# INF2 JAVA – Travaux Pratiques

E. Bruno

19 octobre 2011

# Table des matières

# Chapitre 1

# TP

La correction des TP se trouve sur le svn suivant : https ://lsis.univ-tln.fr/svn/bruno/enseignement/INF2TP/trunk/INF2TP/

La première fois que vous voulez récupérer la correction utiliser la commande suivante ( **ATTENTION cela va créer un répertoire INF2TP dans votre répertoire courant** ).

```
svn checkout https://lsis.univ-tln.fr/svn/bruno/enseignement/INF2TP/trunk/INF2TP/
```

Pour vérifier si les corrections ont été mises à jour et mettre à jour votre copie locale lancer la commande suivante dans le répertoire `INF2TP` :

```
svn update
```

Un exemple d'application se trouve sur le svn suivant :

```
http://lsis.univ-tln.fr/svn/bruno/enseignement/MonEntreprise/branches/experimental/
```

faire un `svn checkout` puis éventuellement des `svn update`

## 1.1 TP1 - Premiers pas en Java

### 1.1.1 Objectif

Les objectifs de ce premier TP sont la mise en place de l'environnement de travail en java et l'étude des fondements du langage. C'est-à-dire d'une part la compréhension de la compilation et de l'exécution de code java, l'arborescence classique d'une application et la génération de la documentation et d'autre part la maîtrise de la création de classes et d'objets.

### 1.1.2 Une première exécution

**Mettre en place l'environnement**

La première chose à faire est de fixer quelle distribution du jdk va être utilisée, pour cela fixer la valeur de la variable d'environnement `JAVA_HOME` pour indiquer son emplacement (`/usr/lib/jvm/java-6-sun`). Ensuite, on ajoute le répertoire `$JAVA HOME/bin` à la valeur de la variable `PATH` pour que les commandes de bases soient accessibles. Vérifier que cela fonctionne en exécutant les commandes `java -version` et `javac -help` (il doit s'agit de la JVM de Sun).

Il est impératif que toutes les documentations suivantes soient ouvertes dans votre navigateur web :

– la documentation sur les API [1] : http ://java.sun.com/javase/6/docs/api/

– la documentation sur les outils Java fournis par Sun : http ://java.sun.com/javase/6/docs/

1. Application Programming Interface

**Compiler et exécuter une application**

A la racine de votre compte ajouter un répertoire `INF2` (ce répertoire sera appelé répertoire de travail). A l'intérieur de votre répertoire de travail ajouter le répertoire `TP` (appelé répertoire de projet). A l'intérieur de votre répertoire de projet ajouter les répertoires standards `src`, `build`, `doc`, `dist` et `lib`.

Dans le répertoire `src`, créer l'arborescence correspondant au paquetage : `fr.univtln.login.tp.tp1`. A l'intérieur de ce paquetage ajouter le programme Java PremierProgramme.java.

Ouvrir et modifier ce programme pour que l'instruction `package` soit correcte.

A partir de votre répertoire de projet, compiler le programme avec la commande suivante :

```
javac -sourcepath src -d build src/fr/univtln/login/tp/tp1/PremierProgramme.java
```

Regarder dans le répertoire `build`. Pour exécuter le programme, depuis votre répertoire de projet :

```
java -classpath build fr.univtln.login.tp.tp1.PremierProgramme Pierre
```

### 1.1.3 Générer la documentation

**Précision sur la forme**

La commande javadoc produit de la documentation en partant de commentaires particuliers insérés dans le code source des classes (/** ... */). On peut ainsi documenter les paquetages, classes ou interfaces, variables d'instance, méthodes,...

