

ATELIER

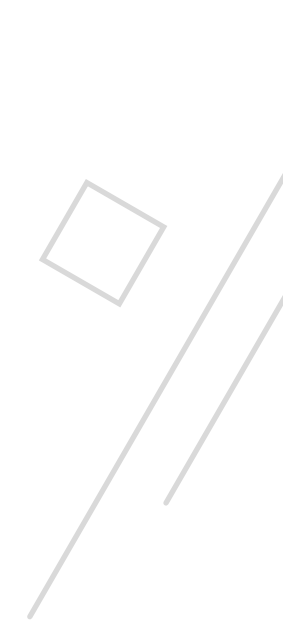
Persistance des données

dans un environnement

CLIENT-SERVEUR



Objectif

- **Savoir** définir la persistance des données
 - **Traiter** la conservation de l'état des applications d'une exécution à une autre
- 

The background is a solid teal color. It features several white geometric elements: a large square in the top-left corner, a smaller square in the top-right, a small square in the center-right, and a small square in the bottom-left. Additionally, there are several white lines of varying lengths and orientations scattered across the slide, some parallel to the edges and others at angles.

Partie I

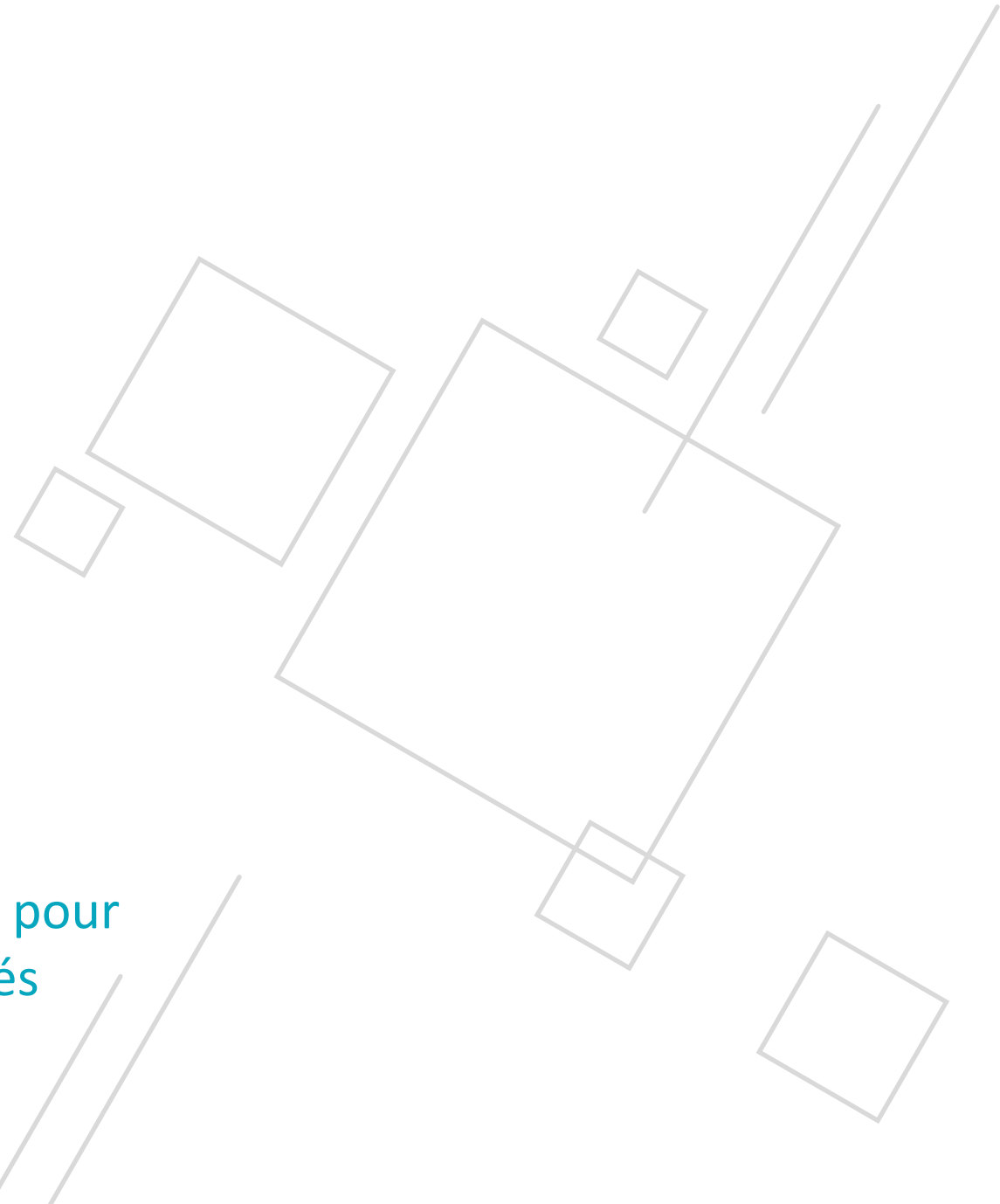
Introduction aux choix du Framework

Introduction aux Frameworks

Ensemble de composants qui structure une application et contraint la manière de développer :

- Lié à un langage de programmation
- Plutôt générique (contrairement à une librairie)

Abstraction de fonctionnalités et d'architecture pour un langage au détriment de certaines possibilités





Avantages du Framework

- Gain de temps et d'efficacité
- Meilleure structuration du code
- Maintenance / Evolution simplifiée
- Sécurisation



Inconvénients du Framework

- Limite les fonctionnalités complexes du langage
- Niveau d'abstraction supplémentaire
- Nouvelle technologie à supporter / intégrer / maintenir / ...
- Contrainte de poids

Comment choisir un **framework** ?

- **Définir** le cadre du projet (besoins, fonctionnalités, ...)
- **Ratio** avantages/inconvénients technologiques
- Quel **impact** au niveau projet ?

Métriques techniques

Exemple

- Facilité d'utilisation +
- Fonctionnalités déjà intégrées +
- Besoins client “standards” +
- Utilisation de fonctionnalités avancées du langage de programmation -
