**JPA (Java Persistence API) e Hibernate**

A Java Persistence API (JPA) é um framework para persistência em Java, que oferece uma API de mapeamento objeto-relacional e soluções para integrar persistência com sistemas corporativos escaláveis.

Com JPA, os objetos são **POJO (Plain Old Java Objects)**, ou seja, não é necessário nada de especial para tornar os objetos persistentes. Basta adicionar algumas anotações nas classes que representam as entidades do sistema e começar a persistir ou consultar objetos.

JPA é uma especificação, e não um produto. Para trabalhar com JPA, precisamos de uma implementação.

O projeto do **Hibernate ORM** possui alguns módulos, sendo que o **Hibernate EntityManage**r é a implementação da JPA que encapsula o **Hibernate Core**. O Hibernate Core é a base para o funcionamento da persistência, com APIs nativas e metadados de mapeamentos em arquivos XML. Possui uma linguagem de consultas chamada **HQL** (parecido com SQL), um conjunto de interfaces para consultas usando critérios (Criteria API), etc.

**Configuração de JPA e Hibernate com Maven**

Como estamos usando Maven, não precisamos acessar o site do Hibernate para baixar os arquivos necessários e incluir manualmente no projeto. Podemos incluir todas as dependências no arquivo **pom.xml**, que o Maven baixará os arquivos necessários automaticamente.

**<dependencies>**

<!-- Núcleo do Hibernate -->

**<dependency>**

**<groupId>**org.hibernate**</groupId>**

**<artifactId>**hibernate-core**</artifactId>**

**<version>**4.3.8.Final**</version>**

**<scope>**provided**</scope>**

**</dependency>**

<!-- Implementação de EntityManager da JPA -->

**<dependency>**

**<groupId>**org.hibernate**</groupId>**

**<artifactId>**hibernate-entitymanager**</artifactId>**

**<version>**4.3.8.Final**</version>**

**<scope>**provided**</scope>**

**</dependency>**

**</dependencies>**

**Definindo uma 'JPA Persistence Entity'**

Para ser capaz de armazenar objetos em um banco de dados relacional usando JPA primeiramente precisamos definir uma **'Persistence Entity'**. Uma JPA Persistence Entity nada mais é que uma **POJO (Plain Old Java Object)** *class*, ou seja, uma classe Java qualquer (com métodos *getters* e *setters* e um construtor padrão) que é marcada (anotada) para ter a capacidade de representar objetos na base de dados.

**Ex**.:

@Entity

public class Produto {

private String descricao;

private Double valor;

// getter's and setter's

}

Como pode-se perceber, na classe acima a única adição para que ela represente o formato JPA foi a inserção da anotação **@Entity** na classe, que marca que a classe agora é uma 'Persistence Entity'.

**Campos persistentes em uma 'Persistence Entity'**

Ao armazenar objetos em um banco de dados, os seus métodos ou código não são persistidos. Somente o estado persistente do objeto gerado a partir da '**Persistence** **Entity**', representado por seus campos persistentes, é armazenado. Por padrão, qualquer campo que não for declarado como @**Transient** é um campo persistente. Em nosso exemplo os campos persistentes de Produto são '**descricao**' e '**valor**'. Os valores que estiverem armazenados nesses dois campos serão armazenados no banco quando o objeto for persistido (salvo).

**Persistence Entity**

**Restrições:**

* Devem ser uma classe de nível superior (**não** uma **classe interna** ou **classe aninhada**).
* Devem ter um **construtor padrão** (sem argumentos)
* Não podem ser '**final**' e não pode conter métodos finais ou variáveis de instância finais.

**Possibilidades:**

* Podem estender outras **'Persistence Entity'**
* Podem estender classes que não são ' **Persistence** **Entity**', porém não podem estender classes de sistemas, tal como **ArrayList**.
* Podem implementar qualquer **interface**.
* Podem conter construtores, métodos, campos e tipos como qualquer modificador de acesso (**public**, **protected** ou **private**).
* Podem ser **concretas** ou **abstratas**.

**Relacionamentos**

Os relacionamentos entre as entidades de domínio devem ser expressos na modelagem através de vínculos entre as classes. Para isso podemos definir quatro tipos de relacionamentos entre classe de acordo com a cardinalidade, são eles:

**One to One** (Um para Um)

**One to Many** (Um para Muitos)

**Many to One** (Muitos para Um)

**Many to Many** (Muitos para Muitos)

Relacionamentos, por padrão, executam carregamento retardado ou preguiçoso (**Lazy** Loading), já campos básicos possuem carregamento imediato (**Eager** Loading).