Les commentaires peuvent contenir du texte simple et des balises HTML [2] de mise en forme de texte (<I> italique, <B> caractère gras, ... ). On peut utiliser la balise <code> pour inclure du code dans les commentaires. Des balises spéciaux appelées annotations qui commencent par le caractère @ (@author, @version, @param,... ) sont définies pour fixer des valeurs standards. Les commentaires doivent être placées juste avant ce qu'ils commentent. Lire cette page : http ://java.sun.com/javase/6/docs/technotes/tools/solaris/javadoc.html

De plus, il est rappelé que le langage Java est entouré d'un ensemble de bonnes pratiques. Cette page présente les habitudes dans le cadre du langage java. Lire cette page : http ://www.oracle.com/technetwork/java/javase/documentation/index-13

Ajouter les commentaires pour les packages (cf. `package-info.java`)

**Génération**

A partir de maintenant, tous vos programmes seront commentés. Générer la documentation depuis votre répertoire de projet avec la commande :

```
javadoc fr.univtln.tp.tp1 src/fr/univtln/bruno/tp/tp1/PremierProgramme.java -d doc
```

### 1.1.4 Générer et exécuter l'archive jar

Génerer l'archive de votre projet depuis votre répertoire de projet avec la commande :

```
cd build; jar cvf ../dist/tp.jar fr; cd ..
```

Exécuter le programme en ajoutant le jar au classpath et en indiquant la classe exécutable avec la commande suivante :

```
java -classpath dist/tp.jar fr.univtln.login.tp.tp1.PremierProgramme Pierre
```

Il est possible de rendre le jar "exécutable" en créant un fichier `manifest` qui indique en particulier quelle est la classe exécutable.

Copier le fichier `monManifest` (monmanifest.zip) dans le répertoire de projet et regarder son contenu. Recréer le fichier `.jar` en précisant le manifeste à ajouter :

```
cd build; jar cvfm ../dist/tp.jar ../monManifest fr; cd ..
```

Puis exécuter directement l'archive :

```
java -jar dist/tp.jar Pierre
```

2. HyperText Markup Language

### 1.1.5 Automatiser le processus avec Apache Ant

Comme vous l'avez vu, la compilation d'un projet complet en Java est une tâche fastidieuse. C'est pourquoi, il est conseillé d'utiliser un outil qui automatise ce travail : Ant (http ://ant.apache.org/). Ant utilise un fichier de configuration qui indique les tâches à accomplir, vous trouverez un exemple de ce fichier sur le wiki build.xml.zip.

Copier ce fichier dans votre répertoire de projet et regarder son contenu. Adapter son contenu pour qu'il indique des chemins corrects.

Créer une variable d'environnement `ANT_HOME` qui indique le répertoire d'installation de ant (`/usr/share/ant`). Ajouter `ANT_HOME/bin` dans le `PATH`.

Pour obtenir l'ensemble des tâches possibles avec un fichier de construction :

```
ant -projecthelp build.xml
```

Pour lancer l'exécution et si nécessaire compiler et générer l'archive :

```
ant run
```

### 1.1.6 Utilisation de l'environnement Eclipse

Nous allons maintenant utiliser l'environnement de développement eclipse : `http://www.eclipse.org`. Comme "workspace" vous pouvez choisir votre espace de travail (c'est là que eclipse stocke ses informations de configuration).

Créer un nouveau projet à partir d'un fichier ant (cf. menu file). Modifier les propriétés du nouveau projet (clic droit sur le projet). Dans "Java compiler" indiquer la compatibilité avec Java 5 (ou 6). Dans les propriétés des chemins vérifier que les répertoires des sources et des binaires sont corrects.

Dans les propriétés du paquetage `JRE_LIB` indiquer où se trouve l'archive de la Javadoc.

Pour lancer une tâche ant, utiliser le menu de droite. Vous pouvez maintenant éditer vos programmes Java.

### 1.1.7 Les concepts de base de Java

#### Création et instanciation d'une classe

Créer une classe Java exécutable nommée `Test` qui appartient au paquetage 'tp1'. La classe 'Test' devra afficher `"Hello"`.

