

Programmation par Objets II

Georges Edouard KOUAMOU



Modifier avec WPS Office

Contenu du Cours

- Rappel
- Renforcer la Gestion des Exceptions
- Programmation par Interface et classes abstraites
 - Gestion de l'héritage multiple
 - Mieux structurer son code
- Collections et gestion des interactions entre les classes
- Gestion des entrées/sorties
 - Flot de caractères (stream)
 - Fichiers séquentiels et directs
- **Notions avancées**
 - Communication Java-SGBD : JDBC
 - Threads: Création – Synchronisation - Communications inter-threads
 - Java FX : concept et utilisation (un autre framework pour les GUI)



Rappel sur la POO

- Concept de base (fondamentaux) de la POO
 - Héritage
 - Classe
 - Objet
 - Interface
 - Polymorphisme
 - Encapsulation
- Classe
 - prototype/maquette/moule à partir duquel les objets sont créés
- Objet
 - Instance d'une classe
 - Entité logicielle qui a un état (la valeur des attributs) et un comportement (méthodes)



Rappel (cont ...)

- Héritage
 - Mécanisme de transmission des propriétés de la classe mère vers les classes filles
- Polymorphisme
 - Le fait d'écrire sous le même nom de méthodes plusieurs comportements de classes différentes
- Interface
 - Partie visible de l'extérieur d'un objet
 - Partie qui assure l'interaction entre un objet et son environnement (tous les objets externes)
- Encapsulation
 - Masquage de l'information, cacher les détails de la réalisation d'un objet



Rappel (cont ...)

- Package
 - Déclaration: *package* <nom du package>
 - Rôle: regroupement des classes entre elles, on verra qu'on peut aussi regrouper les interfaces et d'autres entités à l'aide des packages
- Utilisation
 - Import <nom du package>.<nom de la classe>
 - Ou bien import <nom du package>.* pour importer toutes les classes
- Gestion des entrées au clavier
 - Java.util.Scanner
- Types de données
 - Primitifs: int, short, long, float, double, char, boolean
 - Complexes: String, tableaux
- Déclaration d'un tableau
 - Type_de_données[] nom_du_tableau;



Rappel (cont ...)

- Les opérateurs
 - Arithmétiques: +, -, *, /, %
 - Logiques: &&, ||, !
 - Comparaison: >=, <=, <, >, ==, !=
 - Opérateur binaires: &, |
 - Incrément: ++, +=,
 - Décrément: --, -=
- Instanciation d'un objet: new
- Typologie des méthodes
 - Constructeurs: pas de type retour, porte le nom de la classe
 - Mutateurs: getters et les setters => manipulation des attributs (lecture écriture)



Exercice

- Ecrire un programme Java pour trier dans l'ordre croissant un tableau de taille n (saisi par l'utilisateur). Les valeurs des cellules (entiers) également sont saisies par l'utilisateur
- On effectuera le tri rapide et le tri par sélection, tri insertion
- Indications
 - Créer un projet : **Licence2**
 - Créer un package: **tp1**
 - Créer la classe: **Tri** dans le package
 - Ajouter un attribut tableau des entiers: *tab*
 - Ajouter un autre attribut de type entier : *taille* qui indique le nombre d'objets stockés dans *tab*
 - Ecrire les 2 méthodes: *public void triSelection()*, *public void triInsertion()*
 - Ecrire le constructeur qui initialise *tab* et *taille*: **Tri(int[] t, int n)**
 - Ecrire la classe Test qui crée un tableau et l'utilise pour instancier un objet Tri



Exceptions

Gestion des erreurs



Modifier avec WPS Office

Cours de POO 2

Exceptions

- Définition
 - Une exception est une erreur se produisant dans un programme qui conduit le plus souvent à l'arrêt de celui-ci.
 - Une exception non *capturée* génère un gros message affiché en rouge dans la console d'Eclipse/Netbeans.
- Gérer les exceptions s'appelle aussi « la capture d'exception ».
- Principe:
 - repérer un morceau de code (par exemple, une division par zéro) qui pourrait générer une exception
 - capturer l'exception correspondante
 - et enfin la traiter (i.e afficher un message personnalisé et continuer l'exécution.



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

- Ecrire une méthode main avec le code ci-après

```
int j = 20, i = 0;  
System.out.println(j/i);  
System.out.println("coucou toi !");
```
- Exécuter le code et dire ce que vous constatez dans la console
 - 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*
- La classe **Exception** répertorie les différents cas d'erreur.
- Le bloc **try ... catch** permet de capturer les exceptions



Capter une exception

- Modifier le code précédent pour afficher un message personnalisé lors d'une division par 0

```
public static void main(String[] args) {  
    int j = 20, i = 0;  
    try {  
        System.out.println(j/i);  
    } catch (ArithmeticException e) {  
        System.out.println("Division par zéro !");  
    }  
    System.out.println("coucou toi !");  
}
```

- Le paramètre de la clause **catch** permet de connaître le type d'exception qui doit être capturée
- L'objet **e** peut servir à préciser notre message grâce à l'appel de la méthode `getMessage()`

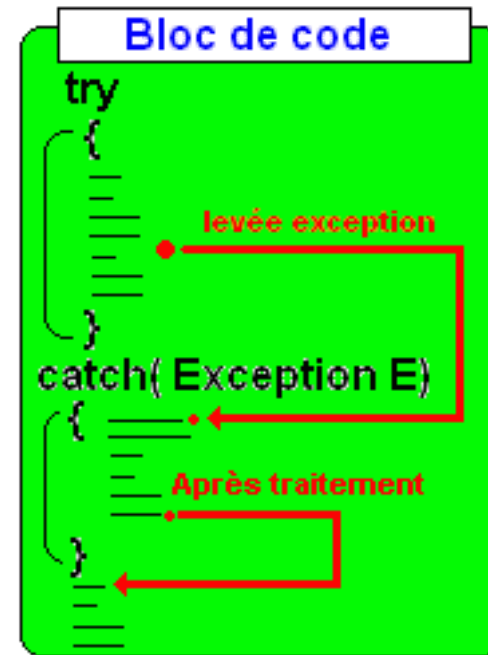
```
System.out.println("Division par zéro !" + e.getMessage());
```

En savoir plus!

Exercices

- Lister les méthodes de la classe Exception
- Enumérer les types d'exception courants
 - Les sous classes ou classes dérivées de la classe Exception
- A quoi sert la clause **finally**?

Schéma de fonctionnement de l'interception



Exceptions personnalisées

- Procédure

1. Créer une classe héritant de la classe Exception (par convention, les exceptions ont un nom se terminant par « Exception »)
2. Renvoyer l'exception levée à la classe ainsi créer
3. Gérer celle-ci

- Les mots clés utiles

- **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** 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 dérivé)



Créer une classe Exception

Une exception
personnalisée

Hérite de la classe
Exception

```
1 class NombreHabitantException extends Exception{
2     public NombreHabitantException(){
3         System.out.println("Vous essayez d'instancier une classe Ville avec un nombre d'habitants
   négatif !");
4     }
5 }
```



Modifier avec WPS Office

Cours de POO 2

Utilisation de l'exception personnalisée

```
1 public Ville(String pNom, int pNbre, String pPays)
2     throws NombreHabitantException
3     {
4         if(pNbre < 0)
5             throw new NombreHabitantException();
6         else
7         {
8             nbreInstance++;
9             nbreInstanceBis++;
10
11             nomVille = pNom;
12             nomPays = pPays;
13             nbreHabitant = pNbre;
14             this.setCategorie();
15         }
16     }
```

indique que si une erreur est capturée, celle-ci sera traitée en tant qu'objet de la classe **NombreHabitantException**

Désormais, entourer l'appel à ce constructeur avec le bloc try {...} catch



Modifier avec WPS Office

Gestion de plusieurs exceptions

- valable pour toutes sortes d'exceptions, qu'elles soient personnalisées ou inhérentes à Java
- lever une exception si le nom de la ville fait moins de 3 caractères
 - Répéter les étapes précédentes

```
1 public class NomVilleException extends Exception {  
2     public NomVilleException(String message){  
3         super(message);  
4     }  
5 }
```

