SCIENCES SORBONNE UNIVERSITÉ

SRCS : Systèmes Répartis Client/Serveur

TME 8 préparatoire : Installation et prise en main de JavaEE sous Eclipse

Objectifs pédagogiques :
- **Installation et configuration d'un serveur d'application JavaEE**
- **programmation sur Eclipse**

Prérequis

S'assurer que le `.bashrc` définisse les variables d'environnement `JAVA_HOME` et `JAVA`. Si tel n'est pas le cas ajoutez-y les les lignes suivantes :

```
export JAVA_HOME=/usr/lib/jvm/votre/dossier/jdk
export JAVA=$JAVA_HOME/bin/java
```

Important : Votre JDK doit être à la version 8.

Exercice 1 – Installation d'un serveur d'application

Nous allons utiliser le serveur GlassFish 5 et l'installer en mode utilisateur.

Question 1

Téléchargez et désarchivez l'archive de la dernière version de GlassFish .

```
wget  http://download.oracle.com/glassfish/5.0/release/glassfish-5.0.zip
unzip glassfish-5.0.zip
```

Effacer éventuellement le fichier zip pour gagner de la place.

Question 2

Ajoutez à votre .bashrc la ligne suivante :

```
GLASSFISH_HOME=/chemin/vers/votre/dossier/glassfish
PATH=$PATH:$GLASSFISH_HOME/bin
```

puis, pour prendre en compte ces modifications dans votre terminal courant, tapez

```
source ~/.bashrc
```

Question 3

Les port 8080 et 8181 font partie des port d'écoutes GlassFish, il se peut que ces ports soit déjà utilisés par une autre application notamment 8080. Pour le savoir tapez

```
netstat -na | grep tcp | grep LISTEN | grep 8080
```

qui ne doit rien afficher. Si le port est déjà utilisé, modifiez le fichier

```
$GLASSFISH_HOME/glassfish/domains/domain1/config/domain.xml
```

en remplaçant par exemple 8080 par 8282.

Question 4

Démarrez le serveur d'application en mode terminal :

```
asadmin start-domain
```

Question 5

Afin de repérer quels sont les ports d'écoute des différents services de Glassfish, placez-vous dans le répertoire d'installation de glassFish et tapez :

```
asadmin get "configs.*" | grep -e "port=[0-9]\+$"
```

Question 6

Déterminez à quel service correspond chaque port d'écoute listés à la question 5. Vous pouvez tester les ports en tapant dans la barre de recherche de votre navigateur `localhost:<num port>`. Sur quel port écoute la console WEB d'administration de GlassFish ?

Question 7

Le serveur glassfish est fourni avec le SGBD Apache Derby pour la persistance des données des applications. Les fichiers binaire de SGBD se trouvent dans le répertoire `javadb` de votre installation GlassFish. Ce service n'est pas démarré automatiquement au démarrage du serveur. Vous devez donc le faire explicitement en tapant :

```
asadmin start-database --dbhome $HOME/MyDB
```

Sur quel port écoute ce service ?

Question 8

Stoppez les services. Le serveur glassfish sera par la suite de ce TME démarré via Eclipse :

```
asadmin stop-database
asadmin stop-domain
```

Exercice 2 – Utilisation d'Eclipse JavaEE

Question 1

La version d'eclipse de base n'est pas adaptée au développement d'application JavaEE. Nous allons donc utiliser une autre version (attention elle prend au moins 1,2 Go d'espace disque). Si vous êtes à la PPTI elle est accessible via la commande :

```
eclipse-jee
```

Si vous êtes sur votre machine personnelle vous devez utiliser une version de Eclipse JEE disponible à cette URL :

```
https://www.eclipse.org/downloads/packages/
```

Question 2

Créez un workspace dédié pour vos code JavaEE (appelons-le *workspace-jee*).