Créer une classe `Personne` sans constructeur explicite qui décrit une personne ayant un nom, un prénom, un âge et un salaire. Créer les accesseurs correspondants à ces attributs (eclipse peut le faire automatiquement). Vous vérifierez qu'un salaire ne peut pas être négatif. Instancier une personne `p1` dans le `main` de la classe `Test`, mettre à jour ses informations et les afficher (`Pierre Truc est agé de 30 ans et gagne 2000€`).

Ajouter des constructeurs dans la classe Personne qui initialisent le nom et le prénom et éventuellement l'âge et la profession. Modifier cette classe pour que le nom et le prénom ne puissent plus être modifiés après l'instanciation.

Ajouter un attribut qui indique l'année de naissance et les accesseurs associés. Modifier la méthode que retourne l'âge. L'année courante s'obtient avec `java.util.Calendar.getInstance().get(Calendar.YEAR)`.

Ajouter une méthode `comparerSalaire()` qui prend en paramètre une personne et retourne -1, 0 ou 1 selon que celle à un salaire supérieur, égal ou inférieur à l'instance courante. Créer une deuxième personne `p2` née la même année que `p1` mais qui gagne plus. Afficher le résultat de la comparaison des salaires de `p1` et `p2`.

Ajouter une méthode `comparerSalaire()` qui prend en paramètre deux personnes et qui retourne -1, 0 ou 1 selon de le salaire de la première est inférieur, égal ou supérieur à celui de la seconde.

Attention `comparerSalaire(Personne)` et `comparerSalaire(Personne, Personne)` sont-elle des méthode de classe ou d'instance ?

Ajouter un attribut de classe `totalDesSalaires` qui indique le total des salaires des personnes et un accesseur. Modifier les méthodes nécessaires pour le salaire total soit maintenu à jour automatiquement.

Afficher le salaire total après la création de chaque personne.

#### Les méthodes de base

**toString()** Afficher directement l'objet `p1` dans la classe `Test`. Ajouter une méthode : `String toString()` à la classe personne qui retourne une chaîne de caractère qui décrit la personne. Exécuter `Test`.

**equals()** On souhaite considérer que deux personnes sont ”égales” dans notre application, si elles sont nées la même année. Pour définir une relation d'égalite d'objets en Java on redéfinit la méthode `Boolean equals(Object)`. Vérifier que `p1` et `p2` sont égales.

#### Définition de classes locales

Pour représenter le fait qu'une Personne à un cerveau ajouter une définition de la Classe `Cerveau` à l'intérieur de la définition de la classe `Personne`. Ajouter un attribut `cerveau` à `Personne` et instancier un cerveau à chaque personne lors de sa création. Compiler et exécuter `Test`. Regarder dans le répertoire `build` les classes qui ont été compilées. Essayer de créer une instance de `Cerveau` directement dans la classe `Test`.

### 1.1.8 Pour finir…

Vous allez maintenant créer une classe `Entreprise` qui permet de représenter des entreprises qui comportent au maximun `MAX_EMPLOYES Personnes`.

Vous ajouterez les méthodes pour créer une entreprise, ajouter des employés et afficher le nombre d'employés, un employé précis et tous les employés (en utilisant le `foreach` de Java5).

Les entreprises doivent appartenir à l'une des trois catégories suivantes : `PUBLIQUE, PME, GRAND GROUPE`. Modifier la classe `Entreprise` pour représenter cela en utilisant un type `enum` et ajouter trois méthodes `isPUBLIQUE()`, `isPME()`, `isGRAND_GROUPE()` qui retournent un booléen.

Ajouter une méthode `miseEnForme()` qui passe le nom de toute les personnes d'une entreprise en majuscule.