Utilisation du mot super(). Avec cette redéfinition, nous pourrons afficher notre message d'erreur en utilisant la méthode


```

1 public Ville(String pNom, int pNbre, String pPays) throws NombreHabitantException,
   NomVilleException
2 {
3     if(pNbre < 0)
4         throw new NombreHabitantException(pNbre);
5
6     if(pNom.length() < 3)
7         throw new NomVilleException("le nom de la ville est inférieur à 3 caractères ! nom = "
   + pNom);
8     else
9     {
10         nbreInstance++;
11         nbreInstanceBis++;
12
13         nomVille = pNom;
14         nomPays = pPays;
15         nbreHabitant = pNbre;
16         this.setCategorie();
17     }
18 }

```

Utilisation de throws,
séparation des type
d'exception par une virgule



Gérer 2 exceptions dans une instructions

- Si vous mettez un nom de ville de moins de 3 caractères et un nombre d'habitants négatif, c'est l'exception du nombre d'habitants qui sera levée en premier, et pour cause : il s'agit de la première condition dans notre constructeur.
- Lorsque plusieurs exceptions sont gérées par une portion de code, pensez bien à mettre les blocs **catch** dans un ordre pertinent

```
3 try {
4     v = new Ville("Re", 12000, "France");
5 }
6
7 //Gestion de l'exception sur le nombre d'habitants
8 catch (NombreHabitantException e) {
9     e.printStackTrace();
10 }
11
12 //Gestion de l'exception sur le nom de la ville
13 catch (NomVilleException e2){
14     System.out.println(e2.getMessage());
15 }
16 finally{
17     if(v == null)
18         v = new Ville();
19 }
```



Modifier avec WPS Office

Multi-catch

- Depuis la version 7 de java
- Catcher plusieurs exceptions dans l'instruction **catch**
- Utiliser l'opérateur |
- permet d'avoir un code plus compact

```
1 public static void main(String[] args){
2     Ville v = null;
3     try {
4         v = new Ville("Re", 12000, "France");
5     }
6     //Gestion de plusieurs exceptions différentes
7     catch (NombreHabitantException | NomVilleException e2){
8         System.out.println(e2.getMessage());
9     }
10    finally{
11        if(v == null)
12            v = new Ville();
13    }
14    System.out.println(v.toString());
15 }
```



Exercice: une pile avec exception

- Une pile est une structure de données dont les opérations se passent à une seule extrémité.
- Structure de l'objet
 - Utiliser un tableau de taille MAX pour stocker les éléments (de type entier) de la pile
 - Un index sommet pour indiquer le niveau de remplissage du tableau
- Les opérations possibles
 - Constructeur de la pile (init)
 - Empiler(int i): met i au sommet de la pile
 - Depiler(): retire l'élément au sommet de la pile
 - estVide: Vrai si la pile ne contient aucun élément
 - estPleine: Vrai si le tableau n'a plus de place
- 1. Utiliser la classe **Exception** pour gérer les situations d'erreur
- 2. Créer une classe **PileException** pour personnaliser la gestion des erreurs
- Délai: 1 jour



Classes abstraites et Interfaces

Améliorer la structure de vos programmes



Modifier avec WPS Office

Cours de POO 2

Classes abstraites et interfaces

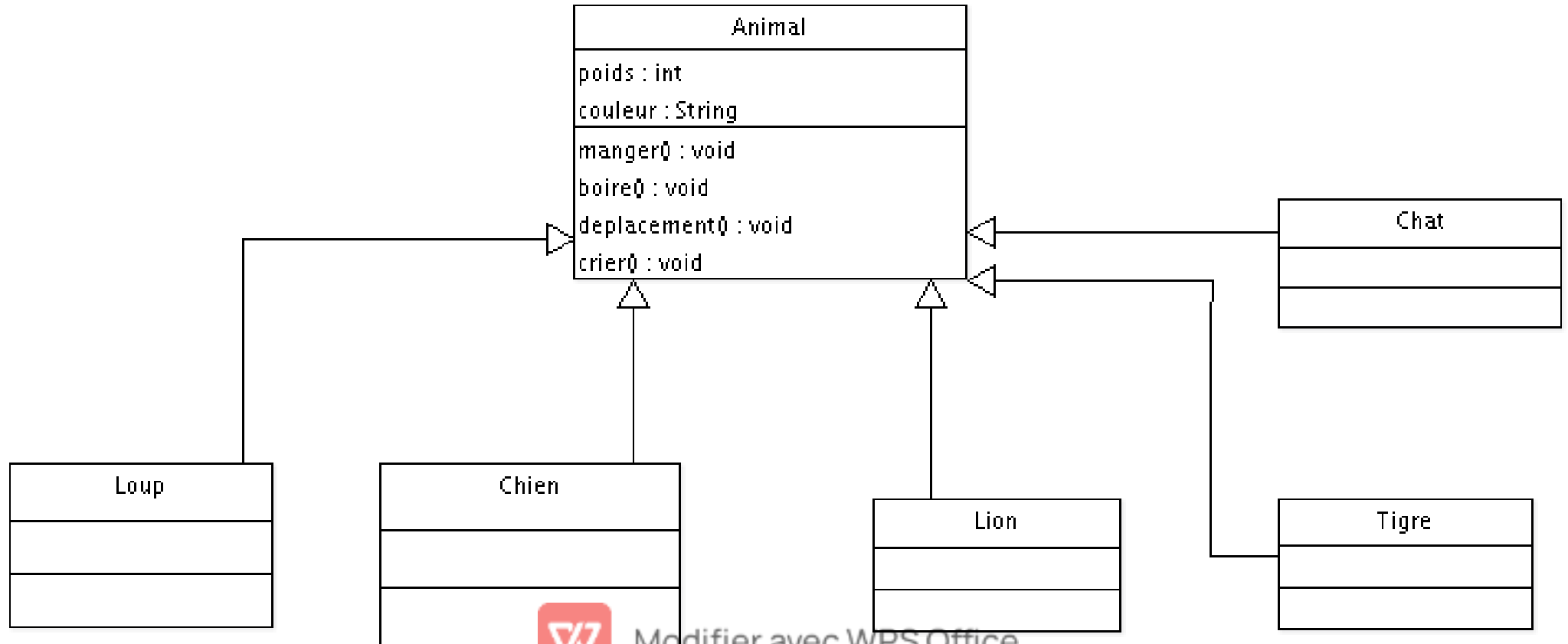
- Une classe **abstraite** est quasiment identique à une classe normale
- les **interfaces** permettent de créer un nouveau supertype
- Ne sont pas instanciables (on ne peut créer des objets de cette classe)
- Aide à bien structurer vos programmes
 - obtenir une structure assez souple pour pallier les problèmes de programmation les plus courants
- Si A est une classe abstraite alors le code suivant signalera une erreur

```
public class Test{  
    public static void main(String[] args){  
        A obj = new A(); //Erreur de compilation !  
    }  
}
```



Modifier avec WPS Office

Exemple: illustration des classes abstraites



Déclaration

- Une classe abstraite doit être déclarée avec le mot clé **abstract**
- Une classe abstraite permet de définir des méthodes abstraites qui présentent une particularité : elle n'ont pas de corps
- on dit « méthode abstraite » : difficile de voir ce que cette méthode sait faire

```
abstract class Animal{  
    abstract void manger(); //Une méthode abstraite  
}
```

