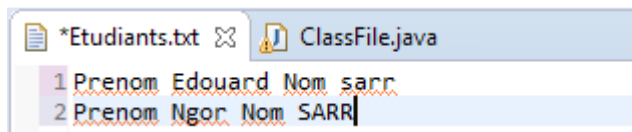
Deuxieme execution :

Donner votre prenom et votre nom

Ngor

SARR

Le fichier :



```
*Etudiants.txt
1 Prenom Edouard Nom sann
2 Prenom Ngor Nom SARR
```

8. Manipulation des objets avec `ObjectInputStream` et `ObjectOutputStream`

Les classes `ObjectInput`(`Output`)`Stream` permettent de lire et écrire des objets du programme

Cas d'utilisation :

- Sauvegarder l'état du programme dans un fichier pour pouvoir le recharger plus tard.
- Envoyer des objets d'un programme à un autre programme.

Elle propose les méthodes :

- *`Void writeObject(Object o)` : écrire un objet vers le flux. L'écriture d'objet consiste à écrire l'état de l'objet (tous les attributs)*
- *`Object readObject()` : reconstituer l'objet qui a été écrit par `writeObject(...)`.*

Les classes des objets <<sérialisable>> doivent implémenter l'interface `java.io.Serializable`. Vous devez savoir que lorsqu'on veut écrire des objets dans



des fichiers, on appelle ça la sérialisation : c'est le nom que porte l'action de sauvegarder des objets.

Exemple 1: Nous souhaitons enregistrer 10 Etudiants dans un fichier nommé etudiant.txt puis de l'afficher par la suite. Créer la classe Etudiants.java :

```
package UCAOI;
import java.io.*;
public class Etudiants implements Serializable{
    String prenom,nom,adresse,tel,classe;
    //*****construction *****
    public Etudiants(String prenom,String nom,String adresse,String
tel,String classe) {
        this.prenom=prenom;
        this.nom=nom;
        this.adresse=adresse;
        this.tel=tel;
        this.classe=classe;
    }
    //***** afficher un etudiant*****
    public String RetournerEtudiant()
    {
        String EtudiantTrouve= " \n Le prenom : "+this.prenom+" \nLe
nom : "+this.nom+" \n l'adresse : "+this.adresse+" \nle telephone : "+this.tel+"
\n la classe: "+this.classe+"\n\n";
        return EtudiantTrouve;
    }
}
```

La classe EtudiantSerialiserAppel :

```
package UCAOI;
import java.io.*;
import java.util.Scanner;
public class EtudiantsUcao {
    public static void main(String[] args) throws IOException,
ClassNotFoundException {
        File MonFichier = new File("etudiant.txt");
        FileOutputStream EcrireFichier= new
FileOutputStream(MonFichier);
```



```
BufferedOutputStream Tampon= new
BufferedOutputStream(EcrireFichier);
ObjectOutputStream Ecrire= new ObjectOutputStream(Tampon);
int i;
Scanner Clavier=new Scanner(System.in);
for (i=0; i<4;i++)
{
    System.out.println("***** creation Nouveau
Etudiant*****");
    System.out.println("Donner le prenom");
    String P = Clavier.next();
    System.out.println("Donner le nom");
    String N = Clavier.next();
    System.out.println("Donner adresse");
    String A = Clavier.next();
    System.out.println("Donner le tel");
    String T = Clavier.next();
    System.out.println("Donner la classe");
    String C = Clavier.next();
    Etudiants Lenouveau= new Etudiants(P,N,A,T,C);
    Ecrire.writeObject(Lenouveau);
    System.out.println("*****Nouveau Etudiant
enregistrer*****");
}
Ecrire.close();
FileInputStream lireFichier= new FileInputStream(MonFichier);
BufferedInputStream Tampon2= new
BufferedInputStream(lireFichier);
ObjectInputStream Lire= new ObjectInputStream(Tampon2);
for (i=0; i<4;i++)
{
    System.out.println(
((Etudiants)Lire.readObject()).RetournerEtudiant() );
}
Lire.close();
}
}
```



Resultat :

*****Nouveau etudiant*****

1eme etudiant*****

Donnez Prenom

Maxim

Donnez nom

Diatta

Donnez Adresse

Dakr

Donnez telephone

778654433

Donnez classe

LPIG3

Donnez age

22

*****Merci etudiant enregistrer*****

*****Nouveau etudiant*****

2eme etudiant*****

Donnez Prenom

Fatou

Donnez nom

dafe

Donnez Adresse

Pikine

Donnez telephone

778654455

Donnez classe

LPIG3

Donnez age

33

*****Merci etudiant enregistrer*****

*****Nouveau etudiant*****

3eme etudiant*****

Donnez Prenom

bruno

Donnez nom

diatta

Donnez Adresse

LPIG3



Donnez telephone

779875564

Donnez classe

LPIG3

Donnez age

23

******Merci etudiant enregistrer******

******Affichage des etudiants :******

*1eme etudiant******

PRENOM: Maxim

NOM : Diatta

ADRESSE: Dakr

TELEPHONE : 778654433

CLASSE : LPIG3

AGE : 22

2eme

*etudiant******

PRENOM: Fatou

NOM : dafe

ADRESSE: Pikine

TELEPHONE : 778654455

CLASSE : LPIG3

AGE : 33

3eme

*etudiant******

PRENOM: bruno

NOM : diatta

ADRESSE: LPIG3

TELEPHONE : 779875564

CLASSE : LPIG3

AGE : 23

Exemple 2 : gestion des Clients

Classes Clients

```
package Personne;  
import java.io.*;  
public class Client implements Serializable {  
    String Prenom,Nom,Adresse;
```



```
public Client(String Prenom,String Nom,String Adresse) {  
    //ici on serialise  
    this.Prenom=Prenom;  
    this.Nom=Nom;  
    this.Adresse=Adresse;  
}  
public String RetournerClient()  
    //ici on déserialise  
    {  
        String ClientRetrouver="\n Le prenom est: "+this.Prenom + "\n Le  
nom est : "+this.Nom + "\n L'adresse est : "+this.Adresse + "\n\n";  
        return ClientRetrouver;  
    }  
}
```