Modifier votre classe `Test`, pour l'on puisse lui passer en paramètre le nom et le type d'une entreprise, puis la liste des personnes qui la compose.

```
java Test maBoite PME Pierre Durand 1950 2000 Paul Dupond 1960 3000 Marie Martin 1975 2500
```

## 1.2 TP2 - Héritage, polymorphisme et interfaces en Java

### 1.2.1 Objectif

L'objectif ce TP est de présenter l'héritage, le polymorphisme et l'utilisation des interfaces en Java. Vous devez écrire la javadoc http ://java.sun.com/j2se/javadoc/writingdoccomments/ au fur et à mesure de l'avancement et il est **fortement** conseillé d'avoir le documentation de java sous les yeux : http ://java.sun.com/j2se/1.5.0/docs/api/index.html

### 1.2.2 Appel d'une méthode la classe mère

Dans cet exercice nous allons modéliser des `plantes`, des `animaux`, des `mammifères` et des `oiseaux` mais aussi des `chiens`, des `aigles`, des `lapins` et des `hommes`.

#### Modélisation par héritage et Instanciation

- Modélisez cette hiérarchie (directement en Java) avec des classes. Ajoutez à chaque classe fille au moins une variable et une méthode nouvelle selon votre imagination (âge, nom, sexe, …).
- Ajoutez une classe exécutable `Test` et créez des instances de chacune des classes. Affichez avec la méthode `System.out.prinln()` chacun des objets.
- Redéfinissez la méthode `toString()` (héritée de `Object`, voir dans la documentation) de la super classe pour qu'elle affiche ”`Je suis un animal`” (ou ”`Je suis une plante`”). Exécutez la classe `Test`. Expliquer le résultat en étudiant la classe `java.lang.System`.
- Modifiez la méthode précédente pour qu'elle affiche ”`Je suis un animal et mon identifiant est X`”.

#### Polymorphisme simple

- Avec quels types de références pouvez-vous manipuler les classes précédentes ?
- Dans la méthode `main` la classe `Test`, créez des tableaux qui permettent de voir certains des animaux crées **dans la section précédente** comme des collections (de taille fixe) d'animaux ou de mammifères.

– Ajouter à chaque classe une méthode `String getInfo()` (qui appellera celle de la superclasse) pour que l'affichage du tableau suivant :

```
1 {new Animal(12), new Animal(), new Chien(5,"Medor"), new Homme(), new Homme(25,"Robert")}
```

soit de la forme :

```
Je suis un animal âgé de 12 an(s).
Je suis un animal.
Je suis un animal âgé de 5 an(s). Je suis un mammifère. Je suis un chien de nom Médor.
Je suis un animal. Je suis un mammifère. Je suis un homme.
Je suis un animal âgé de 25 an(s). Je suis un mammifère. Je suis un homme de nom Robert.
```

**Classes abstraites**

– Modifiez les classes pour s'assurer que tous les Animaux possèdent la méthode `String moyenExpression` qui retourne une phrase du type "`Je parle`", "`J'aboie`", ou "`Je fais des bulles`" (l'aigle trompette, glapit ou glatit ; le lapin clapit).
– Est-ce que le moyen d'expression peut être définit pour toutes les classes ? Peut-on alors créer une instance de la classe Animal, quelle est votre conclusion pour s'assurer de cela ? Faites de même pour toutes les classes pour les lesquelles cela vous semble nécessaire.
– Écrivez une méthode `afficherAnimaux()` dans la classe `Animal` qui prend en paramètre un tableau d'animaux et qui affiche leur référence et leur cri. Testez la sur le tableau de la question précédente.

**Administration de classe et polymorphisme**

– Modifiez vos classes pour que l'on garde automatiquement des références (dans un tableau) vers tous les animaux créés (Vous vérifierez aussi que l'on ne crée pas plus de `NB_MAX_ANIMAUX`).
– Créez une méthode (de classe ou d'instance ?) qui affiche la description complète de tous les animaux crées.

