



# Programmation par objets-JAVA

**Responsable : Dr. DIAW**

**Audience: DUT INFO 2 /DST INFO 2/ DUT 2 TR**



# Sommaire

- Définition
- Histoire de la POO
- Concepts de base
  - Objet
  - Classe
  - Hiérarchie des classes
  - Encapsulation
- Application: Langage JAVA



# Définition

# Définition

- Programmation orientée objet
  - Méthode d'implantation dans laquelle le programme est organisé en collection d'objets coopératifs. Chaque objet est une instance de classe. Toutes les classes sont membres d'une hiérarchie de classes liées par des relations d'héritage



# Histoire de la POO

# ■ Histoire de la POO

- 1967 : Simula, 1<sup>er</sup> langage de programmation à implémenter le concept de type abstrait à l'aide de classes
- 1976 : Smalltalk implémente les concepts fondateurs de l'approche objet (encapsulation, agrégation, héritage) à l'aide de :
  - classes
  - associations entre classes
  - hiérarchies de classes
  - messages entre objets
- 1980 : le 1<sup>er</sup> compilateur C++. C++ est normalisé par l'ANSI et de nombreux langages orientés objets académiques ont étayés les concepts objets : Eiffel, Objective C, Loops, java, Python, Ruby...



# Concepts de base

# ■ Objet

- Représentation abstraite d'une entité du monde réel ou virtuel
- Caractéristique fondamentales d'un objet (informatique)

Objet = État + Comportement + Identité

## ■ État

- Regroupe les valeurs instantanées de tous les attributs d'un objet :
  - attribut est une information qui qualifie l'objet qui le contient
  - Chaque attribut peut prendre une valeur dans un domaine de définition donné

Un étudiant

Matricule

Prénom

Nom

Exemple : Un objet Etudiant regroupe les valeurs des attributs numéro Etudiant, Prénom, Nom et Age

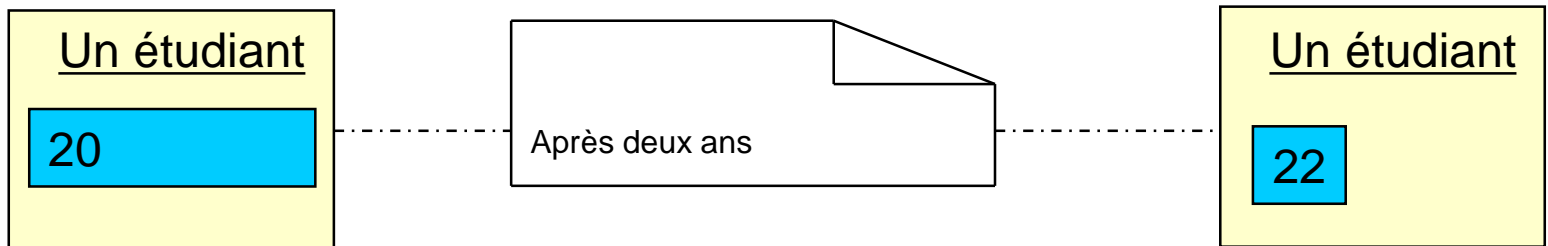


# ■ Objet

- Caractéristiques fondamentales d'un objet (informatique)

## ■ État

- L'état d'un objet à un instant donné, correspond à une sélection de valeurs, parmi toutes les valeurs possibles des différents attributs
- L'état évolue au cours du temps, il est la conséquence de ses comportements passés
  - Un étudiant passe à deux ans de plus, son âge change



- Certaines composantes de l'état peuvent être constantes
  - Le prénom et le nom d'un étudiant par exemple



# ■ Objet

- Caractéristique fondamentales d' un objet (informatique)

- **Le comportement**

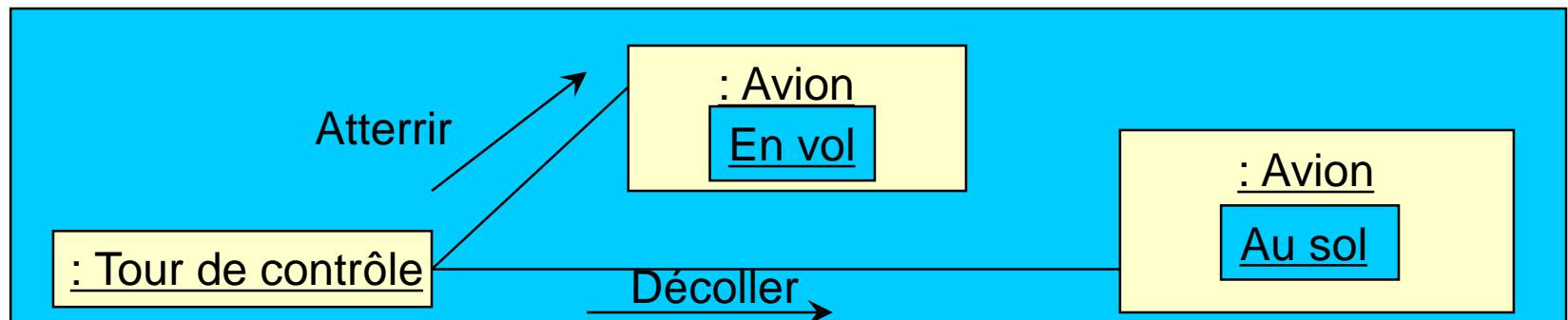
- Regroupe toutes les compétences d' un objet et décrit les actions et les réactions de cet objet
- Chaque atome (partie) de comportement est appelé opération
  - Les opérations d' un objet sont déclenchées suite à une stimulation externe, représentée sous la forme d' un message envoyé par un autre objet
  - L' état et le comportement sont liés

# ■ Objet

## □ Exemple

### ■ Le comportement

- Le comportement à un instant donné dépend de l'état courant et l'état peut être modifié par le comportement
  - Il n'est pas possible de faire atterrir un avion que s'il est en train de voler: le comportement *Atterrir* n'est valide que si l'information *En vol* est valide
  - Après l'atterrissage, l'information *En vol* devient invalide et l'opération *Atterrir* n'a plus de sens



# ■ Objet

## □ Caractéristique fondamentales d' un objet

### ■ L' identité

- Chaque objet possède une identité qui caractérise son existence propre
- L' identité permet de distinguer tout objet de façon non ambiguë, indépendamment de son état
- Permet de distinguer deux objets dont toutes les valeurs d' attributs sont identiques
  - deux pommes de la même couleur, du même poids et de la même taille sont deux objets distincts.
  - Deux véhicules de la même marque, de la même série et ayant exactement les mêmes options sont aussi deux objets distincts.
- En phase de réalisation, l' identité est souvent construite à partir d' un identifiant issu naturellement du domaine du problème.
- Nos voitures possèdent toutes un numéro d' immatriculation, nos téléphones un numéro d' appel