Question 3

Si vous êtes dans une salle de la PPTI, vérifiez que le proxy réseau d'Eclipse est bien configuré. Pour cela :

- Allez dans *Window* → *Preferences*
- Dans l'onglet *General* cliquez sur le sous-onglet *Network Connection*
- Vérifiez que le protocole HTTP soit bien configuré ainsi si vous :

- ◇ **Host :** *proxy.ufr-info-p6.jussieu.fr*
- ◇ **Port :** *3128*

- Vérifiez que les bypass pour *localhost* et *127.0.0.1* sont bien coché

Question 4

Pour une meilleure ergonomie, ouvrir la perspective JavaEE :

- Allez dans *Window* → *Perspective* → *Open Perspective* → *Other...*
- Cliquez sur *Java EE*

Exercice 3 – Configuration de GlassFish sous Eclipse

Vous devez être dans la perspective JavaEE de Eclipse.

Question 1

Nous allons tout d'abord déclarer un serveur d'application dans Eclipse. Pour cela :

- Allez dans *File* → *New* ⇒ *Other...*
- Cliquez sur *Server* dans la rubrique *Server* puis *Next >*
- **Si la rubrique *GlassFish* n'existe pas dans la partie *server type*** suivez les instructions suivantes :
  - ◇ Dans la partie *server type* sélectionnez *GlassFish Tools* dans la rubrique *Oracle* puis *Next >* (ceci peut prendre un moment)
  - ◇ Suivez le processus d'installation (qui doit potentiellement aboutir à un redémarrage d'eclipse, qui peut également prendre un moment).
- Dans la partie *server type* sélectionnez *GlassFish* dans la rubrique *GlassFish*
- Assurez-vous que la machine hôte vous convienne (localhost) puis *Next >*
- Dans le champs GlassFish Location indiquer le chemin absolu vers le dossier d'installation de GlassFish (normalement /votre/home/JavaEE/glassfish5/glassfish)
- Dans le champs Java Location, indiquez le chemin absolue de la jvm du serveur (normalement le jdk8) puis *Next >*.
- Assurez vous que :
  - ◇ le domain path termine bien par *domain1*
  - ◇ le login administrateur soit admin
  - ◇ pas de mot de passe pour l'admin
  - ◇ debug port : aucun

  puis *Next >* et *Finish*

Question 2

L'onglet *Servers* dans la partie basse de la fenêtre Eclipse, permet de configurer et de gérer des serveurs d'application JavaEE. En cliquant sur cet onglet, vous pourrez remarquer que notre serveur GlassFish a bien été déclaré et enregistré dans Eclipse. En double-cliquant dessus, la fenêtre principale affiche un formulaire permettant de configurer votre serveur.

Question 3

Ils vous est possible de démarrer/arrêter le serveur dans l'onglet *server* en le sélectionnant puis en cliquant sur les boutons à droite de l'onglet. Démarrez le serveur.

Question 4

Ouvrez un navigateur et assurez-vous que le serveur d'application est bien à l'écoute sur localhost sur le port d'administration noté précédemment.

Exercice 4 – Application JavaEE sous Eclipse

Dans Eclipse, le développement d'une application JavaEE se traduit par la création d'un projet Eclipse de

type "Enterprise Application Project" (EAP). L'ensemble des composants d'une application sont regroupé dans une archive appelée EAR, qui permet ainsi de déployer l'application entièrement.

Plus précisément, une application JavaEE est composée de plusieurs modules. Il existe plusieurs types de modules : EJB, connector, Dynamic Web, etc. Le développement d'un module dans Eclipse se traduit par la création d'un projet Eclipse dédié. Vous pourrez remarquer qu'il existe pour chaque type de module, un type de projet Eclipse ("EJB project", "Connector project", ...). Chaque projet de développement d'un module est affilié à l'EAR de l'application. Cependant ceci n'est pas obligatoire. Il est possible de déployer dans un serveur d'applications JavaEE un module sans que celui-ci ne soit affecté à une application.

