SIL4 - séance 3

ORM : object-relational mapping

en français, mapping objet/relationnel

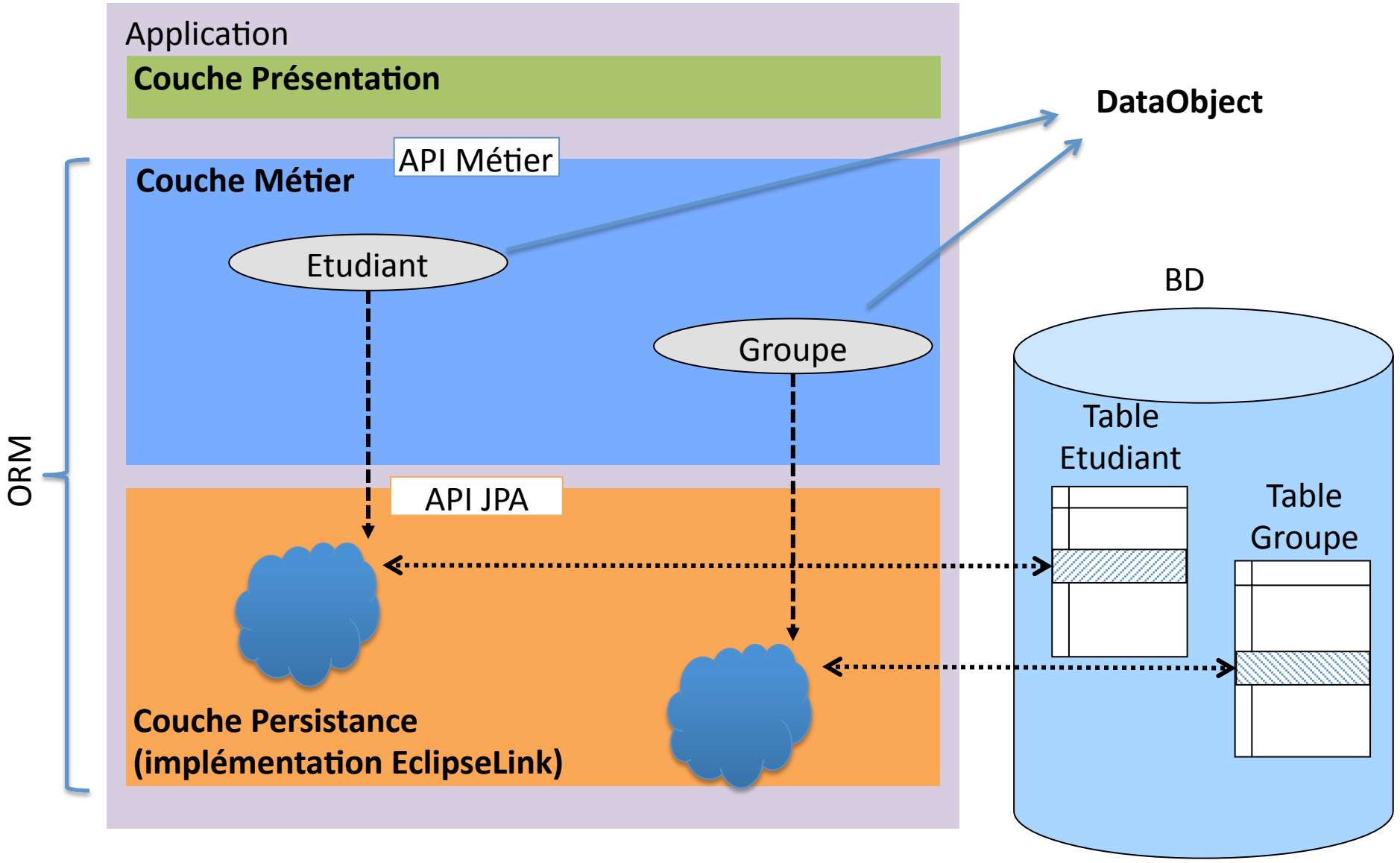
Francis Brunet-Manquat - MIAM (bureau 112)

Jérôme David - SIMO

ORM : object-relational mapping

- Lorsque l'on développe des applications WEB, on est en général amené à manipuler une BD et on est toujours confrontés aux mêmes problèmes récurrents :
  - Trouver des enregistrements (n-uplets)
  - Mettre à jour des enregistrements
  - Ajouter de nouveaux enregistrements
  - Supprimer des enregistrements
  - Traiter une liste d'enregistrements
  - …etc