**Héritage multiple et Interface**

– On souhaite maintenant considérer que certains animaux (pas forcément tous) sont des carnivores (ils comportent la méthode `void manger` avec comme paramètre un autre animal) ou des herbivores (ils comportent la méthode `void manger` avec comme paramètre une plante) ou les deux. Créez deux interfaces pour décrire ces comportements
– Modifiez vos classes pour qu'elles implantent ces interfaces :
  – Les carnivores : quand ils mangent, les chiens affichent "`Je mord X`", et les aigles affichent "`Je déchire X`".
  – Les herbivores : les vaches affichent "`Je broute X`", et les lapins "`Je grignotte X`".
– Les hommes sont à les fois carnivores et herbivores, proposez deux solutions pour le représenter :
  – Dans un premier temps, sans créer de nouvelle interface.
  – Dans un second temps en créant une nouvelle interface `IOmnivore`.
– Le polymorphisme est aussi utilisable avec les interfaces :
  – Appliquez la méthode `manger(unePlante` sur la `Vache` que avez déjà créée en la voyant à travers une référence de type `IHerbivore`. Peut-on appliquer la méthode `moyenExpression()` sur cette référence ?
– Créez deux tableaux `lesCarnivores' et` lesHerbivores" pour manipuler les animaux existants. Créez un lapin, et faites le manger par tous les carnivores, créez une plante et faites la manger par les herbivores.

### 1.2.3 Si vous avez déjà fini...

– Vous pouvez construire depuis eclipse le diagramme UML correspondant à ce TP en utilisant topcased.

## 1.3 TP3 - Les collections en Java

### 1.3.1 Objectif

L'objectif ce TP est de présenter d'une part l'utilisation des collections et d'autre part l'utilisation d'un outils d'édition de diagrammes UML permettant la génération de code Java (Topcased).

### 1.3.2 Utilisation "Simple" des collections

Vous utiliserez les classes du TP précédent.

- Créez un **ensemble** d'animaux (sans utiliser de génériques) et peuplez le en utilisant concrètement un `HashSet`.
- On suppose que tous les animaux ont un nom, indiquez que deux animaux sont égaux s'ils ont le même nom.
- Vérifier que l'on ne puisse pas ajouter deux animaux ayant le même nom (par définition d'un ensemble)
- Affichez cet ensemble en utilisant un itérateur puis avec le foreach de Java 5
- Transformez le support physique de cet ensemble en un `TreeSet`, que faut-il modifier d'autre ?
- Créer une liste d'animaux à partir des animaux de l'ensemble.
- Afficher la liste des animaux triée par l'ordre alphabétique de leur nom.
- Convertissez cet ensemble en un tableau d'animaux (Faites attention au type)
- Modifier cet ensemble en ajoutant la généricité et définissez un ensemble d'animaux (supprimez les transtypages devenus inutiles).
- On suppose que l'on associe à chaque animal un tatouage unique composé de son espèce (une Chaîne de caractères) et d'un entier. Le couple des deux est unique. Ecrire la classe tatouage, surcharger et redéfinir les méthodes nécessaires. Créer une `Map` (avec la généricité) contenant des animaux (*(Aigle,1), (Aigle,2), (Chien,1), (Chien,2)*) et affichez les informations concernant l'animal dont l'identifiant est *(Chien,1).*

### 1.3.3 Ecriture manuelle d'un programme à partir d'un diagramme de classe

L'objectif est d'écrire un programme de gestion d'une bibliothèque qui prête des livres et des ordinateurs à des enseignants et des étudiants.

#### Architecture de base

Ecrire (sans utiliser d'éditeur UML) les classes Java correspondant au diagramme UML donné décrivant une bibliothèque simple. Vous veillerez en particulier à

- redéfinir la méthode `toString()` des différentes classes pour que celles-ci puissent être affichées
- représenter correctement les associations multiples en utilisant des collections (il est conseillé d'utiliser les génériques).
- créer une classe exécutable `Test` dans laquelle vous pourrez :
  - créer une bibliothèque
  - ajouter des adhérents qui doivent avoir un nom, un prénom et un statut
  - ajouter au moins deux livres, deux revues, deux dictionnaires et deux BD
  - ajouter deux ordinateurs portables (un sous linux, un sous windows)

#### Gestion des emprunts

Ecrire le code permettant de gérer les emprunts en ajoutant aux adhérents les méthodes `boolean emprunter(Empruntable e)` et `boolean rendre(Empruntable e)`. Pensez à bien vérifier qu'un objet n'est pas emprunté deux fois (vérifier que cela marche en faisant un test dans la classe `Test`) et qu'un adhérent n'emprunte pas plus de cinq objets.