# ■ Classe

## □ Définition

- Une classe décrit une abstraction d'objets ayant
  - Des propriétés similaires
  - Un comportement commun
  - Des relations identiques avec les autres objets
  - Une sémantique commune
- Par exemple Etudiant (resp. Filière, resp. Cours) est la classe de tous les étudiants (resp. filières, resp. cours)

# ■ Classe

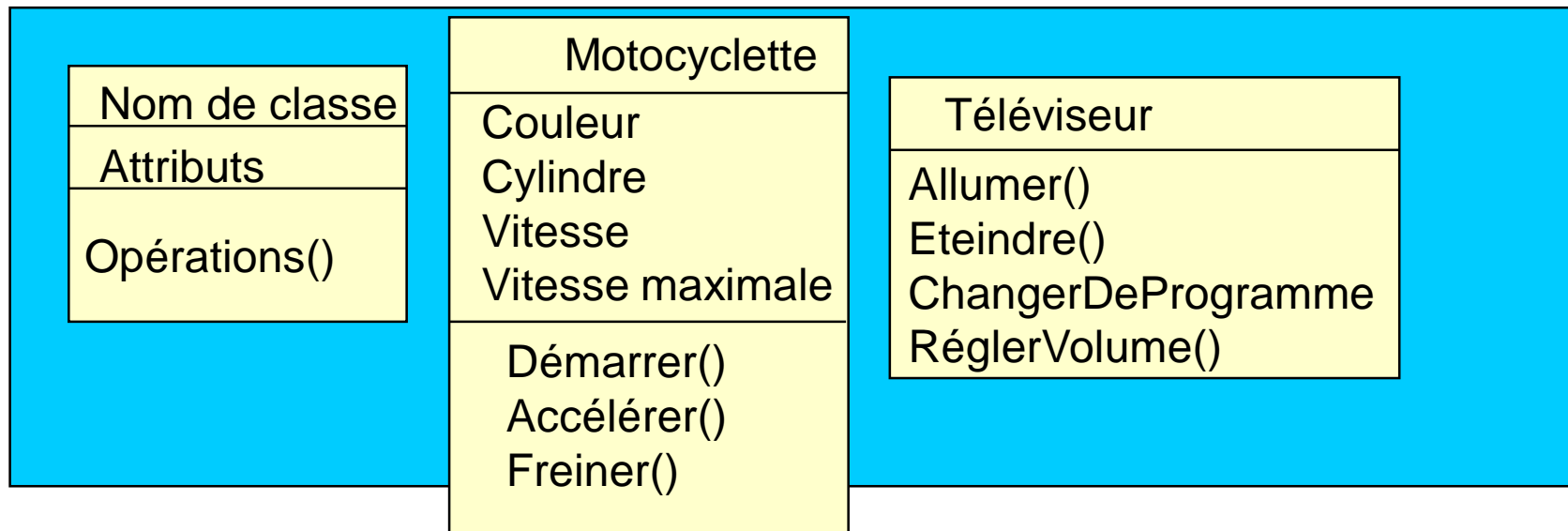
## □ Caractéristiques d' une classe

- Un objet créé par (on dit également appartenant à) une classe sera appelé une *instance de cette classe* ce qui justifie le terme ``*variables d'instances* ' '
- les valeurs des *variables d'instances* sont propres à chaque instances
- Les généralités sont contenues dans la classe et les particularité sont contenues dans les objets
- Les objets sont construits à partir des classes, par un processus appelé instantiation : tout objet est une instance de classe
- Nous distinguons deux types de classes
  - Classe concrète : peut être instanciée
  - Classe abstraite : est une classe non instanciable et contient au moins une méthode abstraite.

# ■ Classe

## □ Représentation graphique d'une classe en UML

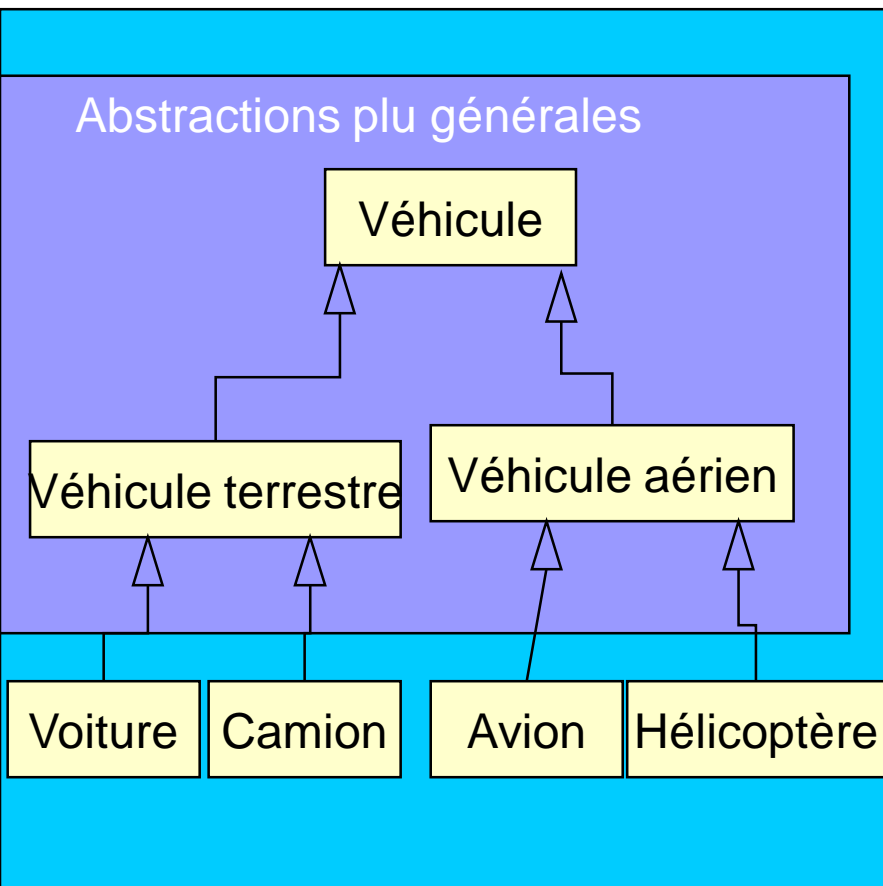
- Chaque classe est représentée sous la forme d'un rectangle divisé en trois compartiments
- Les compartiments peuvent être supprimés pour alléger les diagrammes



# ■ Les hiérarchies de classes

## □ Généralisation et spécialisation

- La généralisation consiste à factoriser les éléments communs (attributs, opérations) d'un ensemble de classes dans une classe plus générale appelée super-classe



