JPA Passo a Passo

Henrique Eduardo M. Oliveira
henrique@voffice.com.br

Palestrante

Henrique Eduardo M. Oliveira (henrique@voffice.com.br)

- **Trabalha:** Arquiteto JEE / Instrutor Java
- **Formação:** Ciências da Computação pela UFSC
- **Experiência:** +7 anos com desenvolvimento de sistemas em Java, especialmente para WEB
- **Certificações:** SCJA, SCJP 5, SCWCD 1.4, SCEA 5
- **Hobby:** Praia e Cinema

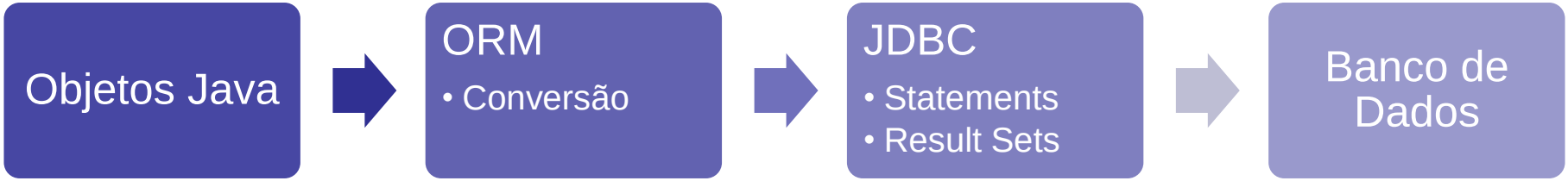
Agenda

- O que é JPA?
- Mapeamento Objeto-Relacional
- Principais funcionalidades da JPA
- Passo a passo para utilização em projetos
  - Mapeamentos de entidades, chaves primárias, colunas, herança, relacionamentos, locking, validação, operações em cascata, consultas
- JPA 2.0
- Conclusões

O que é JPA?

- Java Persistence API
- Especificação padrão para mapeamento objeto-relacional e gerenciamento de persistência da plataforma Java EE 5.0
- Versão 1.0, faz parte da especificação JSR-220 (EJB 3.0)
- Possui amplo suporte pela maioria dos grandes players do mercado: Apache, Oracle, BEA, JBoss