- Ajouter une méthode `void afficherEmprunts()` à la classe `Adherent`
- Ajouter une méthode `void afficherFonds()` à la classe `Bibliothèque` qui affiche les documents et le matériel (chacun affichant par qui il a été emprunté). **Ajoutez les méthodes qui vous semblent utiles pour cela**
- Surcharger les méthodes `emprunter` et `rendre` pour les Livres et les Portables, pour vérifier que l'on emprunte au plus cinq livres mais un seul portable.

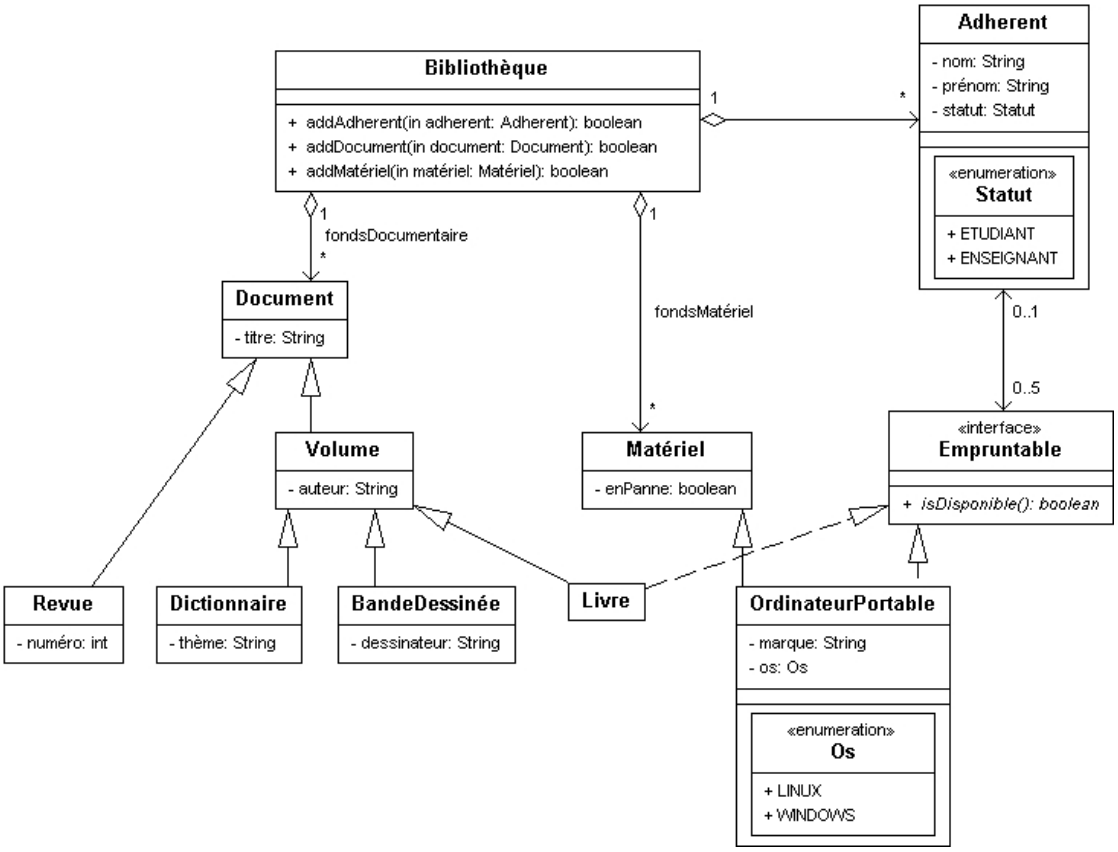


FIGURE 1.1 – fr.univtln.bruno.tp.tp3.Bilbio.eps

**Recherche dans une collection**

Ajouter une méthode `Collection rechercherTitre(String mot)`, qui retourne une collection contenant des références vers les documents qui comportent la chaîne `mot` dans leur titre.

