



HAL
open science

Support de cours du langage de programmation Java

Alexandre Journaux

► **To cite this version:**

Alexandre Journaux. Support de cours du langage de programmation Java. Langage de programmation Java (Initiation au langage Java), 2014, 100 diapositives. hal-02800791

HAL Id: hal-02800791

<https://hal.inrae.fr/hal-02800791v1>

Submitted on 5 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Formation JAVA



Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- Bibliothèques standard
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web

Rapide historique de Java

● Dates clés

- 1990-92 : Langage « Oak » qui s'appuie sur une machine virtuelle
- 1993 : Orientation de « Oak » vers une utilisation sur Internet
- 1995 : Présentation de Java à SunWorld
 - Oak est rebaptisé Java
 - Vif succès : moyen d'animer les pages statiques du web avec les « applets »
- 1996 : Netscape Navigator 2 incorpore une machine virtuelle Java 1.0 en version « beta »

Rapide historique de Java

- Dates clés (suite)

- 1997 : Un premier pas vers une version industrielle Java 1.1

- 1999 : Version industrielle de Java

- Aujourd'hui

- JDK 1.8 (ou Java 8)

- Communauté très active

- <http://java.developpez.com>

Caractéristiques du langage

- Orienté objet

- Basé sur la notion de classe

- Respecte les principes d'abstraction, d'encapsulation et de polymorphisme

- Particularités

- Héritage simple, pas d'héritage multiple (évite les pbs de duplication d'attributs, conflits entre méthode...)

- Notion d'interface

Caractéristiques du langage

● Langage interprété et portable

●● Interprété : Utilise la machine virtuelle

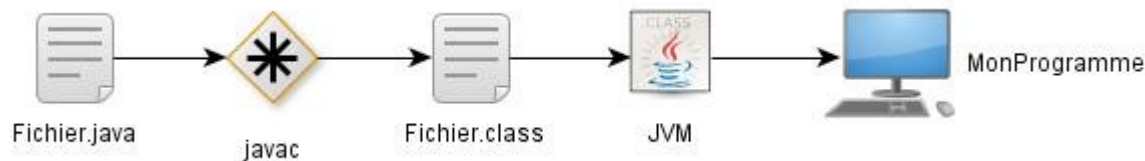
●●● Aucun code natif généré

●●● La machine virtuelle contient l'interpréteur qui traduit à la volée le pseudo-code en code natif

●● Portable

●●● Pseudo-code indépendant de l'architecture matérielle

●●● Prise en charge des aspects spécifiques à chaque système par les différentes implémentations des JVM



Java et ses versions

- Différentes versions de la machine virtuelle
 - Java Micro Edition (Java ME) : terminaux portables
 - Java Standard Edition (Java SE) : postes clients
 - Java Enterprise Edition (Java EE) : serveur d'application basé sur la notion de classe
- Différentes distributions
 - SDK (Software Development Kit) fournit un compilateur et une machine virtuelle
 - JRE (Java Runtime Environment) fournit uniquement une machine virtuelle. Idéal pour le déploiement de vos applications.

Les ressources sur Java

- Site de Java chez Oracle

- <http://www.oracle.com/technetwork/java/index.html>

- API (bibliothèque)

- <http://docs.oracle.com/javase/7/docs/api/>

- Cours, exemples et Forum

- <http://java.developpez.com/>

Les outils

- Simple éditeur ou environnement de développement

Affichage des résultats du sondage: Quel EDI Java utilisez vous en 2011 ?

Eclipse		<u>279</u>	69,40%
NetBeans		<u>155</u>	38,56%
IntelliJ		<u>18</u>	4,48%
RAD / WSAD		<u>5</u>	1,24%
JDeveloper		<u>7</u>	1,74%
JCreator		<u>0</u>	0%
JBuilder		<u>2</u>	0,50%
BEA Workshop Studio		<u>1</u>	0,25%
Editeurs de texte avancés (Emacs, VI, JEdit, UltraEdit, ...)		<u>12</u>	2,99%
Autre (précisez)		<u>2</u>	0,50%

Sondage à choix multiple Votants: **402**. Ce sondage est terminé

Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- Bibliothèques standard
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web

1^{er} exemple : Hello l'INRA

- public class HelloInra
 - Nom de la classe
- public static void main
 - La fonction principale équivalent à la fonction main du C#
- String[] args
 - Permet de récupérer des arguments transmis au programme au moment de son lancement
- System.out.println(« Hello ... »)
 - Méthode d'affichage dans la fenêtre console

```
public class HelloInra {  
  
    public static void main(String[] args) {  
        System.out.println("Hello l'INRA");  
    }  
  
}
```

Mise en œuvre

- Nom de la classe = Nom du fichier java
 - Un seul fichier : HelloInra.java
- Compilation
 - javac HelloInra.java ou javac *.java
 - Génération d'un fichier Byte-Code HelloInra.class
- Exécution
 - java HelloInra
 - Choisir la classe principale à exécuter
 - Ne pas mettre l'extension .class pour l'exécution

Syntaxe

● Proche du C#

- Une instruction se termine par ;
- Le bloc d'instructions : délimité par { et }
- La condition booléenne : délimitée par (et)
- Les tableaux : notation []

● Les commentaires

```
/* voici un commentaire comme en C#  
qui peut être sur plusieurs lignes */  
  
// Voici un commentaire sur une seule ligne  
  
/**  
 * voici un commentaire qui sera utilisé pour  
 * générer automatiquement la javadoc  
 */
```

Syntaxe

- Les types primitifs
 - Ne sont pas des objets
 - Occupent une place fixe en mémoire réservée à la déclaration
 - Entiers : byte (1 octet) - short (2 octets) - int (4 octets) – long (8 octets)
 - Flottants : float (4 octets) - double (8 octets)
 - Booléens : boolean (true ou false)
 - Caractères : char (codage Unicode sur 16 bits)
 - Chacun des types simples possède un alter-ego objet disposant de méthodes de conversion
 - Ex: Integer pour int

Syntaxe

● Déclaration et initialisation d'une variable

```
int n; // Déclaration : type nom  
n = 10; //Affectation : nom=valeur  
int m = 20; //Combinaison : type nom=valeur
```

● Constantes

- Ce sont des variables dont la valeur ne peut être affectée qu'une fois
- Elles ne peuvent plus être modifiées
- Elles sont définies avec le mot clé **final**

```
final int n;  
n = 10;  
final int m = 20;  
  
n=12; // erreur : n est déclaré final
```


Syntaxe

- Opérateurs arithmétiques
 - Unaires : -, --, ++
 - Binaires : +, -, *, /, %
- Opérateurs logiques
 - !, &, |, &&, ||
- Opérateurs de comparaison
 - <, >, <=, >=, ==, !=

```
int i = 6;int j = 6;
if (i==j)
{
    //Ce sont des types primitifs donc la comparaison est vraie
}
Voiture voiture1=new Voiture("123 AXB 31");
Voiture voiture2=new Voiture("123 AXB 31");
if (voiture1==voiture2)
{
    //Ce sont des objets donc la comparaison est fausse
}
```

```
int n = 6;
int m = -n; //m = -6
n++; //n=7
int i = n / 4; //i=1
double j = n / 4; //j=1.0
j = n / 4.0; //j=1.75
String texte=null;
if (texte!=null & texte.length()>4)
{
    //Les 2 tests sont analysés
    //ceci va provoquer une erreur
    //=> java.lang.NullPointerException
}
if (texte!=null && texte.length()>4)
{
    //Le 2e test est analysé que si le 1er est vrai
    //ceci va s'exécuter correctement
}
```

Syntaxe

● Les Wrappers

●● Les types primitifs ne sont pas des objets

●● Un wrapper est la représentation objet d'une primitive

●●● Boolean, Character, Byte, Short, Integer, Long, Float, Double

●● Offrent des méthodes pour traiter les primitives

```
Integer deux=new Integer("2");
double i = deux.doubleValue(); // i = 2.0
char a = 'a';
boolean estUneLettre = Character.isLetter(a);
if (estUneLettre){
    char maj = Character.toUpperCase(a);//maj=A
    System.out.println(maj);
}
```

Syntaxe

● Les tableaux

```
//Déclaration :
int[] tableau;
int[][] tableau2; //tableau à 2 dimensions
//Dimensionnement :
tableau = new int[3];
//Initialisation :
tableau[0] = 10; //comme en C# les indices commencent à zéro
tableau[1] = 20;
tableau[2] = 30;
//Combinaison :
int[] tableau3 = {10,20,30};

//Nombre d'élément
int taille = tableau3.length;

//Parcours d'un tableau
for (int i = 0; i < tableau3.length; i++) {
    System.out.println(tableau3[i]);
}
```

