



Curso de **Java8** para **Web**

Professor
Antonio Benedito Coimbra Sampaio Jr

abc  | Treinamentos

www.abctreinamentos.com.br

Terceira Disciplina

JEE - Persistência de Dados com JDBC e Hibernate

- **UNIDADE 1:** Arquitetura JEE
- **UNIDADE 2:** Introdução a Banco de Dados com Oracle
- **UNIDADE 3:** Persistência de Dados com JDBC
- **UNIDADE 4:** Framework Hibernate
- **UNIDADE 5: Introdução ao JPA**

UNIDADE 5

INTRODUÇÃO

AO JPA

Java Persistence API (JPA)

JPA

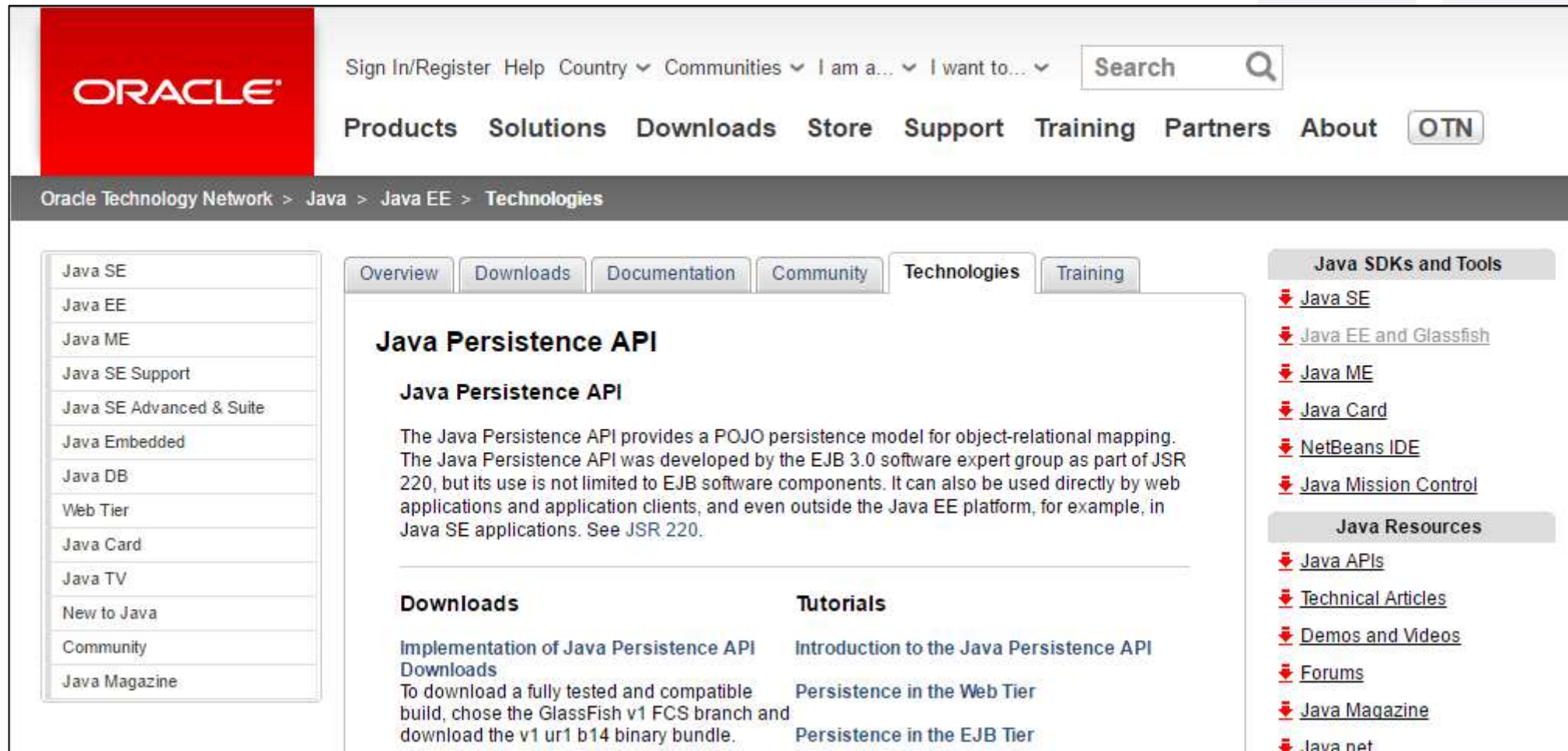
Problemas dos Frameworks ORM

- Construir aplicações Java com acesso a banco de dados utilizando os frameworks de persistência diretamente é complexo, envolve o relacionamento entre as tabelas através de linhas e necessita o uso de consultas.
- Dificuldade e até a impossibilidade de fazer grandes alterações no projeto depois de concluído.
- Falta padronização nos frameworks de persistência.

O Que é o JPA?

- É uma camada que funciona acima dos frameworks de persistência (Hibernate, EclipseLink, Toplink, Spring Data JPA, etc.) e facilita o mapeamento do Banco de Dados.

JPA



Oracle Technology Network > Java > Java EE > Technologies

Sign In/Register Help Country Communities I am a... I want to... Search

Products Solutions Downloads Store Support Training Partners About OTN

Overview Downloads Documentation Community Technologies Training

Java Persistence API

Java Persistence API

The Java Persistence API provides a POJO persistence model for object-relational mapping. The Java Persistence API was developed by the EJB 3.0 software expert group as part of JSR 220, but its use is not limited to EJB software components. It can also be used directly by web applications and application clients, and even outside the Java EE platform, for example, in Java SE applications. See JSR 220.

Downloads

Implementation of Java Persistence API Downloads

To download a fully tested and compatible build, chose the GlassFish v1 FCS branch and download the v1 ur1 b14 binary bundle.

Tutorials

Introduction to the Java Persistence API

Persistence in the Web Tier

Persistence in the EJB Tier

Java SDKs and Tools

- Java SE
- Java EE and Glassfish
- Java ME
- Java Card
- NetBeans IDE
- Java Mission Control

Java Resources

- Java APIs
- Technical Articles
- Demos and Videos
- Forums
- Java Magazine
- Java.net