ORM : object-relational mapping



Qu'est-ce qu'un "DataObject"

- C'est **une classe qui permet de représenter directement un n-uplet** (une entité) stocké dans une table du SGBD
- **La classe aura des membres** (attributs) qui **correspondront aux champs** (colonnes) **de la table** qu'elle représente
- On développera donc une telle classe pour chaque table de la BD

Intérêts de l'utilisation d'un ORM (1/2)

ORM → Object – Relational Mapping

- Implémenter "à la main" un mapping objet/relationnel est une tâche répétitive et fastidieuse

- Il existe **des outils pour réaliser cette corvée de façon "propre" et automatique** :
  - En java : Hibernate, EclipseLink, Ibatis, Torque...
  - En PHP : propel, doctrine, ...

- **Une programmation "BD" plus attractive**
  - Un ORM fournit une API bien définie pour accéder à la BD. Au lieu de requêtes SQL complexe pour trouver et manipuler les données, le développeur utilise une API
  - Un ORM utilise les standards de la POO (exceptions, itérateurs, ...)

Intérêt de l'utilisation d'un ORM (2/2)

- **Un ORM s'intègre naturellement au modèle MVC**
  - L'ORM fournit le socle de la partie "Modèle" de l'architecture MVC.
  - En utilisant l'héritage, on peut facilement développer les objets métiers
- **Un ORM participe à la sécurité de l'application**
  - Torque (par exemple) utilise les "prepared statements" de SQL pour éviter l'injection de code, il "échappe" les chaînes, ...
- **Application portable**
  - Migrer d'un SGBD à un autre est très facile.
- **Conçu pour être adapté et étendu**
  - Les classes générées sont conçues pour être étendues.
  - Des classes dérivées vides sont créées pour que l'on puisse y implémenter la logique métier particulière.
  - On peut aussi y surcharger les méthodes de la classe mère pour les adapter aux besoins

Qu’est-ce que JPA ?

- Standard Java EE
- Spécification qui définit un ensemble de règles pour la gestion de la persistance

Qu’est-ce qu’EclipseLink ?

- implémentation du standard JPA
- Framework ORM open source
- Autres framework : Hibernate, OpenJPA, etc.

JPA par l’exemple

- Démonstration aide projet

- Projet sur l’intratek

Entity (DataObject)