### 1.3.4 Ecriture assistée d'un programme à partir d'un diagramme de classe

En utilisant un outils d'édition de diagramme UML, construisez le diagramme précédent et générer automatiquement le squelette des classes.

### 1.3.5 Si vous avez fini...

Vous pouvez affinez vos connaissances en suivant le tutorial de Sun : http ://java.sun.com/docs/books/tutorial/collections/index.ht
et la documentation http ://download-llnw.oracle.com/javase/6/docs/technotes/guides/collections/overview.html.

## 1.4 TP4 - Entrées/Sorties et IHM

### 1.4.1 Objectif

L'objectif ce TP est de d'illuster l'utilisation simple des entrées/sorties en Java et de mettre en place des IHM de base.

### 1.4.2 Les entrées/sorties

http ://download.oracle.com/javase/tutorial/essential/io/index.html
En vous servant tu TP sur les animaux :

- Ajouter une méthode `void sauverChien(Collection c, String fichier)` et une méthode `sauverChien(OutputStrea os)` à la classe Chien qui sauvegarde l'état interne d'un chien dans un fichier en écrivant les types primitifs directement en binaire dans un fichier.
- Ajouter une méthode `Collection restaurerChien(String fichier)` à la classe Chien qui crée une collection d'instances de Chien à partir des informations stockées dans un fichier.
- Modifier les méthodes précédentes pour qu'un tampon soit utilisé.
- Modifier les méthodes précédentes pour que le fichier soit compressé.
- Créer les méthodes `void sauverChien2(Collection c, String fichier)` et `Collection restaurerChien2(String fichier)` qui font la même chose en utilisant la sérialisation.

### 1.4.3 Les IHM

http ://download.oracle.com/javase/tutorial/uiswing/

#### Construction

Mettez en place une IHM simple qui permet d'afficher et de modifier une instance de la classe `Chien`. Essayer d'utiliser un maximum de composants de Swing (cf. tutoriel). Cette classe sera appelée une vue de Chien.

Compléter l'interface pour qu'elle affiche une "liste de Chiens" représentés par leur nom ou leur tatouage (i.e. leur identifiant).

#### Activation

Compléter votre interface pour qu'elle puisse :

- d'afficher les informations qui viennent d'une instance Chien en utilisant en utilisant un contrôleur de Chien.
- permettre de sélectionner une chien dans le liste et d'afficher des informations (bouton afficher).
- permettre de créer des instances de `Chien` à partir des informations et d'afficher les collections produites (bouton créer).

#### Editeur d'IHM

Pour aller plus loin, vous pouvez utiliser un générateur d'IHM http ://code.google.com/intl/fr/javadevtools/wbpro/quick\textunde (Il faut installer un plugin sous eclipse).

## 1.5 TP5 - Apprendre par vous même l'introspection

- Etape 1 : Suivez et expérimentez le tutorial sur l'introspection (trails Classes et Member).
- Etape 2 : Qu'est-ce qu'un Java bean ?
- Etape 3 : Compilation depuis un programme. Etudiez Janino (Y-a-t-il une alternative ?).
- Des exemples : introspection.java.tgz (NB ces exemples seront bientôt disponible sur le SVN).

## 1.6 TP6 - JDBC

### 1.6.1 Utilisation simple de JDBC

1. En utilisant pgadmin3 ou la ligne de commande et le script journal.zip créer les tables qui représentent des journaux.
2. Créer une classe que se connecte à votre base de donnée postgresql. Pensez à mettre en place correctement les librairies nécessaires. Pour le moment, la connection sera représentée par une variable d'instance.
3. Ajouter une méthode qui affiche la liste des journaux en utilisant un Statment
4. Créer une fonction ajouterJournal(...) qui permet d'insérer un journal dans la base de données
5. En utilisant un PreparedStamement, lire un code de Journal au clavier et afficher les informations correspondantes tant que l'on ne rentre pas 0.