**Herança**

A JPA define três estratégias para se mapear o uso de herança em uma aplicação, são elas:

* **Single Table** (Tabela única)
* **Joined** (Junção)
* **Table per Class** (Tabela por classe)

Mais detalhes: <http://www.devmedia.com.br/tipos-de-heranca-no-hibernate/28641>

**Principais anotações**

**@Table** => Especifica detalhes sobre a tabela no banco.

**@Entity** => Define uma classe como 'Persistence Entity'

**@Column** => Especifica detalhes sobre uma coluna de uma tabela.

**@Basic** => Define como tipo básico o atributo.

**@Id** => Determina a chave primária de uma Entity Class.

**@GeneratedValue** => Define que o valor de uma chave primária será gerado automaticamente.

**@Timestamp** => Especifica o tipo do formato do campo data que será salvo no banco.

**@ElementCollection** => Especifica detalhes sobre uma coleção de tipos básicos.

**@Convert** => Especifica uma converção de um tipo básico.

**@Converts** => Usada para especificar um grupo de converts.

**@JoinColumn** => Especifica detalhes sobre a coluna de junção do relacionamento.

**@OneToOne** => Mapeia o relacionamento entre classes do tipo 'um-para-um'

**@OneToMany** => Mapeia o relacionamento entre classes do tipo 'um-para-muitos'

**@ManyToOne** => Mapeia o relacionamento entre classes do tipo 'muitos-para-um'

**@ManyToMany** => Mapeia o relacionamento entre classes do tipo 'muitos-para-muitos'

**@Lob** => Utilizado para campos com formato binário

**@Inheritance** => Define a estratégia a ser utilizada sobre o comportamento da herança da entidade no banco de dados.

**@DiscriminatorColumn**: Especifica a coluno que irá discriminar objetos no banco de dados quando é utilizada a estratégia **Single Table.**

**@DiscriminatorValue**: Indica o valor de discriminação de um objeto.

**@JoinTable** => Especifica detalhes sobre tabela de junção

**@JoinColumns** => Especifica detalhes sobre colunas de junção, geralmente quando há o uso de chaves compostas. Para especificar mais de uma **@JoinColumn**.

**@Enumerated** => Especifica detalhes para atributos do tipo enum.

**@Embedded** => Especifica que um campo de uma entidade é instancia de uma classe embutida. (Embeddable entity)

**Obtendo um EntityManager**

O objeto **EntityManager** é o responsável por gerenciar (persistir, obter, atualizar ou remover) uma **Persistence Entity.**

**Obs.:** Para aplicações web utilizando servidor de aplicação, temos a vantagem de não se preocupar em como instanciar um objeto **EntityManager**. Ele será instanciado pelo próprio servidor de aplicação.

private final static String UNIT\_NAME = "persistence-unit-name";

@PersistenceContext(unitName = UNIT\_NAME)

protected EntityManager em;

**Principais métodos do EntityManager**

**persist**: usado para salvar objetos na base de dados

**merge**: usado para atualizar objetos na base de dados

**refresh**: usado para sincronizar o objeto com a base de dados.

**remove**: usado para remover um objeto da base de dados.

**find**: usado para buscar um objeto na base de dados através da chave primária.

**createQuery**: usado para obter uma instancia de Query para executar consultas via JPQL, seu retorno é uma lista do objeto que se está procurando.

**createQuery** (dois parametros): mesma finalidade acima, com uma pequena diferença que também é passado a classe de retorno dos objetos no momento da instanciação, afim de trazer o resultado com a tipagem correta, evitando o uso de casting's.

**createNativeQuery**: usado para efetuar consultas nativas do banco, seu retorno padrão é uma Lista de Array de Objetos.

**Ex.:**

Query queryHQL = em.createQuery(consultaHQL);

produtos = (List) queryHQL.getResultList();

Query queryNativa = em.createNativeQuery(consultaNativa);

List resultado = queryNativa.getResultList();

TypedQuery typedQuery = em.createQuery(consultaHQL, Produto.class);

produtos = typedQuery.getResultList();

**Referências**

**Livro de apoio**: <http://alga.works/livro-javaee/>

**Tutorial JPA**: <http://www.objectdb.com/java/jpa>

**JPA e ORM's**: <http://www.caelum.com.br/apostila-java-web/uma-introducao-pratica-ao-jpa-com-hibernate/#14-2-java-persistence-api-e-frameworks-orm>