<http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>

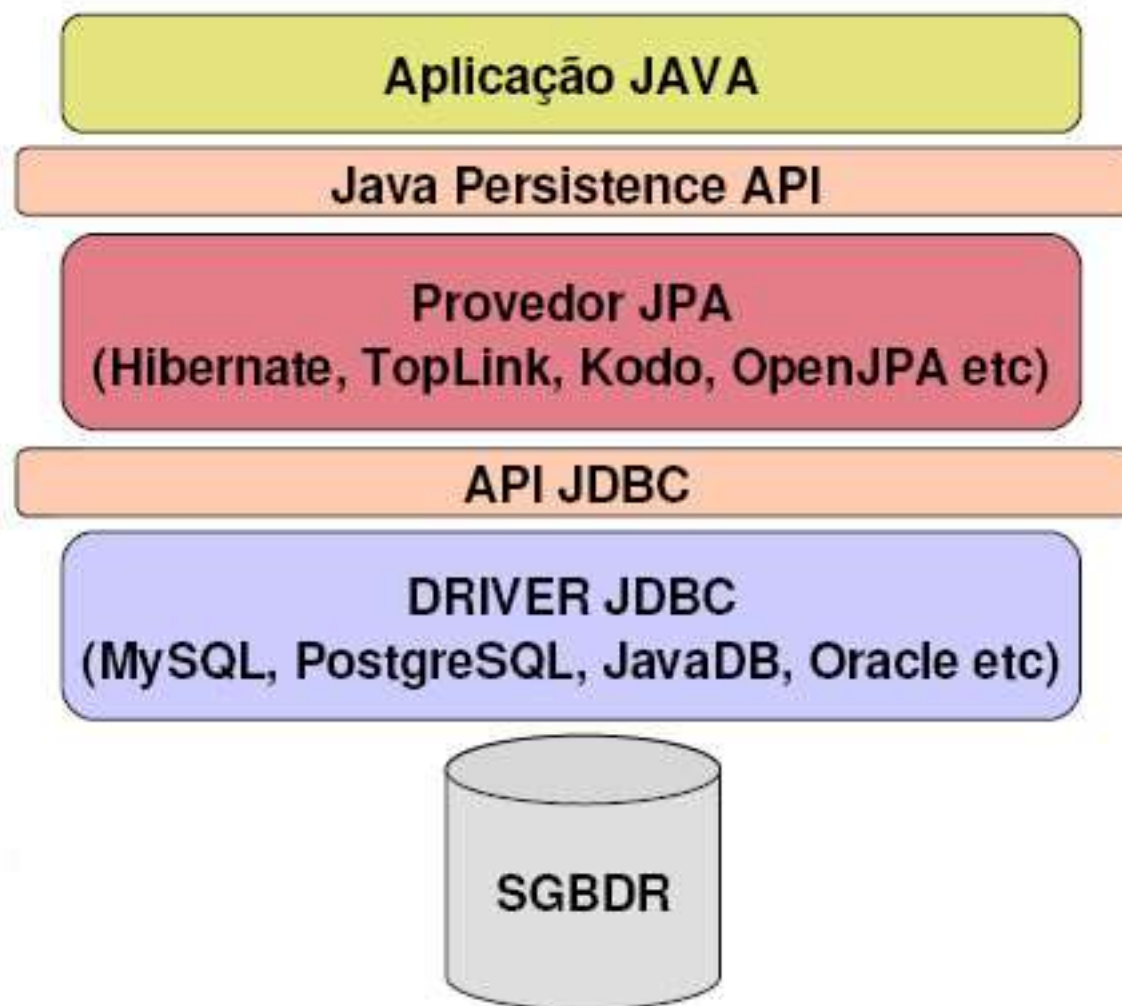
JPA

Características

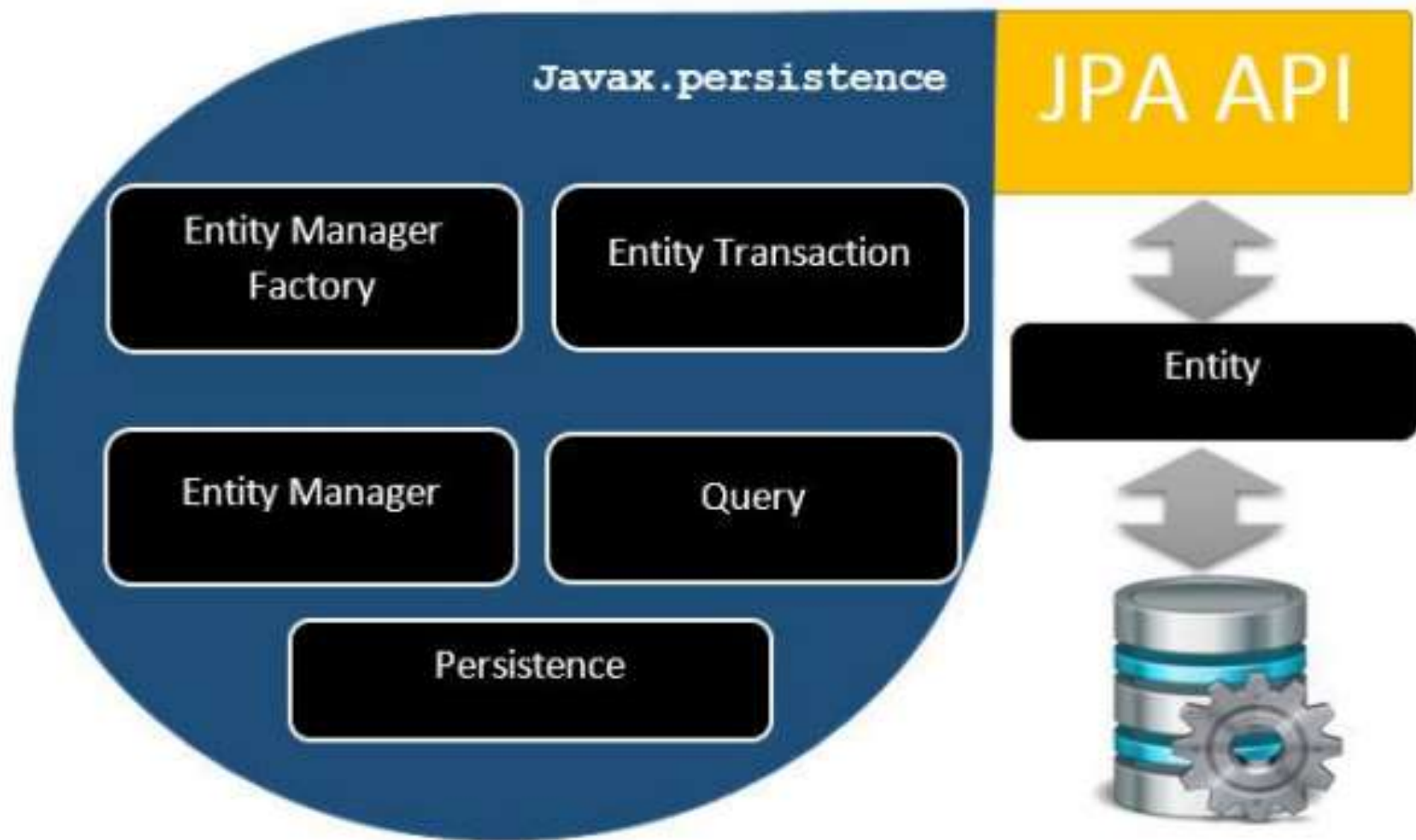
- É uma especificação Java que facilita o uso dos frameworks ORM, sem provocar um aumento no tempo de construção da aplicação.
- Portável para vários frameworks de persistência (Hibernate, TopLink, etc.).
- Arquivo (**persistence.xml**) é utilizado para configurar conexões ao banco de dados e apontar para os entidades que representam as classes.
- Anotações são usadas para definir Tabelas, campos, tipos de relacionamentos (1:1, 1:N, herança) e outras propriedades.

JPA

- A versão mais atual é a 2.1, lançada em 2013 com a nova especificação JEE 7.



Arquitetura JPA



<http://www.tutorialspoint.com/jpa>

Arquitetura JPA

Principais Classes e Interfaces

- As principais classes e interfaces da especificação JPA são utilizadas para armazenar entidades em registros de banco de dados.
- Estas classes e interfaces estão definidas no pacote **javax.persistence** e são utilizadas para auxiliar o desenvolvedor a escrever menos código.

São elas:

- **EntityManager**
 - É uma Interface que gerencia as operações de persistência em objetos.
- **EntityManagerFactory**
 - É uma classe que implementa o padrão de projeto Factory para criar e gerenciar várias instâncias da classe EntityManager.

Arquitetura JPA

São elas:

- **Entity**
 - É a anotação utilizada para informar que uma dada classe pode ser 'persistida' em uma tabela de um banco de dados.
- **EntityManager**
 - É a Interface utilizada para controlar as transações.
- **Persistence**
 - É a classe que possui métodos estáticos para obter um objeto da classe **EntityManagerFactory**.
- **Query**
 - É a interface implementada para controlar a execução de instruções SQL.

Exercícios

1) [CESPE - 2014 - TJ-SE] Julgue o item abaixo, relativo à JPA (Java Persistence API).

A JPA, que foi criada como alternativa para o Hibernate para conexão com os sistemas gerenciadores de banco de dados, está nativa no Java SE a partir da versão 1.3.

a) Certo b) Errado

2) [FEPESE - 2013 - JUCESC] Em relação à JPA e Hibernate, considere as seguintes afirmativas.

1. JPA Especifica uma JSR;
2. Hibernate Especifica uma JSR;
3. Hibernate cuida da camada de persistência enquanto JPA da camada de transação;
4. Hibernate é uma implementações de JSR;
5. JPA é uma Implementação de JSR.

Exercícios

2) [FEPESE - 2013 - JUCESC] Em relação à JPA e Hibernate, considere as seguintes afirmativas.

Assinale a alternativa que indica todas as afirmativas corretas.