Classe GestionClients

```
package Personne;  
import java.io.*;  
import java.util.Scanner;  
public class ClientsUcao {  
    public static void main(String[] args) throws IOException,  
ClassNotFoundException {  
        File MonFichier=new File("client.txt");  
        FileOutputStream EcrireFichier=new  
FileOutputStream(MonFichier);  
        BufferedOutputStream Tampon=new  
BufferedOutputStream(EcrireFichier);  
        ObjectOutputStream Ecrire=new ObjectOutputStream(Tampon);  
        String p,n,a;  
        int i;  
        java.util.Scanner clavier=new java.util.Scanner(System.in);  
        for(i=0;i<4;i++)  
        {  
            System.out.println("*****Creation  
Nouveau Client*****");  
            System.out.println("Donner le prenom");  
            p=clavier.next();  
            System.out.println("Donner le nom");  
            n=clavier.next();  
        }  
    }  
}
```



```
        System.out.println("Donner l'adresse");
        a=clavier.next();
        Client Lenouveau=new Client(p,n,a);
        Ecrire.writeObject(Lenouveau);
        System.out.println("*****Nouveau
Client Enregistrer*****");
    }
    Ecrire.close();
    FileInputStream LireFichier=new FileInputStream(MonFichier);
    BufferedInputStream Tampon2=new
BufferedInputStream(LireFichier);
    ObjectInputStream Lire=new ObjectInputStream(Tampon2);
    for(i=0;i<4;i++)
    {
        // liere methode pour deserialiser
        System.out.println(((Client)Lire.readObject()).RetournerClient());
        Client Monclient=(Client) Lire.readObject();
        String Leclient=Monclient.RetournerClient();
        System.out.println(Leclient);
    }
    Lire.close();
}
}
```

TP : 4

Exo 1 : Ecrire un programme pour créer deux fichiers nommés Fic1.txt et Fic2.txt. Mettre Dans Fic1 votre prenom, votre nom et votre Adresse. Aprese avoir afficher le contenu de Fic, copier le contenu de Fic1 dans Fic2.

Exo 2 : Créer un répertoire téléphonique contenant vos camarades de classe : PRENOM, NOM et TELEPHOEN. Enregistrer les dans un fichier dans le repertoire D et nommé le Rep.txt.

Utiser la serialisation.

Exo 3 : Implémenter une application pour gérer les informations sur les clients de la société SARRIS.



Chapitre 7 : Accès base de données

JDBC (Java Data Base Connectivity)

JDBC est une API Java (ensemble de classes et d'interfaces défini par SUN et les acteurs du domaine des SGBD) permettant d'accéder aux bases de données à l'aide du langage Java via des requêtes SQL (langage permettant de dialoguer avec un SGBDR). Cette API permet d'atteindre de façon quasi transparente des bases Sybase, Oracle, Informix,... avec le même programme Java JDBC.

Les classes de JDBC version 1.0 sont regroupées dans le package `java.sql` et sont incluses dans le JDK à partir de sa version 1.1. La version 2.0 de cette API est incluse dans la version 1.2 du JDK. Pour pouvoir utiliser JDBC, il faut un pilote qui est spécifique à la base de données à laquelle on veut accéder. Avec le JDK, Sun fournit un pilote qui permet l'accès aux bases de données via ODBC. Ce pilote permet de réaliser l'indépendance de JDBC vis à vis des bases de données.

1. Les types de pilote JDBC

Il existe quatre types de pilote JDBC :

Type 1 (JDBC-ODBC bridge) : le pont JDBC-ODBC qui s'utilise avec ODBC et un pilote ODBC spécifique pour la base à accéder. Cette solution fonctionne très bien sous Windows. C'est une solution pour des développements avec exécution sous Windows d'une application locale qui a le mérite d'être universelle car il existe des pilotes ODBC pour la quasi totalité des bases de données. Cette solution "simple" pour le développement possède plusieurs inconvénients :

- *la multiplication du nombre de couches rend complexe l'architecture (bien que transparent pour le développeur) et détériore un peu les performances*
- *lors du déploiement, ODBC et son pilote doivent être installés sur tous les postes où l'application va fonctionner*
- *la partie native (ODBC et son pilote) rend l'application moins portable et dépendante d'une plate-forme*



Type 2 : un driver écrit en Java qui appelle l'API native de la base de données

Ce type de driver convertit les ordres JDBC pour appeler directement les API de la base de données via un pilote natif sur le client. Ce type de driver nécessite aussi l'utilisation de code natif sur le client.

Type 3 : un driver écrit en Java utilisant un middleware. Ce type de driver utilise un protocole réseau propriétaire spécifique à une base de données. Un serveur dédié reçoit les messages par ce protocole et dialogue directement avec la base de données. Ce type de driver peut être facilement utilisé par une applet mais dans ce cas le serveur intermédiaire doit obligatoirement être installé sur la machine contenant le serveur web.