Syntaxe

● La classe String

```
//Création d'une instance :
String texte = "Bonjour";
//Longueur d'une String
int longueur = texte.length(); // longueur=7
//Conversion vers une String;
int i = 10;
String dix = String.valueOf(i);
//plus simple :
dix = ""+10;
//Conversion inverse grâce aux wrappers:
double d = Double.valueOf(dix).doubleValue(); //.doubleValue() n'est pas obligatoire
int j = new Integer(dix);
//Les comparaisons
if ("abc".equals("ABC")){} //faux
if ("abc".equalsIgnoreCase("ABC")){} // vrai
//La recherche
texte.startsWith("Bon"); //retourne vrai si commence par Bon
texte.endsWith("r"); //retourne vrai si termine par r
texte.indexOf("on"); //retourne 1 : l'index de la 1ère occurrence de on ; -1 si pas trouvé
texte.lastIndexOf("o"); //retourne 4 : l'index de la dernière occurrence de o
//L'éditition
texte.toUpperCase(); //retourne BONJOUR
texte.substring(0,3); //retourne Bon
texte.replaceAll("on", "aa"); //retourne Baajour
"Bonjour   ".trim(); //retourne Bonjour
//Concaténation
String chaine = texte+" à vous "+dix; //chaine=Bonjour à vous 10
```

Structures de contrôle

● La condition

```
if (condition){
    //instructions;
}

if (condition){
    //instructions;
} else{
    //instructions;
}

if (condition1){
    //instructions;
} else if (condition2) {
    //instructions;
} else {
    //instructions;
}
```

Structures de contrôle

● Le traitement switch-case

```
switch (valeurEntiere) {  
  case valeur1:  
    //instruction;  
  case valeur2:  
    //instruction;  
    break;  
  default:  
    //instruction;  
    break;  
}
```

● La boucle for

```
for (initialisation;condition;action){  
  //instruction;  
}  
//exemple :  
for (int i = 0; i < 10; i++){  
  //instruction;  
}
```

Structures de contrôle

- La boucle while et la boucle do-while

```
while (condition) {  
    //instruction;  
}  
  
do {  
    //instruction;  
} while (condition);
```

- Action sur une boucle
 - **break** permet de sortir de la boucle
 - **continue** permet d'aller directement à l'évaluation suivante

Travaux pratiques – TP1

- Environnement Eclipse
 - Installation
 - Organisation de l'environnement et des sources
- Créer le programme HelloInra avec paramètre
 - Le nom est en paramètre
 - Affiche : *Hello Jean* (Jean passé en paramètre)
- Créer le programme TrieTableau
 - Tableau d'entier : 4, 1, 7, 3, 2, 9
 - Trie et Affiche le tableau trié

Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- Bibliothèques standard
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web



Classe

● Définition

●● Une classe est constituée :

●●● de données qu'on appelle des attributs

●●● de procédures et/ou des fonctions qu'on appelle des méthodes

●● Une classe est un modèle de définition pour des objets

●●● Ayant même structure (même ensemble d'attributs)

●●● Ayant même comportement (même méthodes)

●● Les objets sont des représentations dynamiques de la classe (instanciation)

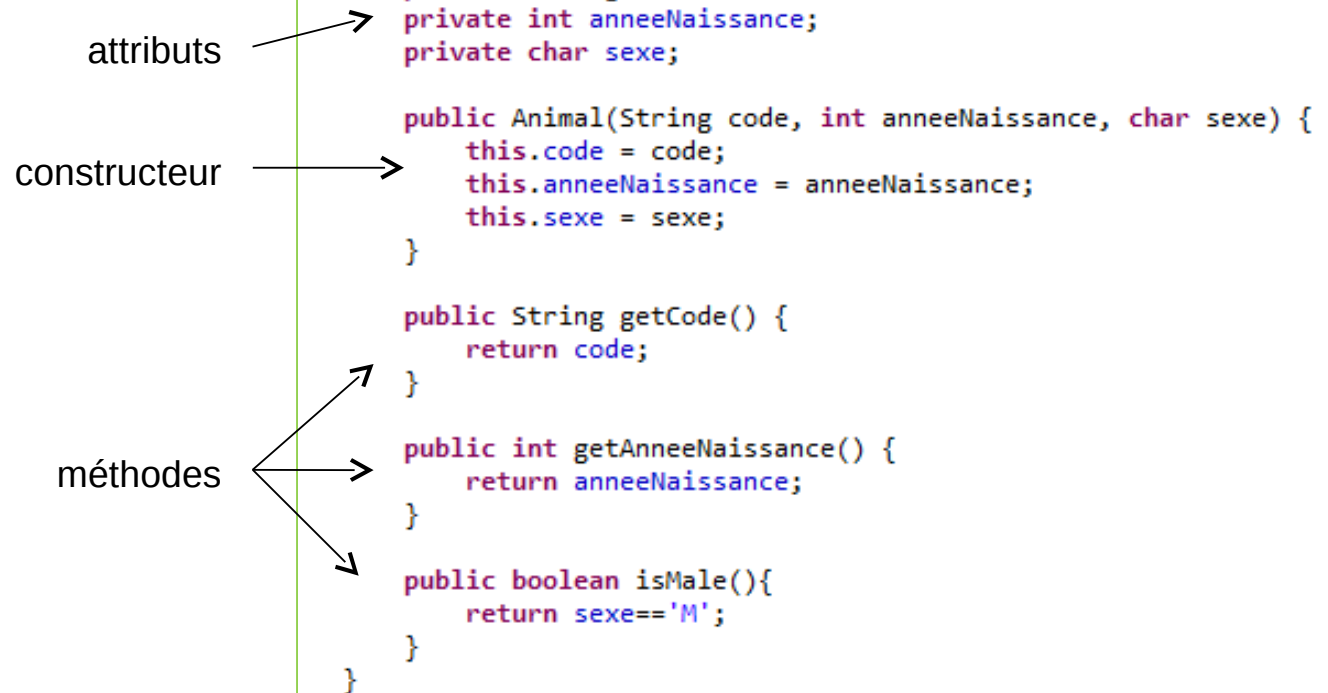
●●● Une classe permet d'instancier (créer) plusieurs objets

●●● Chaque objet est instance d'une classe et une seule

Classe

● Déclaration

- Le code s'écrit dans un fichier qui porte le même nom plus l'extension .java





Classe

- Les attributs
 - Variables globales dans la classe
- Les méthodes
 - Procédures/fonctions qui permettent de manipuler l'objet
 - Les paramètres sont passés
 - Par valeur pour les types primitifs
 - Par référence pour les objets
 - Les constructeurs
 - Méthode d'initialisation qui porte le nom de la classe
 - Il peut y avoir plusieurs constructeurs pour une classe



Classe

- Contrôle d'accès

- Il est possible de préciser l'accès aux classes, attributs, méthodes et constructeurs:

- private : accessible uniquement à l'intérieur de la classe

- protected : accessible par les sous-classe

- public : accessible par n'importe quelle classe

- <aucun> : accessible par toutes les classes du package de la classe

Classe

● Convention de nommage

```
public class JeSuisUneClasse {  
  
    String jeSuisUneVariable;  
    final String JE_SUIS_UNE_CONSTANTE="";  
  
    void jeSuisUneMethode() {  
  
    }  
}
```

- Un fichier par classe, une classe par fichier
- Classe Animal décrite dans le fichier Animal.java
- Il peut exceptionnellement y avoir plusieurs classes par fichier (cas des Inner classes)



Objet

- Un objet est instance d'une seule classe
- Tout objet est manipulé et identifié par sa référence
- Utilisation de pointeur caché
 - « $a = b$ » signifie a devient identique à b
Les deux objets a et b sont identiques et toute modification de a entraîne celle de b
 - « $a == b$ » retourne « true » si les deux objets sont identiques
C'est-à-dire si les références sont les mêmes, cela ne compare pas les attributs

Objet

● Manipulation

```
//Déclaration :  
Animal animal1;  
//Création et allocation mémoire :  
animal1 = new Animal();  
//Déclaration et création en une seule ligne  
Animal animal2 = new Animal("0002",2009,'M');  
//Utilisation :  
animal1.setCode("0001");  
if (animal2.isMale()){  
    System.out.println(animal2.getCode()+" est un mâle");  
}
```


Objet

- Un attribut peut être une instance d'une autre classe

```
public class Animal {
    private Espece espece;
    private String code;
    private int anneeNaissance;
    private char sexe;

    public Animal(Espece espece, String code, int anneeNaissance, char sexe) {
        this.espece = espece;
        this.code = code;
        this.anneeNaissance = anneeNaissance;
        this.sexe = sexe;
    }

    public Animal(String codeEspece, String code, int anneeNaissance, char sexe) {
        this.espece = new Espece(codeEspece);
        this.code = code;
        this.anneeNaissance = anneeNaissance;
        this.sexe = sexe;
    }
}
```