- a) São corretas apenas as afirmativas 1 e 4.
- b) São corretas apenas as afirmativas 2 e 3.
- c) São corretas apenas as afirmativas 3 e 4.
- d) São corretas apenas as afirmativas 1, 2 e 3.
- e) São corretas apenas as afirmativas 3, 4 e 5

Exercícios

1)

A JPA, que foi criada como alternativa para o Hibernate para conexão com os sistemas gerenciadores de banco de dados, está nativa no Java SE a partir da versão 1.3.

a) Certo **b) Errado**

2)

Assinale a alternativa que indica todas as afirmativas corretas.

a) São corretas apenas as afirmativas 1 e 4.

b) São corretas apenas as afirmativas 2 e 3.

c) São corretas apenas as afirmativas 3 e 4.

d) São corretas apenas as afirmativas 1, 2 e 3.

e) São corretas apenas as afirmativas 3, 4 e 5

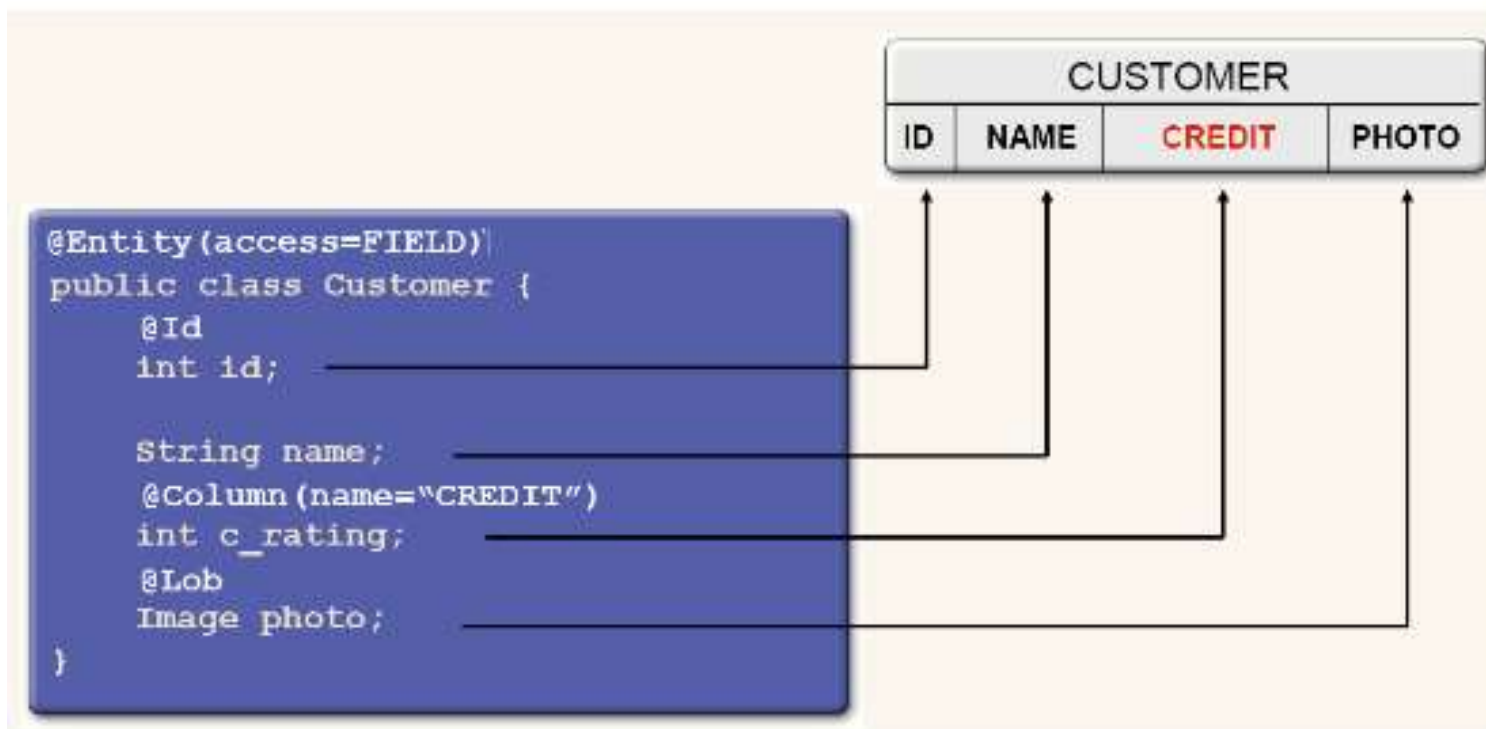
Principais Componentes do JPA

Principais Componentes

São 03:

- Mapeamento das Tabelas em Classes (anotações)
- Arquivo de Configuração (persistence.xml)
- Classe de gerenciamento (EntityManager.java)

Mapeamento das Tabelas em Classes



Mapeamento das Tabelas

- O mapeamento das Tabelas em Classes é feito com o uso de anotações.

Principais Anotações JPA:

- **@Entity**
- **@Table**
- **@Column**
- **@Id**
- **@NamedQuery**
- **@NamedQueries**
- **@ManyToOne, @OneToMany e @ManyToMany**

@Entity e @Table

- **@Entity** é a anotação utilizada para informar que uma dada classe pode ser 'persistida' em uma tabela de um banco de dados.
- **@Table** é a anotação utilizada para informar qual o nome da tabela do banco de dados que a classe vai utilizar.

```
@Entity  
@Table(name = "DEPENDENTE", schema = "NTEC")  
public class Dependente implements java.io.Serializable {
```

- Caso não seja informada esta anotação, o JPA considera o nome da tabela como sendo o mesmo nome da classe.

@Column e @Id

- **@Column** é a anotação utilizada para indicar que o atributo em questão representa uma coluna de uma tabela de banco de dados.

```
@Id
@Column(name = "IDDEPENDENTE", unique = true,
        nullable = false, precision = 22, scale = 0)
public Long getIddependente() {
    return this.iddependente;
}
```

- O atributo “*name*” indica o nome da coluna na tabela; “*unique*” indica se o valor da coluna é único ou não; “*nullable*” indica se o valor da coluna pode ser nulo ou não.
- **@Id** é a anotação utilizada para identificar uma chave primária.

@GeneratedValue

- **@GeneratedValue** é a anotação utilizada para indicar que o valor da chave primária é gerado automaticamente pelo SGBD.

```
@Id
@GeneratedValue (strategy = GenerationType.IDENTITY)
@Column(name = "IDDEPENDENTE", unique = true,
nullable = false, precision = 22, scale = 0)
public Long getIddependente() {
    return this.iddependente;
}
```

- O atributo “*strategy*” indica como será a geração da chave: AUTO, IDENTITY SEQUENCE ou TABLE.
- nome da coluna na tabela; “*unique*” indica se o valor da coluna é único ou não; “*nullable*” indica se o valor da coluna pode ser nulo ou não.

@NamedQuery e @NamedQueries

- **@NamedQuery** é a anotação utilizada para representar uma única consulta a um banco de dados.

```
@Entity
@Table(name = "FUNCIONARIO", schema = "NTEC")
@NamedQuery(name = "Consulta", query = "select f from
FUNCIONARIO f where f.id = :id")
public class Funcionario implements java.io.Serializable {
```

- **@NamedQueries** é a anotação utilizada para representar várias consultas a um banco de dados.

```
@NamedQueries ({
@NamedQuery(name = "Domic.findById", query = "SELECT r FROM Domic r WHERE r.id = :id"),
@NamedQuery(name = "Domic.findByName", query = "SELECT r FROM Domic r WHERE r.nome = :nome")
})
```

@ManyToOne, @OneToMany e @ManyToMany

- **@ManyToOne**, **@OneToMany** e **@ManyToMany** são as anotações utilizadas para representar relacionamentos muitos-para-um, um-para-muitos e muitos-para-muitos, respectivamente.
- **@ManyToOne** e **@OneToMany** se diferenciam pelo ponto de vista da entidade.

```
// Classe Dependente
@ManyToOne(fetch = FetchType.LAZY)
@JoinColumn(name = "IDFUNCIONARIO", nullable = false)
public Funcionario getFuncionario() {


// Classe Funcionario
@OneToMany(cascade = CascadeType.ALL, fetch =
FetchType.LAZY, mappedBy = "funcionario")
public Set<Dependente> getDependentes() {
```

@ManyToOne, @OneToMany e @ManyToMany

- O atributo “*fetch*” indica como o conteúdo dos dados relacionados é acessado:
 - **EAGER**, sempre que o objeto “pai” for trazido da base de dados, o conteúdo das classes filhas também vai ser trazido.
 - **LAZY**, o conteúdo das classes filhas só vai ser trazido quando o objeto filho for utilizado.
- O atributo “*mappedBy*” indica de que ponto de vista o relacionamento está representado.
- O atributo “*cascade*” indica como deve ser propagada a ação realizada na classe ‘1’ para as classes ‘N’.