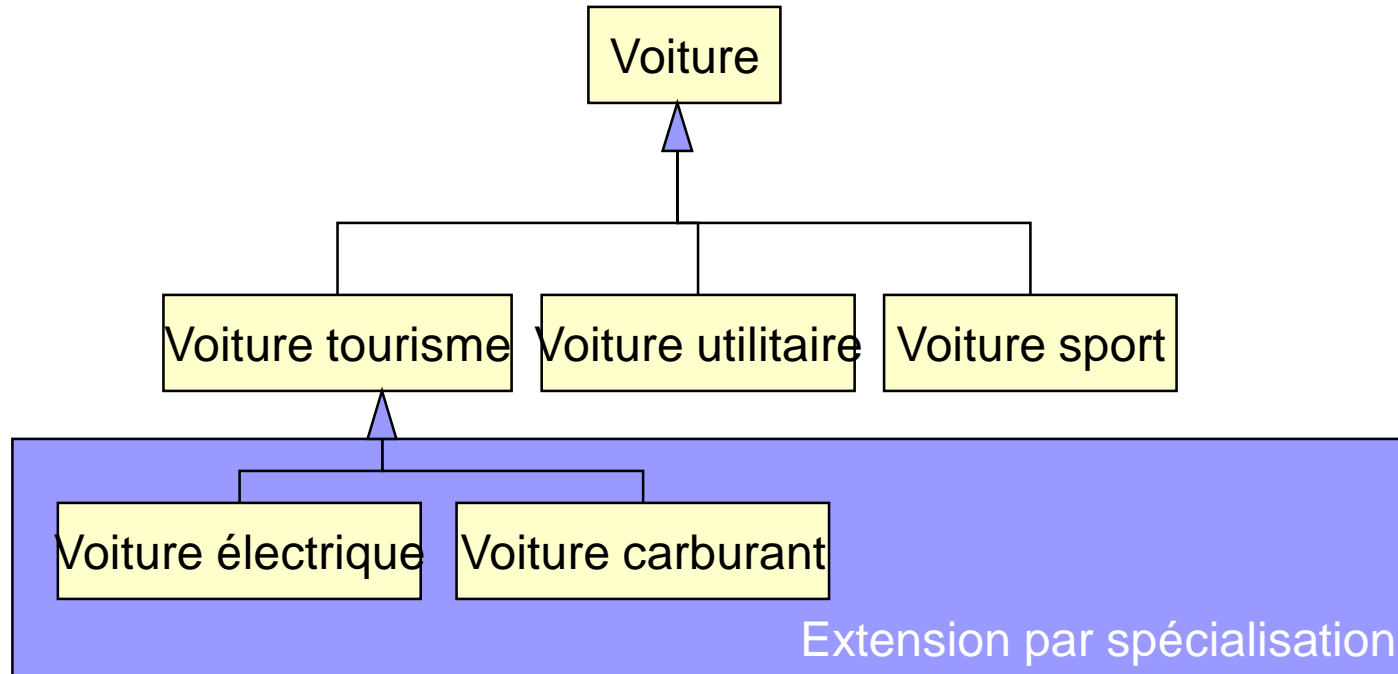
- Les classes sont ordonnées selon une hiérarchie ; une super-classe est une abstraction de ses sous-classes
- Un arbre (une hiérarchie) représentant une généralisation se détermine en partant des feuilles (et non pas de la racine) car les feuilles appartiennent au monde réel alors que les niveaux supérieurs sont des abstractions construites pour ordonner et comprendre



# ■ Les hiérarchies de classes

## ■ Généralisation et spécialisation

- La spécialisation permet de capturer les particularités d'un ensemble d'objets non discriminés par les classes déjà identifiées
  - Les nouvelles caractéristiques sont représentées par une nouvelle classe, sous-classe d'une des classes existantes



# ■ Les hiérarchies de classes

## ■ Généralisation et spécialisation

- La généralisation et la spécialisation sont deux point de vue antagonistes du concept de classification; elle expriment dans quel sens une hiérarchie de classe est exploitée
- La généralisation ne porte aucun nom particulier ; elle signifie toujours : est un ou est une sorte de
- La généralisation ne concerne que les classes, elle n'est pas instanciable en liens
- La généralisation ne porte aucune indication de multiplicité