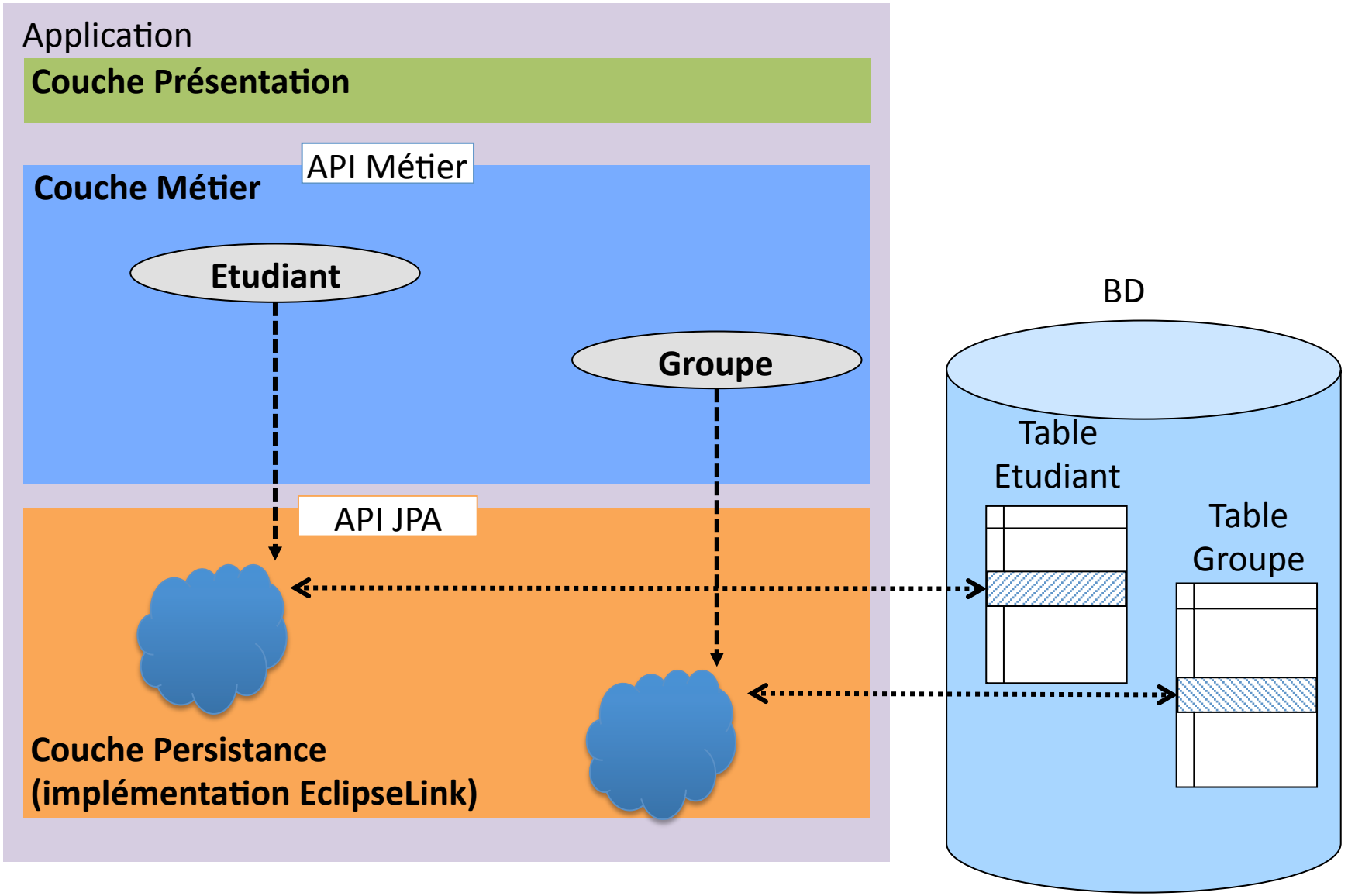
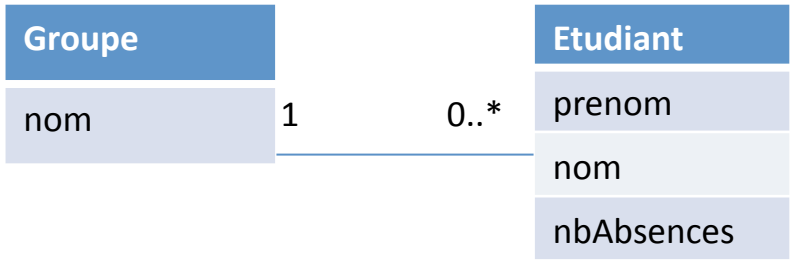


Diagramme de classes UML



Modèle relationnel

Groupe (id, nom*)
Etudiant (id, prenom*, nom*, nbAbsences*, #groupe_id)

	Primary Key
*	Champ obligatoire
#	Foreign Key

Description JPA

Ce que vous devez faire

```
@Entity
public class Groupe implements Serializable {

@Id
@GeneratedValue
private Integer id;

@Column(unique=true, nullable=false)
private String nom;

@OneToMany(mappedBy="groupe", fetch=FetchType.LAZY)
private List<Etudiant> etudiants;

…
```

```
@Entity
public class Etudiant implements Serializable {

@Id
@GeneratedValue
private Integer id;

@Column(nullable=false)
private String prenom;

@Column(nullable=false)
private String nom;

private int nbAbsences;

@ManyToOne
private Groupe groupe;

…
```

Annotation JPA d'une entité (1/2)

```
@Entity
public class Groupe implements Serializable {


@Id
@GeneratedValue
private Integer id;


@Column(unique=true, nullable=false)
private String nom;


@OneToMany(mappedBy="groupe", fetch=FetchType.LAZY)
// LAZY = fetch when needed, EAGER = fetch immediately
private List<Etudiant> etudiants;


private static final long serialVersionUID = 1L;
```

Colonne id, clé primaire, générée automatiquement

Colonne nom, unique, non null

Représentation de la multiplicité oneToMany, propriété du fetch

Annotation JPA d'une entité (2/2)