@ManyToOne, @OneToMany e @ManyToMany

- O *cascade* só é acionado pelo JPA quando a ação for executada pela entidade em que o atributo 'cascade' foi definido.
- São 06 as opções de 'cascade':
 - CascadeType.DETACH
 - CascadeType.MERGE
 - CascadeType.PERSIST
 - CascadeType.REFRESH
 - CascadeType.REMOVE
 - CascadeType.ALL



Orders		
orderid	customerid	orderdate
10001	FRODO	17/04/99
10002	GNDLF	18/04/99
10003	BILBO	19/04/99

❌!DELETE FROM Orders
WHERE OrderID = 10002

deleted		
orderid	customerid	orderdate
10002	GNDLF	18/04/99

OrderDetails		
orderid	partid	quantity
10001	11	12
10001	42	10
10001	72	5
10002	14	9
10002	51	40
10003	41	10
10003	51	35
10003	65	15

Opções de 'cascade'

Tipo	Ação	Disparado por
<code>CascadeType.DETACH</code>	Quando uma entidade for retirada do Persistence Context (o que provoca que ela esteja detached) essa ação será refletida nos relacionamentos.	Persistence Context finalizado ou por comando específico: <code>entityManager.detach()</code> , <code>entityManager.clear()</code> .
<code>CascadeType.MERGE</code>	Quando uma entidade tiver alguma informação alterada (update) essa ação será refletida nos relacionamentos.	Quando a entidade for alterada e a transação finalizada ou por comando específico: <code>entityManager.merge()</code> .
<code>CascadeType.PERSIST</code>	Quando uma entidade for nova e inserida no banco de dados essa ação será refletida nos relacionamentos.	Quando uma transação finalizada ou por comando específico: <code>entityManager.persist()</code> .

Opções de 'cascade'

<code>CascadeType.REFRESH</code>	Quando uma entidade tiver seus dados sincronizados com o banco de dados essa ação será refletida nos relacionamentos.	Por comando específico: <code>entityManager.refresh()</code> .
<code>CascadeType.REMOVE</code>	Quando uma entidade for removida (apagada) do banco de dados essa ação será refletida nos relacionamentos.	Por comando específico: <code>entityManager.remove()</code> .
<code>CascadeType.ALL</code>	Quando qualquer ação citada acima for invocada pelo JPA ou por comando, essa ação será refletida no objeto.	Por qualquer ação ou comando listado acima.

© Hébert Coelho

Exercício

1) [FCC - 2011 - TRT] Considere o trecho de código abaixo:

```
@Entity
@Table(name = "domic")
@NamedQueries ({
    @NamedQuery(name = "Domic.findById", query = "SELECT r FROM Domic r WHERE r.id = :id"),
    @NamedQuery(name = "Domic.findByName", query = "SELECT r FROM Domic r WHERE r.nome = :nome")
})
public class Domic implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Column(name = "id", nullable = false)
    private Integer id;
    @Column(name = "nome")
    private String nome;
    @OneToMany(cascade = CascadeType.ALL, mappedBy = "domicId")
    private Collection<Predio> predioCollection;
}
```

Em relação à JPA (Java Persistence API) é INCORRETO afirmar que

- a) @NamedQuery é aplicada para definir várias consultas.
- b) @Entity define que haverá correspondência da classe com uma tabela do banco de dados.
- c) @Id define que o atributo que está mapeado com tal anotação corresponderá à chave primária da tabela.

Exercício

1) [FCC - 2011 - TRT] Considere o trecho de código abaixo:

Em relação à JPA (Java Persistence API) é INCORRETO afirmar que

d) `@Column(name = "id", nullable = false)` define que o atributo da classe mapeado com tal anotação deve estar associado à coluna cujo nome é "id", além de definir que tal campo não pode ser nulo.

e) `@OneToMany` indica que o atributo contém um conjunto de entidades que a referenciam.

Exercício

1) [FCC - 2011 - TRT] Considere o trecho de código abaixo:

Em relação à JPA (Java Persistence API) é INCORRETO afirmar que

a) **@NamedQuery** é aplicada para definir várias consultas.

Arquivo de Configuração (persistence.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="JPAApp" transaction-type="RESOURCE_LOCAL">
    <class>model.Cliente</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:oracle:thin:@localhost:1521:XE"/>
      <property name="javax.persistence.jdbc.user" value="cursojava"/>
      <property name="javax.persistence.jdbc.password" value="123456"/>
      <property name="javax.persistence.jdbc.driver" value="oracle.jdbc.OracleDriver"/>
    </properties>
  </persistence-unit>
</persistence>
```

Persistence.xml

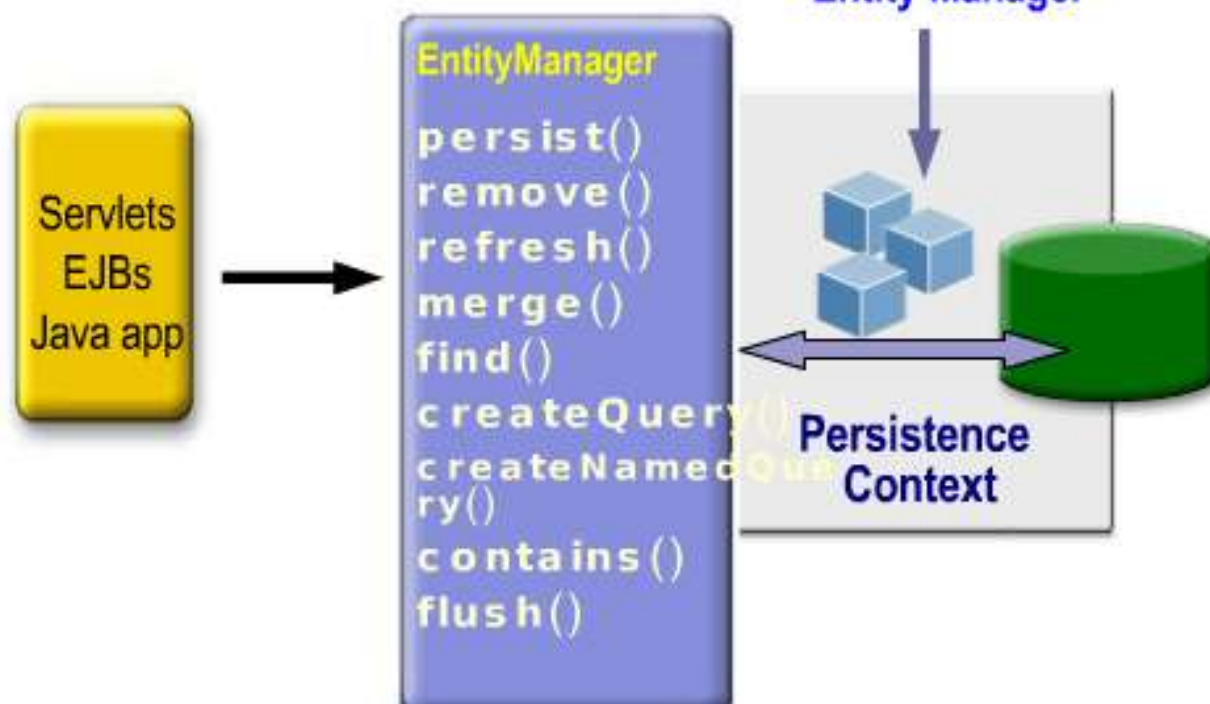
Elementos Básicos

- O elemento **<persistence-unit>** é utilizado para configurar o provedor JPA, definir o *schema* do banco de dados que será utilizado e realizar o mapeamento das classes (entidades).
- É composto por outros elementos:
 - O elemento **<class>** referencia as entidades (mapeamento das tabelas) utilizadas.
 - O elemento **<provider>** informa qual o framework de persistência utilizado.
 - O elemento **<properties>** define as configurações do banco de dados.

Classe de Gerenciamento (EntityManager.java)

- É a Interface utilizada para gerenciar as operações de persistência no banco de dados.

EntityManager API for managing entities



EntityManagerFactory.java

Características

- Tem como principal propósito fornecer instâncias de EntityManager.
- É baseado no Padrão de Projeto Factory.
- É compartilhado pelas Threads da aplicação.

- Criação:

```
EntityManagerFactory emf;  
EntityManager em = emf.createEntityManager();
```