Type 4 : un driver Java utilisant le protocole natif de la base de données

Ce type de driver, écrit en java, appelle directement le SGBD par le réseau. Il est fourni par l'éditeur de la base de données. Les drivers se présentent souvent sous forme de fichiers jar dont le chemin doit être ajouté au classpath pour permettre au programme de l'utiliser. JDBC permet à un programme Java d'interagir localement ou à distance avec une base de données relationnelle. Il fonctionne selon un principe client/serveur

client = le programme Java client

serveur = la base de données

2. Principe

- *le programme Java ouvre une connexion*
- *Il envoie des requêtes SQL*
- *Il récupère les résultats de la requête*
- *Il traite les données recueillies*
- *Il ferme la connexion une fois les traitements terminés*

3. Pilotes (JDBC Drivers)

L'ensemble des classes qui implémentent les interfaces spécifiées par JDBC pour un SGBD particulier est appelé pilote JDBC. Les protocoles d'accès aux bases de données étant propriétaires, il y a donc plusieurs drivers pour atteindre



diverses BD. Chaque BD utilise un pilote qui lui est propre et qui permet de convertir les requêtes dans le langage natif du SGBDR.

Les drivers dépendent du SGBD auquel ils permettent d'accéder. Pour travailler avec un SGBD, il faut disposer de classes (driver) qui implémentent les interfaces de JDBC.

JDBC est totalement indépendant de tout SGBD: la même application peut être utilisée pour accéder à une base Oracle, Sybase, MySQL, etc.

3.1. Chargement du driver

Un programme JDBC débute toujours par le chargement du pilote approprié pour la BD. Mais puisque le programme a la possibilité d'accéder à plusieurs types de BD, il peut avoir plusieurs pilotes. C'est au moment de la connexion à la BD que le DriverManager choisit alors le bon pilote.

DriverManager = gestionnaire de tous les drivers chargés par un programme java.

Structure d'un programme JDBC

Chargement du driver (chargement de la classe du driver dans la JVM)

Quand une classe Driver est chargée, elle doit créer une instance d'elle même et s'enregistrer auprès du DriverManager. Certains compilateurs refusent la notation précédente et demandent :

```
Class.forName(String driverName).new Instance();
```

Exemples:

Pour ODBC

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Pour Oracle

Class.forName("oracle.jdbc.driver.OracleDriver")

;

Pour MySQL

Class.forName("org.gjt.mm.mysql.Driver");

Cette étape 1 constitue l'enregistrement du driver JDBC



Exemple pour MySQL:

```
java.sql.Connection con;  
java.sql.Statement st;  
ResultSet rs;  
String url = "jdbc:mysql://localhost/baseucaao";  
String user = "root";  
String pwd = "";  
String Driver = "org.gjt.mm.mysql.Driver";  
/* Etape 1: chargement du driver */  
Class.forName(Driver).newInstance ();
```

3.2.Établir une connexion à la base de données

Une fois le driver enregistré, une connexion peut être établie avec la BD. Pour se connecter à un SGBD, il faut faire appel à la classe gestionnaire de drivers. Cette connexion se fait avec la méthode **getConnection(...)** de la classe **DriverManager**. Cette méthode peut prendre 3 arguments au plus:

- un URL vers la BD
- un nom de l'utilisateur de la base
- un mot de passe

Cette méthode renvoie un objet de type **Connection**.

```
Connection con = DriverManager.getConnection(" jdbc:mysql://host/Base" ,  
"login", "passwd");
```

Exemple pour MySQL:

```
java.sql.Connection con;  
java.sql.Statement st;  
ResultSet rs;  
String url = "jdbc:mysql://localhost/baseucaao";  
String user = "root";  
String pwd = "";  
String Driver = "org.gjt.mm.mysql.Driver";  
/* Etape 1: chargement du driver */  
Class.forName(Driver).newInstance ();
```



```
/* Etape 2 : connexion à la base de données */  
con = DriverManager.getConnection(url,user,pwd) ;
```

3.3.Création de requêtes SQL en utilisant l'objet de type Connection

Dans cette étape on crée en fait une zone de description de requêtes, ie un espace où l'on pourra exécuter des opérations SQL. Cette zone est un objet de la classe **Statement** que l'on crée avec la méthode **createStatement()**.

```
Statement st = con.createStatement ( ) ;
```

Il existe trois types de Statement:

- *Statement: requêtes statiques simples,*
- *PreparedStatement: requêtes dynamiques pré-compilées (avec paramètres d' I/O),*
- *CallableStatement: procédures stockées.*

Suite Exemple pour MySQL:

```
java.sql.Connection con;  
java.sql.Statement st;  
ResultSet rs;  
String url = "jdbc:mysql://localhost/baseucaao";  
String user = "root";  
String pwd = "";  
String Driver = "org.gjt.mm.mysql.Driver";  
  
/* Etape 1: chargement du driver*/  
Class.forName(Driver).newInstance ( ) ;  
/* Etape 2 : connexion à la base de données */  
con = DriverManager.getConnection(url,user,pwd) ;  
// Etape 3: Création de l'objet statement pour l'exécution des requêtes  
st = con.createStatement() ;
```

3.4.Envoi et exécution de requêtes

Il existe trois types d'exécution de requêtes:



- ***executeQuery(...)***: pour les requêtes *SELECT* qui retournent un *ResultSet* (tuples),
- ***executeUpdate(...)***: pour les requêtes *INSERT*, *UPDATE*, *DELETE*, *CREATE TABLE* *DROP TABLE* qui retourne un entier (*int*) désignant le nombre de tuples traités.
- ***execute()***: pour les procédures stockées (cas rares).

Les méthodes `executeQuery()` et `executeUpdate()` de la classe `Statement` prennent comme argument une chaîne (`String`) indiquant la requête SQL à exécuter.

```
ResultSet rs = st.executeQuery("select * from Client");
```

Remarques

Le code SQL n'est pas interprété par Java. C'est le pilote associé à la connexion (et au finish par le moteur de la BD) qui interprète la requête SQL. Si une requête ne peut s'exécuter ou qu'une erreur de syntaxe SQL a été détectée, l'exception `SQLException` est levée. Le driver `JDBC` effectue d'abord un premier accès à la BD pour découvrir les types des colonnes impliquées dans la requête puis un deuxième pour l'exécuter.