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

public Integer getId() {
        return this.id;
}

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

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

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

public List<Etudiant> getEtudiants() {
        return this.etudiants;
}
```

Constructeur

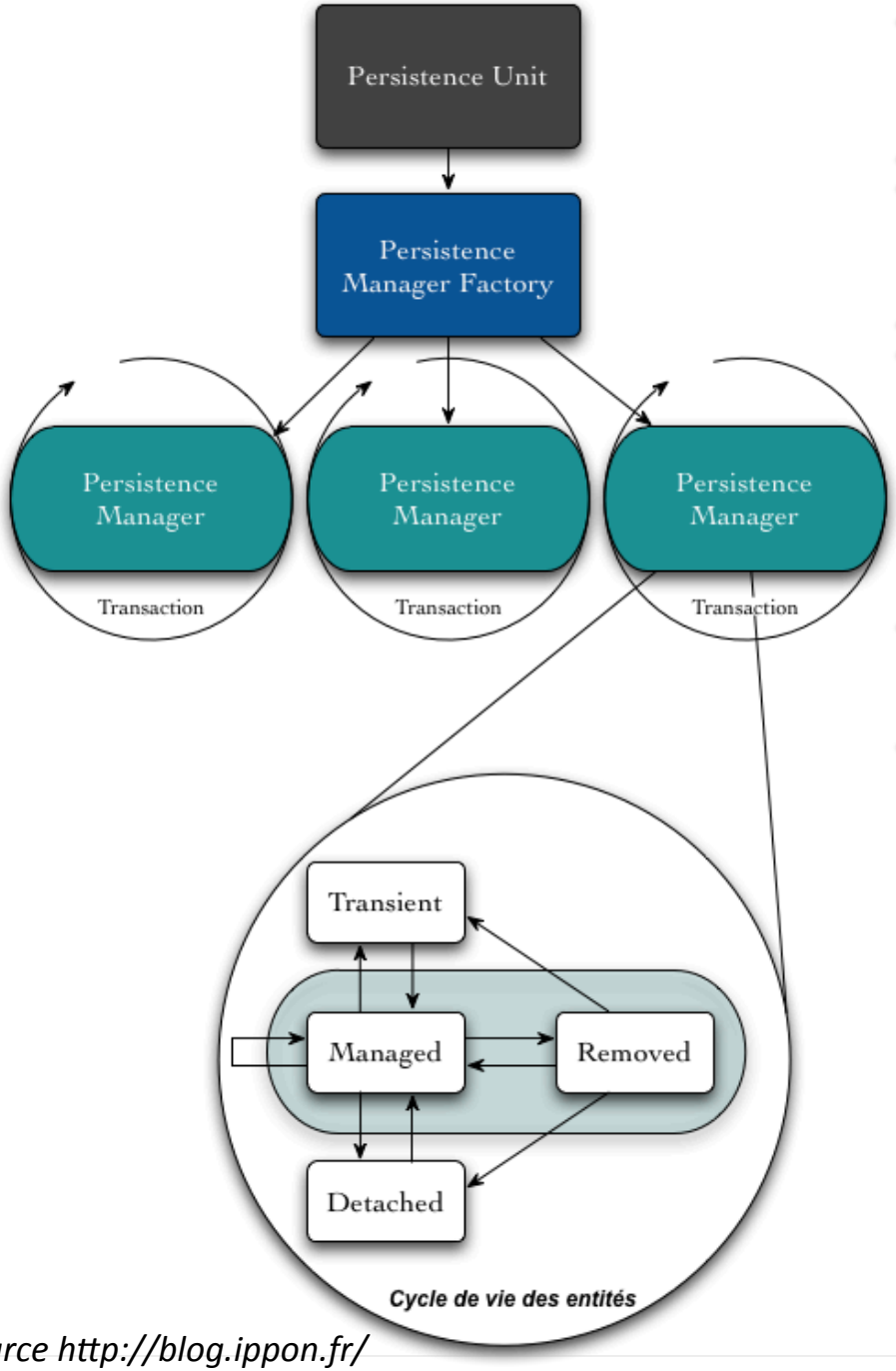
Méthodes getter et setter

Contrôle des données possible

Configuration de la connexion BD

- Fichier persistence.xml (dans le dossier META-INF)
  - Définition du persistence unit
  - Définition du provider
  - Définition des entités
  - Propriétés de connexion JDBC
    - Url, user, driver, etc.
  - Propriétés de l'outil ORM (EclipseLink)
    - Génération, cache, etc.
  - Etc.

Gestion des entités



organise les meta données qui définissent le mapping entre les entités et la base.

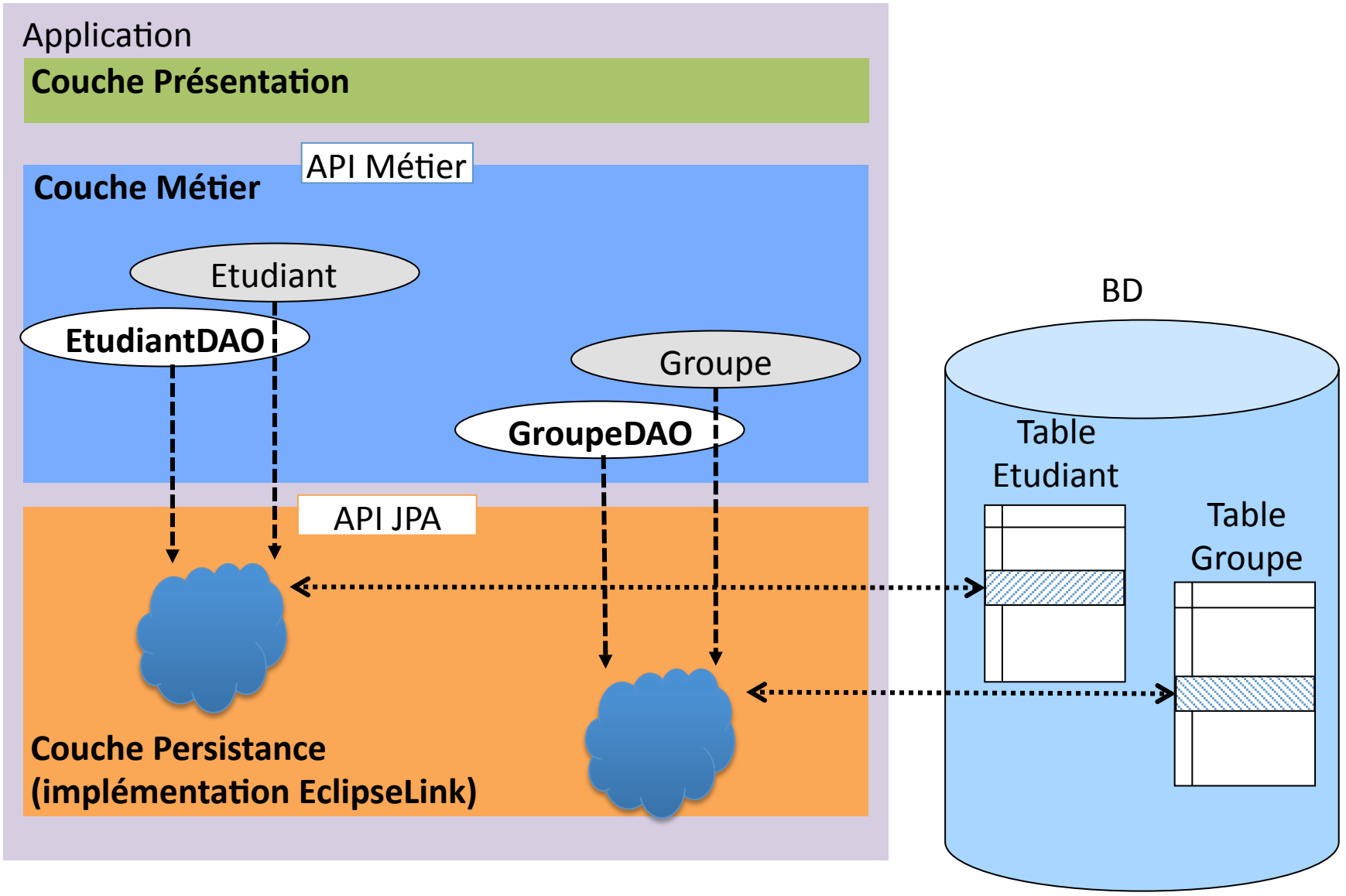
récupère ces metas données et les interprètent pour créer des EntityManager (Persistence Manager).

Une EntityManager (Persistence Manager) gère les échanges entre le code et la base de donnée, c'est à dire le cycle de vie des entités (englobé dans une transaction)

Cycle de vie des entités par rapport à l'EntityManager

Source http://blog.ippon.fr/

DAO (Data Access Object)



DAO (Data Access Object)

En français, *objet d'accès aux données*

- Patron de conception : propose de regrouper les accès aux données persistantes dans des classes à part, plutôt que de les disperser

- Méthodes de la classe GroupeDAO
  - create(String) : crée et retourne un groupe
  - getAll() : retourne tous les groupes
  - removeAll() : supprime tous les groupes

DAO : rechercher un objet par id

- JPA/EclipseLink va exécuter une requête SQL "select" et instantier un objet dont les attributs seront peuplés avec le résultat de cette requête.