```
// EntityManagerHelper  
public static EntityManager getEntityManager() {  
    EntityManager manager = threadLocal.get();  
    if (manager == null || !manager.isOpen()) {  
        manager = emf.createEntityManager();  
        threadLocal.set(manager);  
    }  
    return manager;  
}
```

EntityManager.java

Características

- É obtido de uma instância de EntityManagerFactory;
- Principal interface entre a Aplicação Java e o JPA. Responsável por armazenar e recuperar objetos.

- **Principais Métodos:**

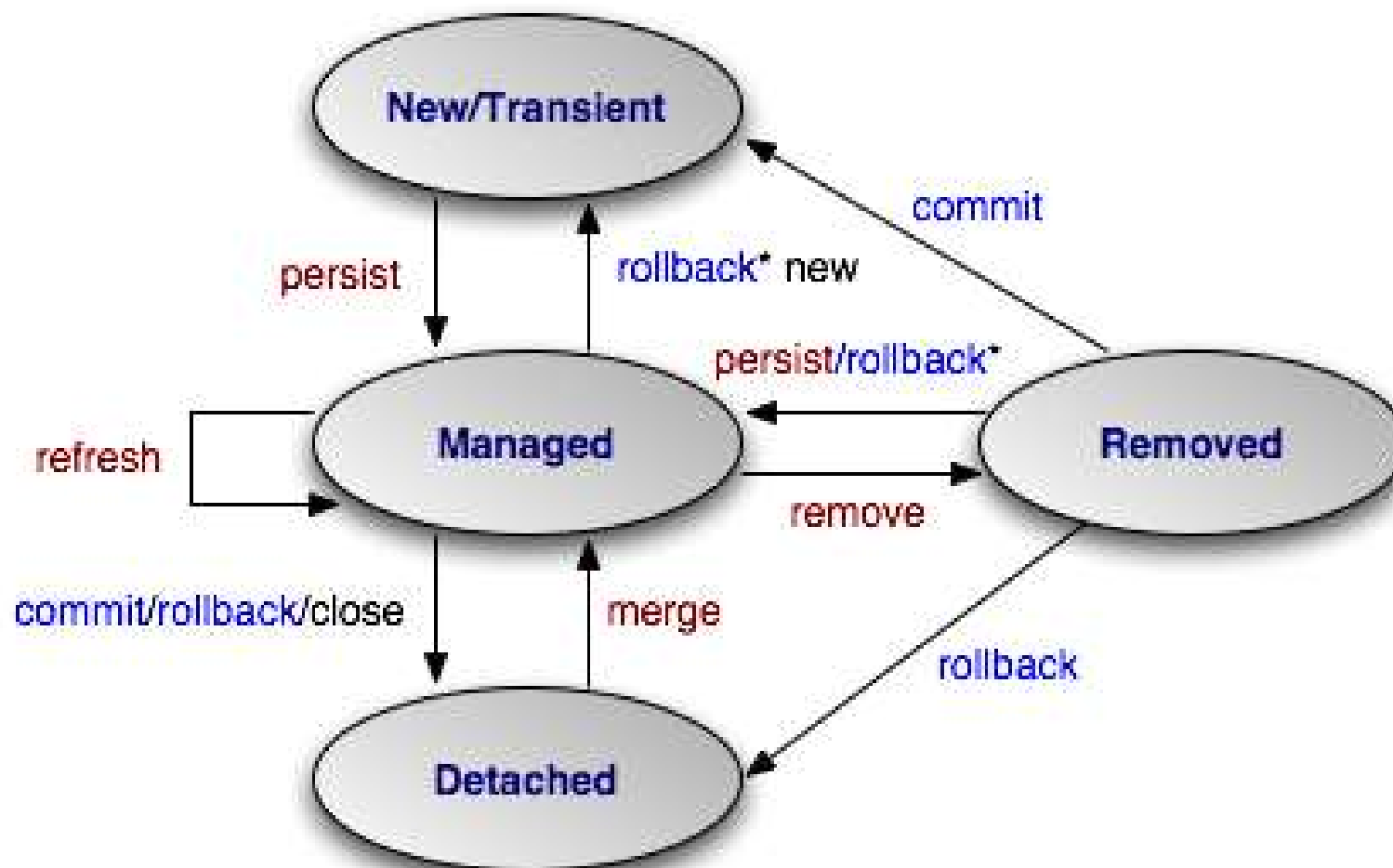
- Inclusão `getEntityManager().persist(entity);`

- Alteração `getEntityManager().merge(entity);`

- Consulta `getEntityManager().find(Funcionario.class,id);`

- Exclusão `getEntityManager().remove(entity);`

Ciclo de Vida JPA



* = Extended persistence context

[http:// openjpa.apache.org](http://openjpa.apache.org)

Exercícios

- 1) [FCC - 2011 - TRT] Os estados do ciclo de vida de uma instância de uma entidade, definidos na JPA 2.1, são
 - a) novo (new), gerenciado (managed), destacado (detached) e removido (removed).
 - b) ativo (active), inativo (inactive) e removido (removed).
 - c) novo (new), temporário (temporary), permanente (permanent) e destacado (detached).
 - d) novo (new), temporário (temporary) e destacado (detached)
 - e) gerenciado (managed), temporário (temporary), permanente (permanent) e destacado (detached).

Exercícios

2) [FCC - 2012 - TJ-PE] Quando se utiliza JPA, um EntityManager mapeia um conjunto de classes a um banco de dados particular. Este conjunto de classes, definido em um arquivo chamado persistence.xml, é denominado

- a) persistence context.
- b) persistence unit.
- c) entity manager factory.
- d) entity transaction.
- e) persistence provider.

3) [FCC - 2012 - TJ-PE] Em uma classe de entidade de uma aplicação que utiliza JPA, a anotação que define um atributo que não será salvo no banco de dados é a

- a) @Optional.
- b) @Transient.
- c) @Stateless.
- d) @Stateful.
- e) @Local.

Exercícios

- 1) [FCC - 2011 - TRT] Os estados do ciclo de vida de uma instância de uma entidade, definidos na JPA 2.1, são
 - a) **novo (new), gerenciado (managed), destacado (detached) e removido (removed).**
- 2) [FCC - 2012 - TJ-PE] Quando se utiliza JPA, um EntityManager mapeia um conjunto de classes a um banco de dados particular. Este conjunto de classes, definido em um arquivo chamado persistence.xml, é denominado
 - a) persistence context.
 - b) **persistence unit.**
 - c) entity manager factory.
 - d) entity transaction.
- 3) [FCC - 2012 - TJ-PE] Em uma classe de entidade de uma aplicação que utiliza JPA, a anotação que define um atributo que não será salvo no banco de dados é a
 - a) @Optional.
 - b) **@Transient.**

Projeto Prático com o JPA

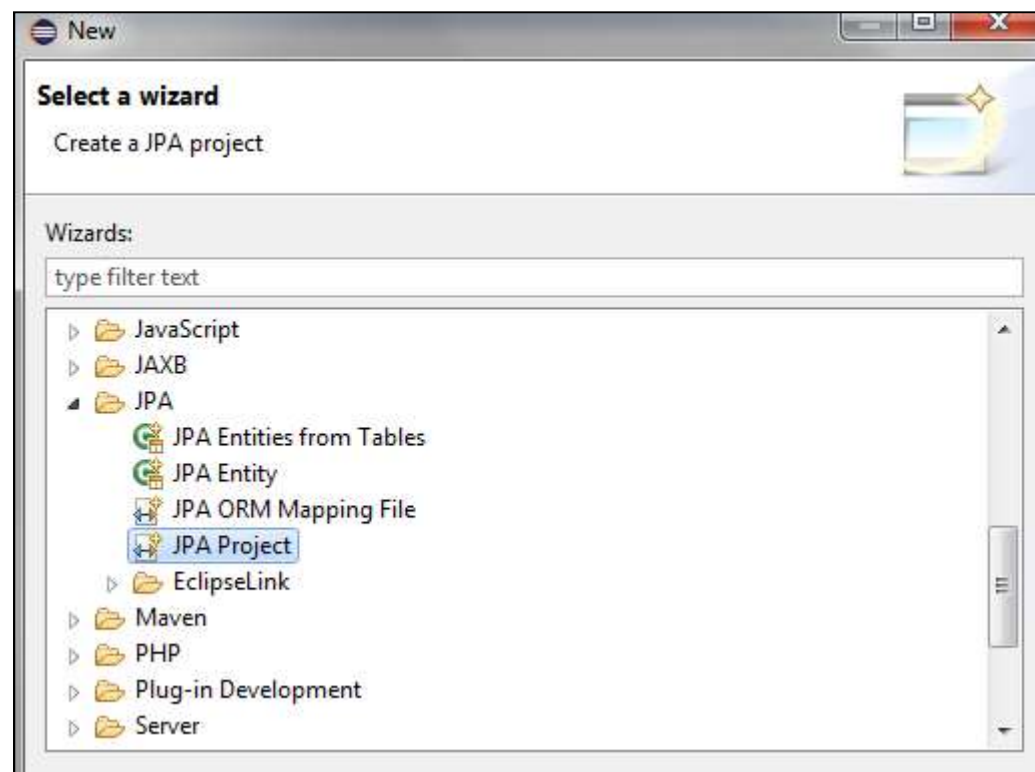
Projeto Prático com JPA

Após a realização dos 06 Passos para configurar o Hibernate (visto na Unidade 4), tornam-se necessários:

- 1. Criar um projeto JPA e configurar o seu arquivo de configuração (**persistence.xml**)