Objet

● Gestion des objets

```
//Récupérer son type  
animal1.getClass(); // Retourne un objet de type Class  
  
//Tester son type  
if (animal1 instanceof Animal) {} // C'est vrai  
if (animal1.getClass() == Animal.class) {} // C'est vrai
```

Objet

- Variables de classes (variables statiques)
- Ce sont des constantes liées à une classe
- Elles sont écrites en MAJUSCULES

déclaration

```
public class Espece {  
  
    private String code;  
    public static final String OVIN="01";  
    public static final String CAPRIN="02";  
  
    /**  
    public Espece(String code) {  
        super();  
        this.code = code;  
    }  
}
```

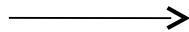
utilisation

```
Espece espece1 = new Espece(Espece.OVIN);
```

Objet

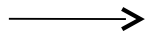
- Méthodes de classes (méthodes statiques)
 - Ce sont des méthodes qui ne s'intéressent pas à un objet particulier
 - Utiles pour des calculs intermédiaires internes à une classe

déclaration



```
public static Espece getMouton(){  
    return new Espece(OVIN);  
}
```

utilisation



```
Espece espece2 = Espece.getMouton();
```



Les Packages

- Un package est un espace de nommage
- C'est un regroupement de classes et d'interfaces logiquement liées
- Les noms des packages sont construits en utilisant une notation pointée
- Exemples :
 - java.lang : rassemble les classes de base Java (Object, String, ...)
 - java.util : rassemble les classes utilitaires (Collections, Date, ...)
 - java.io : lecture et écriture

Les Packages

● Visibilité

- Par défaut, une classe n'est accessible que par les classes du même package
- Seules les classes déclarés *public* sont accessibles

● Utilisation

```
//Utilisation avec le nom complet du package :  
fr.inra.ga.sanitaire.Animal animal = new fr.inra.ga.sanitaire.Animal();  
// => écriture très lourde : préférer l'utilisation de l'import
```

```
import fr.inra.ga.sanitaire.Animal;  
import java.lang.String; // Ne sert à rien java.lang est le package par défaut  
import java.io.FileWriter;  
  
import fr.inra.ga.sanitaire.*;  
import java.io.*;
```

Les Packages

- Le nom du package doit apparaitre au début du fichier source

```
package fr.inra.ga.sanitaire;  
  
import java.io.File;  
  
public class Animal {
```

- Le source .java et la classe compilé .class doivent être dans le répertoire défini par cette règle :
 - Prendre le nom de package
 - Remplacer les . par des /

```
C:\Java\ides\eclipse\workspace\Formation\src\fr\inra\ga\sanitaire
```



Les Héritages

- Mot clé : *extends*
- Java ne permet que l'héritage simple
 - Une seule classe parent

```
public class Mammifere extends Animal{  
  
}
```


Les Héritages

- Accès à la classe parent
- Les méthodes de la classe dérivée peuvent accéder aux attributs *public* et *protected* de la classe parent
- Le mot de clé *super* représente la classe parent

```
public Mammifere(Espece espece, String code, int anneeNaissance, char sexe, int qteLaitParJour) {  
    super(espece, code, anneeNaissance, sexe); //appel du constructeur parent  
    this.qteLaitParJour = qteLaitParJour;  
}
```

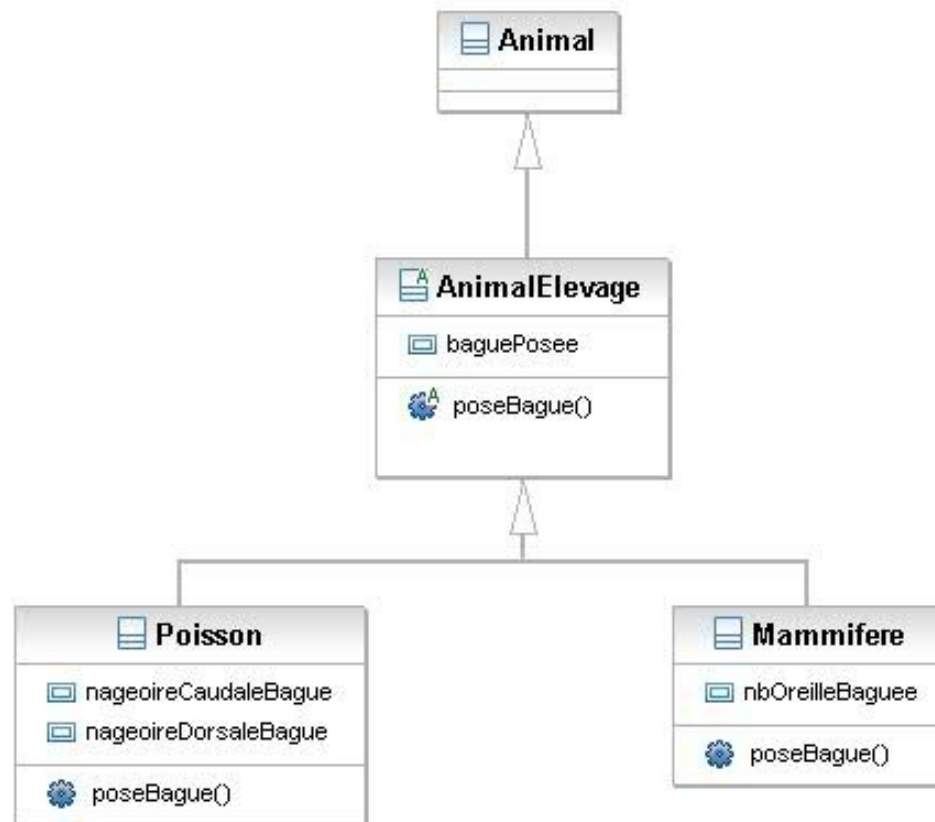
```
anneeNaissance=2007; //attribut définie dans la classe dérivée  
super.anneeNaissance=2007; //attribut définie dans la classe parente  
  
isMale(); //méthode de la classe dérivée  
super.isMale(); //méthode de la classe parente
```

Les classes abstraites

● Intérêt

- On ne connaît pas toujours le comportement par défaut d'une opération commune à plusieurs sous-classes
- Ex : Type de bague d'un animal d'élevage. On sait que tous les animaux d'élevage sont bagués mais la pose des bagues est différente d'un animal à un autre
- 3 règles :
- Si une seule des méthodes d'une classe est abstraite, alors la classe devient aussi abstraite
- On ne peut pas instancier une classe abstraite
- Toutes les classes filles héritant de la classe mère abstraite doivent implémenter toutes ses méthodes

Les classes abstraites



Les classes abstraites

```
public abstract class AnimalElevage extends Animal {  
  
    protected boolean baguePosee;  
    public abstract void poseBague();  
}
```

```
public class Poisson extends AnimalElevage {  
  
    private boolean nageoireDorsaleBague=false;  
    private boolean nageoireCaudaleBague=false;  
  
    @Override  
    public void poseBague() {  
        if (!nageoireDorsaleBague){  
            nageoireDorsaleBague=true;  
        } else if (!nageoireCaudaleBague){  
            nageoireCaudaleBague=true;  
        } else {  
            System.out.println("Nageaoires baguées !");  
        }  
        baguePosee=true;  
    }  
}
```

```
public class Mammifere extends AnimalElevage {  
  
    private int nbOreilleBaguee=0;  
  
    @Override  
    public void poseBague() {  
        if (nbOreilleBaguee < 2) {  
            nbOreilleBaguee++;  
            baguePosee=true;  
        } else {  
            System.out.println("Déjà baguées !");  
        }  
    }  
}
```



Les interfaces

● Intérêt

- Une classe ne peut pas hériter de plusieurs classes
- Ceci peut être contournée à travers la notion d'interface
- Une classe peut implémenter plusieurs interfaces
- Une interface peut étendre plusieurs interfaces

● Particularités

- Une interface ne possède pas d'attribut
- Une interface peut posséder des constantes
- Les interfaces ne sont pas instanciables (comme les classes abstraites)

Les interfaces

- Mise en œuvre
- Mot clé interface
- Déclaration des signatures des méthodes uniquement

```
public interface Deplacable {  
    public void changeDeSite();  
}
```