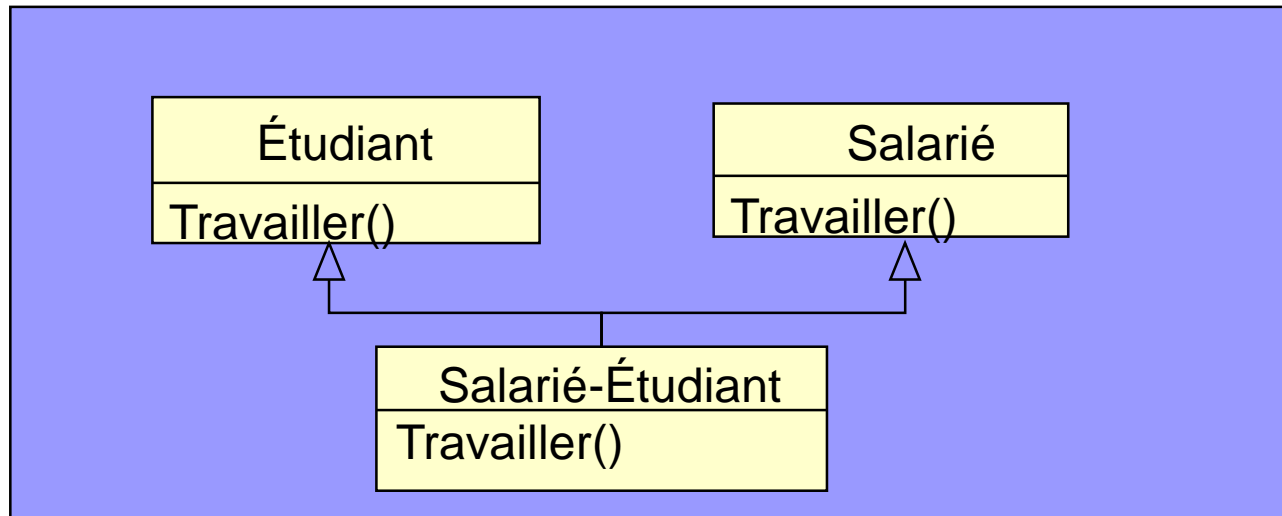
# ■ Les hiérarchies de classes

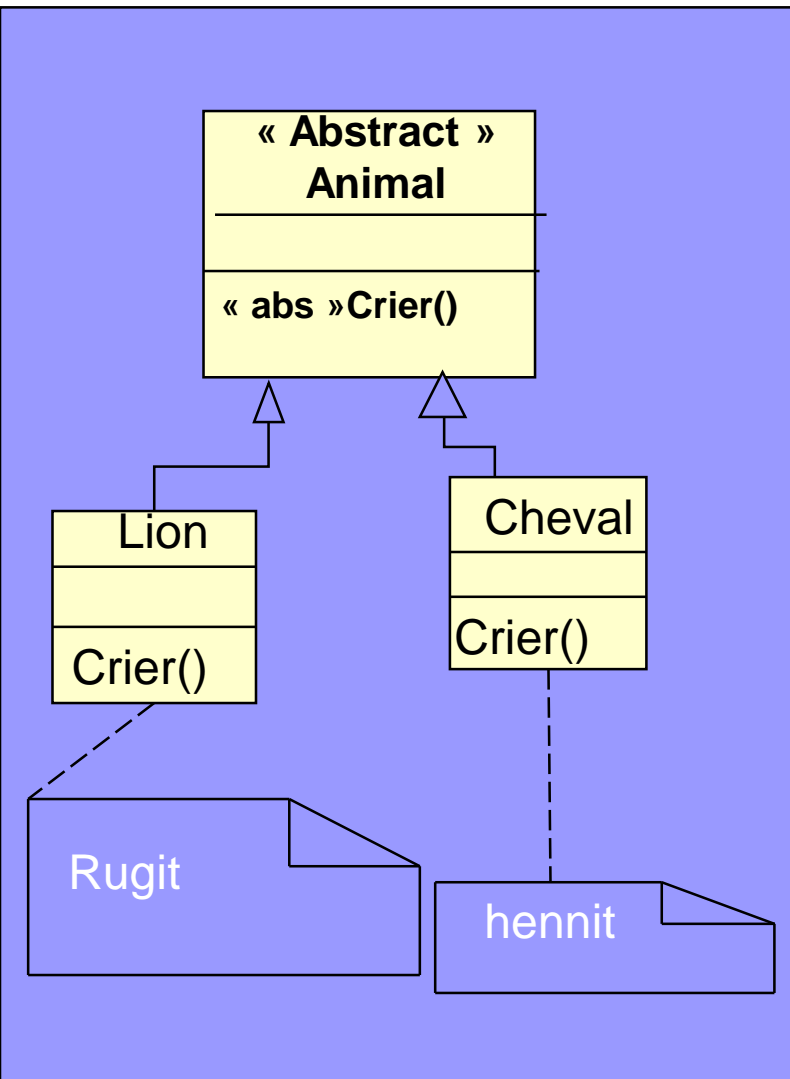
## ■ Généralisation et spécialisation

- La généralisation est une relation non réflexive : une classe ne peut pas dériver d'elle-même
- La généralisation est une relation non symétrique : si une classe B dérive d'une classe A, alors la classe A ne peut pas dériver de la classe B
- La généralisation est une relation transitive : si une classe C dérive d'une classe B qui dérive elle-même d'une classe A, alors C dérive également de A
- Les objets instances d'une classe donnée sont décrit par la propriétés caractéristiques de leur classe, mais également par les propriétés caractéristiques de toutes les classes parents de leur classe

# ■ Les hiérarchies de classes

- Généralisation multiple :
- Elle existe entre arbres de classes disjoints
  - Une classe ne peut posséder qu'une fois une propriété donnée





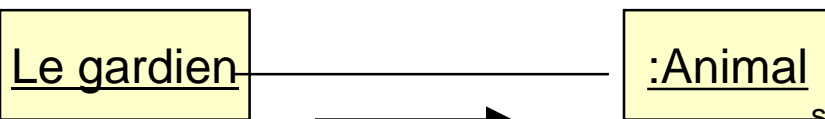
## ■ Les hiérarchies de classes

### ■ Polymorphisme : notion de surcharge

- Le polymorphisme signifie qu'une même opération peut se comporter différemment sur différentes classes

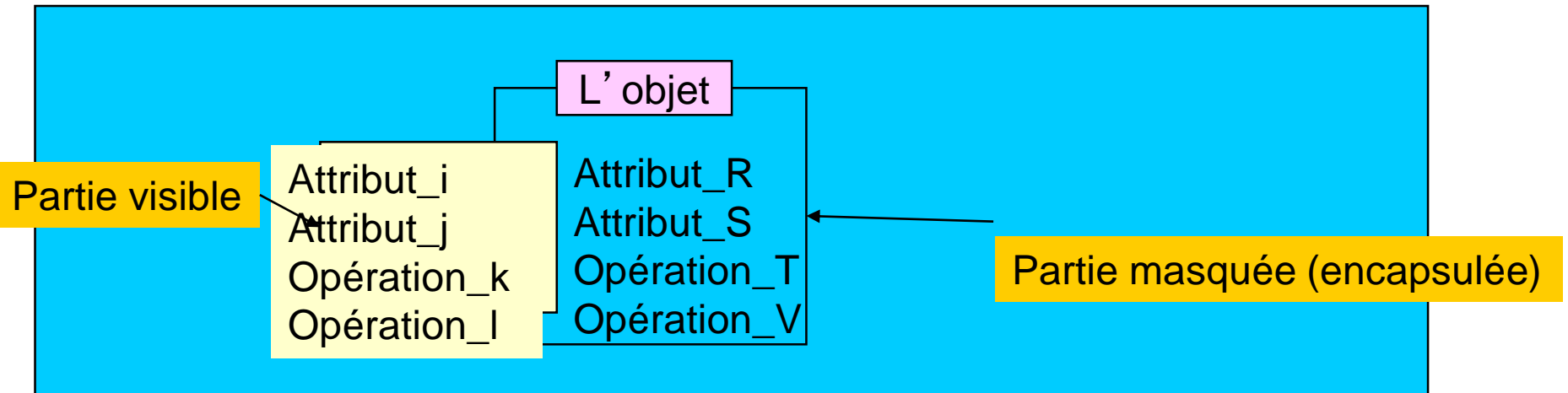
### ■ L'opération **Crier()** agira de manières différentes sur un lion, un cheval ou un chien.

- Le polymorphisme signifie que les différentes méthodes d'une opération ont la même signature
  - Lorsque une opération est invoquée sur un objet, celui-ci « connaît » sa classe et par conséquent est capable d'invoquer automatiquement la méthode correspondante.



# ■ Encapsulation

- Consiste à masquer les détails d'implémentation d'un objet en définissant une interface
- L'interface est la vue externe d'un objet (spécification), elle définit les services accessibles (offerts) aux utilisateurs de l'objet.
- Est la séparation entre les propriétés externes, visibles des autres objets, et les aspects internes, propres aux choix d'implantation d'un objet.