ORM: Mapeamento Objeto-Relacional

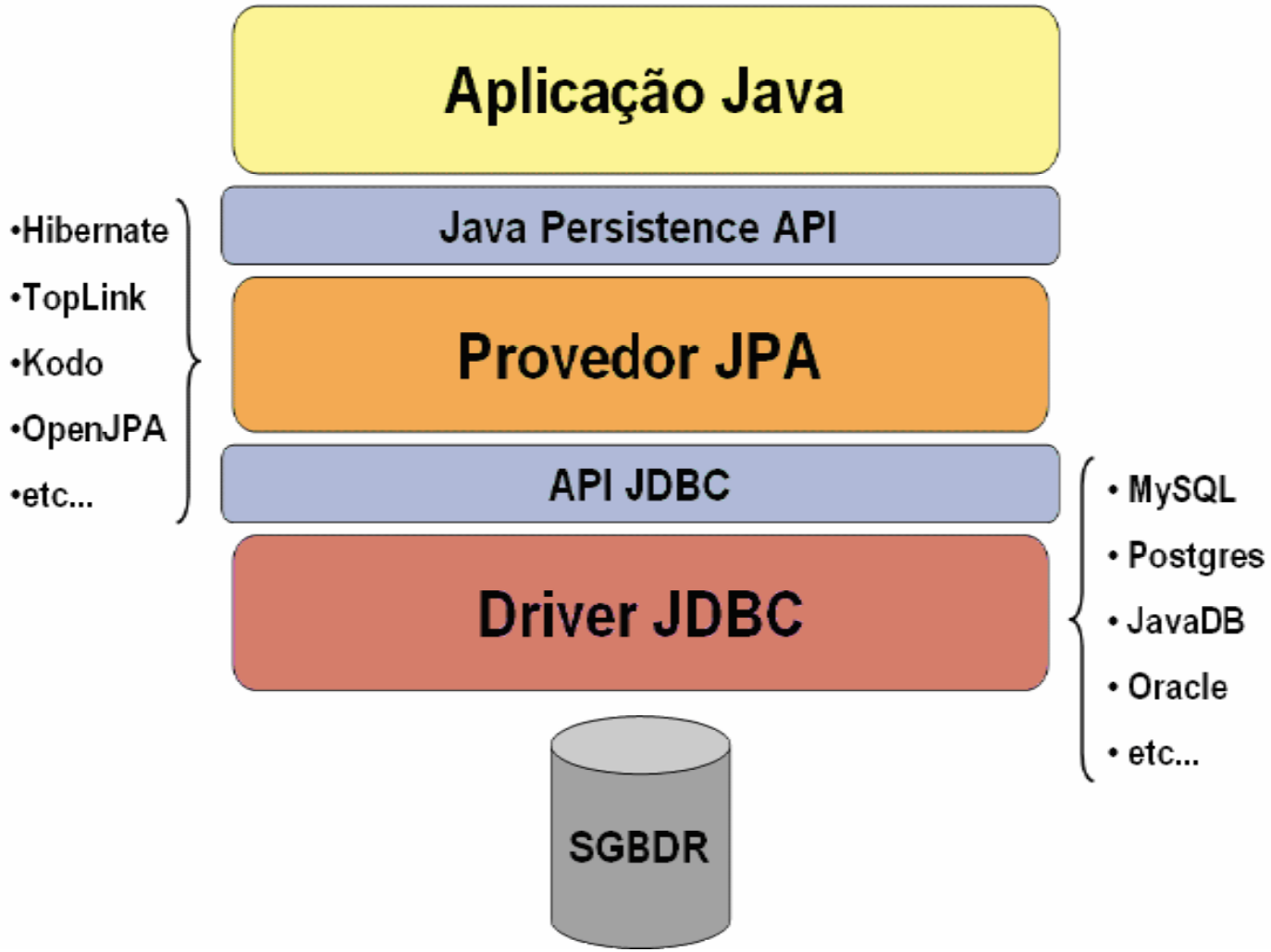


- Modelo OO vs Modelo Relacional
  - Classe = Tabela
  - Objeto = Linha
  - Atributo = Coluna
  - Associação = Chave Estrangeira
- Mapeamento via XML ou Annotations

Funcionalidades da JPA

- Padroniza Mapeamento Objeto-Relacional
- Utiliza POJO's ao invés de Entity Beans
- Pode ser usado com Java SE e Java EE
- Suporta utilização de diferentes *Providers*
- Possui uma linguagem de consulta extendida
- Suporta herança, polimorfismo

Você pode utilizar seu framework preferido!



Passo a passo para utilização

- Download do JPA Provider
- Preparar banco de dados e driver JDBC
- Mapeamento Objeto-Relacional
- Configurar arquivo persistence.xml
- Implementar acesso a dados via EntityManager

Passo a passo para utilização

- Download do JPA Provider
- Preparar banco de dados e driver JDBC
- Mapeamento Objeto-Relacional
- Configurar arquivo persistence.xml
- Implementar acesso a dados via EntityManager

JPA Providers

- Hibernate
  - http://jpa.hibernate.org
- Toplink Essentials
  - http://oss.oracle.com/toplink-essentials-jpa.html
- Open JPA
  - http://openjpa.apache.org
- Netbeans 6.5 já vem com Hibernate, Toplink e EclipseLink!!