- Impact sur l'architecture / difficulté à changer de Framework -

Métriques projet

Exemple

- Stabilité +
- Maturité (techno et/ou paradigme) +
- Communauté +
- Maintenance / évolutions du Framework non garanties -
- Difficulté de formation -

Partie II

JPA – La persistance des données

La persistance des données

JPA : Java **P**ersistence **A**PI

- **Evite de long fichiers** Optimisation
- **Annotation** Transformation du code en requête SQL qui ramène des objets
- **Standard** Comment on doit coder un **ORM**

ORM : Object **R**elational **M**apping

- **Synchronisation** du monde objet et du monde de la base de données relationnelle

ORM : Object Relational Mapping

Java [\[edit \]](#)

- [Apache Cayenne](#), open-source for Java
- [Apache OpenJPA](#), open-source for Java
- [DataNucleus](#), open-source JDO and JPA implementation (formerly known as JPOX)
- [Ebean](#), open-source ORM framework
- [EclipseLink](#), Eclipse persistence platform
- [Enterprise JavaBeans](#) (EJB)
- [Enterprise Objects Framework](#), Mac OS X/Java, part of Apple [WebObjects](#)
- [Hibernate](#), open-source ORM framework, widely used
- [Java Data Objects](#) (JDO)
- [JOOQ Object Oriented Querying](#) (jOOQ)
- [Kodo](#), commercial implementation of both [Java Data Objects](#) and [Java Persistence API](#)
- [TopLink](#) by Oracle

.NET [\[edit \]](#)

- [Base One Foundation Component Library](#), free or commercial
- [Dapper](#), open source
- [Entity Framework](#), included in .NET Framework 3.5 SP1 and above
- [iBATIS](#), free open source, maintained by [ASF](#) but now inactive.
- [LINQ to SQL](#), included in .NET Framework 3.5
- [NHibernate](#), open source
- [nHydrate](#), open source
- [Quick Objects](#), free or commercial

PHP [\[edit \]](#)