● Utilisation

```
public class Animal implements Soignable, Deplacable{
```

Les interfaces

● Exemple

```
public class Animal implements Soignable, Deplacable {  
  
    private int nbElevage=1;  
  
    @Override  
    public void changeDeSite() {  
        nbElevage++;  
    }  
}
```

```
public class Agent implements Deplacable {  
  
    private boolean nouvelleAdresse;  
  
    @Override  
    public void changeDeSite() {  
        nouvelleAdresse=true;  
    }  
}
```

```
public class Decideur  
{  
    void deplace(Deplacable deplacable) {  
        deplacable.changeDeSite();  
    }  
}
```

```
Decideur chef= new Decideur();  
Agent agent = new Agent();  
Animal animal = new Animal();  
chef.deplace(agent);  
chef.deplace(animal);
```



La javadoc

- Intérêts
- Rédaction de la documentation technique des classes au fur et à mesure du développement de ces mêmes classes puis génération finale du html
- Utilisation
- Compris entre `/**` et `*/`
- Utilisation possible de balise html
- Utilisation de tags définis par javadoc permettant de typer certaines informations



La javadoc

● Utilisation

●● Exemple de tags :

●●● @author : Nom de l'auteur

●●● @version : Identifiant de version

●●● @param : Nom et signification de l'argument (méthodes uniquement)

●●● @return : Valeur de retour

●●● @throws : Classe de l'exception et conditions de lancement

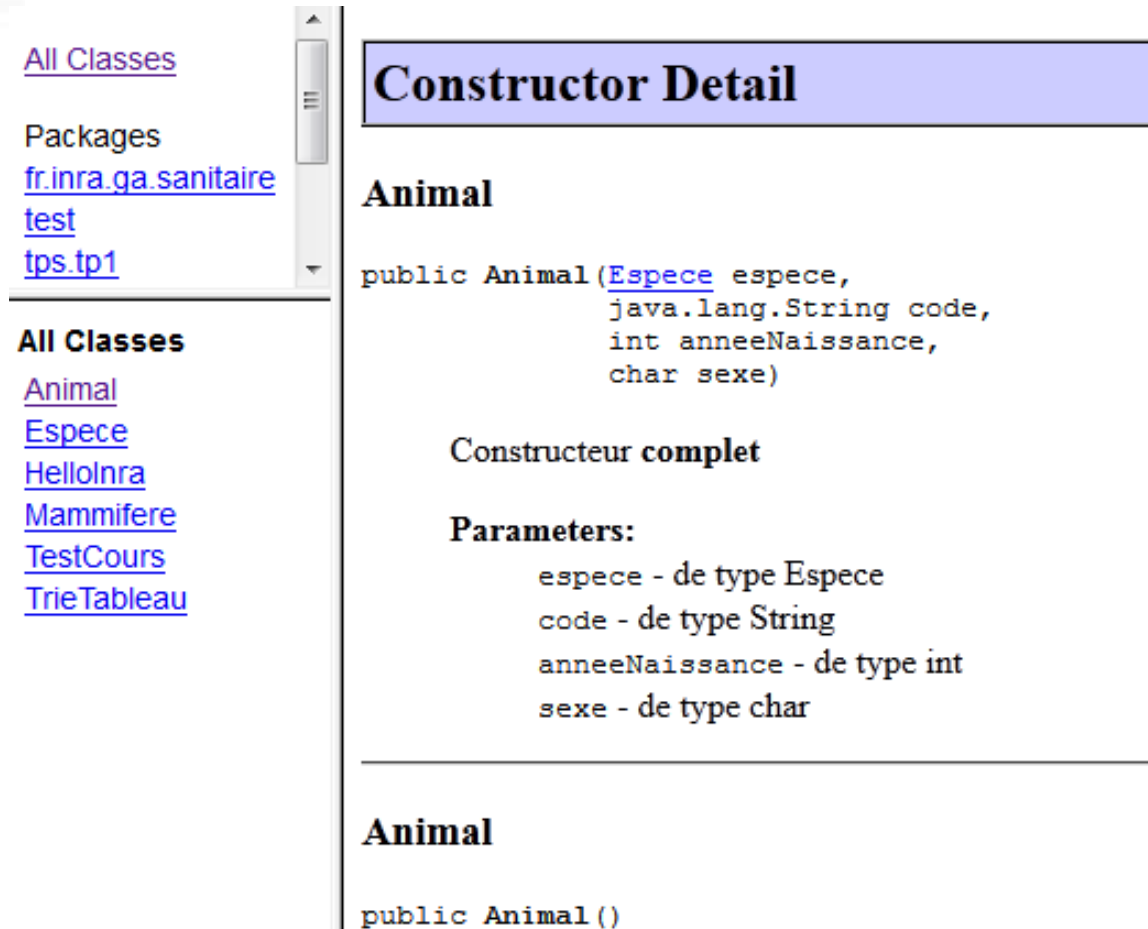
La javadoc

● Exemple

```
/**
 * Constructeur <b>complet</b>
 * @param espece de type Espece
 * @param code de type String
 * @param anneeNaissance de type int
 * @param sexe de type char
 */
public Animal(Espece espece, String code, int anneeNaissance, char sexe) {
    super();
    this.espece = espece;
    this.code = code;
    this.anneeNaissance = anneeNaissance;
    this.sexe = sexe;
}
```

La javadoc

● Résultat :



The screenshot shows a Javadoc viewer interface. On the left, there is a navigation pane with a tree view containing 'All Classes', 'Packages' (with sub-items 'fr.inra.ga.sanitaire', 'test', 'tps.tp1'), and 'All Classes' (with sub-items 'Animal', 'Espece', 'HelloInra', 'Mammifere', 'TestCours', 'TrieTableau'). The main content area is titled 'Constructor Detail' and displays the following information:

```
Animal
```

```
public Animal(Espece espece,  
              java.lang.String code,  
              int anneeNaissance,  
              char sexe)
```

Constructeur **complet**

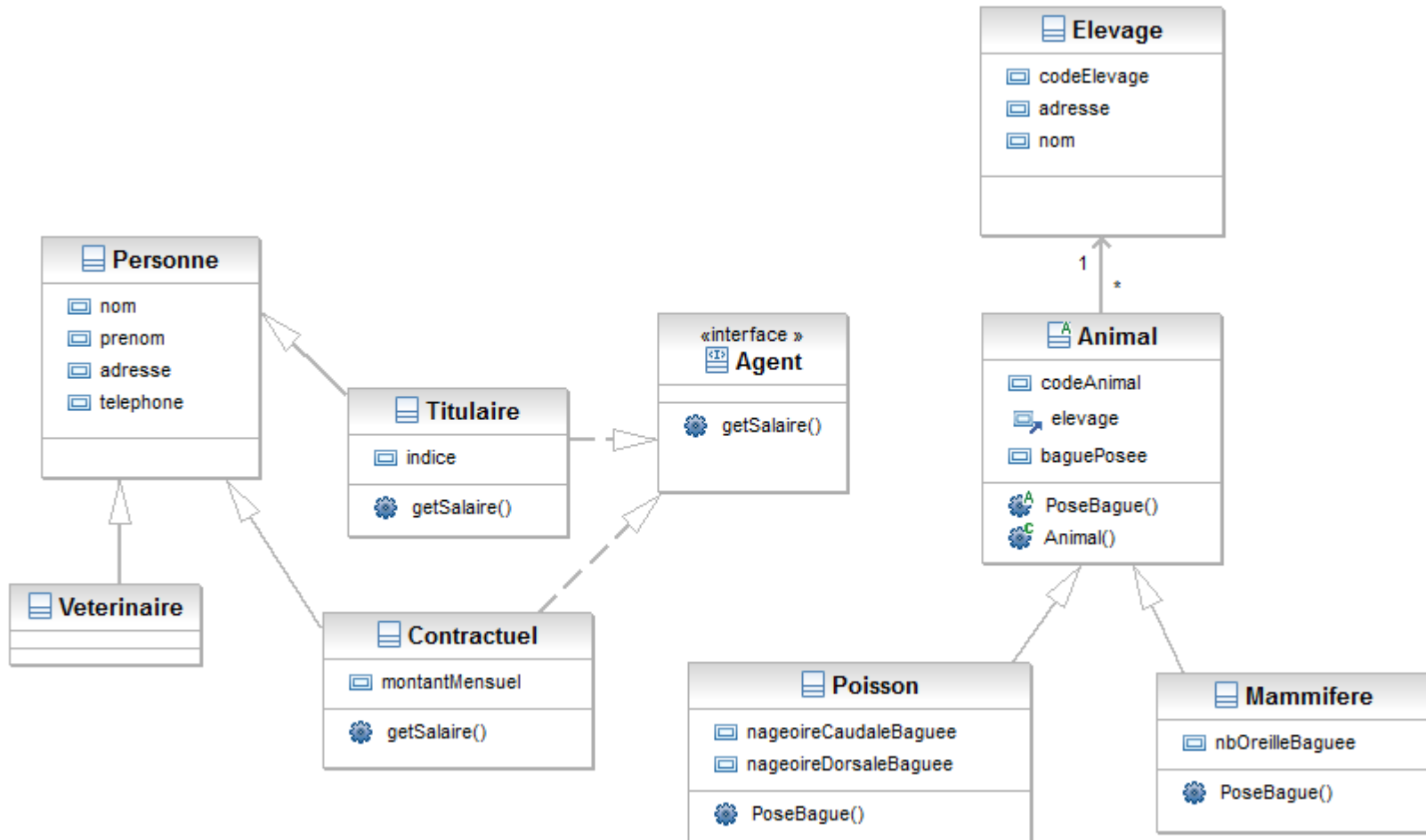
Parameters:

