UNIVERSITE DE RENNES 1	**MDI ESIR2**	istic Informatique Électronique

à la découverte de JPA

Les objectifs de ce travail pratique sont les suivants :

- Comprendre les mécanismes de JPA
- Réaliser une application en utilisant JPA en se plaçant dans un cadre classique de développement sans serveur d'application au départ.

Recommendations. Ce TP utilise des technologies abordé en cours d'un point de vu théorique, mais la documentation technique peut être facilement trouvé sur internet. Quelques exemples de sites qui fournissent de la documentation sur JPA sont les suivantes:
https://en.wikibooks.org/wiki/Java_Persistence
http://www.oracle.com/technetwork/middleware/ias/toplink-jpa-annotations-096251.html

Être autodidacte est une compétence essentielle pour tout informaticien; n'hésitez pas à chercher des tutoriels si vous êtes bloqués.

Sujet

L'objectif du projet est de construire une application de prise de RDV type doodle mais en y ajoutant un certain nombre de services. L'idée est de partir de la création d'un sondage par un utilisateur sur un choix de dates pour une réunion ayant au moins un intitulé et un résumé. Les participants au sondage renseignent leur nom, leur prénom et leur mail (ils participent au sondage au travers d'un lien Web unique). Quand le créateur du sondage valide une date, si la date contient une pause déjeuner (ou pause), les utilisateurs reçoivent automatiquement un mail avec un lien unique pour renseigner leur préférence alimentaire et leurs allergies. L'application envoie aussi un mail avec un clear code qui contient un code qui pourra être demandé pour entrer dans le bâtiment et un lien vers un pad qui permettra de partager les notes de réunion. Le pad sera un service tierce.
Le système doit permettre de sauvegarder l'ensemble des sondages, les choix des participants, les préférences alimentaires des participants, …
L'email servira d'identifiant pour les participants.
Le pad sera initialisé avec le titre de la réunion et une liste des présents et absents en fonction des réponses au sondage.
Un groupe slack sera aussi initialisé pour les communications au sein de la réunion.

Dans une première étape, nous nous intéressons uniquement au modèle métier de cette application. Faites un premier diagramme de classes sur feuille blanche de ce modèle métier, créer les classes Java correspondantes et utilisons JPA pour créer la base de données et la couche d'accès aux données.

Organisation.

Vous trouverez un template de projet sur https://github.com/barais/tpjpa2019sir/
Vous pouvez forker le projet ou récupérer l'archive contenant :

- Un template de projet pour la construction d'application autonome utilisant JPA, hibernate et hsqldb.

Décompressez ce projet sur votre compte et importez dans eclipse ou intelliJ.