## ■ Encapsulation

- garantit l'intégrité des données, car elle permet d'interdire ou de restreindre l'accès direct aux attributs des objets (utilisation d'accesseurs)

## Règle de visibilité

+ Attribut public  
# Attribut protégé  
- Attribut privé  
Attribut de niveau package

+ Opération publique()  
# Opération protégée()  
- Opération privée()

## Salarié

+ nom  
# age  
- salaire

+ donnerSalaire()  
# changerSalaire()  
- calculerPrime()

# Encapsulation

- Il est possible d'assouplir le degré d'encapsulation au profit de certaines classes utilisatrices bien particulière en définissant des niveaux de visibilité
- Les niveaux de visibilité sont:
  - **Niveau privé** : c'est le niveau le plus fort; la partie privée de la classe est totalement opaque et n'est pas visible aux autres objets. Un attribut défini dans la partie privée d'une classe n'est visible que par les méthodes de cette classe. Pour toutes les autres classe, l'attribut reste invisible
  - **Niveau Paquetage**: Un attribut défini dans ce niveau est visible dans toutes les classes appartenant au paquetage



## Règle de visibilité

+ Attribut public  
# Attribut protégé  
- Attribut privé  
Attribut de niveau package

+ Opération publique()  
# Opération protégée()  
- Opération privée()

## Salarié

+ nom  
# age  
- salaire

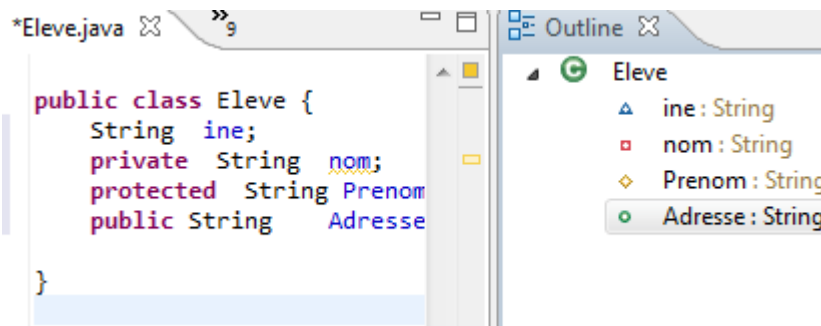
+ donnerSalaire()  
# changerSalaire()  
- calculerPrime()

# Encapsulation

- Il est possible d'assouplir le degré d'encapsulation au profit de certaines classes utilisatrices bien particulière en définissant des niveaux de visibilité
- Les niveaux de visibilité sont:
  - **Niveau protégé** : les attributs placés dans la partie protégée sont visibles
    1. dans toutes les classes appartenant au même paquetage
    2. Dans toutes les classe filles de la classe mère ou l'attribut est défini
  - **Niveau public** : ceci revient à se passer de la notion d'encapsulation et de rendre visibles les attributs pour toutes les classes

# Attributs

- Une classe peut contenir des attributs
- La syntaxe d'un attribut est :  
*visibilité nom : type*
- La visibilité est:
  - ☐ par défaut ( pour package)
  - ☐ '+' pour public
  - ☐ '#' pour protected
  - ☐ '-' pour private





# Java



# Sommaire

1. Les variables et types
2. Les opérations arithmétiques
3. Les opérateurs
4. Structures de contrôle
5. Les classes
6. Tableaux (vecteurs et matrices)
7. Chaines de caractères
8. Les énumérations
9. Interface et classe d'implémentation
10. Les exceptions

# Syntaxe

- Syntaxe de Java est inspirée de celle du langage C
- Java est sensible à la casse
- Les blocs de code sont encadrés par des accolades
- Chaque instruction se termine par un ;
- Une instruction peut se tenir sur plusieurs lignes

# Variable

- Variable est décrite par son identificateur et son type
- Le premier caractère doit être une lettre, le caractère de soulignement ou le signe dollar
- Un identificateur ne peut pas appartenir à la liste des mots réservés en Java
- Exemple `int nombre; long _x1; String $test;`



# Les types primitifs

- boolean : Valeur logique true or false
- byte: octet signé (8 bites): -128 à 127
- short: entier court signé (16 bits)
- char: caractère unicode (16 bits)
- int: entier signé (32 bits)
- long: entier long (64 bits)
- float: virgule flottante simple précision
- double: virgule flottante double précision

# Affectation

- = Exemple:  $a=10$
- $\text{opérateur} =$  Exemple:  $a \text{ opérateur } = 10$   
équivalent  $a = a \text{ opérateur } 10$



# Opérateurs de comparaison

- Opérateurs arithmétiques (+, \*, /, -, %)
- Opérateurs d'affectation ou d'assignement (=, +=, -=)
- Opérateurs de comparaison (<, >, <=, >=, ==, !=)
- Opérateurs bit à bit: (&, ^, |)
- Opérateurs logiques: ( &&, ||, !)
- Opérateurs d'incrémentation et de décrémentation (++ , --)
- ?: Opérateur ternaire condition ? b:c
  - Renvoie b si condition est vraie et c sinon

# Priorité des opérateurs

- Parenthèses ()
- opérateur d'incrément: ++
- opérateur de décrémentation: --
- Opérateurs arithmétiques: \*,/,%
- Opérateurs de comparaison ou relationnels: <,>,<=,>=,==,!=
- Opérateurs logiques bit à bit: ^, &, |
- Opérateurs logiques booléens: &&, ||, !
- Opérateurs d'assignement ou d'affectation composée (+=, -=)

# Structures de contrôle

## ■ Les boucles

- `for (initialisation;condition; modification){ instr;}`
- `while (boolean) { instruction; }`
- `do {instruction; } while (boolean)`

## ■ Les branchements conditionnels

- `if (boolean) { instruction }`
- `else if (boolean) { instruction;}`
- `else{instruction;}`
- `switch (expression){`
- `case C1: instruction; break;`
- `case C2: instruction; break;`
- `default: instruction;}`

# Classe en JAVA

## Etudiant

-INE: String  
-Prénom: String  
-Nom: String

```
public class Etudiant  
{  
    private String Ine;  
    private String Prénom;  
    private String Nom;  
}
```

# Définition des méthodes

```
public class Etudiant {  
    //définition des attributs  
    private String Ine;  
    private String Prénom;  
    private String Nom;  
    //définition des méthodes  
    public String getIne () { return Ine; }  
    public void setIne (String vIne)  
    { Ine=vIne; }  
}
```

## Etudiant

-INE: String

-Prénom: String

-Nom: String

+getINE (): String

+setINE (String): void



# Portée d'une variable

- Portée de classe: la variable est visible dans toute la classe
- Portée de méthode: la variable est visible à l'intérieur de la méthode
- Portée de bloc: la variable est visible à l'intérieur du bloc