- espece - de type Espece
- code - de type String
- anneeNaissance - de type int
- sexe - de type char

```
Animal
```

```
public Animal()
```

Travaux pratiques – TP2



Travaux pratiques – TP2

- Créer la classe Titulaire qui hérite de Personne
 - Package : fr.inra.ga.tp2
 - Créer les attributs et les méthodes
 - Créer une classe Test qui effectuera dans sa méthode main() :
 - Création d'un titulaire
 - Lui affecter un indice
 - Affiche sur la console son nom et son salaire

Valeur du point d'indice : 55,5635 € brut annuel

Travaux pratiques – TP2

- Créer la classe Contractuel
- Créer l'interface Agent avec la méthode getSalaire()
- Faire implémenter cette interface sur les classes Titulaire et Contractuel
- Implémenter la méthode getSalaire() sur ces 2 classes
- Calculer la masse salariale annuelle dans la méthode main() de la classe Test :
- Création d'un tableau d'Agent[]
- Y ajouter un titulaire et un contractuel
- Affiche sur la console la masse salariale annuelle en utilisant une boucle for sur le tableau

Travaux pratiques – TP2

- Créer la classe Abstraite Animal
 - Avec la méthode abstraite PoseBague()
 - Créer les 2 classes Poisson et Mammifere qui hérite de Animal
 - Coder les méthodes PoseBague() dans les 2 sous-classes
- Dans la classe Test
 - Créer un tableau Animal[]
 - Y ajouter des Poissons et des Mammiferes
 - En parcourant le tableau Animal[] poser les bagues qu'aux Poissons
 - Afficher le code de l'animal avec les nageoires baguées

Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- **Librairies standard**
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web

StringBuffer (java.lang)

- Version modifiable d'une chaîne de caractères

```
StringBuffer sb = new StringBuffer();  
//Méthode append  
sb.append("Hello");  
sb.append(" l'INRA !");  
//Méthode substring  
String inra = sb.substring(8,12);  
//Méthode toString();  
System.out.println(sb.toString());
```

Les collections

- Plusieurs Objets pour grouper un ensemble d'éléments

- List

- Séquence d'éléments ordonnée par indice

- Peut contenir des éléments en double

- Set

- Ensemble d'éléments uniques (pas de doublons)

- Map

- Stock des paires clé-valeur

- Pas de doublons pour les clés

Les collections

- Ce sont des Interfaces qui implémentent toutes l'interface Collection
 - Méthodes communes
 - add, get, contains, toArray, clear, ...
 - Pas instanciable, mais il existe pour chacune des implémentations
 - List : ArrayList, LinkedList
 - Set : HashSet, TreeSet
 - Map : HashMap, TreeMap

```
List<String> liste = new ArrayList<String>();  
Set<Integer> set = new TreeSet<Integer>();  
Map<Integer,String> map = new HashMap<Integer,String>();
```

Iterator

- Permet de parcourir les éléments d'une collection
- Deux méthodes principales
- next(), hasNext()

```
List<String> liste = new ArrayList<String>();
liste.add("élément 1");
liste.add("élément 2");

for (Iterator<String> iterator = liste.iterator(); iterator.hasNext();) {
    String element = iterator.next();
    System.out.println(element);
}
```

Les dates

● Date (java.util)

●● Pour stocker

●●● Permet de stocker une date complète (avec le temps)

●●● Pas de facilité pour créer une date spécifique

● Calendar (java.util)

●● Pour manipuler

●●● Possibilité d'ajout/suppression d'unité de temps

●●● Info sur les dates (ex: est-ce un week-end ?)

● DateFormat / SimpleDateFormat (java.text)

●● Pour afficher

●●● Chaîne de caractères ← Date

Les dates

```
//Création et récupération de la date du jour
Date dateDuJour = new Date();

//Création d'un calendrier
Calendar calendrier = new GregorianCalendar();
calendrier.setTime(dateDuJour); //Se positionne sur la date du jour
Calendar calendrier2 = new GregorianCalendar();
calendrier2.set(1973,1,6); //Se positionne au 6 février 1973
//Récupère le jour de la semaine :
int jourSemaine = calendrier2.get(Calendar.DAY_OF_WEEK);
//Récupère l'objet Date correspondant :
Date date2 = calendrier2.getTime();

//Création d'un formateur pour lire ou afficher une date du type 21/01/2013
SimpleDateFormat analyseur = new SimpleDateFormat("dd/MM/yyyy");
//Affiche la date
System.out.println(analyseur.format(date2));
//Analyse à partir d'une chaîne
Date date3 = analyseur.parse("21/01/2013");
```

Travaux pratiques – TP3

- Utiliser les collections pour stocker les animaux dans un élevage
 - Ajouter l'attribut `List<animal>` animaux
 - Modifier le constructeur pour le prendre en compte
 - Implémenter une méthode ajoute qui permet d'ajouter un animal à l'élevage
 - Implémenter une méthode `getAnimal` qui permet de retrouver un animal dans l'élevage à partir de son code
- Dans une classe Test
 - Créer un élevage avec 10 poissons
 - Ajouter un 11e poisson en utilisant la méthode ajoute
 - Le rechercher avec la nouvelle méthode `getAnimal`

Travaux pratiques – TP3

- Ajouter à la classe `Animal` l'attribut `DateDeNaissance`
- Prendre en compte ce nouvel attribut dans le constructeur de la classe
- Implémenter une méthode `affiche()` sur la Classe `Animal` afin d'afficher : `codeAnimal : dd/mm/yyyy`
- Dans la Classe `Test`
- Afficher la liste des animaux de l'élevage en utilisant la méthode `affiche` de la classe `Animal`

Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- Bibliothèques standard
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web

Les Exceptions

● Définition

- Une exception est un signal indiquant que quelque chose d'exceptionnelle (comme une erreur) s'est produit
- Elle interrompt le flot d'exécution normal du programme
- Cela facilite la gestion des erreurs
- Permet de séparer clairement le code d'exécution normale du code de gestion des erreurs

Les Exceptions

● Bloc *try*

- Instructions susceptibles de lever une ou plusieurs exceptions

● Bloc *catch*

- Instructions exécutés à la levée de l'exception
- Plusieurs blocs catch peuvent suivre un bloc *try*

● Bloc *finally*

- Instructions exécutés en sortie du bloc *try* avec ou sans levés d'exception
- Si exception levée, les instructions du bloc *finally* seront exécutés après les instructions du bloc *catch*

Les Exceptions

```
configureLogger();  
try {  
    traiterArguments(args);  
} catch (Exception e) {  
    logger.append("Erreur sur les arguments : " + e.getMessage());  
}  
finally {  
    closeLogger();  
}
```