Suite Exemple pour MySQL:

```
java.sql.Connection con;  
java.sql.Statement st;  
ResultSet rs;  
String url = "jdbc:mysql://localhost/baseucaao";  
String user = "root";  
String pwd = "";  
String Driver = "org.gjt.mm.mysql.Driver";  
/* Etape 1: chargement du driver */  
Class.forName(Driver).newInstance();  
/* Etape 2 : connexion à la base de données */  
con = DriverManager.getConnection(url,user,pwd);  
// Etape 3: Création de l'objet statement pour l'exécution des requêtes  
st = con.createStatement();  
// Etape 4: execution de la requête et liberation de la ressource  
String requetesql = "insert into etudiants values (4,'Edouard'";
```



```
, 'Sarr', 'LPIG4' );  
    rs = st.executeUpdate(requetesql) ;  
    st.close ( );  
    con.close ( );
```

NB: chaque instruction dans la pratique est enveloppe d'un bloc Try

4. Traitement des données

L'objet ResultSet (retourné par l'exécution de executeQuery()) permet d'accéder aux champs des enregistrements (tuples) sélectionnés. Seules les données demandées sont transférées en mémoire par le driver JDBC. Il faut donc les lire manuellement et les stocker dans des variables pour un usage ultérieur.

La méthode **next()** de ResultSet permet de parcourir itérativement ligne par ligne l'ensemble des tuples sélectionnés. Cette méthode :

- *retourne false si le dernier tuple est lu, true sinon,*
- *chaque appel fait avancer le curseur sur le tuple suivant,*
- *initialement, le curseur est positionné avant le premier tuple*
- *Exécuter next() au moins une fois pour avoir le premier.*

```
while (rs.next()) { //traitement tuple par tuple }
```

Les colonnes d'une table de la BD sont référencées par leur numéro (commençant par 1) ou par leur nom. L'accès aux valeurs des colonnes se fait par des méthodes de la forme getXXX(...) permettant la lecture de données du type XXX dans chaque colonne du tuple courant.

Exemple : **int val = rs.getInt(3) ; // acces à la 3e colonne**

```
String Pre = rs.getString("Prenom") ;// accès à la colonne Prenom
```

Résumé :

- `java.sql.DriverManager`
 - `java.sql.Connection`
 - `java.sql.Statement`
 - `java.sql.PreparedStatement`
 - `java.sql.CallableStatement`
 - `java.sql.ResultSet`
 - `java.sql.DatabaseMetaData`
- Classes de connexion
- Interfaces de requête sql
- Classe de traitement des résultats
- Classe de récupération d'info sur la structure de la BD

Type Java	méthode	Type SQL
boolean	<code>getBoolean</code>	BIT
String	<code>getString</code>	CHAR
double	<code>getDouble</code>	FLOAT
float	<code>getFloat</code>	REAL
int	<code>getInteger</code>	INTEGER
long	<code>getLong</code>	BIG INT
byte	<code>getByte</code>	TINYINT
<code>java.math.BigDecimal</code>	<code>getBigDecimal</code>	NUMERIC
<code>byte[]</code>	<code>getBytes</code>	BINARY
<code>java.sql.Date</code>	<code>getDate</code>	DATE
<code>java.sql.Time</code>	<code>getTime</code>	TIME

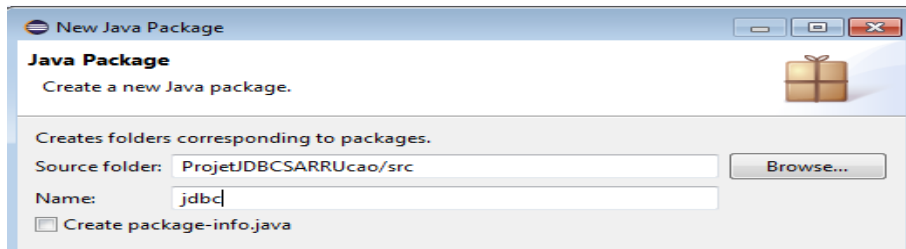
5. Exercice d'application :