- **NB. une méthode abstraite ne peut exister que dans une classe abstraite**

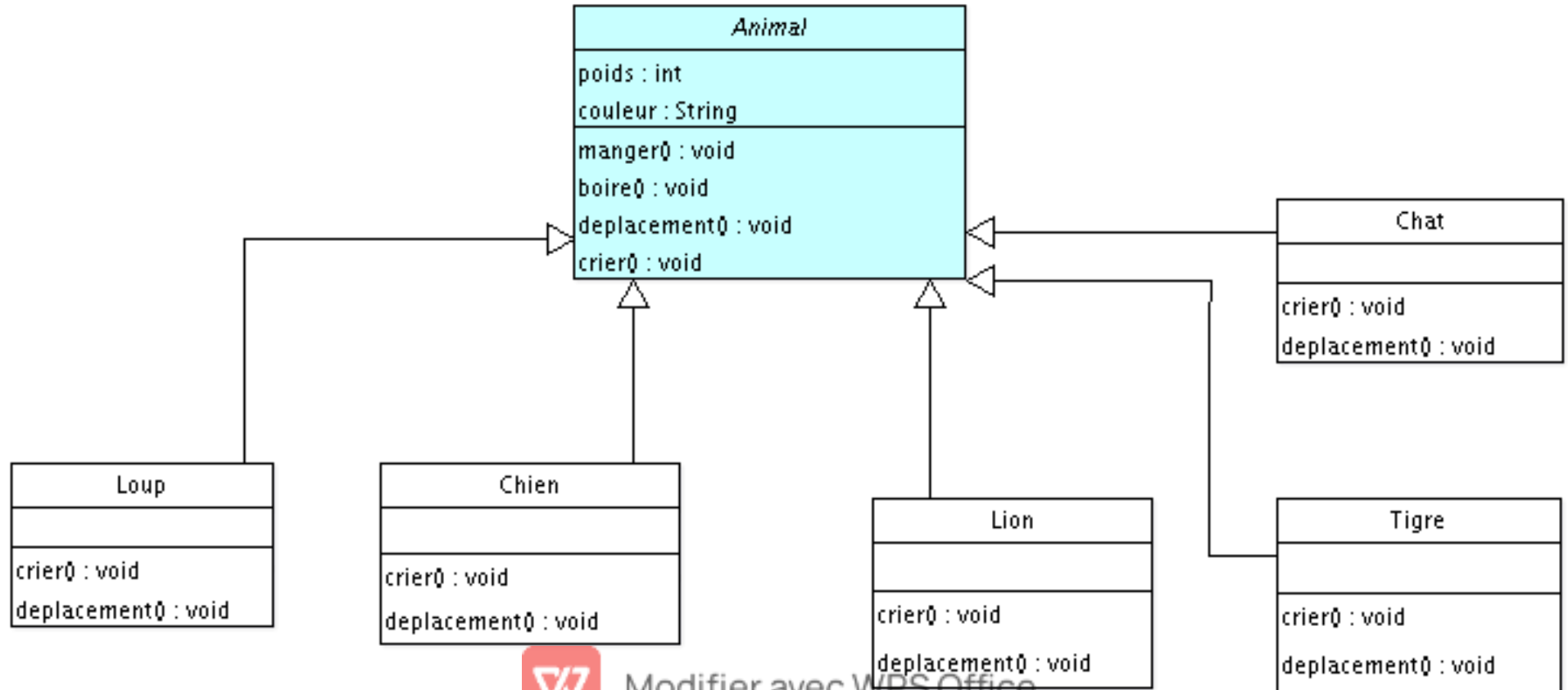


Détail de l'exemple

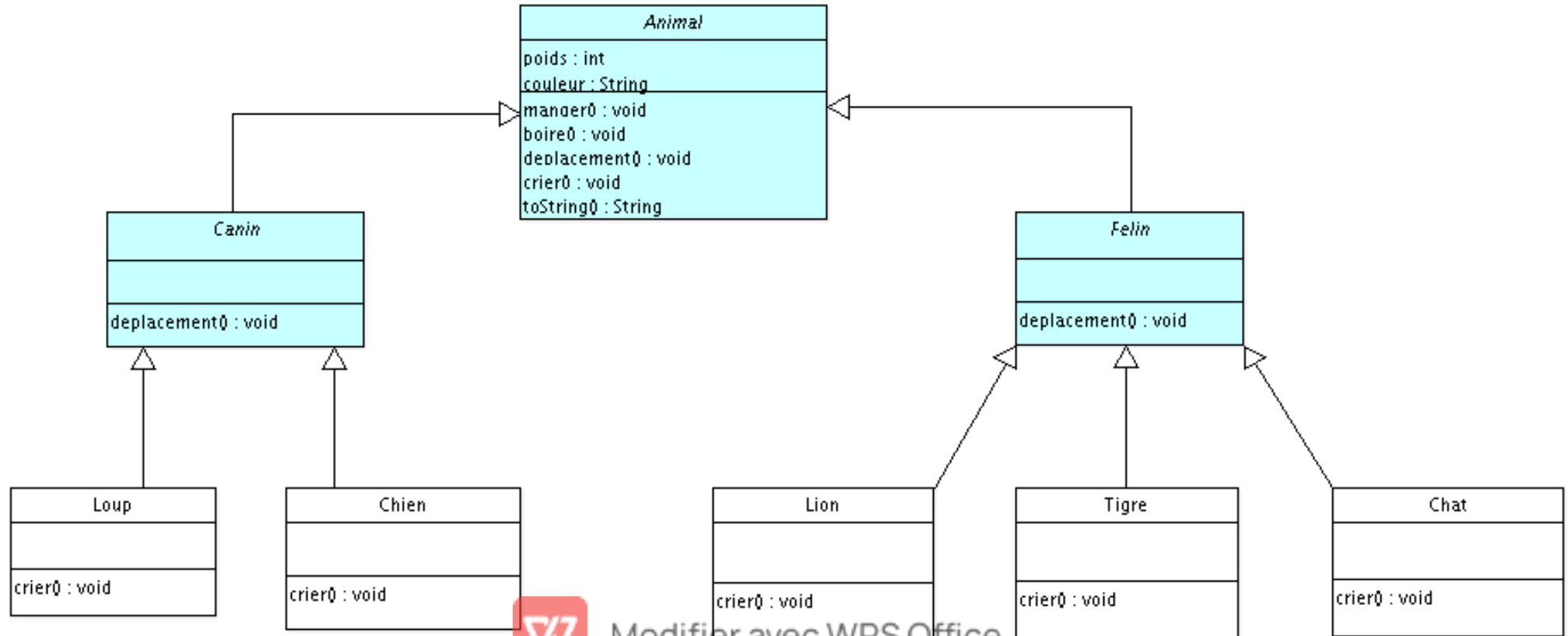
- Nos objets seront probablement tous de couleur et de poids différents.
 - Nos classes auront donc le droit de modifier ceux-ci.
- Nous partons du principe que tous nos animaux mangent de la viande.
 - La méthode **manger()** sera donc définie dans la classe Animal
- Idem pour **boire()**. Ils boiront tous de l'eau
- Ils ne crieront pas et ne se déplaceront pas de la même manière.
 - Nous emploierons donc des méthodes polymorphes et déclarerons les méthodes **déplacement()** et **crier()** abstraites dans la classe Animal



Hiérarchie des classes



Autre hiérarchie



```
1 abstract class Animal {
2
3     protected String couleur;
4     protected int poids;
5
6     protected void manger(){
7         System.out.println("Je mange de la viande.");
8     }
9
10    protected void boire(){
11        System.out.println("Je bois de l'eau !");
12    }
13
14    abstract void deplacement();
15
16    abstract void crier();
17
18    public String toString(){
19        String str = "Je suis un objet de la " + this.getClass() + ", je suis " +
20        this.couleur + ", je pèse " + this.poids;
21        return str;
22    }
23 }
```

Qve fait la methode
getClass



Modifier avec WPS Office

Felin.java

```
1 public abstract class Felin extends Animal {  
2     void deplacement() {  
3         System.out.println("Je me déplace seul !");  
4     }  
5 }
```

Canin.java

```
1 public abstract class Canin extends Animal {  
2     void deplacement() {  
3         System.out.println("Je me déplace en meute !");  
4     }  
5 }
```



Modifier avec WPS Office

```

1 public class Chien extends Canin {
2
3     public Chien(){
4
5     }
6
7     public Chien(String couleur, int poids){
8         this.couleur = couleur;
9         this.poids = poids;
10    }
11
12    void crier() {
13        System.out.println("J'aboie sans raison !");
14    }
15 }

```

```

1 public class Loup extends Canin {
2
3     public Loup(){
4         // // //
5     }
6
7     public Loup(String couleur, int poids){
8         this.couleur = couleur;
9         this.poids = poids;
10    }
11
12    void crier() {
13        System.out.println("Je hurle à la Lune en faisant ouhouh !");
14    }
15 }

```



Modifié avec WPS Office

```
1 public class Lion extends Felin {
2
3     public Lion(){
4
5     }
6
7     public Lion(String couleur, int poids){
8         this.couleur = couleur;
9         this.poids = poids;
10    }
11
12    void crier() {
13        System.out.println("Je rugis dans la savane !");
14    }
15 }
```

Classe Tigre.java

Classe Chat.java

```
void crier() {
    System.out.println("Je miaule sur les toits !");
}
```

```
void crier() {
    System.out.println("Je grogne très fort !");
}
```