# Instanciation

- On appelle instance d'une classe, un objet avec un comportement et un état, tous deux définis par sa classe.
- L'instanciation est l'opération qui consiste à créer un objet à partir d'une classe En Java, le mot-clé new provoque une instanciation en faisant appel à un constructeur de la classe instanciée
- Un constructeur est une méthode qui a le même nom que la classe
- Un constructeur n'a pas de valeur de retour
- Plusieurs constructeurs peuvent exister dans une même classe (avec des arguments différents)
- Il faut au moins un constructeur dans une classe pour en instancier des objets
- L'appel au constructeur affecte une nouvelle adresse en mémoire pour le nouvel objet créé

# Constructeur sans paramètre

```
package ecole;

public class Etudiant {
    //définition des attributs
    private String Ine;
    private String Prénom;
    private String Nom;
    //définition du constructeur
    public Etudiant () {}
    //définition des méthodes
    public String getIne () { return Ine; }
    public void setIne (String vIne)
    {Ine=vIne;}
}
```

## Etudiant

-INE: String  
-Prénom: String  
-Nom: String

+getINE (): String  
+setINE (String): void



# Constructeur avec paramètres

```
package ecole;

public class Etudiant {
    //définition des attributs
    private String Ine;
    private String Prénom;
    private String Nom;
    //définition du constructeur sans paramètre
    public Etudiant () {}
    //définition du constructeur avec paramètre
    public Etudiant( String vIne, String vPrénom, String vNom)
    {
        Ine=vIne;
        Prénom=vPrénom;
        Nom=vNom;
    }
    //définition des accesseurs
}
```

## Etudiant

-INE: String -Prénom: String -Nom: String
+getINE (): String +setINE (String): void



# Les tableaux

- Ce sont des objets: ils sont dérivés la classe Object. Il est possible d'utiliser les méthodes héritées telles que equals () ou getClass().
- Le premier élément d'un tableau possède l'indice 0
- length détermine la taille d'un tableau

# Déclaration (Vecteur)

- Java permet de placer les crochets avant ou après le nom du tableau
- déclaration `int T1[]; //ou int []T1;`
- Allocation `T1= new int [20];`
- Déclaration et allocation
  - `int T1[]= new int [20];`

# Initialisation explicite

```
//initialisation explicite
```

```
int T1[]={4,5,1,2};
```

```
//Affichage verticale
```

```
for (int i=0; i<T1.length; i++)
```

```
System.out.println(T1[i]);
```

```
//Affichage horizontale
```

```
for (int i=0; i<T1.length; i++)
```

```
System.out.print(T1[i]+ " \t");
```

# Initialisation implicite

//initialisation implicite

```
import java.util.Scanner;
Scanner sc=new Scanner(System.in);
System.out.println("Taille du tableau ");
int n =sc.nextInt();
int T2[]= new int [n];
for (int i=0; i<T2.length; i++)
{
System.out.println("Elément position "+i);
T2[i]=sc.nextInt();
}
```

//Affichage horizontale

```
for (int i=0; i<T2.length; i++)
System.out.print(T2[i]+ " \t");
```

# Déclaration (matrice)

- Java permet de placer les crochets avant ou après le nom du tableau
- déclaration `int m1[] [];`
- Allocation `m1 = new int [10] [10];`
- Déclaration et allocation
  - `int m1[] [] = new int [10] [10];`

# Déclaration

- matrice dont les vecteurs n'ont pas le même nombre
- Déclaration et allocation
  - `int m1[] []= new int [3] [];`
  - `m1[0]=new int [4];`
  - `m1[1]=new int [5];`
  - `m1[2]=new int [2];`

# Initialisation explicite d'une matrice

```
int mat[][]={{4,5,2}, {9,10,11}, {12,13,15}};  
int mat[][]={{4,5}, {9,10,11}, {12,13,15,18}};
```



# Initialisation d'une matrice carrée

```
public static void main(String[] args) {  
    int mat [][];  
    Scanner in= new Scanner (System.in);  
    System.out.print("Donner le nombre de vecteurs ");  
    int n=in.nextInt();  
    mat=new int[n] [n]; //matrice carrée  
    for (int i=0; i<n; i++)  
    {  
        for (int j=0; j<n; j++)  
        {  
            System.out.print("Donner la valeur de l'éléement "+ i+ ", "+j+" ");  
            int val=in.nextInt();  
            mat[i][j]=val;  
        }  
    }  
}
```

# Initialisation d'une matrice non carrée

```
public static void main(String[] args) {  
    int mat [][];  
    Scanner in= new Scanner (System.in);  
    System.out.print("Donner le nombre de vecteurs ");  
    int n=in.nextInt();  
    mat=new int[n] [];  
    for (int i=0; i<n; i++)  
    {  
        System.out.print("Donner le nombre d'elements du vecteur numéro "+ i+ " ");  
        int m=in.nextInt();  
        mat[i]=new int [m];  
        for (int j=0; j<m; j++)  
        {  
            System.out.print("Donner la valeur de l'éléement "+ i+ ", "+j+ " ");  
            int val=in.nextInt();  
            mat[i][j]=val;  
        }  
    }  
}
```



# Chaines de caractères

- Les variables de Type String sont des objets
- Si une chaine de caractères est déclarée avec une constante: le compilateur génère un objet de type String avec le contenu spécifié
- Equals()

# Fonctions sur les chaines

- Addition (+)
- Remplacement
  - **String replace(char c, char x)** remplace le caractère c par le caractère x dans la chaine
- Caractère à une position :
  - char **charAt(j)** retourne le caractère à la position j
- Sous-chaine
  - String **substring(int i, int j)** retourne la sous-chaine de la position i à la position j-1
  - String **substring(int i)** retourne la sous-chaine de la position i à la position fi,

# Opérations

- Majuscule
  - **String toUpperCase()**
- Minuscule
  - **String toLowerCase()**
- Concaténation
  - **String concat (String)**
- Type en une chaîne
- **String valueOf (Type)**
  - Type: int, long, char, boolean, double, float, Object, char [], etc.)