Question 1

Créez une application JavaEE :

- Allez dans *File* → *New* ⇒ *Enterprise Application Project*
- nommez le projet "*HelloApplication*"
- Vérifiez que la *target runtime* est bien pour GlassFish 5 et que l'EAR version est bien la dernière version puis *Finish*

Question 2

Créez un projet EJB nommé "HelloEJB". Vérifiez que la runtime cible soit bien pour GlassFish 5 et que la version d'EJB soit supérieure ou égale à 3.1. Ajoutez le projet à l'EAR créée précédemment, cliquez deux fois sur *next >*, puis décochez *"create an EJB CLient JAR module ..."*. Enfin cliquez sur *Finish*.

Question 3

Dans le projet HelloEJB, créez un package "beans" puis y créer deux beans de type session (*clic droit sur le package* → *new* → *SessionBean*) :

- un bean SayHello de type stateless
- un bean Counter de type statefull

Dans les deux cas, cochez la création de l'interface remote.

Question 4

Dans l'interface SayHelloRemote définissez une méthode `hello` qui prend en paramètre une String et qui renvoie une string dont la valeur est le mot "Hello" suivi du paramètre. Voici la classe d'implémentation correspondante :

```
package beans;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

/**
 * Session Bean implementation class SayHello
 */
@Stateless
@LocalBean
public class SayHello implements SayHelloRemote {

    /**
     * Default constructor.
     */
    public SayHello() {
        // TODO Auto-generated constructor stub
    }

        @Override
```

```
	public String hello(String name) {
		return "Hello "+name;
	}

}
```

Question 5

Dans l'interface CounterRemote, définissez trois méthodes :
- `getValue` qui renvoie la valeur du compteur
- `increment` sans retour et qui incrémente de 1 le compteur
- `decrement` sans retour et qui décrémente de 1 le compteur

Voici la classe d'implémentation correspondante :

```
package beans;

import javax.ejb.LocalBean;
import javax.ejb.Stateful;

/**
 * Session Bean implementation class Counter
 */
@Stateful
@LocalBean
public class Counter implements CounterRemote {

	private int value=0;
    /**
     * Default constructor.
     */
    public Counter() {
        // TODO Auto-generated constructor stub
    }

	@Override
	public int getValue() {
		return value;
	}

	@Override
	public void increment() {
		value++;
	}

	@Override
	public void decrement() {
		value--;
	}

}
```

Question 6

Déployez votre application "HelloApplication". Pour cela :

- clic droit sur le projet *HelloApplication* → Run As → Run on Server
- Choisissez le serveur GlassFish créé précédemment puis *Next >*
- Vérifiez que le projet EJB se situe bien dans la partie Configured puis *Finish*

Vous pourrez remarquer que le module est bien déclaré dans notre serveur dans l'onglet *Servers* et que le serveur est bien démarré et synchronisé. Ceci est également cohérent avec l'interface Web d'administration de GlassFish dans l'onglet *Applications*.

Question 7

Le nom JNDI est normalement (si vous n'avez rien changez dans les nommages par défaut)

- `"java:global/HelloApplication/HelloEJB/SayHello!beans.SayHelloRemote"` pour le Hello
- `"java:global/HelloApplication/HelloEJB/Counter!beans.CounterRemote"` pour le Counter.

Il vous est possible de voir ces noms dans le fichier de log `server.log` présent dans le dossier `JavaEE/glassfish5/glassfish/domains/domain1/logs`. Ces logs ont du être produits lorsque vous avez déployé votre application dans Glassfish.

Exercice 5 – Programmation d'un client

Nous souhaitons écrire un programme client qui testera la fonctionnalité des deux EJB codés précédemment. Le code du client sera développé dans une application Java SE classique qui sera ainsi cloisonnée du développement de l'application JavaEE.

Question 1

Créez un nouveau projet Java classique *"HelloClient"*. Ce projet pourra par la suite centraliser l'ensemble des code client de vos application JavaEE.