Lançons ECLIPSE

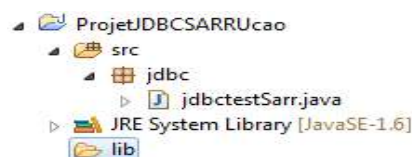
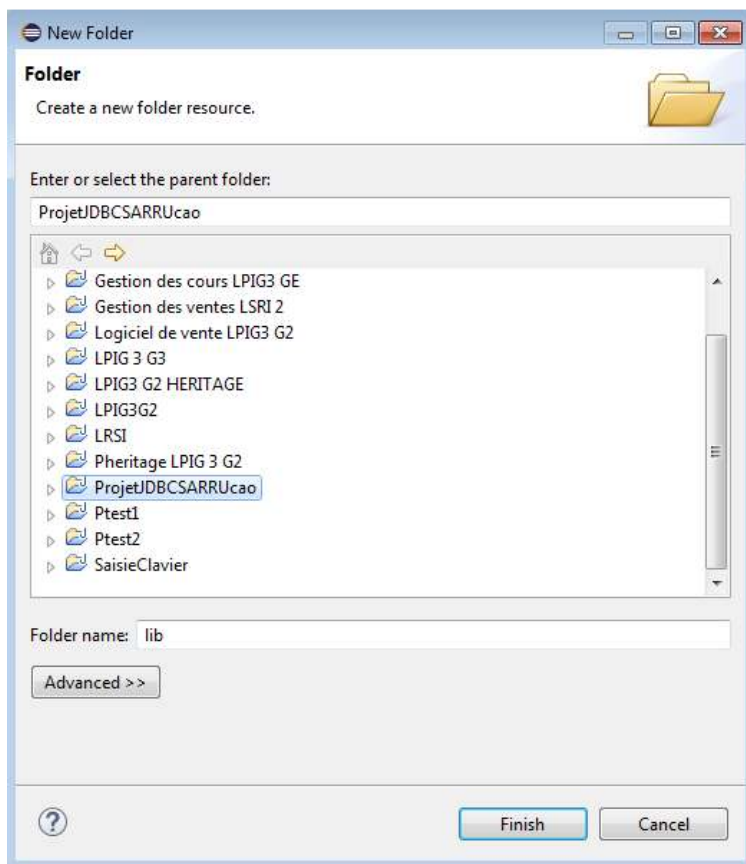
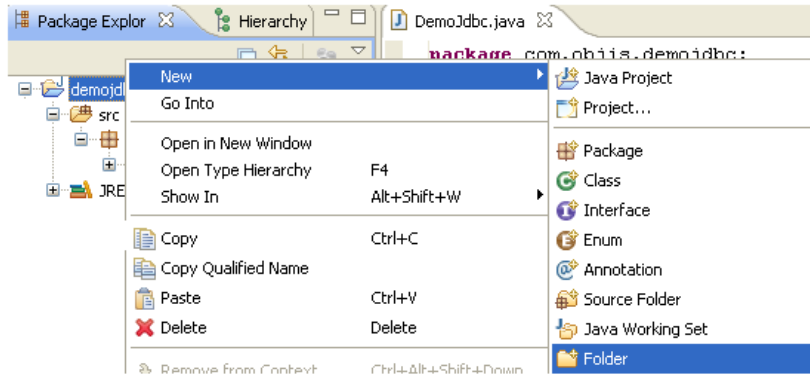
Créons un projet java nommé **ProjetJDBCSARRUcao**.



Créons un package JDBC