# Palindrome

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    //String T="rotor";  
    Scanner entree=new Scanner(System.in);  
    String T;  
    System.out.print(" Saisissez une chaine de caractère: ");  
    T=entree.next();  
    T=T.toUpperCase();  
    int i=0;  
    int j=T.length()-1;  
    while(i<j && T.charAt(i)==T.charAt(j))  
    {i++; j--;}  
    if (T.charAt(i)!=T.charAt(j))  
        System.out.println(T + " n'est pas un palindrome");  
    else  
        System.out.println(T + " est un palindrome");  
} // Fin Main
```

# Fonction Palindrome

```
public static boolean estunPalindrome(String ch)
{
    int i=0;
    int j=ch.length()-1;
    while(i<j && ch.charAt(i)==ch.charAt(j))
    {
        i+=1;
        j-=1;
    }
    if (ch.charAt(i)!=ch.charAt(j))
        return false;
    else
        return true;
}
```



# Les énumérations

- Les énumération sont des types dont l'ensemble des valeurs n'est pas indéfini



# Définition d'une énumération en JAVA

```
package enumeration;
//définition d'une énumération
public enum EnumSexe {
    M, F;
}

//test de l'énumération
public class TestEnumSexe{
    public static void main (String [] args)
    {
        for (EnumSexe sexe:EnumSexe.values())
            System.out.println(sexe+"\t"+sexe.ordinal());
    }
}
```

# Tester une énumération

```
package Enumeration;
import java.util.Scanner;
//tester l'énumération E numSexe
public class EssaiEnumSexe {
public static void main(String[] args) {
    System.out.print("Donner le Sexe: ");
    String vsex=in.next();
    EnumSexe sexe=EnumSexe.valueOf(vsex);
    if (sexe==EnumSexe.M)
        System.out.print("Masculin");
    else if (sexe==EnumSexe.F)
        System.out.print("Feminin");
    else
        System.out.print("Erreur");
    }//fin main
} // fin classe
```

# Définition d'une énumération en JAVA

```
package enumeration;  
//définition de l'énumération EnumJour  
public enum EnumJour{  
    LUNDI, MARDI, MEcredi, JEUDI,  
    VENDREDI, SAMEDI, DIMANCHE;  
}
```

# Tester une énumération

```
package Enumeration;
import java.util.Scanner;
public class EssaiEnumJour {
    /**
     * @param args
     */
    //Affiche les objets de l'énumération EnumJour
    private static void afficherEnumJourSem()
    {
        for (EnumJour j:EnumJour.values())
            System.out.print(j+"("+ j.ordinal()+")"+"", " ");
        System.out.println();
    }
}
```

# Tester une énumération

```
private static boolean Test(String param)
{
    boolean trouve=false;
    EnumJour ji=EnumJour.valueOf(param);
    for (EnumJour j:EnumJour.values())
        if (j==ji)
        {
            trouve=true;
            break;
        }
    return trouve;
}
```

# Tester une énumération

```
public static void main(String[] args) {  
    // TODO Auto-generated method stub  
    System.out.println("Affichage de l'énumération EnumJour");  
    afficherEnumJourSem();  
    Scanner in=new Scanner(System.in);  
    System.out.print("Donner un Jour de la semaine: ");  
    String jour=in.next();  
    EnumJour j=EnumJour.valueOf(jour);  
    if (j==EnumJour.SAMEDI)  
        System.out.print("Fin de semaine");  
    switch(j)  
    {case SAMEDI:    System.out.print("Fin de semaine");  
      break;  
      case DIMANCHE: System.out.print("Jour de repos");  
      break;  
      default: System.out.print("Jour de travail");  
    }  
}
```

# Notions d'héritage

```
public class Personne
{
    private String nom;
    private String prenom;
    public Personne () {}
    public Personne (String vnom, String vprenom)
    { nom=vnom;
      prenom=vprenom; }
    public String getNom()
    { return nom;}
    public String getPrenom()
    {return prenom;}
    public void setNom( String vnom)
    {nom=vnom;}
    public void setPrenom( String vprenom)
    {prenom=vprenom;}
}
```

# Notions d'héritage

```
public class Etudiant extends Personne {  
    private String ine;  
    public Etudiant () {}  
    public Etudiant (String vine)  
    { ine=vine;  
    }  
    public String getine() { return ine; }  
    public void setine(String vine)  
    { ine=vine; }  
}
```



# Notions d'héritage

```
public class Personne
{
    protected String nom;
    protected String prenom;
    public Personne () {}
    public Personne (String vnom, String vprenom)
    {
        nom=vnom;  prenom=vprenom;
    }
    public String getNom()
    { return nom;}
    public String getPrenom()
    {return prenom;}
    public void setNom( String vnom)
    {nom=vnom;}
    public void setPrenom( String vprenom)
    {prenom=vprenom;}
}
```

# Notions d'héritage

```
public class Etudiant extends Personne {  
    private String ine;  
    public Etudiant () {}  
    public Etudiant (String vine, String vnom, String vprenom)  
    {  
        ine=vine;  
        nom=vnom;  
        prenom=vprenom;  
        //super (vnom, vprenom);  
    }  
    public String getine() {return ine;}  
    public void setine(String vine) {ine=vine;}  
    //public String getNom() {return nom;}  
}
```



# Classe abstraite

- Une classe abstraite contient au moins une méthode abstraite
  - Une méthode abstraite est une méthode dont on connaît juste la signature (pas de corps)

# Classe abstraite

```
public abstract class Personne
{
protected String nom;
protected String prenom;
    public Personne () {}
    public Personne (String vnom, String vprenom)
    {
nom=vnom; prenom=vprenom;
    }
    //méthodes abstraites
    public String getNom();
    public String getPrenom();
    //fin des méthodes abstraites
    public void setNom( String vnom)
    {nom=vnom;}
    public void setPrenom( String vprenom)
    {prenom=vprenom;}
}
```

# Classe abstraite

```
public class Etudiant extends Personne {  
    private String ine;  
    public Etudiant () {}  
    public Etudiant (String vine, String vnom, String vprenom)  
    {  
        ine=vine;  
        nom=vnom;  
        prenom=vprenom;  
        //super (vnom, vprenom);  
    }  
    public String getine() {return ine;}  
    public void setine(String vine) {ine=vine;}  
    //public String getNom() {return nom;}  
}
```



# Définition d'une interface

- Une interface ne contient que des méthode abstraites et des constantes non modifiables
  - Une méthode abstraite est une méthode dont on connaît juste la signature (pas de corps)

# Définition d'une interface

```
package MesInterfaces;

public interface IEtudiant {
    String nom=null; //nom est une constante non modifiable
    String prenom=null; //prenom est une constante non modifiable
    public String getNom();
    public String getPrenom();
    public void setPrenom (String vprenom);
    public void setNom(String vnom);
}
```

# Implémentation d'une interface