- 2. Copiar os arquivos .jar do Hibernate e do driver Oracle para este Projeto
- 3. Realizar a Engenharia Reversa das Tabelas **Cliente, Curso e Pagamento**
- 4. Fazer os Ajustes necessários nos Códigos Gerados

Projeto Prático com JPA

- 1. Criar um projeto JPA e configurar o seu arquivo de configuração (**persistence.xml**)
- Selecionar **“File” ⇒ “New” ⇒ “Other...” ⇒ “JPA” ⇒ “JPA Project”**



Projeto Prático com JPA

- 1. Criar um projeto JPA e configurar o seu arquivo de configuração (**persistence.xml**)
- Selecionar **“Next >”**

New JPA Project

JPA Project

Configure JPA project settings.

Project name: JPAAApp

Project location

☒ Use default location

Location: C:\cursoJava8\JPAAApp Browse...

Target runtime

jdk1.8.0_11 New Runtime...

JPA version

2.1

Configuration

Basic JPA Configuration Modify...

A general starting point for a JPA application.

EAR membership

☐ Add project to an EAR

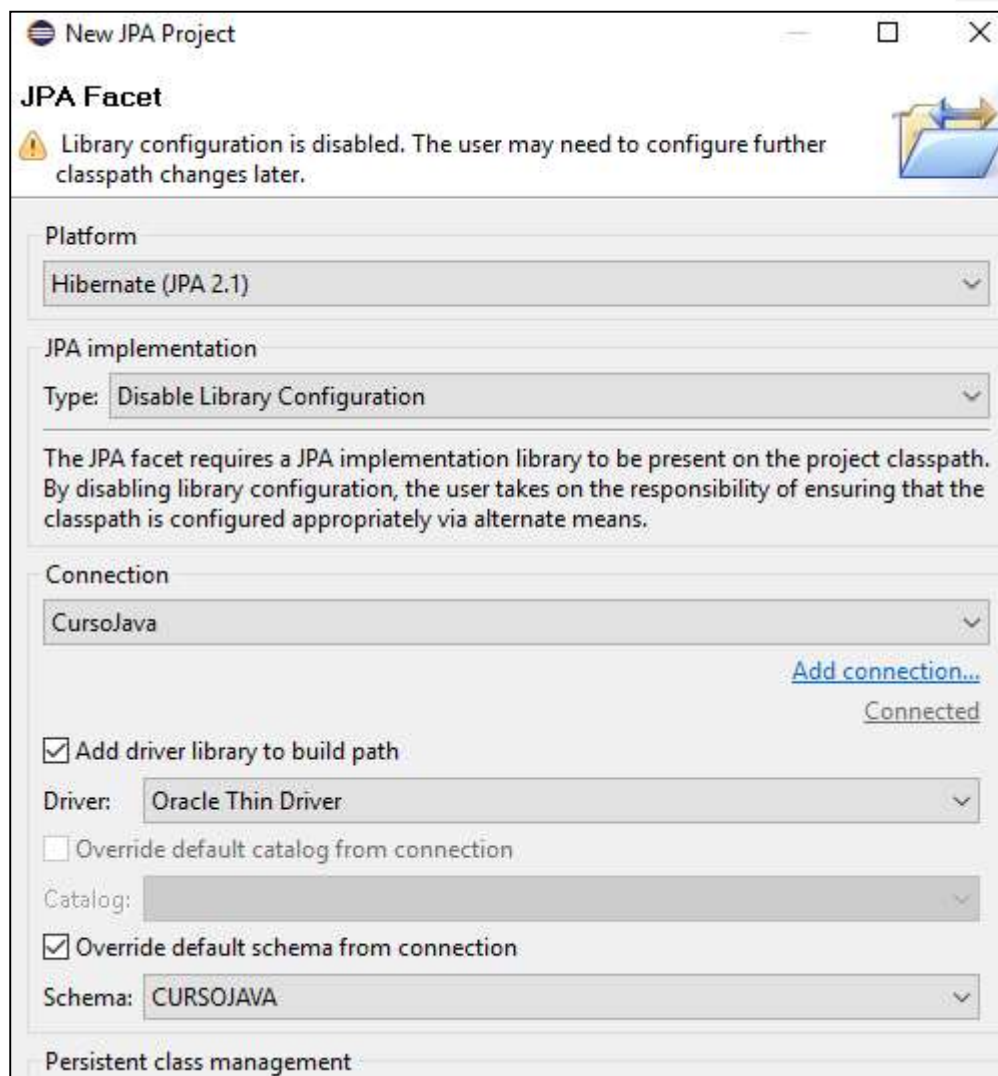
EAR project name: New Project ...

Working sets

☐ Add project to working sets

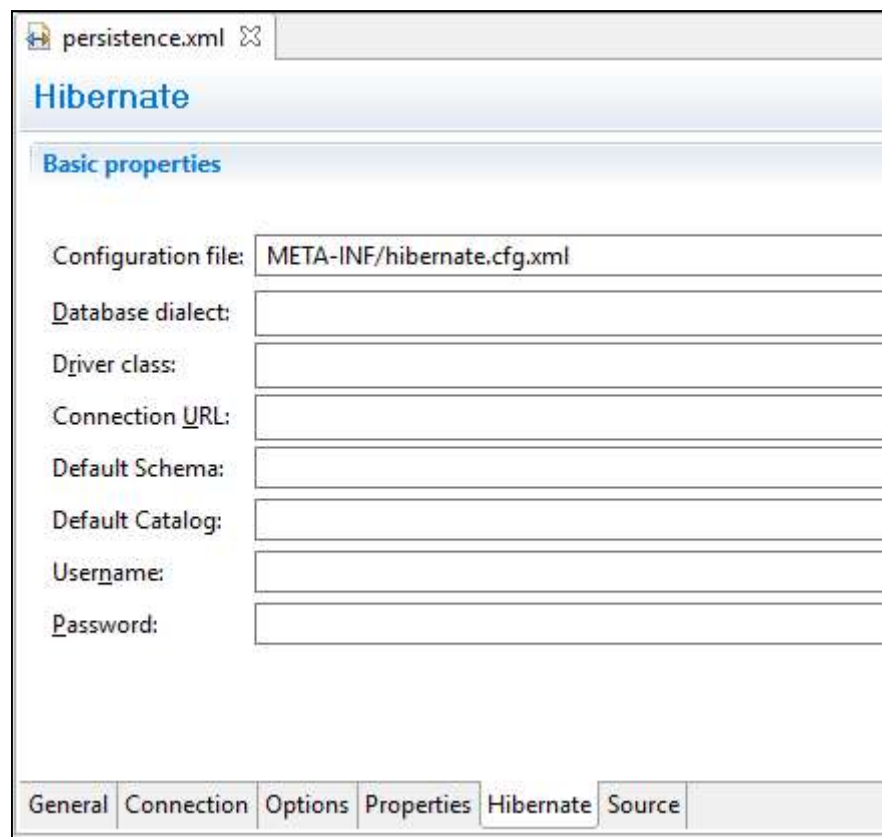
Projeto Prático com JPA

- 1. Criar um projeto JPA e configurar o seu arquivo de configuração (**persistence.xml**)
- Selecionar **“Next >”**
- Selecionar **“Finish”**



Projeto Prático com JPA

- 1. Criar um projeto JPA e configurar o seu arquivo de configuração (**persistence.xml**)
- Selecionar a aba “**Hibernate**” ⇒ “**Setup**” ⇒ “**Create new...**”



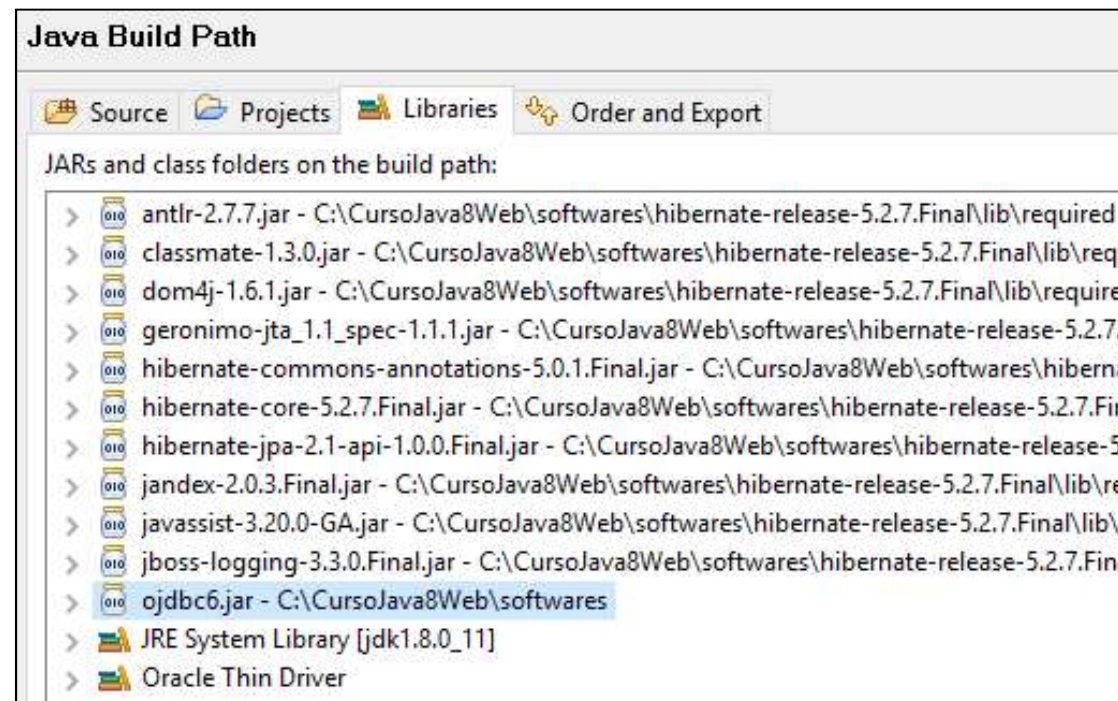
The screenshot shows the Eclipse IDE interface for configuring a new JPA project. The top tab bar shows 'persistence.xml'. The main editor area displays the 'Hibernate' configuration page. The 'Basic properties' section is expanded, showing the following fields:

- Configuration file: META-INF/hibernate.cfg.xml
- Database dialect:
- Driver class:
- Connection URL:
- Default Schema:
- Default Catalog:
- Username:
- Password:

The bottom tab bar shows the following tabs: General, Connection, Options, Properties, **Hibernate**, and Source. The 'Hibernate' tab is currently selected.

Projeto Prático com JPA

- **2. Copiar os arquivos .jar do Hibernate e do driver Oracle para este Projeto**
- Adicionar as principais bibliotecas do Hibernate e JPA nesse projeto:
 - ...\\hibernate-release-5.2.7.Final\\lib\\required
- Adicionar também o driver Oracle (**ojdbc6.jar**).



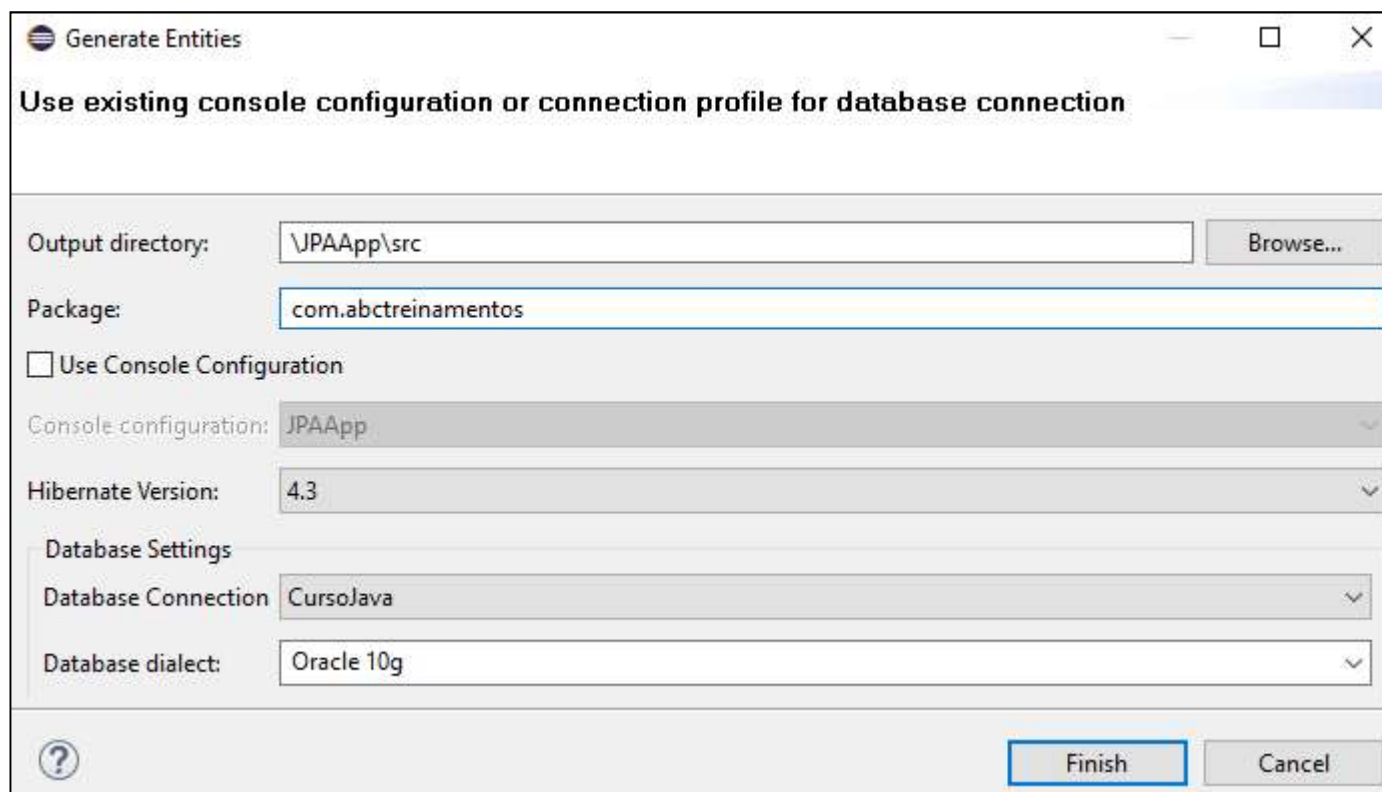
Projeto Prático com JPA

- 3. Realizar a Engenharia Reversa das Tabelas **Cliente**, **Curso** e **Pagamento**
- Clicar com o botão direito no projeto **JPAApp** e Selecionar **“JPA Tools”** ⇒ **“Generate Entities from Tables...”**



Projeto Prático com JPA

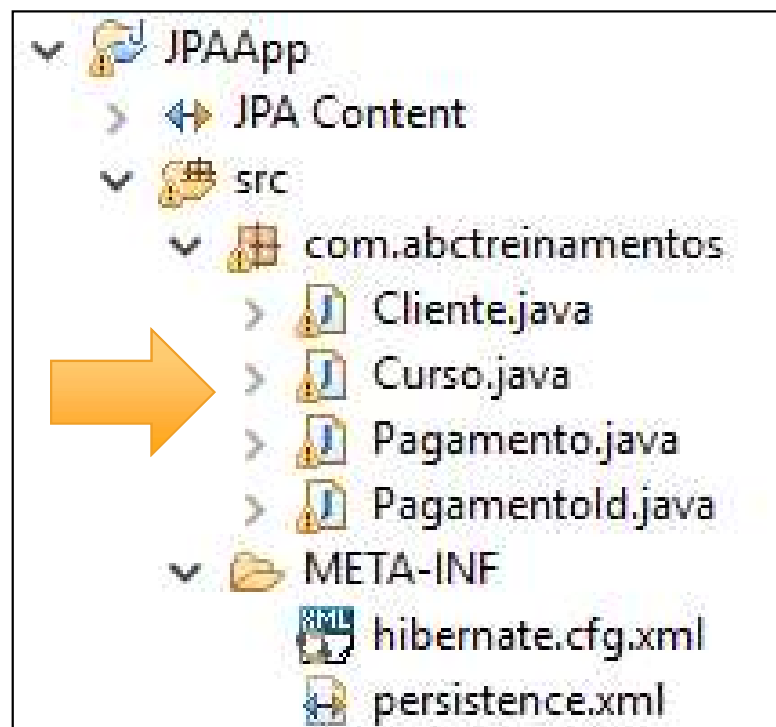
- 3. Realizar a Engenharia Reversa das Tabelas **Cliente**, **Curso** e **Pagamento**



Projeto Prático com JPA

- 3. Realizar a Engenharia Reversa das Tabelas **Cliente**, **Curso** e **Pagamento**

VOs



Projeto Prático com o Hibernate

- **4. Fazer os Ajustes necessários nos Códigos Gerados**

- A) Incluir os códigos abaixo no arquivo **persistence.xml**