Modifier avec WPS Office

```
1 public class Test {  
2     public static void main(String[] args) {  
3         Loup l = new Loup("Gris bleuté", 20);  
4         l.boire();  
5         l.manger();  
6         l.deplacement();  
7         l.crier();  
8         System.out.println(l.toString());  
9     }  
10 }
```

Remplacez le type de référence (ici, Loup) par Animal, essayez avec des objets Chien, etc. Vous verrez comment tout fonctionne



Modifier avec WPS Office

Interfaces

- Une interface définit:
 - Un type
 - Des méthodes sans leur code (méthodes abstraites)
- Une interface ne peut pas être instanciée
 - On peut définir des variables de type d'une interface
- Elle est destinée à être « **implémentée** » par des classes
 - L'implémentation se fait dans les classes
 - À qui elle donnera son type
 - Qui fourniront des définitions pour les méthodes déclarées (code)
- L'héritage est possible entre interfaces
- **Structure**: une interface contient
 - Les constantes
 - La signature des méthodes (nom, paramètres et type retour)
 - Pas d'implémentation



Déclaration et utilisation

Mot clé
interface

```
1 public interface I{
2     public void A();
3     public String B();
4 }
```

Une classe qui utilise l'interface
I. Mot clé **implements**

```
1 public class X implements I{
2     public void A(){
3         //...
4     }
5     public String B(){
6         //...
7     }
8 }
```

On dit qu'une
classe implémente
une interface
*La classe X
implémente
l'interface I .*

Une classe peut implémenter
plusieurs interfaces

Utiliser pour l'héritage multiple

```
1 public interface I2{
2     public void C();
3     public String D();
4 }
```

```
1 public class X implements I, I2{
2     public void A(){
3         //...
4     }
5     public String B(){
6         //...
7     }
8     public void C(){
9         //...
10    }
11    public String D(){
12        //...
13    }
14 }
```

Exemple

- Ajouter des responsabilités à la classe Chien
 - sans modifier la classe existante
 - Permettre au chien d'être amical

```
1 public interface Rintintin{
2     public void faireCalin();
3     public void faireLechouille();
4     public void faireLeBeau();
5
6 }
```

```
1 public class Chien extends Canin implements Rintintin {
2
3     public Chien(){
4
5     }
6     public Chien(String couleur, int poids){
7         this.couleur = couleur;
8         this.poids = poids;
9     }
10
11     void crier() {
12         System.out.println("J'aboie sans raison !");
13     }
14
15     public void faireCalin() {
16         System.out.println("Je te fais un GROS CÂLIN");
17     }
18
19     public void faireLeBeau() {
20         System.out.println("Je fais le beau !");
21     }
22
23     public void faireLechouille() {
24         System.out.println("Je fais de grosses léchouilles...");
25     }
26 }
```



Classe Test

- Tester le polymorphisme de notre implémentation
- Objectif: définir deux superclasses afin de les utiliser comme supertypes et de profiter pleinement du **polymorphisme**
 - La méthode effectivement appelée sur les objets sera la plus précise possible, en fonction du type réel de cet objet

```
2
3 public static void main(String[] args) {
4
5     //Les méthodes d'un chien
6     Chien c = new Chien("Gris bleuté", 20);
7     c.boire();
8     c.manger();
9     c.deplacement();
10    c.crier();
11    System.out.println(c.toString());
12
13    System.out.println("-----");
14    //Les méthodes de l'interface
15    c.faireCalin();
16    c.faireLeBeau();
17    c.faireLechouille();
18
19    System.out.println("-----");
20    //Utilisons le polymorphisme de notre interface
21    Rintintin r = new Chien();
22    r.faireLeBeau();
23    r.faireCalin();
24    r.faireLechouille();
25 }
```



Modifier avec WPS Office

A retenir: les membres d'une interface

- Contiennent des déclarations de méthodes publiques
 - Toutes les méthodes sont **abstract** ou **public abstract**
 - même si non spécifié
- Peuvent définir des champs publics et constants
 - Tous les champs sont **public final static**
 - Le compilateur ajoute les mot clés
- Il n'est **pas possible d'instancier** une interface
 - On ne peut que déclarer des variables avec leur type
 - Ces variables pourront recevoir des références à des objets qui sont des instances d'une classe qui implémente l'interface

```
public interface Surfaceable {  
    double surface(); // equivaut à  
    public abstract double surface();  
}
```

```
public interface I {  
    int field = 10; // equivaut à  
    public final static int field = 10;  
}
```



A retenir: les classes abstraites

- Dans une **interface**, tout doit être abstrait (méthodes)
 - Les méthodes doivent toutes être abstraites
 - Elles ne peuvent être que publiques
- Dans une **classe**, tout doit être concret
 - Les méthodes doivent être définies (leur code)
 - Elles peuvent avoir des modificateurs de visibilité différents
- Une **classe abstraite** permet de créer un type à mi-chemin
 - Il s'agit d'une classe qui peut avoir des **méthodes abstraites**
 - Elle est considérée comme partiellement implantée, donc **non instanciable**
 - Elle peut éventuellement n'avoir aucune méthode abstraite
 - Le mot-clé `abstract` fait qu'elle n'est pas instanciable



Exercice 1. Classes abstraites

- Un parc auto se compose des voitures et des camions qui ont des caractéristiques communes regroupées dans la classe Véhicule.
 - Chaque véhicule est caractérisé par son matricule, l'année de son modèle, son prix.
 - Lors de la création d'un véhicule, son matricule est incrémenté selon le nombre de véhicules créés.
 - Tous les attributs de la classe véhicule sont supposés privés. ce qui oblige la création des accesseurs (get...) et des mutateurs (set...) ou les propriétés.
 - La classe Véhicule possède également deux méthodes abstraites démarrer() et accélérer() qui seront définies dans les classes dérivées et qui afficheront des messages personnalisés.
 - La méthode ToString() de la classe Véhicule retourne une chaîne de caractères qui contient les valeurs du matricule, de l'année du modèle et du prix.
 - Les classes Voiture et Camion étendent la classe Véhicule en définissant concrètement les méthodes accélérer() et démarrer() en affichant des messages personnalisés.
- Travail à faire:
 - Créer la classe abstraite Véhicule.
 - Créer les classes Camion et Voiture.
 - Créer une classe Test qui permet de tester la classe Voiture et la classe Camion



Collections

Structures de Données Dynamiques



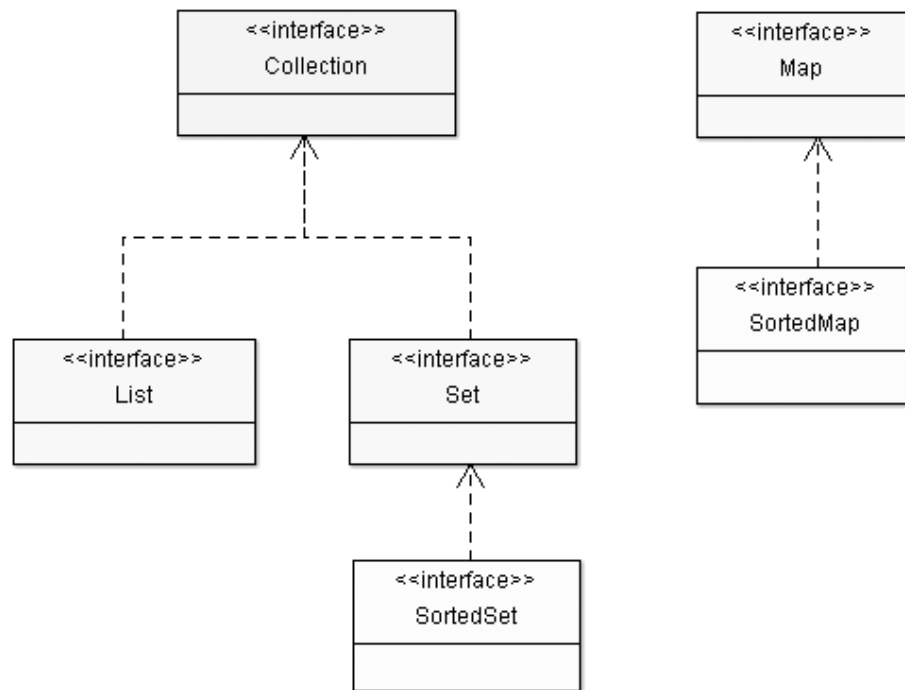
Modifier avec WPS Office

Cours de POO 2

Présentation

- Structures de données dynamiques
 - impossible de dépasser leur capacité
- Les principaux sont dans le package **java.util**
 - List (Interface)
 - Vector (*très peu utiliser actuellement*)
 - ArrayList
 - LinkedList
 - Set (Interface)
 - HashSet
 - LinkedHashSet
 - TreeSet