```
public class EtudiantImpl implements IEtudiant {
    protected String nom=IEtudiant.nom;
    protected String prenom=IEtudiant.prenom;
    public EtudiantImp(){}
        public String getNom() {return this.nom;}
        public void setNom( String vnom)
            {nom=vnom;}
        public String getPrenom() {return this.prenom;}
    public void setPrenom( String vprenom)
        {prenom=vprenom;}
    public static void main (String [ ] args)
    {
        EtudiantImpl E=new EtudiantImpl();
        E.setnom("Moussa");
        E.setPrenom("FALL");
        System.out.println(E.getNom());
        System.out.println(E.getPrenom());
    }
}
```



# Les exceptions

- En Java, les erreurs se produisent lors d'une exécution sous la forme d'**exceptions**
- Une exception :
  - est un objet, instance d'une classe d'exceptions (java.lang.Exception)
  - peut provoquer la sortie d'une méthode
  - correspond à un type d'erreur
  - contient des informations sur cette erreur

# Déclaration des exceptions

- Une méthode déclare, par le mot-clé `throws`, dans sa signature les classes d'exception qu'elle peut envoyer
- **Exemple de la méthode `substring()` de la classe `String`**
  - `public class String {`
  - `...`
  - `public String substring(int beginIndex, int endIndex)`
  - `throws IndexOutOfBoundsException {`
  - `...`
  - `}`
  - `...`
  - `}`



# Traitement des exceptions

- **Propagation**

- L'exception est renvoyée à la méthode ayant invoquée la méthode déclarant l'exception (mots-clés throws et throw)

- **Interception**

- L'exception est traitée dans la méthode appelant la méthode émettant l'exception (mots-clés try et catch)



# Traitement des exceptions

- **Propagation**

- L'exception est renvoyée à la méthode ayant invoquée la méthode déclarant l'exception (mots-clés throws et throw)

- **Interception**

- L'exception est traitée dans la méthode appelant la méthode émettant l'exception (mots-clés try et catch)

# Connexion JDBC

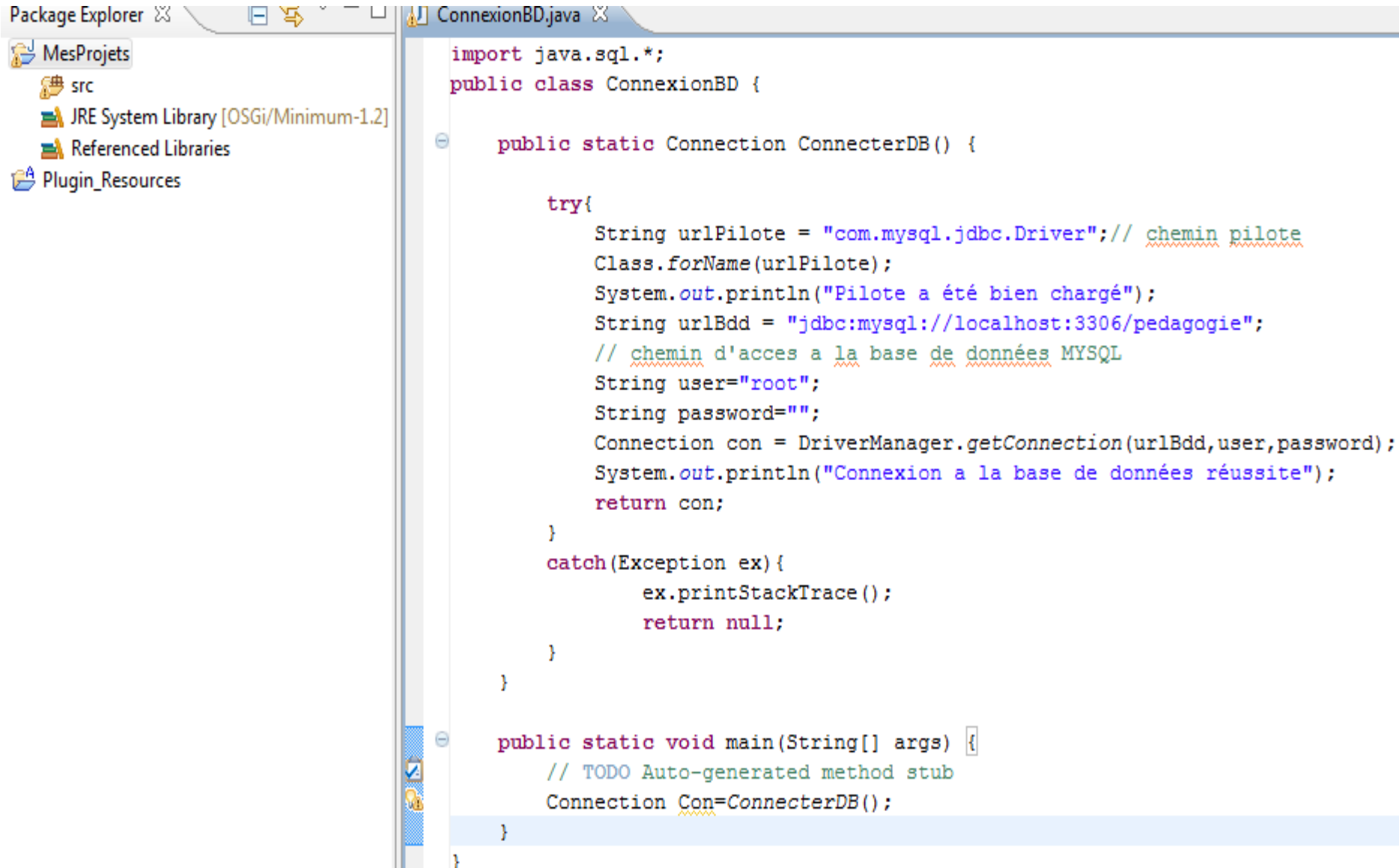
```
* @param args the command line arguments
*/
public static void main(String[] args) {
    // TODO code application logic here
}

public static Connection connecterDB(){
    try{
        Class.forName("com.mysql.jdbc.Driver");
        System.out.println("Driver ok!");
        String url="jdbc:mysql://localhost:3306/db_gstproduit";
        String user="root";
        String password="";
        Connection cnx=DriverManager.getConnection(url,user,password);
        System.out.println("Connexion bien établie");
        return cnx;
    }
    catch(Exception e){
        e.printStackTrace();
        return null;
    }
}
```

# Connexion JDBC Suite

```
public static void main(String[] args) {  
    // TODO code application logic here  
    Connection cnx = connectorDB();  
}
```

# JDBC



```
import java.sql.*;

public class ConnexionBD {

    public static Connection ConnectorDB() {

        try{
            String urlPilote = "com.mysql.jdbc.Driver";// chemin pilote
            Class.forName(urlPilote);
            System.out.println("Pilote a été bien chargé");
            String urlBdd = "jdbc:mysql://localhost:3306/pedagogie";
            // chemin d'accès à la base de données MYSQL
            String user="root";
            String password="";
            Connection con = DriverManager.getConnection(urlBdd,user,password);
            System.out.println("Connexion à la base de données réussite");
            return con;
        }
        catch(Exception ex){
            ex.printStackTrace();
            return null;
        }
    }

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Connection Con=ConnectorDB();
    }
}
```