Question 2

Configurez le build path du projet *"HelloClient"* (*clic droit sur le projet → Build Path → Configure Build Path...*) :

- Dans l'onglet *Projects* cliquez sur *add* et sélectionner le projet EJB et/ou JPA souhaité. Ceci permet d'intégrer dans votre projet client l'accès aux interfaces et classes compilés.
- Dans l'onglet *Libraries* :
  - ◇ Cliquez sur *Add External JARs...*
  - ◇ Sélectionnez les fichiers .jars se situant à la racine du répertoire `glassfish/lib` (appserv-rt.jar, gf-client.jar, javaee.jar, jndi-properties.jar)
- Cliquez sur OK

Question 3

Dans le projet client, créez un package *client*.

Question 4

Dans le package *clients* créez une classe JUnit *TestClient* dont le contenu est donné ci-dessous et exécutez-la.

```
package test;

import static org.junit.Assert.assertEquals;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.junit.Test;

```

```
import beans.CounterRemote;
import beans.SayHelloRemote;

public class TestClient {

	@Test
	public void test() throws NamingException {
		//Properties props = new Properties();
		//si machine distante, changer ci-dessous
		//props.setProperty("org.omg.CORBA.ORBInitialHost", "localhost");
		//props.setProperty("org.omg.CORBA.ORBInitialPort", "8686");
		Context context = new InitialContext();
		Object o = context.
		  lookup("java:global/HelloApplication/HelloEJB/SayHello!beans.
		      SayHelloRemote");
		SayHelloRemote sh_rem = (SayHelloRemote) o;
		System.out.println(sh_rem.getClass());
		assertEquals("Hello World", sh_rem.hello("World"));

		o = context.
		  lookup("java:global/HelloApplication/HelloEJB/Counter!beans.
		      CounterRemote");
		CounterRemote cpt_rem = (CounterRemote) o;
		int val = cpt_rem.getValue();
		cpt_rem.increment();
		cpt_rem.increment();
		cpt_rem.increment();
		cpt_rem.increment();
		cpt_rem.decrement();
		assertEquals(val + 3, cpt_rem.getValue());
	}

}
```

Exercice 6 – Configuration d'une connexion à une base de données sous Eclipse

Nous avons besoins de configurer dans Eclipse une connexion à notre serveur de base de données Derby fournie avec l'installation GlassFish. **Si vous rencontrez des difficultés pour configurer la connexion avec Derby, vous pourrez trouver en annexe de ce document comment configurer une connexion avec SQLite.**

Question 1

Avant toute chose n'oubliez pas de démarrer votre serveur Derby via le terminal :

```
asadmin start-database --dbhome $HOME/JavaEE/MyDB
```

Question 2

Dans Eclipse

- Allez dans *Window* → *Show View* → *Other...*
- Sélectionnez *Data Source Explorer* dans la rubrique *Data Management*

Question 3 -

Dans l'onglet *Data Source Explorer*, clic droit sur *Database Connections* → *new...*

- choisir Derby, choisir un nom dans le champs name (disons "*MonDerby*") puis *Next >*
- Cliquez sur L'icône *new driver definition* juste à droite de sélecteur de driver :
  - ◇ Choisir *Derby Client JDBC Driver*, version 10.2 dans l'onglet *Name/Type*
  - ◇ Dans l'onglet *Jar list*, cliquez sur le nom *derbyclient.jar* et cliquez sur le bouton *Edit Jar/Zip* à droite
  - ◇ Sélectionnez le fichier *derbyclient.jar* présent dans le dossier d'installation de derby (dossier *javadb/lib* de glassfish)
  - ◇ Cliquez sur *OK*
- Vérifiez/renseignez les champs suivants :
  - ◇ le champs *Database* indique le nom du dossier qui contiendra les données : appelons le *sample*. Ce dossier sera créé dans le dossier indiqué par l'option *–dbhome* dans la commande de démarrage du serveur Derby.
  - ◇ le champs *host* est bien *localhost*
  - ◇ le port d'écoute correspond bien à celui que vous avez noté dans l'exercice 1 (normalement sa valeur est 1527)
  - ◇ Les champs user name et Password doivent être tous les deux égal à *APP* (en majuscules).
  - ◇ cochez la case save password et vérifiez que la case *create database (if required)* est bien cochée
- Vous pouvez tester la connexion avec votre base de données en cliquant sur *Test Connection*. Ceci aura pour effet de créer le dossier de votre base de données.
- Cliquez sur *Next >* (un récapitulatif s'affiche) et *Finish*