Hiérarchie des interfaces



Particularités

- Les collections sont des objets qui permettent de gérer des ensembles d'objets
- List: collection d'éléments ordonnés et accepte les doublons
 - La liste étant ordonnée, un élément peut être accédé à partir de son index
 - d'interagir avec un élément de la collection en utilisant sa position
 - d'insérer des éléments null
- Set : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons
- Map : collection sous la forme d'une association de paires clé/valeur



Opérations usuelles sur les collections

- Indice: Revoir le cours de structure de données
- Insérer(Element e)
- Supprimer(int position)
- Modifier(Element e, int position)
- Trier()
- Rechercher(Element e): int
- Taille(): int nombre d'éléments contenus dans la collection



Opérations de ArrayList

- Créer une liste
 - **ArrayList()** → crée un ArrayList vide de capacité initiale de 10.
 - **ArrayList(int taille)** → crée une liste vide de capacité taille.
 - **ArrayList(Collection c)** → Crée un ArrayList contenant les éléments de la collection c
- Taille d'une liste
 - Utiliser la méthode `size()`
- Tester si une liste est vide
 - La méthode `isEmpty()`
- Autre méthode
 - `add()` permet d'ajouter un élément ;
 - `get(int index)` retourne l'élément à l'indice demandé ;
 - `remove(int index)` efface l'entrée à l'indice demandé ;
 - `isEmpty()` renvoie « vrai » si l'objet est vide ;
 - `removeAll()` efface tout le contenu de l'objet ;
 - `contains(Object element)` retourne « vrai » si l'élément passé en paramètre est dans l'ArrayList.



Exemple

- Ecrire une classe Pile
 - de taille illimitée
 - Capable de contenir tout type d'objets
- Les opérations
 - CreerPile(): Pile retourne une pile vide => constructeur
 - Empiler(Objet o): ajoute l'objet o au sommet de la pile
 - Depiler(): Object : enlève l'objet au sommet de la pile et le retourne à l'appelant
 - estVide(): boolean: teste si la pile contient des éléments ou pas
 - Sommet(): Object: retourne l'élément au sommet de la pile sans le détruire
- On exploitera les méthodes sur les ArrayList pour implémenter les opérations sus-citées
- N'oublier pas de gérer les exceptions
- Écrire une classe Test pour valider votre implémentation



Exemple 2: Pile d'entier

- Comment peut-on faire évoluer le code précédent pour restreindre uniquement aux entiers?
 - Les génériques
- Ecrire une classe PileEntier



Exercice: classes abstraites & Collections

- Enoncé

- Un parc auto se compose des voitures et des camions qui ont des caractéristiques communes regroupées dans la classe Véhicule. Chaque véhicule est caractérisé par son matricule, l'année de son modèle, son prix.
- Lors de la création d'un véhicule, son matricule est incrémenté selon le nombre de véhicules créés.
- Tous les attributs de la classe véhicule sont supposés privés. ce qui oblige la création des accesseurs (get...) et des mutateurs (set....) ou les propriétés.
- La classe Véhicule possède également deux méthodes abstraites démarrer() et accélérer() qui seront définies dans les classes dérivées et qui afficheront des messages personnalisés.
- La méthode ToString() de la classe Véhicule retourne une chaîne de caractères qui contient les valeurs du matricule, de l'année du modèle et du prix.
- Les classes Voiture et Camion étendent la classe Véhicule en définissant concrètement les méthodes accélérer() et démarrer() en affichant des messages personnalisés.

- Travail à faire:

- Créer la classe abstraite **Véhicule**.
- Créer les classes **Camion** et **Voiture**.
- Créer une classe **Test** qui permet de tester la classe Voiture et la classe Camion

- Indication: utiliser une Collection ArrayList pour contenir les voitures et les camions. Utiliser la taille de la collection pour calculer le matricule



Flux d'entrée/sortie

Utilisation des Stream



Modifier avec WPS Office

Cours de POO 2

Présentation

- Stream = flux
 - consiste en un échange de données entre le programme et une autre source, par exemple la mémoire, un fichier
 - médiateur entre la source des données et sa destination
- Schéma des opérations d'entrée/sortie
 - ouverture, lecture/écriture, fermeture du flux
- 2 Catégories de flux
 - Flux d'entrée (in) => pour la lecture
 - Exemple: System.in
 - Flux de sortie (out) => pour l'écriture
 - System.out



L'objet File

- L'objet **File** permet de manipuler un fichier en tant que entité du (système de fichier du) système d'exploitation
- Les méthodes usuelles
 - `getAbsolutePath()` : chemin absolue d'accès au fichier
 - `getName()` : nom du fichier
 - `exists()` : teste l'existence du fichier
 - `isDirectory()` : est ce un dossier/repertoire ou non
 - `isFile()` : est ce un fichier
 - `delete()` : effacer le fichier
 - `mkdir()` : créer un dossier/repertoire



Exemple

```
•import java.io.File;//Package à importer afin d'utiliser l'objet File
•public class TestStream {
•public static void main(String[] args) {
•    File f = new File("test.txt");//Création de l'objet File
•    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 !
•    }
•    }
•}
```



Modifier avec WPS Office

Écriture et lecture dans un flux

- Lire dans un fichier: utiliser l'objet `FileInputStream`
- écrire dans un fichier: utiliser l'objet `FileOutputStream`
- Ces classes héritent des classes abstraites `InputStream` et `OutputStream`
 - `package java.io`.

 Pour que l'objet `FileInputStream` fonctionne, le fichier doit exister ! Sinon l'exception `FileNotFoundException` est levée. Par contre, si vous ouvrez un flux en écriture (`FileOutputStream`) vers un fichier inexistant, celui-ci sera créé automatiquement !



FileInputStream

Méthode	Description
<code>int available()</code>	Utiliser pour estimer le nombre d'octets qui peuvent être lus du flux d'entrée
<code>int read()</code>	Utiliser pour lire un octet. Retourne -1 si la fin du fichier est atteinte
<code>int read(byte[] b)</code>	Utiliser pour lire <code>b.length</code> octet de données du flux d'entrée. Retourne le nombre d'octets lus ou -1 si on est à la fin du fichier
<code>int read(byte[] b, int off, int nbre)</code>	Utiliser pour lire <code>nbre</code> octets de données à partir de la position <code>off</code> .
<code>FileDescriptor getFD()</code>	Retourne le descripteur de fichier qui est un objet.
<code>protected void finalize()</code>	Assure que la method <code>close()</code> est appelé lorsqu'il n'y a plus de reference qui pointe sur le flux.
<code>void close()</code>	Ferme le flux.



Exemple... lecture d'un caractère

```
1. import java.io.FileInputStream;
2. public class DataStreamExample {
3.     public static void main(String args[]){
4.         try{
5.             FileInputStream fin=new FileInputStream("D:\\testout.txt");
6.             int i=fin.read();
7.             System.out.print((char)i);
8.             fin.close();
9.         }catch(Exception e){
10.             System.out.println(e.getMessage());
11.         }
12.     }
13. }
```

Chemin absolue ou
relatif d'accès au fichier

Que fait ce
programme?
Modifier le afin qu'il
recupère le contenu
entier d'un fichier



Modifier avec WPS Office

Cours de POO 2

Exemple... lecture du contenu entier

```
1. import java.io.FileInputStream;
2. public class DataStreamExample {
3.     public static void main(String args[]) {
4.         try {
5.             FileInputStream fin=new FileInputStream("D:\\testout.txt");
6.             int i=0;
7.             while((i=fin.read())!=-1) {
8.                 System.out.print((char)i);
9.             }
10.            fin.close();
11.        }catch(Exception e){System.out.println(e);}
12.    }
13.}
```



BufferedInputStream

- Pour accélérer la lecture des fichiers en réduisant les accès disques
- Permet de lire sur un flux de données en utilisant une **zone mémoire tampon** *buffer*).
- Cette classe se charge de lire sur le flux de données un grand nombre d'octets qu'elle garde dans un tampon.
- Tous les appels à la méthode *read ()* ultérieurs sur une instance de cette classe renvoient des données provenant de ce tampon tant qu'il y reste des octets à lire ;
- le tampon est rechargé à partir du fichier quand on a épuisé toutes les informations qui s'y trouvent.