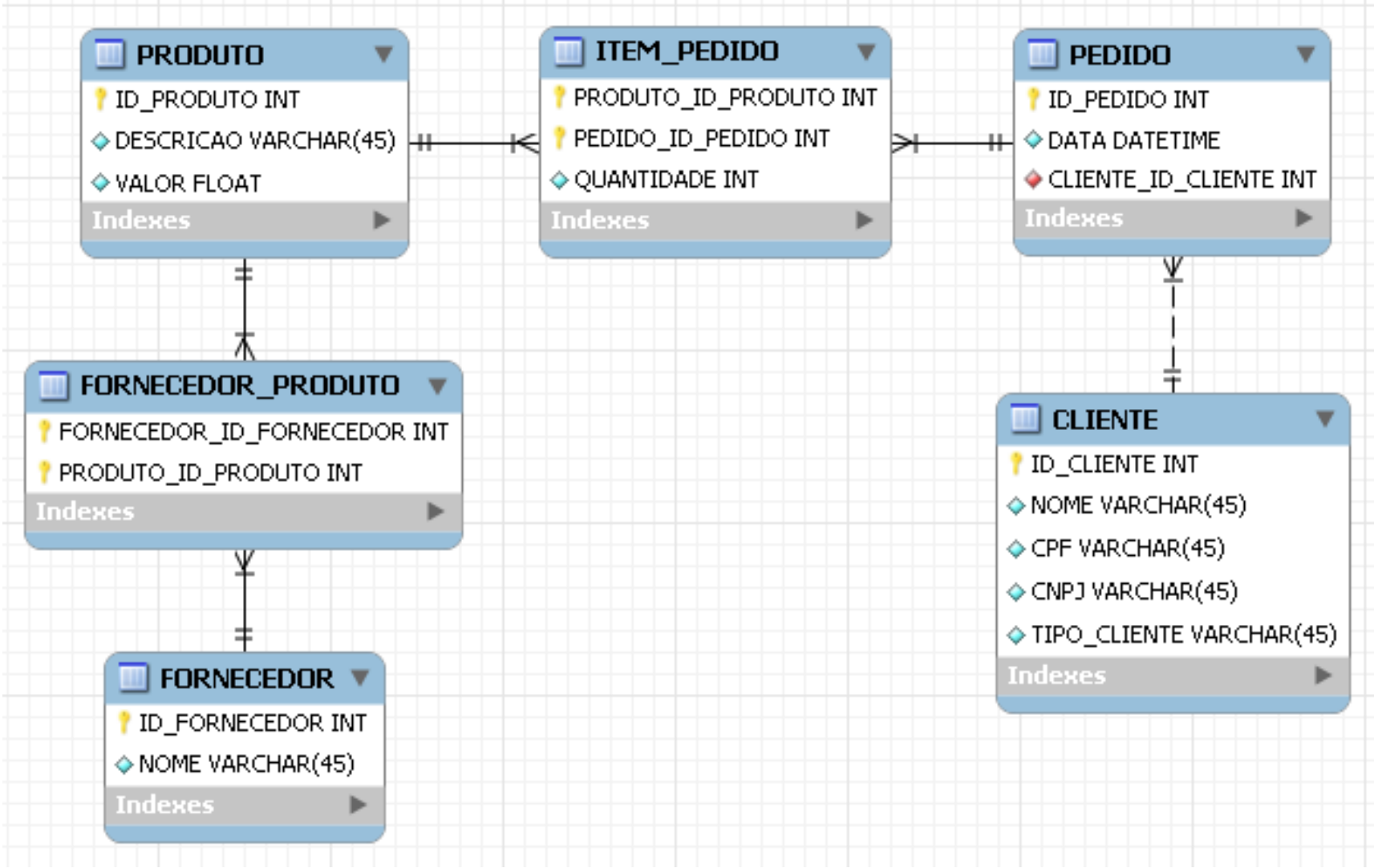
Passo a passo para utilização

- Download do JPA Provider
- Preparar banco de dados e driver JDBC
- Mapeamento Objeto-Relacional
- Configurar arquivo persistence.xml
- Implementar acesso a dados via EntityManager