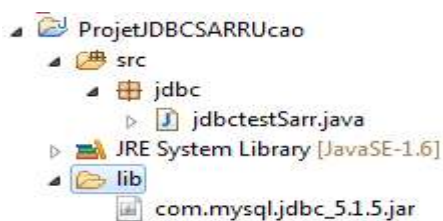
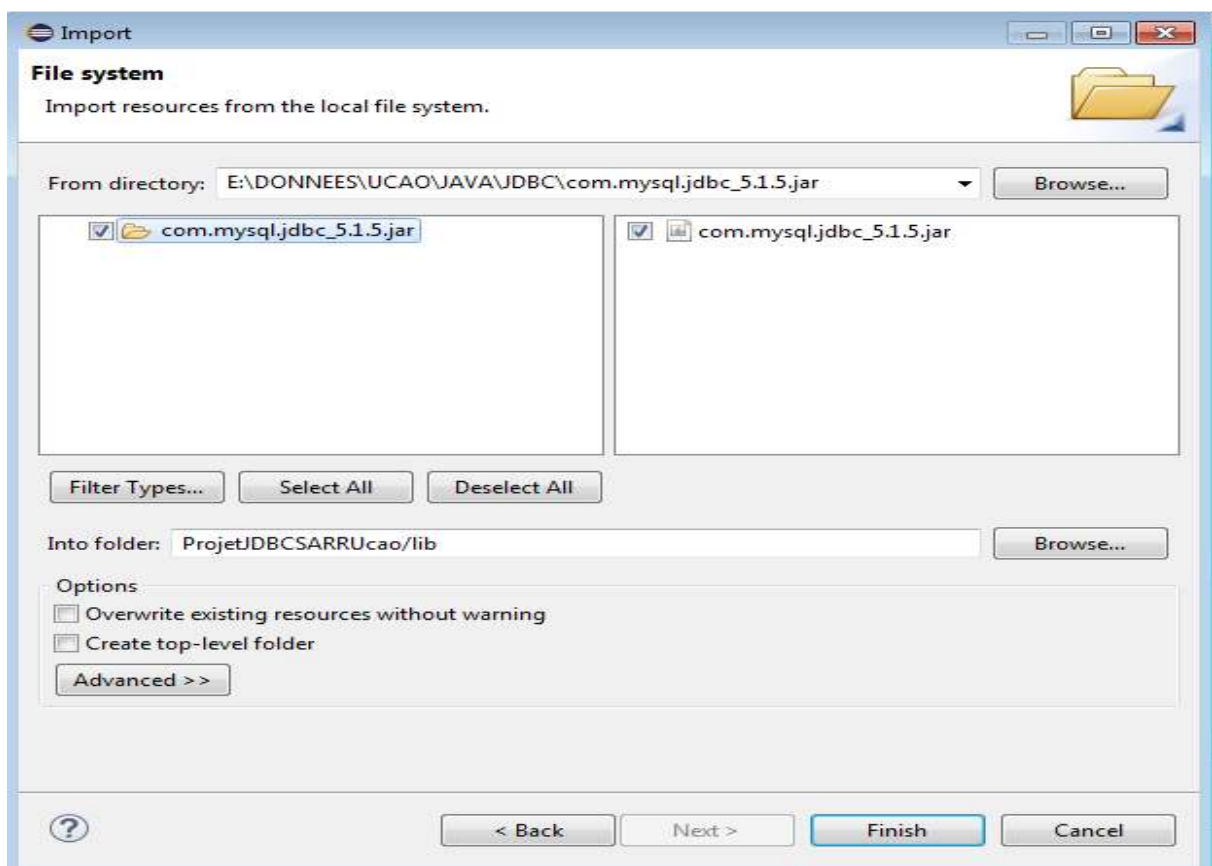
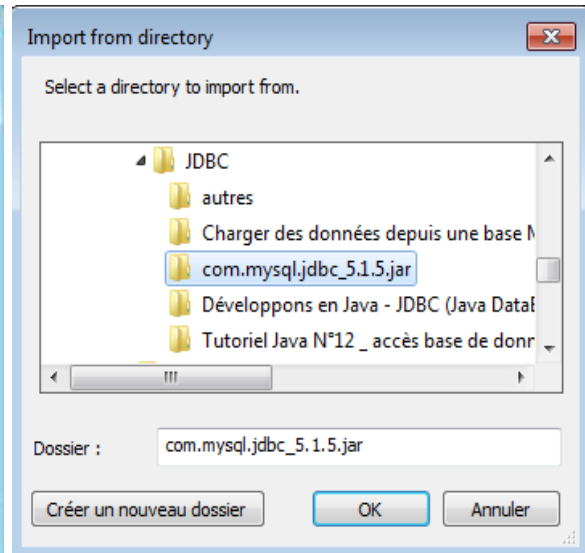
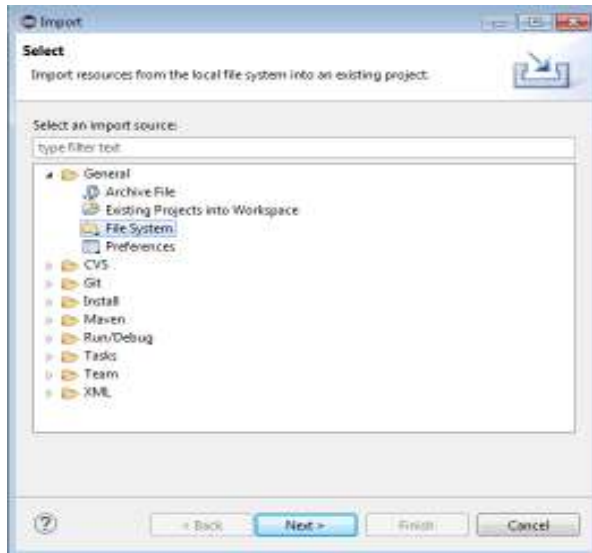
Ajout du driver JDBC MySql : Créer un repertoire lib dans la racine du projet