Les Exceptions

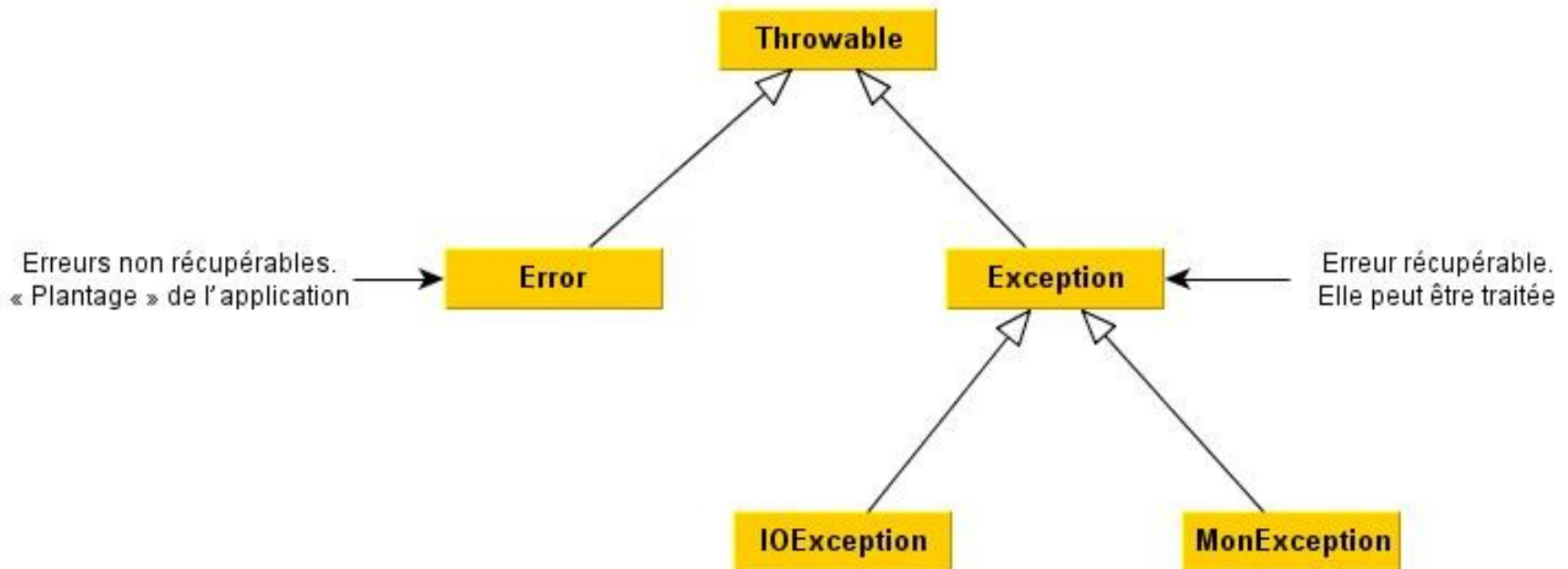
- Instructions *throw* et *throws*
 - Une méthode peut intercepter une exception et la renvoyer
 - soit telle quelle sans la modifier,
 - soit en construisant une exception d'un autre type

```
private void renvoieException() throws IOException {
    //Instructions susceptibles de lever l'exception IOException
}

private void renvoieNouvelleException() throws MonException {
    try {
        //Instructions susceptibles de lever l'exception IOException
    } catch (IOException e) {
        throw new MonException(e);
    }
}
```

Les Exceptions

- Créer son Exception
- Toute exception doit héritée de Exception, sous-classe de la classe Throwable



Les Exceptions

● Créer son Exception

```
public class AnimalDeJaPresentException extends Exception {  
  
    private Animal animal;  
  
    public AnimalDeJaPresentException() {  
    }  
  
    public AnimalDeJaPresentException(Animal animal) {  
        super();  
        this.animal = animal;  
    }  
  
    @Override  
    public String getMessage() {  
        return "Problème : "+animal.getCodeAnimal()+" est déjà présent.";  
    }  
}
```

Travaux pratiques – TP4

- Créer une Exception pour un animal non trouvé dans un élevage
 - Si l'animal n'est pas trouvé levé cette exception
 - Dans la classe Test chercher un animal absent
 - Vérifier que l'exception est bien levée

Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- Bibliothèques standard
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web

Les entrées/sorties

- Plusieurs classes du paquetage java.io permettent de gérer les flux de données en entrée et en sortie
 - sous forme de caractères (exemple fichiers textes)
 - BufferedReader, FileReader pour lire des caractères
 - BufferedWriter, FileWriter pour écrire des caractères
 - ou sous forme binaire (octets)
 - FileInputStream pour lire des octets
 - FileOutputStream pour écrire des octets

Les entrées/sorties

● Lecture/écriture caractère par caractère

```
FileWriter fileJava;
try {
    fileJava = new FileWriter("c:/temp/java.txt");
    fileJava.write("C'est facile le java.");
    fileJava.close();
} catch (IOException e) {
    e.printStackTrace();
}

FileReader fileJavaReader;
int contenu;
try {
    fileJavaReader = new FileReader("c:/temp/java.txt");
    while ((contenu=fileJavaReader.read())!=-1){
        System.out.print((char)contenu); //Affiche chacune des lettres
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

Les entrées/sorties

● Lecture/écriture ligne par ligne

```
BufferedWriter writer;
try {
    writer = new BufferedWriter(new FileWriter("C:/temp/java.txt",true)); //append=true
    for (int i = 0; i < 10; i++) {
        writer.write("Poisson"+i);
        writer.newLine();
    }
    writer.close();
} catch (IOException e) {
    e.printStackTrace();
}

BufferedReader reader;
String ligne = null;
try {
    reader = new BufferedReader(new FileReader("C:/temp/java.txt"));
    while ( (ligne = reader.readLine()) != null){
        System.out.println(ligne);
    }
    reader.close();
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
```

Les entrées/sorties

- Accès aux fichiers

- *File* donne accès aux informations d'un fichier ou d'un répertoire

```
File repertoire = new File("c:/temp/");
if (!repertoire.exists()){
    repertoire.mkdir();
}
System.out.println(repertoire.getPath());
File[] files = repertoire.listFiles(); //retourne dans un tableau les fichiers du répertoire
for (int i = 0; i < files.length; i++) {
    System.out.println(files[i].getName());
}

File fichier = new File("c:/temp/java.txt");
if (fichier.exists()) {
    fichier.delete();
} else {
    fichier.createNewFile();
}
//Création du writer à partir du File
BufferedWriter writer = new BufferedWriter(new FileWriter(fichier,true)); //append=true
writer.write("Nouveau texte");
writer.close();
```

Les entrées/sorties

- Les fichiers de propriétés
 - Système de fichier clé-valeur
 - Utilisés pour l'accès aux fichiers de configuration
 - ex :

```
annotation.processing.enabled=true
annotation.processing.enabled.in.editor=false
annotation.processing.run.all.processors=true
annotation.processing.source.output=${build.generated.sources.dir}/ap-source-output
application.title=OracleToMySQL
application.vendor=Theron
build.classes.dir=${build.dir}/classes
build.classes.excludes=**/*.java,**/*.form
# This directory is removed when the project is cleaned:
build.dir=build
build.generated.dir=${build.dir}/generated
build.generated.sources.dir=${build.dir}/generated-sources
# Only compile against the classpath explicitly listed here:
build.sysclasspath=ignore
build.test.classes.dir=${build.dir}/test/classes
build.test.results.dir=${build.dir}/test/results
```

Les entrées/sorties

● Les fichiers de propriétés

```
File file = new File("c:/temp/inra.properties");
if (!file.exists()) file.createNewFile();
FileOutputStream out = new FileOutputStream(file);
//Déclaration de l'objet Properties :
Properties props = new Properties();
//Ajout de propriétés :
props.setProperty("elevage", "LANGLADE");
props.setProperty("especes", "OVIN,LAPIN");
props.setProperty("installation", "C:/Program Files (x86)");
//Sauvegarde des propriétés :
props.store(out, "commentaire");

FileInputStream in = new FileInputStream(file);
//Chargement des données :
props.load(in);
//Accès aux données :
String elevage=props.getProperty("elevage");
String especes=props.getProperty("especes");
String tableau[] = especes.split(",");
System.out.println(elevage);
for (int i = 0; i < tableau.length; i++) {
    System.out.println(tableau[i]);
}
```

```
#commentaire
#Mon Jan 28 10:03:03 CET 2013
especes=OVIN,LAPIN
elevage=LANGLADE
installation=C\:/Program Files (x86)
```

Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- Bibliothèques standard
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web

Connexion aux BD

- JDBC : Java DataBase Connectivity
 - API permettant de se connecter aux bases de données relationnelles
 - Le package *java.sql* contient les spécifications JDBC
 - Des interfaces : *Connection*, *Statement*, *PreparedStatement*, *ResultSet*, *Driver*, ...
 - Des exceptions : *SQLException*, *SQLWarning*, *DataTruncation*, ...
 - Des classes unitaires : *Date*, *Time*, *DriverPropertyInfo*, *Types*, ...

Connexion aux BD

Utilisation