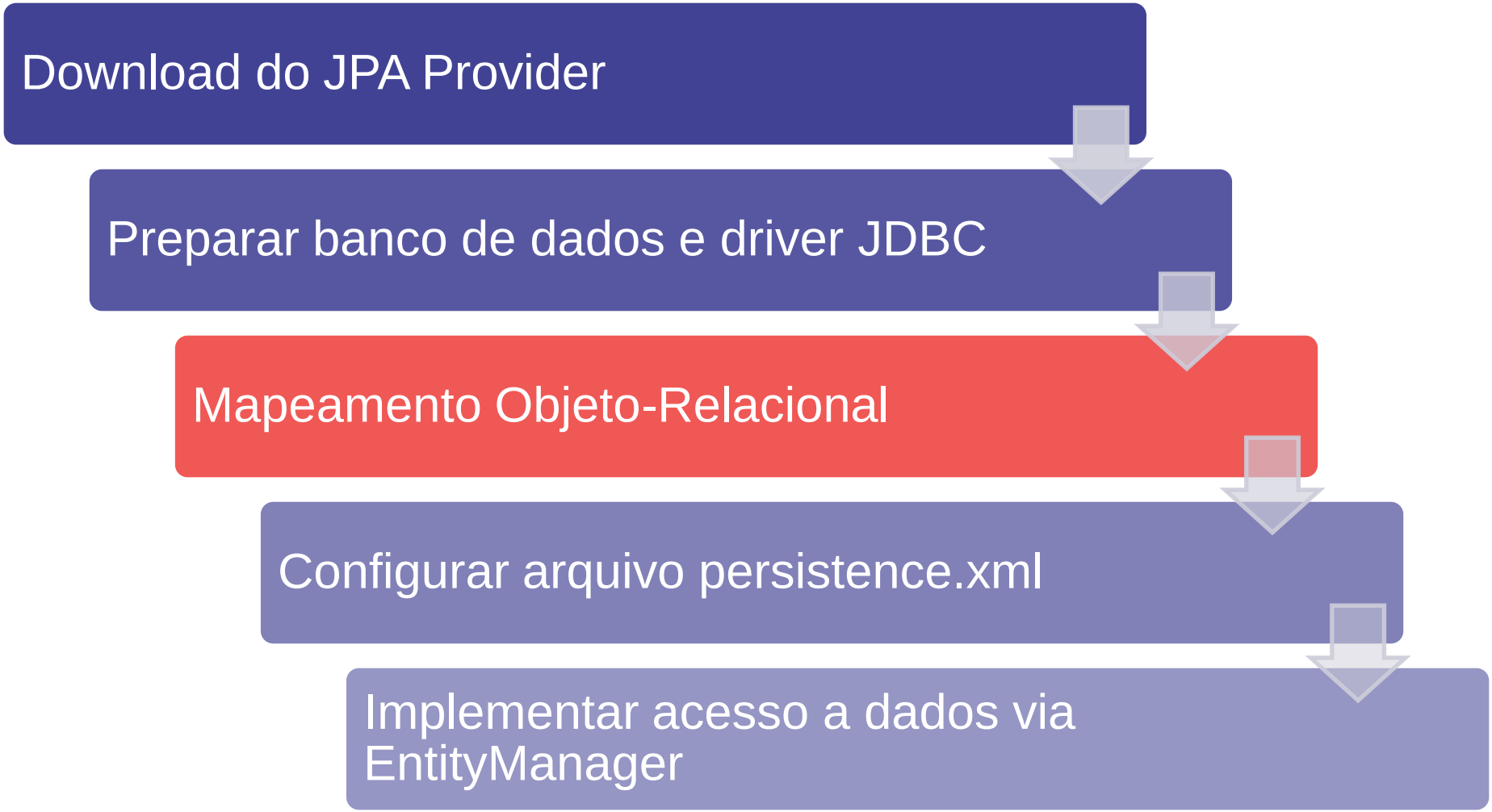
Banco de Dados e Driver JDBC

- MySQL Community Server
  - http://dev.mysql.com/downloads/mysql/5.0.html
- MySQL Connector/J 5.1
  - http://dev.mysql.com/downloads/connector/j/5.1.html
- Ou banco de dados de sua preferência!

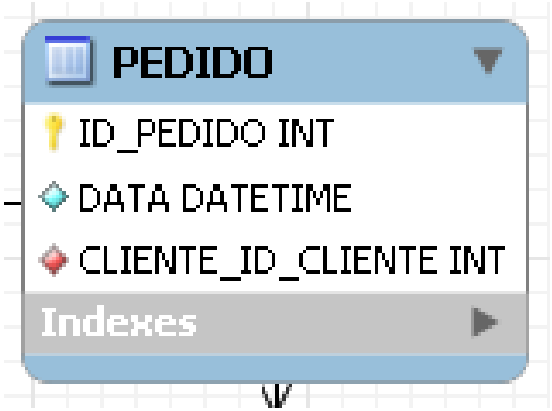
Modelo Relacional



Passo a passo para utilização



Mapeamento Objeto-Relacional




```
@Entity
@Table(name="PEDIDO")
public class Pedido implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_PEDIDO", nullable = false)
    private Integer idPedido;

    @Column(name = "DATA", nullable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private Date data;

    @JoinColumn(name = "CLIENTE_ID_CLIENTE", referencedColumnName = "ID_CLIENTE")
    @ManyToOne(optional = false, fetch = FetchType.EAGER)
    private Cliente cliente;
```