- [Laravel](#), framework that contains an ORM called "Eloquent" an ActiveRecord implementation.
- [Doctrine](#), open source ORM for PHP 5.2.3, 5.3.X., 7.4.X Free software (MIT)
- [CakePHP](#), ORM and framework for PHP 5, open source (scalars, arrays, objects); based on database introspection, no class extending
- [CodeIgniter](#), framework that includes an ActiveRecord implementation
- [Yii](#), ORM and framework for PHP 5, released under the BSD license. Based on the ActiveRecord pattern
- [FuelPHP](#), ORM and framework for PHP 5.3, released under the MIT license. Based on the ActiveRecord pattern.
- [Laminas](#), framework that includes a table data gateway and row data gateway implementations
- [Propel](#), ORM and query-toolkit for PHP 5, inspired by [Apache Torque](#), free software, MIT
- [Qcodo](#), ORM and framework for PHP 5, open source
- [QCubed](#), A community driven fork of [Qcodo](#)
- [Redbean](#), ORM layer for PHP 5, for creating and maintaining tables on the fly, open source, BSD
- [Skipper](#), visualization tool and a [code/schema generator](#) for PHP [ORM frameworks](#), commercial

- **Hibernate** utilise à la fois ces propres mécanismes mais aussi ceux de **JPA**

ORM : Object Relational Mapping

L'ORM s'appuie pour cela sur une configuration **associant les classes du modèle fonctionnel et le schéma de la base de données.**

L'ORM **génère des requêtes SQL** qui permettent de **matérialiser ce graphe ou une partie de ce graphe en fonction des besoins.**

L'ORM va permettre d'obtenir **les 4 actions CRUD** :

- Create
- Update
- Read
- Delete

Des codes plus robuste en terme de sécurité **cependant si l'ORM est mal maîtrisé celui-ci peut dégradé les performances de vos programmes !**

Hibernate

Hibernate est une solution open source de type **ORM** (Object Relational Mapping) développé par RedHat qui permet de faciliter **le développement de la couche persistance d'une application**.

Hibernate permet donc de représenter une base de données en objets Java et vice versa

Ceci afin d'abstraire l'implémentation de la base de données du code (La plupart des BD sont supportées)

Annotation d'entité

L'annotation **@Entity** nous indique que cette classe est une classe persistante.

L'annotation **@Table** permet de fixer le nom de la table dans laquelle les instances de cette classe vont être écrites.

Pour chaque table, un fichier portant le nom de celui-ci doit être créé, ici le fichier est **MyEntity.java**

```
@Entity // il s'agit d'une classe à persister
@Table(name = "MY_ENTITY",
        uniqueConstraints = { @UniqueConstraint(columnNames = { "Id" }) }) // Insertion d'une contrainte
                                Unique sur la colonne Id

public class MyEntity {
    private Integer myId;
    private String myString;

    @Id // @Id indique une clé primaire
    public Integer getMyId() {
        return myId;
    }
}
```

Annotation de champs

@Column : Cette annotation peut se poser sur un champ ou sur un getter

```
...  
public class MyEntity {  
    ...  
  
    @Column  
    public String getMyString() {  
        return myString;  
    }  
}
```


Annotation de champs

L'annotation **@Column** expose les attributs optionnels suivants :

- **name** : permet de fixer le nom de la colonne.
- **length** : pour les chaînes de caractères, cet attribut permet de fixer la taille du champ SQL associé.
- **unique** : ajoute une contrainte d'unicité sur la colonne.
- **nullable** : empêche cette colonne de porter des valeurs nulles. Cette contrainte est utile pour la définition de clés étrangères.
- **insertable, updatable** : empêche la modification des valeurs de cette colonne, utilisée pour les clés étrangères.

Annotation de champs

@Column : Cette annotation peut se poser sur un champ ou sur un getter

...

```
public class MyEntity {  
  
    @Column(name="family_name", length=50)  
    private String nom ;  
  
    @Column(name="first_name", length=50)  
    private String prenom ;  
    private int age ;  
  
    @Column(name="civility", length=12)  
    @Enumerated(EnumType.ORDINAL) // ou EnumType.STRING  
    private Civility civilite ;  
}
```

Relation d'entités

Tout comme en SQL, on peut définir quatre types de relations entre entités JPA :

- relation **1:1** : annotée par **@OneToOne**
- relation **n:1** : annotée par **@ManyToOne**
- relation **1:n** : annotée par **@OneToMany**
- relation **n:n** : annotée par **@ManyToMany**