Dans ce répertoire importez le driver JDBC :

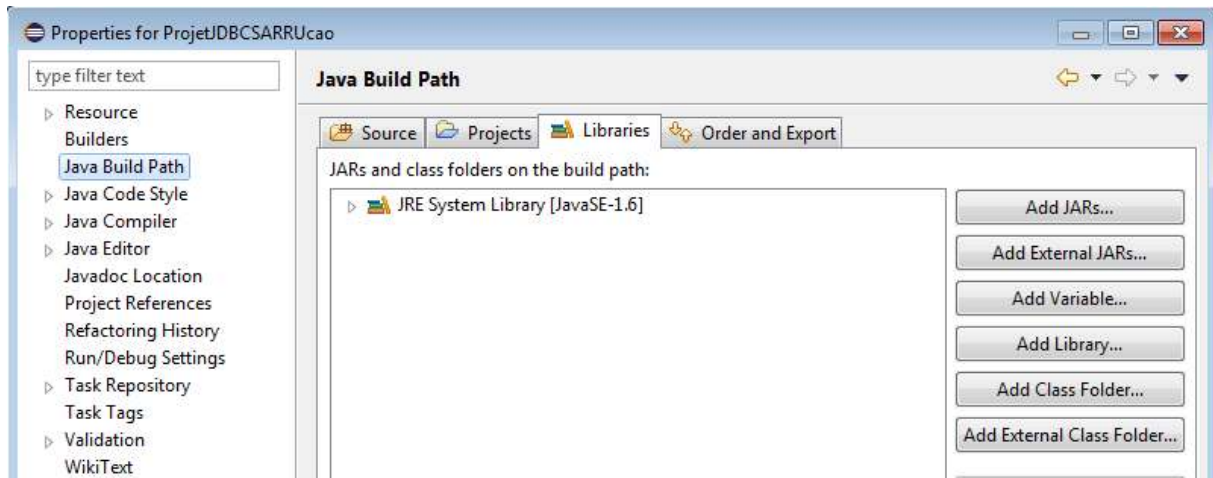
clicquez-droit puis import

General/File System

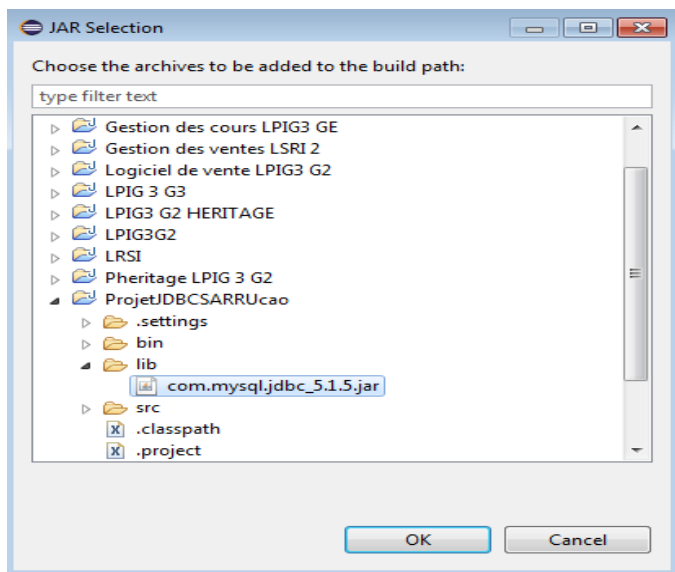


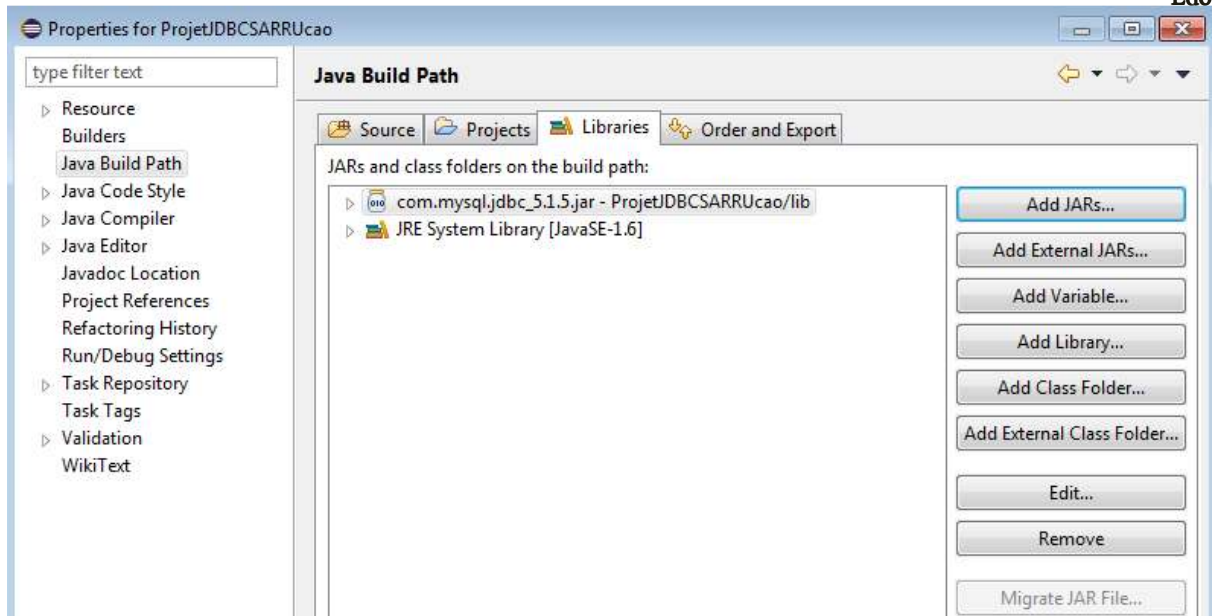
Informez Eclipse de l'existence de ce driver

Cliquez-droit sur le projet->properties. L'écran suivant apparaît. Ensuite java Build Path et l'onglet Librairie

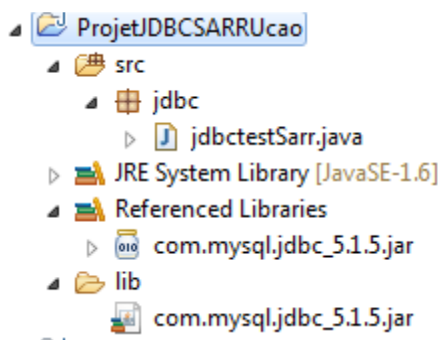


Dans l'onglet 'Librairies' cliquez sur bouton 'Add Jars' car le driver est déjà dans le projet (Sinon il faudrait cliquer sur Add External Jars)

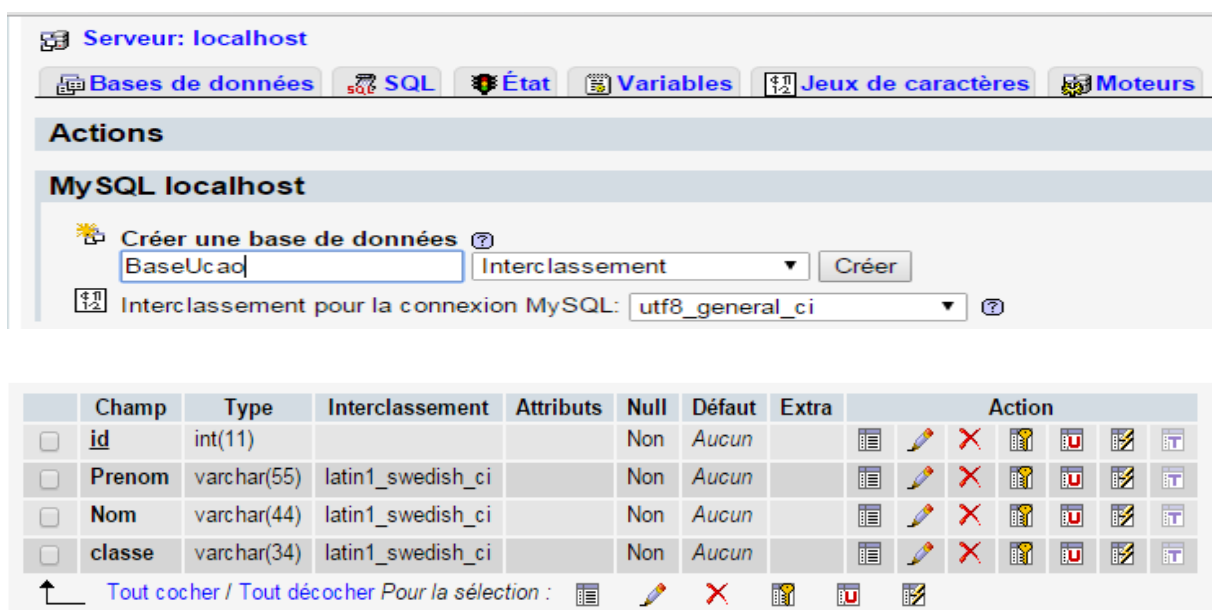




sélectionnez le driver puis OK. Le driver apparaît désormais dans 'Referenced librairies'.