Question 4

Il faut maintenant déclarer la connexion à la base de données au serveur Glassfish. Pour cela :

- Ouvrez un navigateur
- aller à l'URL d'administration de glassfish (normalement *localhost :4848*)
- Á gauche, dans le menu JDBC, cliquez sur DerbyPool dans la rubrique *JDBC Connection Pools*
- Dans la partie principale de la page, cliquez sur l'onglet *Additional Properties* et vérifiez que les informations que vous avez saisie dans la question précédente corresponde bien. Normalement il faut modifier le paramètre *DatabaseName* à la valeur *sample* (ou la valeur que vous avez renseignée dans le champs *Database* précédemment).
- Cliquez sur *Save* en haut à droite de la page.
- Revenez dans Eclipse, dans la vue *Servers*, déroulez *GlassFish Management* et *Resources*. Clic droit sur *JDBC* → *Refresh*.

Exercice 7 – Projet JPA sous Eclipse

Un projet JPA vous permettra de développer des EJB entité et de les lier à la base de données.

Question 1

Créez un projet JPA :

- Allez dans *File* → *New* ⇒ *Other...*
- Cliquez sur *JPA Project* dans la rubrique *JPA* puis *Next >*
- Configurer le projet comme suit :
  - ◇ nommez votre projet "*HelloJPA*"
  - ◇ choisissez comme *target Runtime* GlassFish 5
  - ◇ JPA version à 2.1
  - ◇ cochez *add project to an EAR* et choisir l'application JavaEE créée précédemment
- Puis cliquez sur *Next >* 2 fois (vous arrivez à la fenêtre JPA Facet normalement)

- Choisissez *EclipseLink* dans le champs Platform et choisissez la connexion à votre base de données ("MonDerby" normalement) dans la rubrique *connection*
- Notez que le champs *Schema* correspond au nom de votre schéma de base de données dans lequel les tables de votre projets seront créées. Par défaut ce nom est *APP* (`DEFAULT` pour SQLite).
- Cliquez sur *Finish*

Question 2

Créez une nouvelle entité `Employe`. Pour cela :

- clic droit sur le projet *HelloJPA* → *New >* → *JPA Entity*
- dans le package mettez par exemple *org.entreprise*
- dans le champs *Class Name* mettre *Employe* puis *Next >*
- Vous pouvez ici déclaré les attributs de votre entité. Nous considérerons qu'un employé possède :
  - ◇ un nom de type string (qui fera office de clé primaire)
  - ◇ un salaire de type flottant
- cliquez sur *Finish*.

Question 3

La classe de l'entité `Employe` est :

```
package org.entreprise;

import java.io.Serializable;
import javax.persistence.*;


/**
 * The persistent class for the EMPLOYE database table.
 *
 */
@Entity
@NamedQuery(name="Employe.findAll", query="SELECT e FROM Employe e")
public class Employe implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	private String nom;

	private double salaire;

	public Employe() {
	}

	public String getNom() {
		return this.nom;
	}

	public void setNom(String nom) {
		this.nom = nom;
	}

	public double getSalaire() {
		return this.salaire;
	}
```

```
	public void setSalaire(double salaire) {
		this.salaire = salaire;
	}

}
```

Question 4

Vous pouvez remarquer que Eclipse détecte une erreur au niveau de l'annotation `@Entity` car la table *Employe* n'existe pas dans votre base de données. Pour cela :