```
<persistence-unit name="JPAAp">  
  <class>com.abctreinamentos.Cliente</class>  
  <class>com.abctreinamentos.Curso</class>  
  <class>com.abctreinamentos.Pagamento</class>  
  <class>com.abctreinamentos.Pagamentold</class>  
  <properties>...
```

- B) Trocar **BigDecimal** por **long** nas **Classes Geradas**

- C) Anular o método toString() nas **Classes Geradas**

```
@Override  
public String toString() {  
    return "Cliente [cpf="+cpf+" nome="+nome+" email="+email+"]";  
}
```

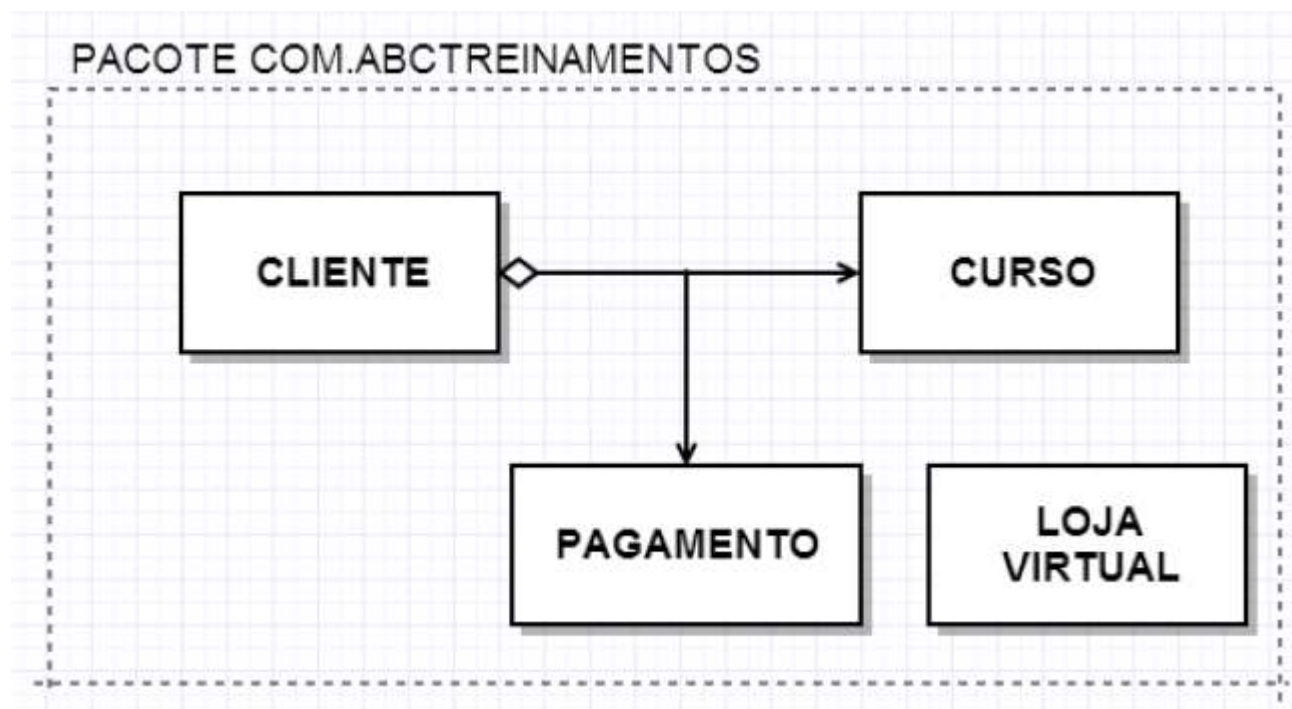
Projeto Prático com o Hibernate

- **4. Fazer os Ajustes necessários nos Códigos Gerados**
- D) Inserir o código abaixo para realizar as operações de **CRUD**