Entidades

- @Entity
  - Especifica que uma classe é uma entidade
  - Uma entidade é um objeto que pode ser persistido
  - Representa uma tabela no banco de dados relacional

- @Table
  - Especifica nome da tabela no banco de dados

```
@Entity
@Table(name = "pedido", schema="jpa")
public class Pedido implements Serializable {
```

Atributos

- @Column
  - Mapeia um atributo ou uma propriedade (getter) a um campo do banco de dados
  - Possui diversas opções de validação
    - Lança javax.persistence.PersistenceException

```
@Column(name = "NOME", nullable = false, length = 45,
        insertable = true, updatable = true, unique = false)
private String nome;
```

Chave Primária Simples

- @Id
  - Cada entidade precisa possuir uma chave primária
  - Mapeia uma chave primária simples
  - Chave pode ser gerada automaticamente:
    - IDENTITY, AUTO, SEQUENCE, TABLE

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "ID_PEDIDO", nullable = false)
private Integer idPedido;
```

Chave Primária Composta

- @Embeddable
  - Define que uma classe pode fazer parte de uma entidade

```
@Embeddable
public class ItemPedidoPK implements Serializable {

    @Column(name = "PRODUTO_ID_PRODUTO", nullable = false)
    private int idProduto;

    @Column(name = "PEDIDO_ID_PEDIDO", nullable = false)
    private int idPedido;
```

Chave Primária Composta

- @EmbeddedId
  - Define uma propriedade que é embeddable como chave primária

```
@Entity
@Table(name = "item_pedido")
public class ItemPedido implements Serializable {

    @EmbeddedId
    protected ItemPedidoPK itemPedidoPK;
```

Herança

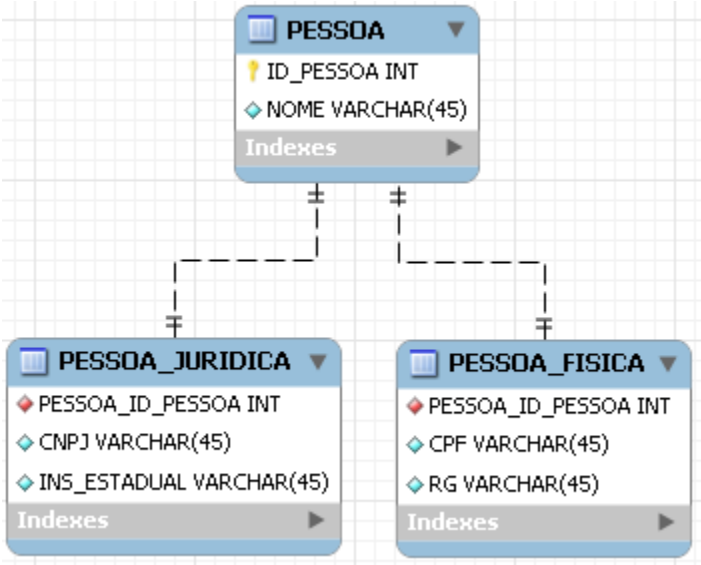
- Single Table
  - Apenas 1 tabela para toda a hierarquia
  - Modelo de herança padrão
- Joined Subclass
  - 1 tabela com campos para entidade pai na hierarquia e uma tabela para cada entidade filha contendo somente os campos específicos da subclasse
- Table per Class
  - 1 tabela separada com todos os campos para cada uma das entidades filhas
  - Suporte é opcional

Herança com Single Table


```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TIPO_CLIENTE")
public abstract class Cliente implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_CLIENTE", nullable = false)
    private Integer idCliente;

    @Column(name = "NOME", nullable = false, length = 45)
    private String nome;


@Entity
@DiscriminatorValue("pessoaJuridica")
public class ClientePessoaJuridica extends Cliente {

    @Column(name = "CNPJ", length = 19)
    private String cnpj;
```

Herança com Joined Subclass