```
Groupe groupe = em.find(Groupe.class, id);
// groupe est maintenant un objet de la classe Groupe
// ou NULL si le groupe n'existe pas
```

DAO : créer un objet (1/2)

- Pour créer un nouvel objet persistant, il suffit de l'instancier, de donner des valeurs à ses attributs et de le sauvegarder dans la BD par la méthode **persist**.

- JPA/EclipseLink traduira cela par une requête SQL "insert"

```
Groupe groupe = new Groupe();
groupe.setNom(nom);
em.persist(groupe);


// On peut maintenant accéder au champ id de l'objet créé
// (champ autoincrémenté)
int id = groupe.getId();
```

DAO : créer un objet (2/2)

- Attention à la gestion des entités

```
// Creation de l'entity manager
EntityManager em = GestionFactory.factory.createEntityManager();

// Début transaction
em.getTransaction().begin();

// Create new etudiant
Etudiant etudiant = new Etudiant();
etudiant.setPrenom(prenom);
etudiant.setNom(nom);
etudiant.setGroupe(groupe);
em.persist(etudiant);

// Commit
em.getTransaction().commit();

// Close the entity manager
em.close();
```

Méthodes de l'EntityManager

- PERSIST : rendre une entité persistante
- REMOVE : rendre une entité non persistante
- REFRESH : synchroniser l'état d'une entité persistante avec son état en base
- DETACH : détacher une entité persistante de l'EntityManager
- MERGE : attacher une entité persistante à l'EntityManager courant

DAO : rechercher un objet (1/2)
JPA Query Language (JPQL)

```
// Creation de l'entity manager
EntityManager em = GestionFactory.factory.createEntityManager();

// Recherche
Query q = em.createQuery("SELECT e FROM Etudiant e WHERE e.nbAbsences > 0");

@SuppressWarnings("unchecked")
List<Etudiant> listEtudiant = q.getResultList();
```

Pour plus de critères :
http://www.objectdb.com/java/jpa/query

DAO : rechercher un objet (2/2)
JPA Query Language (JPQL)

```
// Creation de l'entity manager
EntityManager em = GestionFactory.factory.createEntityManager();

//
em.getTransaction().begin();

//
em.createQuery("DELETE FROM Etudiant AS e WHERE e.id = :id")
        .setParameter("id", id)
        .executeUpdate();

// Commit
em.getTransaction().commit();

// Close the entity manager
em.close();
```

Pour plus de critères :
http://www.objectdb.com/java/jpa/query

Webographie

- Manuel

http://www.objectdb.com/java/jpa

- JPA : une magie qui se mérite

http://blog.ippon.fr/2011/10/11/jpa-une-magie-qui-se-merite-retour-aux-sources-de-jpa/

- openClassRooms

http://openclassrooms.com/courses/creez-votre-application-web-avec-java-ee/la-persistance-des-donnees-avec-jpa

Intégration à votre application web

1. Ajouter les librairies JPA/EclipseLink au WebContent
   - Ajouter les jar contenu dans l'archive JPA-lib.zip sur l'intratek dans le dossier WebContent/WEB-INF/lib/

2. Modifier votre servlet frontale (ou créer une servlet) pour initialiser et détruire correctement l'EntityManagerFactory
   - Méthode init() devra contenir GestionFactory.*open();*
   - Méthode destroy() devra contenir GestionFactory.*close();*

3. Mettre le fichier persistence.xml dans le dossier WebContent/WEB-INF/classes/META-INF