BufferedInputStream

Constructor	Description
<code>BufferedInputStream(InputStream IS)</code>	Cree un <code>BufferedInputStream</code> avec un tampon par défaut
<code>BufferedInputStream(InputStream IS, int size)</code>	Creation d'un <code>BufferedInputStream</code> avec la taille du tampon
Method	Description
<code>int available()</code>	Retourne une estimation du nombre d'octets qui peuvent être lues du flux.
<code>int read()</code>	Lit le prochain octet .
<code>int read(byte[] b, int off, int ln)</code>	Lit les octets à partir de la position off, les met dans un tableau b.
<code>void close()</code>	Ferme le flux input stream et libère les ressources système.



Exemple

- `import java.io.*;`
- `public class BufferedInputStreamExample {`
- `public static void main(String args[]){`
- `try{`
- `FileInputStream fin=new`
`FileInputStream("D:\\testout.txt");`
- `BufferedInputStream bin=new`
`BufferedInputStream(fin);`
- `int i;`
- `while((i=bin.read())!=-1){`
- `System.out.print((char)i);`
- `}`
- `bin.close();`
- `fin.close();`
- `} catch(Exception e){`
- `System.out.println(e);`
- `}`
- `}`
- `}`



Bon a savoir

- Aux flux d'entrée (lecture) **InputStream** correspondent les homologues en écriture **OutputStream**
 - FileInputStream => FileOutputStream
 - BufferedInputStream => BufferedOutputStream
- Aux méthodes **read()** correspondent les méthodes **write()**
- **Exercice:**
 - Ecrire un programme Java qui lit une séquence au clavier et les écrit dans un fichier sur le disque
 - Ecrire un programme qui copie un fichier vers un nouveau fichier
 - Conseillez-vous l'utilisation d'un buffer? Justifier votre opinion



ObjectInputStream

- La classe **ObjectInputStream** est utilisée pour lire des objets java (y compris les types primitifs) dans un flux d'entrée (*InputStream*)
- Les objets doivent supporter l'interface **java.io.Serializable**

Le Constructeur

public ObjectInputStream(InputStream out) throws IOException {}

Crée un ObjectInputStream qui lit dans le InputStream spécifié.

Méthodes	Description
1) public final void readObject(Object obj) throws IOException {}	Lis l'objet spécifié dans le ObjectInputStream.
2) public void flush() throws IOException {}	Vide le output stream courant
3) public void close() throws IOException {}	Ferme le flux courant



Modifier avec WPS Office

ObjectOutputStream

- La classe **ObjectOutputStream** est utilisée pour écrire des objets java (y compris les types primitifs) dans un flux de sortie *OutputStream*)
- Les objets doivent supporter l'interface **java.io.Serializable**

Le Constructeur

public ObjectOutputStream(OutputStream out) throws IOException {}

Crée un ObjectOutputStream qui écrit dans le OutputStream spécifié.

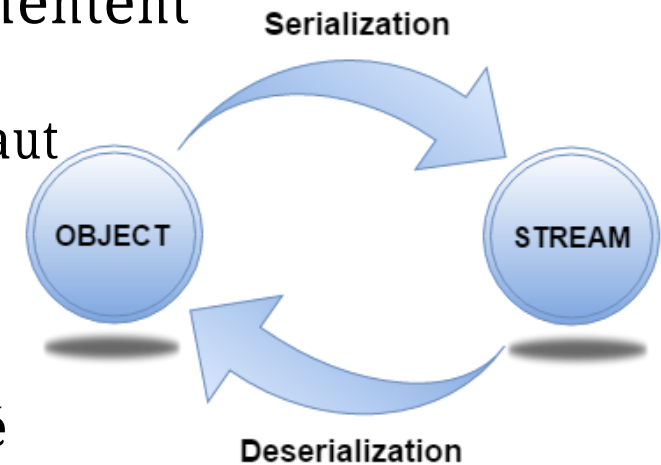
Méthodes	Description
1) public final void writeObject(Object obj) throws IOException {}	Écris l'objet spécifié dans le ObjectOutputStream.
2) public void flush() throws IOException {}	Vide le output stream courant
3) public void close() throws IOException {}	Ferme le flux courant



Modifier avec WPS Office

Un mot sur la **sérialisation**

- Mécanisme de java pour écrire l'état d'un objet dans un flux d'octets
- Avantages
 - Principalement pour transmettre un objet sur le réseau (marshaling)
- La classe dont les objets seront persistés implémentent
 - L'interface **java.io.Serializable**
 - La classe String implémente cette interface par défaut
- C'est une interface de marquage
 - Pas d'attributs
 - Pas de méthodes
 - Donne aux objets de la classe une certaine capacité



Exemple (serializable)

```
1. import java.io.Serializable;
2. public class Student implements Serializable
   {
3.     int id;
4.     String name;
5.     public Student(int id, String name) {
6.         this.id = id;
7.         this.name = name;
8.     }
9.     public String toString(){
10.         return "id=" + this.id + " name=" + this.
            name;
11.     }
12. }
```

```
1. import java.io.*;
2. class Persist {
3.     public static void main(String args[])throws Exception
       {
4.         Student s1 =new Student(211,"ravi");
5.         FileOutputStream fout=new FileOutputStream("f.txt");
6.         ObjectOutputStream out=new ObjectOutputStream(fout)
           t)
7.         out.writeObject(s1);
8.         out.flush();
9.         System.out.println("success");
10.     }
11. }
```



Synthèse

	Flux en lecture	Flux en sortie
Flux de caractères	BufferedReader CharArrayReader FileReader InputStreamReader LineNumberReader PipedReader PushbackReader StringReader	BufferedWriter CharArrayWriter FileWriter OutputStreamWriter PipedWriter StringWriter
Flux d'octets	BufferedInputStream ByteArrayInputStream DataInputStream FileInputStream ObjectInputStream PipedInputStream PushbackInputStream SequenceInputStream	BufferedOutputStream ByteArrayOutputStream DataOutputStream FileOutputStream ObjectOutputStream PipedOutputStream PrintStream



Modifier avec WPS Office

Typologie des filtres

- **Buffered** : ce type de filtre permet de mettre les données du flux dans un tampon. Il peut être utilisé en entrée et en sortie
- **Sequence** : ce filtre permet de fusionner plusieurs flux.
- **Data** : ce type de flux permet de traiter les octets sous forme de type de données
- **LineNumber** : ce filtre permet de numéroté les lignes contenues dans le flux
- **PushBack** : ce filtre permet de remettre des données lues dans le flux
- **Print** : ce filtre permet de réaliser des impressions formatées
- **Object** : ce filtre est utilisé par la sérialisation
- **InputStream / OutputStream** : ce filtre permet de convertir des octets en caractères



Accès aux bases de données

L'api JDBC



Modifier avec WPS Office

Cours de POO 2

Présentation

- JDBC : nom déposé et pas un acronyme
 - En général il se définit par **Java DataBase Connectivity**
- C'est un ensemble d'interfaces et de classes qui **permettent l'accès, à partir de programmes Java, à des données tabulaires,** généralement des bases de données contenues dans des SGBD relationnels
- JDBC permet
 - Des connexions aux SGBD
 - D'envoyer des requêtes SQL et de recevoir les résultats
 - De manipuler les métadonnées de connexion, des résultats
 - De traiter les erreurs retournées par le SGBD



Stockage des données

- **Dans des objets** et leurs attributs
 - ceux-ci ne restent en mémoire que de manière temporaire, cette durée étant déterminée par leur portée
 - Lorsque l'application se termine, les données sont perdues ;
- **Dans des fichiers**
 - Vous savez écrire en manipulant les flux d'entrée et sortie.
 - Les données ainsi écrites sur le disque ont le mérite d'être sauvegardées de manière permanente, et sont accessibles peu importe que l'application en cours d'exécution ou non
 - Le souci: manipuler les données volumineuses et gérer les jointures (produit cartésien) entre elles.
- Les données sont sans arrêt lues, écrites, modifiées ou supprimées
 - mission impossible sans un système de stockage efficace.
- **Base de données**
 - Système miracle qui permet d'enregistrer des données de façon organisée et hiérarchisée



Base de données

- Une **base de données** englobe un ensemble de **tables**
- Une **table** est une structures contenant des **lignes** et des **colonnes** et dans lesquelles les données sont rangées
- une **entrée** désigne une ligne
- un **champ** désigne une colonne

Une colonne/champ

id	pseudo	email	age
1	Coyote	coyote@bipbip.com	25
2	Thunderseb	jadorejquery@unefois.be	24
3	Kokotchy	decapsuleur@biere.org	27
4	Marcel	marcel@laposte.net	47

Une entrée



Modifier avec WPS Office

SGBD et SQL

- SGBD: Système de Gestion des Bases de Données
 - Logiciel permettant le stockage, la consultation, la mise à jour, la structuration ou encore le partage d'informations dans une base de données
 - Les principaux
 - MySQL et ses variante MariaDB, ... : solution libre et gratuite
 - PostgreSQL : solution libre et gratuite, moins connue du grand public
 - Oracle : solution propriétaire et payante, massivement utilisée par les grandes entreprises. C'est un des SGBD les plus complets, mais un des plus chers également
 - SQL Server : la solution propriétaire de Microsoft
- SQL: Structured Query Language
 - Langage pour écrire les requêtes.