Relation 1:1 unidirectionnelle

```
@Entity
public class Maire implements Serializable {

    @Id
    private Long id;

    @Column(length=40)
    private String nom ;

    // suite de la classe

}
```

```
@Entity
public class Commune implements Serializable {

    @Commune_Id
    private Long id;

    @Column(length=40)
    private String nom ;

    ➡ @OneToOne
      @JoinColumn(name="Maire_fk")
      private Maire maire ;

    // suite de la classe

}
```

Relation 1:1 bidirectionnelle

```
@Entity
public class Maire implements Serializable {

    @Id
    private Long id;

    @Column(length=40)
    @OneToOne(mappedBy="maire_fk")
    // référence la relation dans la classe Commune
    private String nom ;

    // suite de la classe
}
```

```
@Entity
public class Commune implements Serializable {

    @Commune_Id
    private Long id;

    @Column(length=40)
    private String nom ;

    @OneToOne
    @JoinColumn(name="Maire_fk")
    private Maire maire ;

    // suite de la classe
}
```

Relation 1:n unidirectionnelle

```
@Entity
public class Marin implements Serializable {

    @Id
    private Long id;

    // suite de la classe
}
```

```
@Entity
public class Bateau implements Serializable {

    @Commune_Id
    private Long id;

    ➡ @OneToMany
    private Collection<Marin> marins ;

    // suite de la classe
}
```

Relation 1:n bidirectionnelle

```
@Entity
public class Marin implements Serializable {

    @Id
    private Long id;

    ➡ @ManyToOne
    private Collection<Marin> marins ;

    // suite de la classe
}
```

```
@Entity
public class Bateau implements Serializable {

    @Commune_Id
    private Long id;

    @OneToMany
    private Collection<Marin> marins ;

    // suite de la classe
}
```

Relation n:n unidirectionnelle

```
@Entity
public class Musicien implements Serializable {

    @Id
    private Long id;

    ➡ @ManyToMany
    private Collection<Instrument> instruments ;

    // suite de la classe
}
```

```
@Entity
public class Instrument implements Serializable {

    @Commune_Id
    private Long id;

    // suite de la classe
}
```


Relation n:n bidirectionnelle

```
@Entity
public class Musicien implements Serializable {

    @Id
    private Long id;

    ➡ @ManyToMany
    private Collection<Instrument> instruments ;

    // suite de la classe
}
```

```
@Entity
public class Instrument implements Serializable {

    @Commune_Id
    private Long id;

    ➡ @ManyToMany(mappedBy="instruments")
    private Collection<Musicien> musiciens ;
    // suite de la classe
}
```

Relation et annotation Fetch lazy ou fetch eager

Lorsque JPA doit charger une entité depuis la base de données, comment les information doivent-il être chargées ?

- (fetch = FetchType.**EAGER**) signifie que l'information doit être chargée systématiquement lorsque l'entité est chargée. Cette stratégie est appliquée par défaut pour **@Basic**, **@OneToOne** et **@ManyToOne**.
- (fetch = FetchType.**LAZY**) signifie que l'information ne sera chargée qu'à la demande (par exemple lorsque la méthode **get** de l'attribut sera appelée). Cette stratégie est appliquée par défaut pour **@OneToMany** et **@ManyToMany**.