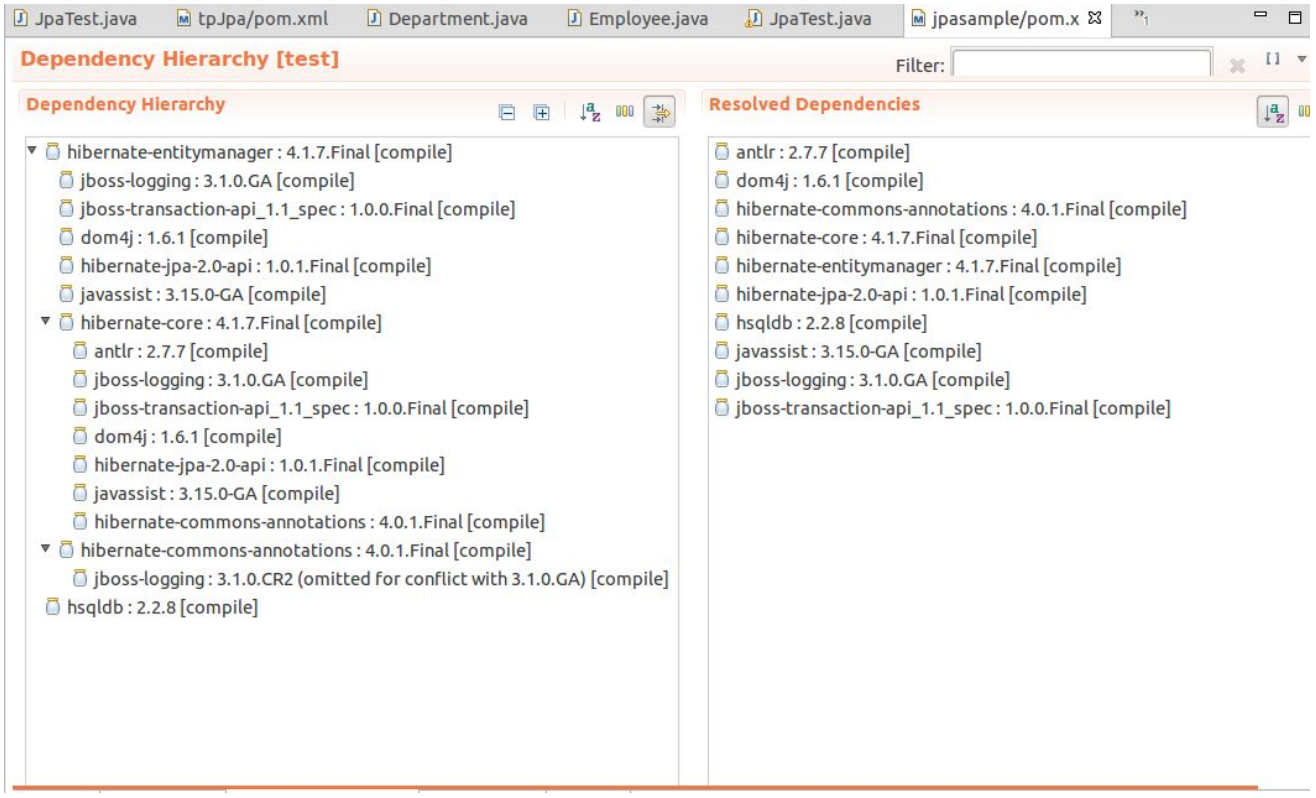
Pour eclipse 4.X
Depuis eclipse 4.X, le support de maven s'est amélioré. Pour importer votre projet. File -> import -> maven -> existing maven project.
Votre projet est configuré.

Démarrage de la base de données. Puis dans la version copiée sur votre compte allez dans le répertoire du projet. Vous trouverez là le script de démarrage de la base de données (run-hsqldb-server.sh) et le script du démarrage du Manager (show-hsqldb.sh). Lancez le système de base de données, puis le *Manager*. Connectez vous à la base de données (login : sa – et pas de mot de passe : -- URL de connexion : jdbc:hsqldb:hsql://localhost/). Le fichier de données se trouve dans le répertoire Data. Vous pourrez supprimer l'ensemble des fichiers de ce répertoire si vous souhaitez réinitialiser complètement votre système de base de données.

Question 0.

Regardez rapidement le pom.xml, vous constaterez que c'est un projet simple avec deux dépendances (hibernate et hsqldb (driver jdbc pour hsqldb)).

Ce qui donne au final les dépendances suivantes.



Question 1.

Transformez une première classe en entité.

Travaillez uniquement sur le champ id et les attributs de la classe. Créez plusieurs instances de cette classe. Rendez ces instances persistantes. Regardez dans le manager de la base de données le résultat.

Vous devez obtenir ce genre de code mais sur vos classes métier

```
package test.testjpa.domain;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToOne;

@Entity
public class Employee {
	private Long id;

	private String name;

	private Department department;

	public Employee() {
	}

	public Employee(String name, Department department) {
		this.name = name;
		this.department = department;
	}

	public Employee(String name) {
		this.name = name;
	}

	@Id
	@GeneratedValue
	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}
```

```
	public void setName(String name) {
		this.name = name;
	}

	@ManyToOne
	public Department getDepartment() {
		return department;
	}

	public void setDepartment(Department department) {
		this.department = department;
	}

	@Override
	public String toString() {
		return "Employee [id=" + id + ", name=" + name + ", department="
				+ department.getName() + "]";
	}

}
```

Question 2.