- dans l'explorateur de projet, clic droit sur la classe Employe.java → *JPA Tools* → *add to persistent unit*. Ceci permet de déclarer dans le fichier *persistence.xml* de votre projet JPA que le type *Employe* est un type persistant
- Cliquez sur le fichier *persistence.xml* dans l'onglet *JPA Content* dans le projet JPA. Vérifiez dans l'onglet *General* que la classe *Employe* a bien été ajouté dans la colonne *Managed Classes*, notez également le champs name qui correspond au nom du persistence context qui permettra de créer votre EntityManager.
- Toujours dans l'explorateur de projet, clic droit sur le projet *HelloJPA* → *JPA Tools* → *Generate Tables from Entities...*
- Cliquez sur le choix *both* puis *Finish*. Ceci aura pour effet de créer la table *Employe* dans votre base de données. Vous pouvez d'ailleurs remarquer dans la vue *Data Source Explorer*, l'état de votre base de données (Schémas enregistrés, tables enregistrées pour chaque schéma ...).

Question 5

Ajouter au Build path des projets HelloEJB et HelloClient, le projet HelloJPA pour que les entités y étant codées soient connues par le compilateur.

Question 6

Pour ajouter ou modifier votre table *Employe*, vous devez créer un EJB session qui permettra de faire cela. Créez un EJB session Stateless nommé *DRH* dans votre projet *HelloEJB*. Cet EJB offre les services suivants :

- `boolean recrutement(String nom)` : qui pour un nom d'employé donné ajoute une nouvelle entrée dans la table correspondante avec un salaire égal à 1300. Si l'employé existe déjà (on considérera qu'il n'y a pas d'homonyme) la méthode renvoie faux, vrai sinon.
- `double getSalaire(String nom);` : qui renvoie le salaire d'un employé donné.
- `void promotion(String nom, double v)` : augmente de `v` euros le salaire de l'employé
- `Employe[] list_personnel();` : qui renvoie la liste de tous les employés de l'entreprise
- `void tu_es_vire(String nom);` : qui supprime de la base l'employé en question.

```
package beans;

import java.util.List;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.Query;

import org.entreprise.Employe;

/**
```

```
 * Session Bean implementation class DRH
 */
@Stateless
@LocalBean
public class DRH implements DRHRemote {

    /**
     * Default constructor.
     */
    public DRH() {
        // TODO Auto-generated constructor stub
    }

    @PersistenceContext(unitName="HelloJPA")
    private EntityManager em;

        @Override
        public boolean recrutement(String nom,double salaire) {
                Employe emp = em.find(Employe.class, nom);
                 if(emp != null){
                  return false;
                 }
                 emp = new Employe();
                 emp.setNom(nom);
                 emp.setSalaire(salaire);
                 em.persist(emp);
                 em.flush();
                 return true;
        }

        @Override
        public double getSalaire(String nom) {
               Employe emp = em.find(Employe.class, nom);
                 if(emp != null){
                  return emp.getSalaire();
                 }
                  return -1.0;
        }

        @Override
        public void promotion(String nom, double v) {
               Employe emp = em.find(Employe.class, nom);
                 if(emp != null){
                  emp.setSalaire(emp.getSalaire() + v);
                  em.flush();
                 }

        }

        @Override
        public Employe[] list_personnel() {
               Query q = em.createQuery("SELECT e FROM Employe e ");
```

```
                List<?> l =q.getResultList();
                Employe[] res=new Employe[l.size()];
                return l.toArray(res);
        }

        @Override
        public void tu_es_vire(String nom) {
                Employe emp = em.find(Employe.class, nom);
                if(emp != null){
                 em.remove(emp);
                 em.flush();
                }

        }

}
```

Question 7

Redéployez votre application JavaEE sur le serveur pour prendre en compte ce nouveau bean ainsi que le module HelloJPA.