```
//1 - Chargement du pilote :
Class.forName("com.mysql.jdbc.Driver");
//2 - Ouverture d'une connexion :
Connection conn=DriverManager.getConnection("jdbc:mysql://germinal.toulouse.inra.fr/dbmanagerclass","dbman","dbman!!");
//3 - Création d'une instruction :
Statement statement = conn.createStatement();
//4a - Exécution d'une requête de mise à jour :
String queryMaj = "UPDATE ANIMAL SET ANIMAL_NOM='Blanchette' WHERE ID_ANIMAL = 1";
int rows = statement.executeUpdate(queryMaj);//retourne le nombre de lignes affectées
//4b - Exécution d'une requête de sélection :
String query = "SELECT ID_ANIMAL, ANIMAL_NOM from ANIMAL";
ResultSet resultSet = statement.executeQuery(query);
//5 - Traitement des résultats
while (resultSet.next()) //parcours des données
{
    //Attention la numérotation des colonnes commence à 1
    int idAdnimal = resultSet.getInt(1);
    String animalNom = resultSet.getString(2);
    System.out.println("Animal "+idAdnimal+" : "+animalNom);
}
//6 - Fermeture de la connexion
//but : libérer la connexion et les ressources associées
//A faire dans l'ordre :
resultSet.close();
statement.close();
conn.close();
```

Connexion aux BD

● Les différents pilotes

●● MySQL

●●● Jar : mysql-connector-java-5.1.22-bin.jar

●●● Class name : com.mysql.jdbc.Driver

●●● URL : jdbc:mysql://<hostname>[<:port>]/<dbname

●● Oracle

●●● Jar : ojdbc5.jar

●●● Class name : oracle.jdbc.driver.OracleDriver

●●● URL : jdbc:oracle:thin:[user/password]@[host][:port]:SID

Connexion aux BD

● Optimisation

- *Statement* est utilisé pour les requêtes statiques
- *PreparedStatement* permet de gérer des requêtes dynamiques

```
PreparedStatement ps = conn.prepareStatement("UPDATE ANIMAL set ANIMAL_NOM=? where ID_ANIMAL=?");  
ps.setString(1,"Blanchette");  
ps.setInt(2, 1);  
ps.executeUpdate();
```

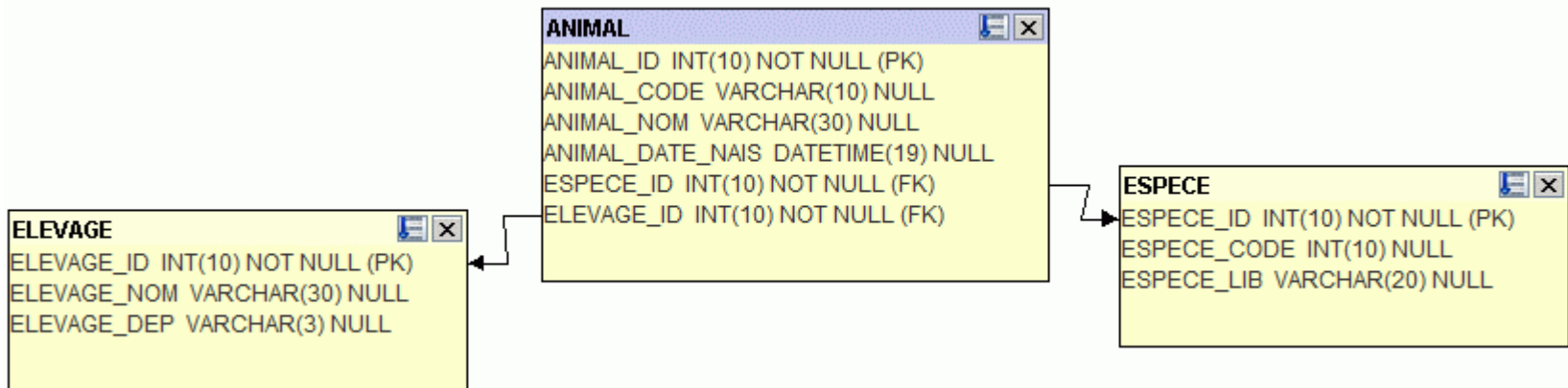
- Par défaut, une nouvelle connexion est auto-commit

```
connection.setAutoCommit(false); //permet de changer ce mode  
//Gestion manuel du commit :  
connection.commit();  
connection.rollback();
```

Travaux pratiques – TP5

- Créer un fichier texte contenant une liste de mouton
- Dans la méthode main de la classe Test créer ce fichier

```
Mouton Margueritte  
Mouton Margau  
Mouton Mirtille  
Mouton Michele  
Mouton Martine
```



Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- Bibliothèques standard
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web

IHM Windows avec Swing

- SWING
- Framework graphique
- Permet de dessiner des fenêtres, des boutons, des arbres, des listes, ...

IHM Windows avec Swing

● Les fenêtres

●● JWindow

●●● Fenêtre basique. Pas de menu, titre, ...

●●● Utilisé pour faire des fenêtres d'attente

●● JDialog

●●● Fenêtre pour les boites de dialogue

●●● Peut être modale

●● JFrame

●●● Fenêtre principale de l'application

●●● Possède barre de titre

●●● Peut accueillir des menus, des labels, des boutons ...

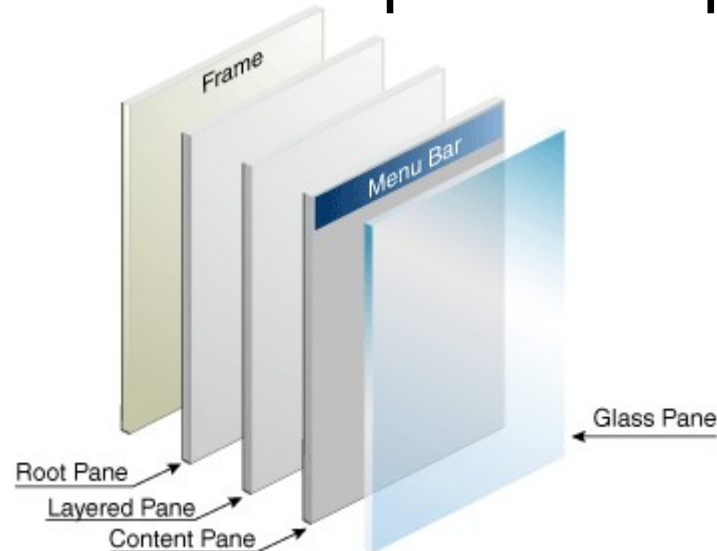
IHM Windows avec Swing

● Création d'une JFrame

```
final JFrame fenetre = new JFrame();
fenetre.setTitle("Ma première fenêtre"); //On donne un titre à l'application
fenetre.setSize(640,240); //On donne une taille à notre fenêtre
fenetre.setLocationRelativeTo(null); //On centre la fenêtre sur l'écran
fenetre.setResizable(false); //On interdit la redimensionnement de la fenêtre
fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //On dit à l'application de se fermer lors du clic sur la croix
fenetre.setVisible(true);
```

IHM Windows avec Swing

- Une fenêtre est composée de plusieurs parties



- RootPane : container principal, c'est lui qui contient les autres
- LayeredPane : forme juste un panneau composé du ContentPane et de la barre de menu (MenuBar)
- GlassPane : couche par dessus le tout utilisée pour intercepter les actions de l'utilisateur avant qu'elles ne parviennent aux composants
- ContentPane : composant qui contient la partie utile de la fenêtre, c'est-à-dire, celle dans laquelle on va afficher nos composants.

IHM Windows avec Swing

- Les **composants** sont destinés à afficher du texte, à permettre à l'utilisateur de saisir du texte, à afficher un bouton, ...
- On ne peut pas ajouter un composant directement sur la fenêtre
- Les **conteneurs** sont destinés à accueillir les composants et d'autres conteneurs
- C'est les conteneurs qui seront ajoutés à la fenêtre

IHM Windows avec Swing

● Création d'une JPanel (conteneur)

```
JPanel panel = new JPanel(); //Création du conteneur
panel.setLayout(new FlowLayout()); //FlowLayout : type d'organisation des composants
panel.setBackground(Color.white); // Définit la couleur de fond
fenetre.setContentPane(panel); //Affectation du layout à la fenêtre
```

● Ajout d'un composant

```
//Composant label
JLabel label = new JLabel("Bienvenue sur ma première fenêtre");
panel.add(label);

//Composant zone de texte
final JTextField jtf = new JTextField();
jtf.setColumns(10);
panel.add(jtf);

//Composant bouton
JButton boutonFermer = new JButton("Fermer");
boutonFermer.addActionListener(new ActionListener() {
    //Evènement déclenché lors du clic sur le bouton
    @Override
    public void actionPerformed(ActionEvent e) {
        fenetre.dispose();
    }
});
panel.add(boutonFermer);
```

IHM Windows avec Swing

● Utilisation du MigLayout

```
MigLayout layout = new MigLayout("fill,wrap 3"); //MigLayout : type d'organisation des composants
//fill : permet de répartir tout l'espace entre les différents composants aussi bien en hauteur qu'en largeur
//wrap 3 : permet un retour à la ligne après l'ajout de 3 composants
panel.setLayout(layout);

//Composant label
JLabel label = new JLabel("Bienvenue sur ma première fenêtre");
panel.add(label,"span 3, align center");
//span 3 : indique que le composants s'étend sur 3 colonnes
//align center : indique que le composant sera centré