Création une base de données BaseUcao et une table Etudiants





Insertion de données en base

Créons une classe `Insertion.java`

```
package jdbc;
public class Insertion {
    public Insertion() {
    }
    public static void main(String[] args) {
    }
}
```

Implémentons cette classe

```
package jdbc;
import java.sql.*;
import com.mysql.jdbc.*;
public class Insertion {
    /*******declaration des variables
    static java.sql.Connection con;
    static java.sql.Statement st;
    static int rs;
    static String url ="jdbc:mysql://localhost/baseucaao";
    static String user="root";
    static String pwd="";
    static String Driver="org.gjt.mm.mysql.Driver";
    public static void main(String[] args) {
        /* ***** Etape 1: chargement du driver*/
        try{
            Class.forName(Driver).newInstance ( ) ;
        }
        catch(Exception e){
            System.out .println("Erreur de chargement du driver:
"+e.getMessage() ) ;
        }
        /* ***** Etape 2 : Connexion à la base de données basesarr
avec l'utilisateur root et le mot de passe sarr*/
        try {
            con = DriverManager.getConnection(
url,user,pwd) ;
```

```

    }
    catch (Exception ez ){
        System.out.println("Erreur de connexion à la base"+
ez.getMessage());
    }
    // Etape 3: Création de l'objet statement pour l'exécution des requêtes
    try {
        st = con.createStatement() ;
    }
    catch (SQLException t){
        System.out.println ("Erreur de Statement "+t.getMessage());
    }
    /* Etape 4 : Exécution de la requête de sélection puis stock du résultat
dans le ResultSet rs et affichage*/
    try {
        String requetesql="insert into etudiants values (1,'fatou'
,'SARR','LPIG3')";
        rs = st.executeUpdate(requetesql) ;
    }
    catch (Exception er){
        System.out .println("Erreur ResultSet "+er.getMessage ( ) );
    }
    // Etape 5: Fermer toutes les connexion avec la BD
    try {
        st.close ( ); con.close ( );
    }
    catch(Exception d)
    {}
    //*****
}
}

```

+ Options

	id	Prenom	Nom	classe
<input type="checkbox"/>	1	fatou	SARR	LPIG3

↑ Tout cocher / Tout décocher Pour la sélection :

+ Options

	id	Prenom	Nom	classe
<input type="checkbox"/>	2	Edouard	SARR	LPIG2
<input type="checkbox"/>	1	fatou	SARR	LPIG3

↑ Tout cocher / Tout décocher Pour la sélection :

Suppression de données en base

C'est le même principe seulement que la requête change

/ Etape 4 : Exécution de la requête de sélection puis stock du résultat dans le ResultSet rs et affichage*/*

```
try {  
    String requetesql="Delete from etudiants where id=1";  
    rs = st.executeUpdate(requetesql) ;  
}  
catch (Exception er){  
    System.out.println("Erreur ResultSet "+er.getMessage ( ) );  
}
```

+ Options

	id	Prenom	Nom	classe
<input type="checkbox"/>	2	Edouard	SARR	LPIG2

↑ Tout cocher / Tout décocher Pour la sélection :   

Modification de données en base

/ Etape 4 : Exécution de la requête de sélection puis stock du résultat dans le ResultSet rs et affichage*/*

```
try {  
    String requetesql="UPDATE etudiants Set Prenom= 'Edouard  
Ngor' where id=2";  
    rs = st.executeUpdate(requetesql) ;  
}  
catch (Exception er){  
    System.out.println("Erreur ResultSet "+er.getMessage ( ) );  
}
```

+ Options

	id	Prenom	Nom	classe
<input type="checkbox"/>	2	Edouard Ngor	SARR	LPIG2

↑ Tout cocher / Tout décocher Pour la sélection :   

NB ; le résultat de la requête est :

Type entier pour : INSERT, UPDATE et DELETE

Type **ResultSet** pour SELECT



Recherche de données en base

+ Options						
			id	Prenom	Nom	classe
			3	aliou	Faye	LPIG3
			1	Fatou	SARR	LPIG2
			2	Edouard Ngor	SARR	LPIG2
			4	Samba	Sene	LPIG4

```
package jdbc;
import java.sql.*;
import com.mysql.jdbc.*;
public class Selection {
    //*****declaration des variables
    static java.sql.Connection con;
    static java.sql.Statement st;
    static ResultSet rs;
    static String url = "jdbc:mysql://localhost/baseucaao";
    static String user = "root";
    static String pwd = "";
    static String Driver = "org.gjt.mm.mysql.Driver";
    public static void main(String[] args) {
        /* ***** Etape 1: chargement du driver*/
        try{
            Class.forName(Driver).newInstance ( ) ;
        }
        catch(Exception e){
            System.out .println("Erreur de chargement du driver:
"+e.getMessage() ) ;
        }
        /* ***** Etape 2 : Connexion à la base de données basesarr
avec l'utilisateur root et le mot de passe sarr*/
        try {
            con = DriverManager.getConnection(
url,user,pwd) ;
        }
        catch (Exception ez ){
            System.out.println("Erreur de connexion à la base"+
ez.getMessage());
        }
        // Etape 3: Création de l'objet statement pour l'exécution des requêtes
        try {
```




```
        st = con.createStatement() ;
    }
    catch (SQLException t){
        System.out.println ("Erreur de Statement "+t.getMessage());
    }
    /* Etape 4 : Exécution de la requête de sélection puis stock du résultat
    dans le ResultSet rs et affichage*/
    try {
        rs = st.executeQuery("select * from etudiants") ;
        while (rs.next() ){
            System.out .println(rs.getObject (1)+"      "+rs.getObject(2)+"
            "+rs.getObject(3) ) ;
        }
    }
    catch (Exception er){
        System.out .println("Erreur ResultSet "+er.getMessage ( ) );
    }
    // Etape 5: Fermer toutes les connexion avec la BD
    try {
        st.close ( ); con.close ( );
    }
    catch(Exception d)
    { }

    /*******
    }
}
```

Résultat :

```
<terminated> Selection [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (11 janv.
3      aliou  Faye
1      Fatou  SARR
2      Edouard Ngor  SARR
4      Samba  Sene
```