Question 8

Testez votre bean DRH avec le test JUnit TestDRH

```
package test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;
import java.util.stream.Collectors;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import org.entreprise.Employe;
import org.junit.After;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;

import beans.DRHRemote;

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class TestDRH {
```

```
30
31
32  private static Map<String,Double> employes= new HashMap<>();
33  private Context context;
34  private DRHRemote drh ;
35
36
37  private static String urlDRH = "java:global/HelloApplication/HelloEJB/DRH!beans.
        DRHRemote";
38
39
40  @BeforeClass
41  public static void beforeAllTests() {
42          employes.put("Dupuis", 1250.0);
43          employes.put("Durand", 1500.0);
44          employes.put("Dupont", 5620.0);
45          employes.put("Martin", 2890.0);
46  }
47
48  @Before
49  public void beforeEachTest() throws NamingException {
50          context = new InitialContext();
51          drh = (DRHRemote)context.lookup(urlDRH);
52  }
53
54
55  @After
56  public void afterEachTest() throws NamingException {
57          drh=null;
58          context.close();
59  }
60
61
62  //vider la base de données des employes
63  @Test
64  public void test1() {
65
66          for(Employe e : drh.list_personnel()) {
67                  drh.tu_es_vire(e.getNom());
68          }
69          assertEquals(0, drh.list_personnel().length);
70  }
71
72
73  //recrutement de tous les employes
74  @Test
75  public void test2() {
76          for(Entry<String,Double> e : employes.entrySet()) {
77                  assertTrue(drh.recrutement(e.getKey(), e.getValue()));
78          }
79  }
80
```

```
	@Test
	//on vérifie que tous le monde est bien dans la base et a bien le bon salaire
	public void test3() {
		Set<String> tmp = Arrays.stream(drh.list_personnel()).map(emp->emp.getNom
		    ()).collect(Collectors.toSet());
		assertEquals(employes.keySet(),tmp);

		for(Entry<String,Double> e : employes.entrySet()) {
			assertEquals(Double.valueOf(drh.getSalaire(e.getKey())), e.
			    getValue());
		}

	}
	//on vérifie que tous les employe ne peuvent pas être recrutés deux fois
	@Test
	public void test4(){
		for(Entry<String,Double> e : employes.entrySet()) {
			assertFalse(drh.recrutement(e.getKey(), e.getValue()));
		}
	}

	//on fait une augmentation de salaire
	@Test
	public void test5(){
		String nom = employes.keySet().iterator().next();
		double salaire_initial = employes.get(nom);
		double promotion = 500;
		double nouveau_salaire = salaire_initial + promotion;
		drh.promotion(nom, promotion);
		assertEquals(Double.valueOf(nouveau_salaire), Double.valueOf(drh.
		    getSalaire(nom)));
	}

	//on vire quelqu'un
	@Test
	public void test6(){
		String nom = employes.keySet().iterator().next();
		drh.tu_es_vire(nom);
		assertFalse(Arrays.stream(drh.list_personnel()).anyMatch(e -> e.getNom().
		    equals(nom)));
	}
}
```

Annexe : Configuration de connexion à SQLite

SQLite diffère des SGBD traditionnels comme MySQL dans le sens où il ne reproduit pas l'architecture client-serveur habituelle. En effet c'est une bibliothèque permettant d'intégrer un SGBD directement dans les programmes qui utilisent le SQL pour requêter des données persistantes. L'ensemble de la base de données est stockée dans un seul fichier.

Question 1

Télécharger la dernière version du driver JDBC de SQLite à l'adresse suivante :
https://bitbucket.org/xerial/sqlite-jdbc/downloads/

Question 2

Intégrez le fichier téléchargé dans les librairies de GlassFish avec la commande suivante (il doit être démarré) :

```
asadmin add-library <le_fichier_jar_telechargé>
```

Ceci a pour effet de copier votre jar dans le répertoire `glassfish5/glassfish/domains/domain1/lib`.

Question 3

Redémarrez GlassFish.