JLabel labelSaisir = new JLabel("Saisir le texte :");
panel.add(labelSaisir,"align right");
//align center : indique que le composant sera aligné à droite

//Composant zone de texte
final JTextField jtf = new JTextField();
jtf.setColumns(10);
panel.add(jtf,"growx");
//growx : permet au composant d'utiliser tout l'espace qui lui est alloué horizontalement
//growy : permet au composant d'utiliser tout l'espace qui lui est alloué verticalement
//grow : dans les 2 sens

JButton bouton2 = new JButton("Efface");
bouton2.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        jtf.setText("");
    }
});
panel.add(bouton2,"align center");

JButton boutonFermer = new JButton("Fermer");
boutonFermer.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        fenetre.dispose();
    }
});
panel.add(boutonFermer,"span 3, align center");
```

Formation JAVA

- Généralités
- Notion de base du langage
- Programmer Objet en Java
- Bibliothèques standard
- Les erreurs et exceptions
- Les entrées-sorties
- Connexion aux bases de données
- IHM Windows avec Swing
- Java et le Web



Java et le Web

● Les JSP

- JSP : Java Server Pages

- Technologies qui permet la génération de pages web dynamiques

- Code java dans des tags prédéfinis à l'intérieur d'une page HTML

- Fichier avec extension .JSP

Java et le Web

● Les Tags de directives

●● <%@page ... %>

●●● Permet de définir les options qui s'appliquent à toute la JSP

●●● En principe, placée en début de fichier

●●● S'écrit sous la forme : option=valeur

●● <%include ... %>

●●● Utile pour insérer un élément commun (ex: en-tête)

●● <%! ... %> , <%= ... %> et <% ... %>

●●● Permet d'insérer du code java

●●● <%! déclarations %>

●●● <%= expression %>

●●● <% code Java %>

●● <!-- ... --> OU <%!-- ... %>

●●● Permet d'insérer des commentaires

Java et le Web

● Les Tags de directives

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>GlassFish JSP Page</title>
  </head>
  <body>
    <%@include file="entete.htm"%>
    <h1>Hello World!</h1>
    <%! Date dateDuJour; %>
    <% dateDuJour = new Date();%>
    <p align="center">Date du jour : <%= dateDuJour %></p>
    <!-- Cette page a ete generee le <%= new Date() %> et c'est un commentaire généré dans le HTML-->
    <!-- Commentaire cachés : non généré dans le HTML --%>
  </body>
</html>
```

Java et le Web

● Les Tags useBean, setProperty, getProperty

```
<jsp:useBean id="animal" scope="session" class="fr.inra.formation.business.Animal" />
<p>nom initial = <%=animal.getNom() %></p>
<%animal.setNom("BLANCHETTE");%>
<p>nom mis à jour une fois = <%= animal.getNom() %></p>
```

```
<jsp:setProperty name="animal" property="nom" value="NOIRAUDE" />
<p>nom mise à jour 2e fois = <jsp:getProperty name="animal" property="nom" /></p>
```

● <jsp:include>

●● Permet d'inclure le contenu généré par une jsp

●●● <jsp:include page=« relativeURL » />

Java et le Web

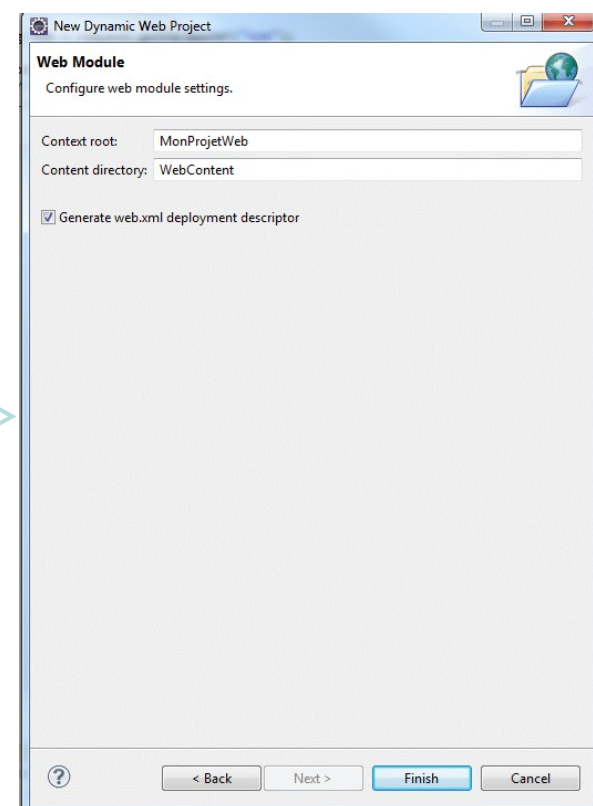
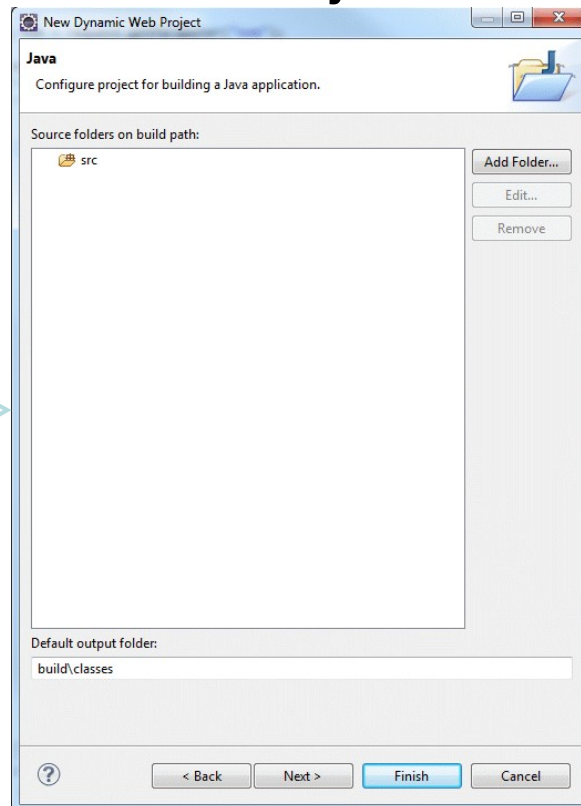
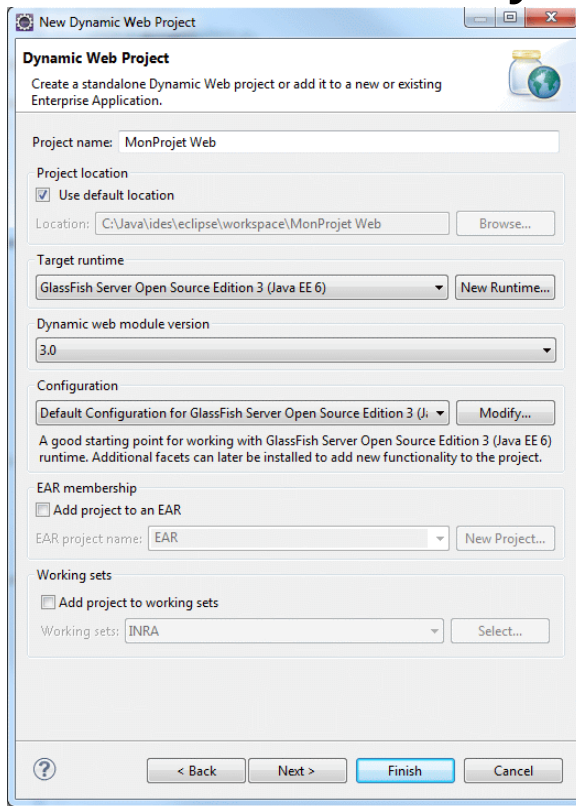
● Un exemple simple

```
<FORM METHOD=POST ACTION="TestJSPAccueil.jsp">
Entrez votre nom :
<INPUT TYPE=TEXT NAME="nom">
<INPUT TYPE=SUBMIT VALUE="OK" >
</FORM>
```

```
<HTML>
<HEAD>
<TITLE>Accueil</TITLE>
</HEAD>
<BODY>
<%
String nom = request.getParameter("nom");
%>
<H2>Bonjour <%= nom %></H2>
</BODY>
</HTML>
```

Java et le Web

- Créer un projet Java Web avec Eclipse
- File → New → Project...
- Web → Dynamic Web Project





Java et le Web

- Pour aller plus loin...

- Cours sur les JSP :

- <http://jmdoudoux.developpez.com/cours/developpons/java/chap-jsp.ph>

- Struts

- Framework qui enrichit les tags JSP pour faciliter le développement

- Cours :

- <http://jmdoudoux.developpez.com/cours/developpons/java/chap-struts.ph>