Même travail avec une première association entre deux entités.

Vous devez obtenir un code qui ressemble à cela mais sur vos classes métier. N'oubliez pas de remplacer les attributs mis à *Transient* sur vos classes précédentes.

```
package test.testjpa.domain;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;

@Entity
public class Department {

	private Long id;

	private String name;

	private List<Employee> employees = new ArrayList<Employee>();

	public Department() {
		super();
	}
```

```
	public Department(String name) {
		this.name = name;
	}

	@Id
	@GeneratedValue
	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	@OneToMany(mappedBy = "department", cascade = CascadeType.PERSIST)
	public List<Employee> getEmployees() {
		return employees;
	}

	public void setEmployees(List<Employee> employees) {
		this.employees = employees;
	}
}
```

Question 3.

Finissez le modèle métier présenté précédemment. Intégrer une classe de service permettant de peupler la base mais aussi de faire des requêtes sur la base de données.

Le code pour créer les entités peut ressembler à cela :

```
package test.testjpa.jpa;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;

import test.testjpa.domain.Employee;
```

```
import test.testjpa.domain.Department;

public class JpaTest {

	private EntityManager manager;

	public JpaTest(EntityManager manager) {
		this.manager = manager;
	}
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		EntityManagerFactory factory =
			Persistence.createEntityManagerFactory("example");
		EntityManager manager = factory.createEntityManager();
		JpaTest test = new JpaTest(manager);

		EntityTransaction tx = manager.getTransaction();
		tx.begin();
		try {
			test.createEmployees();
		} catch (Exception e) {
			e.printStackTrace();
		}
		tx.commit();

		test.listEmployees();

		 manager.close();
		System.out.println(".. done");
	}

	private void createEmployees() {
		int numOfEmployees = manager.createQuery("Select a From Employee a",
Employee.class).getResultList().size();
		if (numOfEmployees == 0) {
			Department department = new Department("java");
			manager.persist(department);

			manager.persist(new Employee("Jakab Gipsz",department));
			manager.persist(new Employee("Captain Nemo",department));

		}
	}

	private void listEmployees() {
		List<Employee> resultList = manager.createQuery("Select a From Employee a",
Employee.class).getResultList();
		System.out.println("num of employess:" + resultList.size());
		for (Employee next : resultList) {
			System.out.println("next employee: " + next);
		}
```

```
        }
}
```

Question 4. Connexion à une base mysql

Modifiez le fichier *persistence.xml* afin de vous connecter sur une base de données *MySQL* de l'istic.
Pour ce faire connecter vous sur http://anteros.istic.univ-rennes1.fr pour vous créer votre base de données. Puis en vous inspirant de l'exemple suivant http://snipplr.com/view/4450/ modifier votre fichier persistence.xml pour vous connecter à votre base de données.
Il est aussi nécessaire d'ajouter le driver jdbc vers mysql. Pour ce faire ajoutez la dépendance dans le pom.xml

```
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.47</version>
</dependency>
```

Question 5. Portez votre application et gérer au minimum une relation d'héritage, les requêtes, une requête nommée.

Pour l'héritage, on peut imaginer des sondages type choix parmi une liste, des sondages de date, des sondages de lieu et des sondage de date et de lieu.

Faites vos requêtes en utilisant les criteria query.
http://stackoverflow.com/questions/5705291/select-in-equivalent-in-jpa2-criteria

Question 6. Mise en évidence du problème de n+1 select.

Pour comprendre le problème du n+1 select.
récupérez le code joint depuis github.

https://github.com/barais/tpM2s.git

Comme d'habitude c'est un projet Maven.
Lancez JPAtest pour peupler la base de données.
Lancez N1select pour faire une requête en chargement paresseux (problème du n+1 select)
Lancez JoinFetch pour faire une requête en chargement au plus tôt (sans le problème du n+1 select)
Comparez les performances et le nombre de requêtes réellement effectuées.

Bon TP

Johann et Olivier