Question 4

On va maintenant déclarer une nouvelle ressource de connexion pour GlassFish. Pour cela
- Ouvrez un navigateur
- aller à l'URL d'administration de glassfish (normalement *localhost :4848*)
- À gauche, dans le menu JDBC, cliquez sur *JDBC Connection Pools*
- Cliquez sur *New...* en haut de la liste
- Remplissez les champs suivants :
  - ◇ **Pool Name :** *SQLITE*
  - ◇ **Resource Type** : `javax.sql.ConnectionPoolDataSource`

  puis cliquez sur *next*.
- Remplissez les champs suivants :
  - ◇ **Datasource Classname** : `org.sqlite.javax.SQLiteConnectionPoolDataSource`
  - ◇ tout en bas de la page ajouter la property suivante (cliquez sur *Add Property*) :
    - · **Name** : *url*
    - · **Value** : *jdbc :sqlite :/votre/chemin/absolu/vers/le/fichier/de/la/BD*
- Cliquez sur *Finish*. Une nouvelle ressource a du apparaître dans la liste des *JDBC Connection Pools*. Cliquez sur la nouvelle ressource créée et testez la connexion avec le bouton *ping*.

Question 5

Nous allons à présent créer une ressource JBDC dans ClassFish. Pour cela
- cliquez sur *JDBC Resources*, puis sur le bouton *New....*
- Renseignez comme valeur du champs **JNDI Name** *jdbc/SQLite* et choisissez comme Pool Name le nom du pool de connexion créé ci-avant.
- Cliquez sur *OK* pour valider.

Question 6

Nous allons configurer la ressource JDBC par défaut pour que celle ci utilise la ressource SQLite. Pour celà
- Cliquez sur la ressource *jdbc/__default* dans les *JDBC Resources* et choisissez comme Pool Name *SQLITE*
- cliquez sur *save* pour prendre en compte les modifications

Question 7

Redémarrez GlassFish.

Question 8

Dans eclipse vous pourrez vous assurer que les deux ressources créées (*SQLITE* pour les jdbc-connection-pool et *jdbc/SQLite* pour les jdbc-resource) sont bien affichés dans l'onglet Server

Question 9

Nous allons maintenant déclarer une connexion à SQLite pour Eclipse afin que les projets JPA puissent se connecter à la base de données pour créer les tables des entities. Pour cela

- Dans l'onglet *Data Source Explorer*, clic droit sur *Database Connections* → *new...*
- choisir SQLite, choisir un nom dans le champs name (disons "*MonSQLite*") puis *Next >*
- Cliquez sur l'icône *new driver definition* juste à droite de sélecteur de driver :
  - ◇ Choisir *SQLite JDBC Driver*, dans l'onglet *Name/Type*
  - ◇ Dans l'onglet *Jar list*, cliquez sur *Clear All* puis sur le bouton *Add Jar/Zip...* à droite
  - ◇ Sélectionnez le fichier jar téléchargé précédemment (utilisez celui du dossier de GlassFish *glassfish5/glassfish/domains/domain1/lib* de préférence)
  - ◇ Cliquez sur *OK*
- Vérifiez/renseignez les champs suivants :
  - ◇ le champs *Database* indique le nom du dossier qui contiendra les données : appelons le *sample*.
  - ◇ le champs *Database Location* est bien le même que celui renseigné plus haut c'est à dire */votre/chemin/absolu/vers/le/fichier/de/la/BD*
- Vous pouvez tester la connexion avec votre base de données en cliquant sur *Test Connection*.
- Cliquez sur *Next >* (un récapitulatif s'affiche) et *Finish*
- Vous pourrez remarquer ensuite dans la vue *Data Source Explorer*, en déroulant les différentes rubriques de votre connexion qu'il existe un seul Schéma appelé *DEFAULT* dans votre base de données.

Vous êtes maintenant prêt pour créer un projet JPA pouvant utiliser la connexion à SQLite.