```
public static void main(String[] args) {  
    EntityManagerFactory emf =  
        Persistence.createEntityManagerFactory("JPAAApp");  
    EntityManager em = emf.createEntityManager();  
    EntityTransaction tx = em.getTransaction();  
    tx.begin();  
    //OPERAÇÕES DE CRUD  
    tx.commit();  
    em.close();  
    emf.close();  
}
```

Exercícios

- 1) Criar as classes **ClienteApp**, **CursoApp** e **PagamentoApp** para realizar as operações de **CRUD** nas suas respectivas Tabelas.
- 2) **ATIVIDADE EXTRA:** Copiar o arquivo **LojaVirtual** escrito na **Unidade 3** e adaptá-lo às operações de CRUD realizadas no exercício (1).



Tipos de Consultas no JPA

Tipos de Consultas no JPA

São três os tipos de consultas no JPA:

- Java Persistence Query Language (JPQL);
 - EntityManager;
 - SQL “puro”.
-
- A maioria das consultas são resolvidas com JPQL e o EntityManager. As mutio específicas são resolvidas com o SQL “puro”.

JPA Query Language

- O JPQL é uma linguagem parecida com o SQL, porém, “orientada a objetos”.
- Possibilita descrever consultas polimórficas e consultas sobre coleções.

```
Select p from Pessoa p where upper(p.nome) like 'MARIA%'
```

- A consulta acima retorna todos os objetos da classe pessoa e de suas subclasses que tenham o nome começado por 'MARIA'.

```
String queryString = "select pessoa from Pessoa pessoa  
                        where upper(pessoa.nome) like :NOME ";  
Query query = getEntityManager().createQuery(queryString);  
List usuarios = query.getResultList();
```

- A consulta acima retorna todos os objetos da classe pessoa e de suas subclasses que tenham o nome começado pelo valor definido na variável NOME.

Considerações Finais

- Existem muitas ferramentas de apoio ao desenvolvimento usando JPA.
- O uso adequado dessas ferramentas deixa apenas o trabalho estritamente necessário para o desenvolvedor.
- O maior esforço para usá-lo está na construção e manutenção dos mapeamentos.

Principais Vantagens do JPA

- Aumento da Produtividade, pois o mapeamento do banco de dados é feito automaticamente.
- Mantém poucas linhas de código e é de fácil entendimento.
- É uma API bastante estável e especificada por uma JSR.
- API rica em funcionalidades.

Exercícios

- 1) [FCC - 2012 - TJ-PE] Quando se utiliza JPA, um EntityManager mapeia um conjunto de classes a um banco de dados particular. Este conjunto de classes, definido em um arquivo chamado persistence.xml, é denominado
 - (a) persistence context. (b) persistence unit.
 - (c) entity manager factory. (d) entity transaction.
 - (e) persistence provider.
- 2) [FCC - 2012 - TJ-PE] Em uma classe de entidade de uma aplicação que utiliza JPA, a anotação que define um atributo que não será salvo no banco de dados é a
 - (a) @Optional. (b) @Transient. (c) @Stateless.
 - (d) @Stateful. (e) @Local.
- 3) Incluir o método **consultarTodos** nas classes **ClienteApp**, **CursoApp** e **PagamentoApp**.

Exercícios

- 1) [FCC - 2012 - TJ-PE] Quando se utiliza JPA, um EntityManager mapeia um conjunto de classes a um banco de dados particular. Este conjunto de classes, definido em um arquivo chamado persistence.xml, é denominado
(a) persistence context. **(b) persistence unit.**
(c) entity manager factory. (d) entity transaction.
(e) persistence provider.
- 2) [FCC - 2012 - TJ-PE] Em uma classe de entidade de uma aplicação que utiliza JPA, a anotação que define um atributo que não será salvo no banco de dados é a
(a) @Optional. **(b) @Transient.** (c) @Stateless.
(d) @Stateful. (e) @Local.
- 3) Incluir o método **consultarTodos** nas classes **ClienteApp**, **CursoApp** e **PagamentoApp**.

RESUMO

TÓPICOS APRESENTADOS

- Neste conjunto de videoaulas nós vimos:
 - **Java Persistence API (JPA)**
 - **Principais Componentes do JPA**
 - **Projeto Prático com o JPA**
 - **Tipos de Consultas no JPA**

ATIVIDADES PARA SE APROFUNDAR

- 1) Implementar as operações de **Novo, Consultar, Alterar e Excluir** na aplicação **LojaVirtual**.