### 1.6.2 Utilisation des métadonnées

1. Ajouter une méthode qui affiche les métadonnées correspondant à la base de données
2. Ajouter une méthode qui afficher les métadonnées correspondant à la requête `select * from journal`

### 1.6.3 Batch Update

1. Utiliser les batch update pour ajouter des journaux en lisant un fichier texte formaté (`code_j:titre:prix:type:periode::ac`

### 1.6.4 Utilisation d'un pool de connexion

En utilisant la classe suivante, utiliser maintenant un pool de connexion au lieu d'une variable globale.

```
1  package fr.univ_tln.bruno.mycompany.jdbc.pool;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.SQLException;
6  import java.util.LinkedList;
7  import java.util.Queue;
8
9  /**
10  * The Class DatabaseManager.
11  */
12 public class DatabaseManager {
13
14   /** The Constant freeConnections. */
15   private static final Queue<Connection> freeConnections = new LinkedList<Connection>();
16
17   /** The Constant numberOfInitialConnections. */
18   private static final int numberOfInitialConnections = 5;
19
20   /** The Constant password. */
21   private static final String password = System
22       .getProperty("MyCompany.database.password");
23
24   /** The Constant url. */
25   private static final String url = System
26       .getProperty("MyCompany.database.url");
27
28   /** The Constant user. */
29   private static final String user = System
30       .getProperty("MyCompany.database.user");
31
32   static {
33     for (int i = 0; i < numberOfInitialConnections; i++) {
34       try {
35         freeConnections.add(DriverManager.getConnection(url, user,
36             password));
37       } catch (SQLException e) {
38         // TODO Auto-generated catch block
39         e.printStackTrace();
40       }
41     }
42   }
43
44   /**
45    * Gets the connection.
46    *
47    * @return the connection
48    *
49    * @throws SQLException
50    *             the SQL exception
51    */
52   public static synchronized Connection getConnection() throws SQLException {
53     Connection connection = null;
54     if (freeConnections.isEmpty()) {
55       connection = DriverManager.getConnection(url, user, password);
56     } else {
57       connection = freeConnections.remove();
58     }
59     return connection;
60   }
61
62   /**
63    * Release connection.
64    *
65    * @param connection
66    *            the connection
```

```
67      */
68     public static synchronized void releaseConnection(Connection connection) {
69       if (freeConnections.size() < numberOfInitialConnections) {
70         freeConnections.add(connection);
71       } else {
72         try {
73           connection.close();
74         } catch (SQLException e) {
75           e.printStackTrace();
76         }
77       }
78     }
79 }
```

Attention, les login et password doivent être définit pas des ”properties” (cf `System.setProperty()`). Ensuite, ils pourront être lus dans un fichier de configuration.

### 1.6.5 Pour finir

1. Créer une interface `Entity` qui impose les méthodes `create`, `remove`, et `merge`
2. Créer une classe qui spécialise la classe Chien en implante l'interface précédente et qui permet de
   (a) Créer une instance de Chien
   (b) D'invoquer `create` pour ajouter ce Chien dans la base de données
   (c) De modifier l'instance de Chien en mémoire (éventuellement plusieurs fois)
   (d) De faire une mise à jour dans la base de données avec `merge`
   (e) Et finalement de supprimer le Chien avec remove.
3. Implanter des méthodes qui permettent de retrouver et d'instancier un ou des Chiens :
   (a) Chien findById(...)
   (b) List<Chien> findByName(...), ...
4. En vous inspirant de cet exemple http ://onjava.com/pub/a/onjava/excerpt/swinghks\textunderscorehack24/index.html, créer une JTable qui permet d'afficher les Chiens contenus dans la base de données.