Structure de SQL

- Langage de Définition des Données (LDD)
 - Créer les bases de données (CREATE DATABASE
 - Création des tables (CREATE TABLE)
 - Supprimer une base de données (DROP DATABASE ...)
 - Supprimer une table (DROP TABLE ...)
 - Modification de la structure (ALTER TABLE ...)
- Langage de Manipulation des Données (LMD)
 - Insertion d'un tuple (INSERT INTO
 - Supprimer un tuple (DELETE FROM ...)
 - Mis à jour (UPDATE ... SET ...)
 - Consultation, Interrogation (SELECT ... FROM ...WHERE ...)
- Langage de contrôle des données (LCD)
 - Gestion des privilèges (GRANT, REVOKE, CONNECT, COMMIT, ROLLBACK, SET)



Pratique de SQL

- Gestion des inscriptions dans votre institution
- Les tables
 - Etudiant(**matricule**, nom, genre, datenaissance, telephone, email)
 - Versement(**matricule**, **dateversement**, montant)
- Questions
 - Liste des étudiants
 - Quels sont les étudiants inscrits
 - Nom des étudiants ayant versé la totalité
 - Le téléphone et le nom des étudiants n'ayant effectué aucun versement
 - Volume d'argent encaissé
- Travail à faire
 - Créer la BD en question
 - Insérer quelques données
 - Répondre aux questions ci-dessus



Définition des données

- **CREATE TABLE** `etudiant` (
 - `MATRICULE` **VARCHAR(10)** NOT NULL,
 - `NOM` **VARCHAR(50)** NOT NULL,
 - `GENRE` **CHAR(1)** NOT NULL DEFAULT "",
 - `DATENAISSANCE` **INT(4)** NOT NULL DEFAULT 0,
 - `TELEPHONE` **VARCHAR(10)** NOT NULL DEFAULT '0',
 - `EMAIL` **VARCHAR(50)** NOT NULL DEFAULT '0',
 - **PRIMARY KEY** (`MATRICULE`)
 -);



Définition des données

- **CREATE TABLE** `versement` (
• `MATRICULE` **VARCHAR**(10) **NOT NULL**,
• `DATEVERSEMENT` **INT** **NOT NULL**,
• `MONTANT` **INT** **NOT NULL**,
• **PRIMARY KEY** (`MATRICULE`, `DATEVERSEMENT`),
• **CONSTRAINT** `FK__etudiant` **FOREIGN KEY** (`MATRICULE`) **REFERENCES** `etudiant`
(`MATRICULE`)
•);



Réponse

- Démarrer WAMP server et s'assurer que le service mySQL est lancé
- Lancer le client (workbench ou autre)
- **CREATE DATABASE `bdPOO`;** // création de la BD
- **USE `bdpoo`;** // rendre la nouvelle BD active
- **CREATE TABLE `etudiant` (
 - `matricule` **VARCHAR(9) NOT NULL,**
 - `nom` **VARCHAR(50) NOT NULL,**
 - `genre` **ENUM('M','F') NOT NULL,**
 - `dateNaissance` **DATE NULL,**
 - `telephone` **VARCHAR(12) NULL,**
 - `email` **VARCHAR(20) NULL****
- **)COLLATE='utf8_bin' ENGINE=InnoDB;**
- **ALTER TABLE `etudiant` ADD PRIMARY KEY (`matricule`);**



Réponse ...

- **CREATE TABLE** `versement` (
 - `matricule` **VARCHAR(9) NOT NULL**,
 - `dateVersement` **DATE NOT NULL**,
 - `montant` **INT NOT NULL**
- **) COLLATE='utf8_bin' ENGINE=InnoDB;**
- **ALTER TABLE** `versement` **ADD CONSTRAINT** `FK_versement_etudiant` **FOREIGN KEY** (`matricule`) **REFERENCES** `etudiant` (`matricule`);



Insertion des données

- **Insert into etudiant values**

- ('1718L024','SONG','M','2002-04-12','677453212','song@gmail.com'),
- ('1718L005','ONANA','M','1999-05-24','699876540','onana@yahoo.cm'),
- ('1718L019','LOTCHOUANG','F','2004-09-27','675432345','lotchouang@gmail.cm');

- **Insert into versement values**

- ('1718L019','2018-10-31',250000),
- ('1718L024','2018-10-15',650000),
- ('1718L024','2018-11-11',30000);

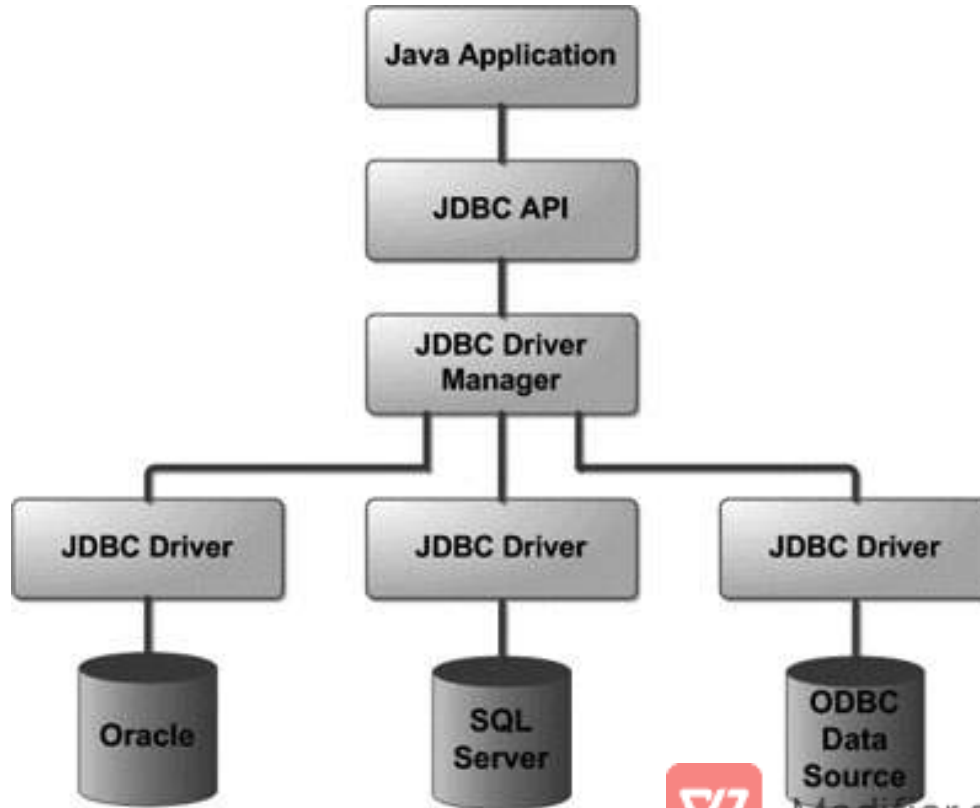


Réponses aux questions

- `SELECT * FROM etudiant;`
- `SELECT distinct nom FROM etudiant,versement WHERE etudiant.matricule=versement.matricule`
 - `SELECT distinct nom FROM etudiant e INNER JOIN versement v ON e.matricule=v.matricule`
- **`SELECT distinct nom, sum(montant) m FROM etudiant,versement WHERE etudiant.matricule=versement.matricule GROUP BY versement.matricule HAVING m>=650000`**
- `SELECT nom,telephone FROM etudiant e WHERE e.matricule not in (SELECT matricule FROM versement)`
- `SELECT sum(montant) as total FROM versement`



Architecture de JDBC



- **API JDBC:** fournit la connexion de l'application au Gestionnaire (JDBC Manager).
- **API du Pilote JDBC:** Cette option prend en charge la connexion JDBC Gestionnaire-Pilote.
- L'API JDBC utilise un gestionnaire de pilotes et des pilotes spécifiques à la base de données pour fournir une connectivité transparente aux bases de données hétérogènes.
- Le gestionnaire de pilotes JDBC s'assure que le pilote correct est utilisé pour accéder à chaque source de données.
 - Le gestionnaire de pilotes est capable de prendre en charge plusieurs pilotes simultanés connectés à plusieurs bases de données hétérogènes.