Cette annotation peut se poser sur un champ ou sur un getter

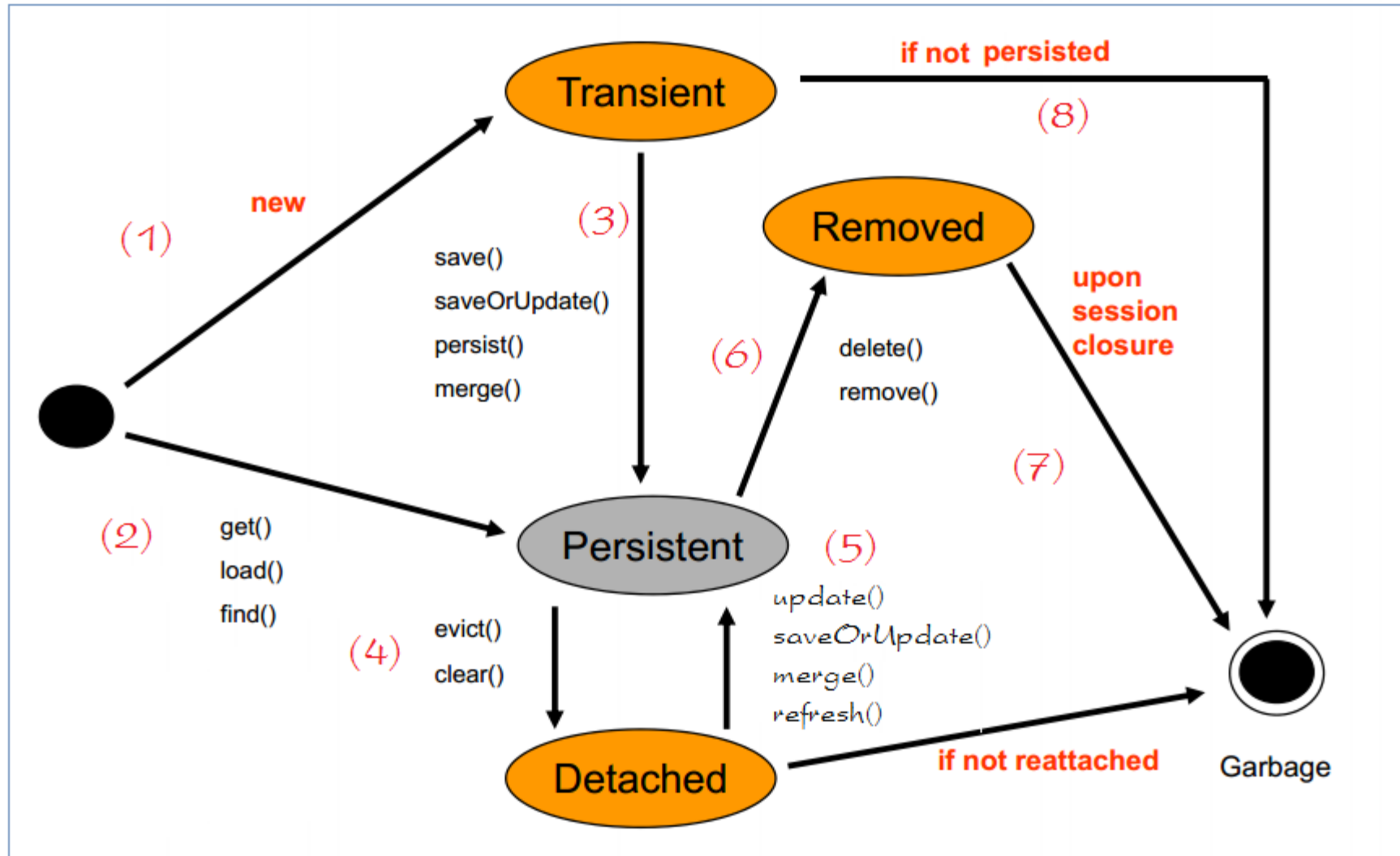
Manipuler des entités

À partir d'une instance d'**EntityManager**, nous allons **pouvoir manipuler les entités** afin de les créer, les modifier, les charger ou les supprimer. Pour cela, nous disposons de **six méthodes** :

- find : **EntityManager.find (Class<T>, Object)**
- persist : **EntityManager.persist**
- merge : **EntityManager.merge(T)**
- detach : **EntityManager.detach(Object)**
- refresh : **EntityManager.refresh(Object)**
- remove : **EntityManager.remove(Object)**

L'EntityManager va prendre en charge la relation avec la base de données et la génération des requêtes SQL nécessaires.

4 états d'un objet



TP

En suivant le tutoriel d'initiation à Hibernate, écrivez le programme répondant à l'énoncé suivant :

Un boulanger souhaite optimiser la prise de commande depuis une tablette. Avec l'aide d'Hibernate vous allez mettre en place la persistance des données de l'application qui possède les fonctions suivantes :

- Enregistrement d'un nouveau client (nom, prenom, telephone)
- Enregistrement des produits du boulanger et de leur prix (produit, prix)
- Enregistrement des commandes des clients (date, heure, article, client)

- Afficher la dernière commande d'un client
- Afficher toutes les commandes d'un client

Initiation à Hibernate :

<https://devstory.net/10201/java-hibernate>