Modifier avec WPS Office

Les interfaces de l'API JDBC

- **DriverManager**: Cette classe gère une liste de pilotes de base de données. Correspond aux demandes de connexion de l'application Java avec le pilote de base de données approprié à l'aide du sous-protocole de communication. Le premier pilote qui reconnaît un certain sous-protocole sous JDBC sera utilisé pour établir une connexion à la base de données.
- **Driver**: cette interface gère les communications avec le serveur de base de données. Vous allez interagir directement avec les objets Driver très rarement. Au lieu de cela, vous utilisez les objets DriverManager, qui gère les objets de ce type. Il résume également les détails associés à l'utilisation d'objets Driver.
- **Connection**: Cette interface avec toutes les méthodes pour contacter une base de données. L'objet de connexion représente le contexte de communication, c'est-à-dire que toute communication avec la base de données s'effectue uniquement via l'objet de connexion.
- **Statement**: vous utilisez des objets créés à partir de cette interface pour soumettre les instructions SQL à la base de données. Certaines interfaces dérivées acceptent les paramètres en plus d'exécuter des procédures stockées.
- **ResultSet**: ces objets contiennent des données extraites d'une base de données après l'exécution d'une requête SQL à l'aide d'objets Statement. Il agit comme un itérateur pour vous permettre de parcourir ses données.
- **SQLException**: cette classe gère les erreurs qui se produisent dans une application de base de données.



Avant de commencer (Prerequis)

- Installer Mysql et démarrer le service
- Installer le pilote (driver): JConnector pour Mysql



Liaison Mysql et projet Java

- Créer un projet Java dénommé « jdbc »
- Ajouter le driver de mysql au projet. il y a 2 possibilités
 - Click droit sur **Library > Add library** permet d'ajouter le jar natif de Netbeans
 - Click droit sur **Library>Add JAR** permet d'ajouter un fichier jar externe
- Pour la suite
 - découvrir les nouveaux objets et interfaces qui doivent intervenir dans notre application afin d'établir une communication avec la base de données
 - Apprendre à lire des données depuis la base vers notre application
 - Ecrire des données depuis notre application vers la base



Chargement du Driver

```
1  /* Chargement du driver JDBC pour MySQL */
2  try {
3      Class.forName( "com.mysql.jdbc.Driver" );
4  } catch ( ClassNotFoundException e ) {
5      /* Gérer les éventuelles erreurs ici. */
6  }
```

Inutile de charger le driver avant chaque connexion, une seule fois durant le chargement de l'application suffit

- ligne 3 de notre exemple permet de charger le driver
 - ici « `com.mysql.jdbc.Driver` »
 - Le nom du driver est fourni par le constructeur
 - Pour un autre SGBD, vous renseigner sur le site de son distributeur pour trouver son nom exact
- Si une exception de type `ClassNotFoundException` est renvoyée
 - le fichier .jar contenant le driver JDBC pour MySQL n'a pas été correctement placé dans le classpath
 - Retirer le driver que nous avons ajouté en tant que bibliothèque externe et constater



Modifier avec WPS Office

Connexion à la BDD: URL

- Syntaxe de l'URL : `jdbc:mysql://localhost:3306/bdpo`

Protocole

le nom de l'hôte sur lequel le serveur MySQL est installé. S'il est en place sur la même machine que l'application Java exécutée, alors vous pouvez simplement spécifier **localhost**. une adresse IP comme 127.0.0.1

le port TCP/IP écouté par votre serveur MySQL. Par défaut, il s'agit du port **3306**

le nom de la base de données à laquelle vous souhaitez vous connecter



Modifier avec WPS Office

Cours de POO 2

Connexion à la BDD: les objets utiles

- L'objet [DriverManager](#). Permet d'établir la connexion
 - Appeler sa méthode statique [getConnection\(\)](#) pour récupérer un objet de type [Connection](#).
 - Les paramètres: l'adresse de la base de données, le nom d'utilisateur et le mot de passe associé.
- L'appel à cette méthode peut retourner des erreurs de type [SQLException](#):
 - si une erreur **SQLException: No suitable driver** est envoyée, alors cela signifie que le driver JDBC n'a pas été chargé ou que l'URL n'a été reconnue par aucun des drivers chargés par votre application ;
 - si une erreur **SQLException: Connection refused** ou **Connection timed out** ou encore **CommunicationsException: Communications link failure** est envoyée, alors cela signifie que la base de données n'est pas joignable.



Objet connexion

```
•public static void main(String[] args) {
•    // TODO code application logic here
•    String url = "jdbc:mysql://localhost:3306/bdpoo";
•    String utilisateur = "utilisateur";
•    String motDePasse = "MotDePasse";
•    Connection connexion = null;
•    try {
•        /* Chargement du driver JDBC pour MySQL */
•        Class.forName("com.mysql.jdbc.Driver");
•        /* Connexion à la base de données */
•        connexion = DriverManager.getConnection(url, utilisateur, motDePasse);
•        System.out.println("Connexion créée: " + connexion.toString());
•    } catch (ClassNotFoundException e) {
•
•        /* Gérer les éventuelles erreurs ici. */
•    } catch (SQLException ex) {
•        ex.printStackTrace();
•    } finally {
•        if (connexion != null) {
•            try {
•                /* Fermeture de la connexion */
•                connexion.close();
•            } catch (SQLException ignore) {
•                /* Si une erreur survient lors de la fermeture, il suffit de l'ignorer. */
•            }
•        }
•    }
•}
```



Création d'une requete

- Créer un objet de type [Statement](#)
 - d'une interface dont le rôle est de permettre l'exécution de requêtes
 - Initialisation:
 - d'appeler la méthode [createStatement\(\)](#) de l'objet Connection
 - `Statement statement = connexion.createStatement();`
- Méthodes usuelles de l'objet Statement (pour exécuter une requete)
 - [executeQuery\(\)](#): cette méthode est dédiée à la lecture de données via une requête de type **SELECT**;
 - [executeUpdate\(\)](#): cette méthode est réservée à l'exécution de requêtes ayant un effet sur la base de données (écriture ou suppression), typiquement les requêtes de type **INSERT, UPDATE, DELETE**, etc.
 - retourne un objet de type [ResultSet](#) contenant le résultat de la requête
- Une requête de lecture
 - `ResultSet resultat = statement.executeQuery("SELECT * FROM etudiant;");`



ResultSet

- Similaire à un tableau, qui contient les éventuelles données retournées par la base de données sous forme de lignes
- Pour accéder à ces lignes de données, vous avez à votre disposition un curseur, que vous pouvez déplacer de ligne en ligne
- Méthodes usuelles
 - **next()**: déplace le curseur sur la ligne suivante, retourne un **true** tant qu'il reste des données à parcourir
 - **getInt()** pour récupérer un entier ;
 - **getString()** pour récupérer une chaîne de caractères ;
 - **getBoolean()** pour récupérer un booléen ;
 - etc.
- **libérer les ressources** utilisées lors d'un échange avec la base de données, en fermant les différents objets.
 - Appel de la méthode **close()** sur chacun



Application

```
Statement st = connexion.createStatement();  
// Exécution d'une requête  
ResultSet rs = st.executeQuery("Select * From etudiant;");  
//Recupération des données  
while(rs.next()) {  
    System.out.print(rs.getString("matricule") + "\t");  
    System.out.print(rs.getString("nom")+ "\t");  
    System.out.print(rs.getDate("dateNaissance"));  
    System.out.println("\n-----");  
}  
//Libérer les ressources utilisées  
rs.close();  
st.close();
```