```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Pessoa implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID_PESSOA", nullable = false)
    private Integer idPessoa;
```




```
@Entity
@Table(name = "PESSOA_FISICA")
@PrimaryKeyJoinColumn(name = "PESSOA_ID_PESSOA",
                      referencedColumnName = "ID_PESSOA")
public class PessoaFisica extends Pessoa {

    @Column(name = "CPF", length = 14)
    private String cpf;
```

Relacionamentos

- @ManyToOne
  - Entidade Pedido

```
@JoinColumn(name = "CLIENTE_ID_CLIENTE", referencedColumnName = "ID_CLIENTE")
@ManyToOne(optional = false, fetch = FetchType.EAGER)
private Cliente cliente;
```

- @OneToMany
  - Entidade Cliente

```
@OneToMany(mappedBy = "cliente")
private Collection<Pedido> pedidoCollection;
```

- FetchType.EAGER
- FetchType.LAZY

Relacionamentos

- @ManyToMany
  - Entidade Produto

```
@JoinTable(name = "fornecedor_produto",
           joinColumns = {@JoinColumn(
                   name = "PRODUTO_ID_PRODUTO",
                   referencedColumnName = "ID_PRODUTO")},
           inverseJoinColumns = {@JoinColumn(
                   name = "FORNECEDOR_ID_FORNECEDOR",
                   referencedColumnName = "ID_FORNECEDOR")})
@ManyToMany
private Collection<Fornecedor> fornecedorCollection;
```

  - Entidade Fornecedor

```
@ManyToMany(mappedBy = "fornecedorCollection")
private Collection<Produto> produtoCollection;
```

Operações em cascata

- CascadeType:
  - **PERSIST**: Quando uma nova entidade é persistida, todas as entidades na coleção são persistidas
  - **MERGE**: Quando uma entidade desconectada é atualizada, todas as entidades na coleção são atualizadas
  - **REMOVE**: Quando uma entidade existente é removida, todas as entidades na coleção são removidas
  - **ALL**: Se aplicam todas as regras acima

Lock Otimista

- @Version
  - Define uma coluna para armazenar informação de versão para controle de lock otimista
  - Lança javax.persistence.OptimisticLockException

```
@Version
@Column(name = "VERSAO")
private int versao;
```

Herança de Mapeamento

- @MappedSuperclass
  - Designa uma classe cujos mapementos serão herdados pelas subclasses
  - A classe anotada não possui tabela no banco de dados

```
@MappedSuperclass
public class BaseEntity implements Serializable {

    @Version
    @Column(name = "VERSAO")
    private int versao;


@Entity
public class Pedido extends BaseEntity {
```

Métodos de Callback

- PostLoad, PostPersist, PostRemove, PostUpdate
- PrePersist, PreRemove, PreUpdate
  - Podem ser utilizados pra adicionar funcionalidades extras: Validação por exemplo

```
@PrePersist
@PreUpdate
public void validate() {
    if (cpf == null || !cpf.matches("\\d{3}\\.\\d{3}\\.\\d{3}-\\d{2}")) {
        throw new ValidationException("CPF Inválido!");
    }
}
```

Consultas

- Java Persistence Query Language (JP-QL)
  - Define linguagem para consulta de entidades
  - Consultas baseadas nas entidades e suas propriedades, independente da modelagem física do banco de dados
  - Utiliza sintaxe próxima a SQL
  - Consultas estáticas (named queries)
  - Consultas dinâmicas

Consultas

- Consulta estática
  - Anotada na classe ou em arquivo XML separado

```
@Entity
@NamedQueries(
     {@NamedQuery(name = "Pedido.findPedidoByIdCliente",
      query = "SELECT p FROM Pedido p WHERE cliente.idCliente = :idCliente")})
public class Pedido extends BaseEntity {
```

- Consulta dinâmica

```
String jpql = "SELECT c FROM Cliente c WHERE 1 = 1";
for (String paramName : filter.keySet()) {
    Object paramValue = filter.get(paramName);
    jpql += " AND " + paramName + " = '" + paramValue + "'";
}
Query query = em.createQuery(jpql);
return query.getResultList();
```

Consultas

> Criação de objetos

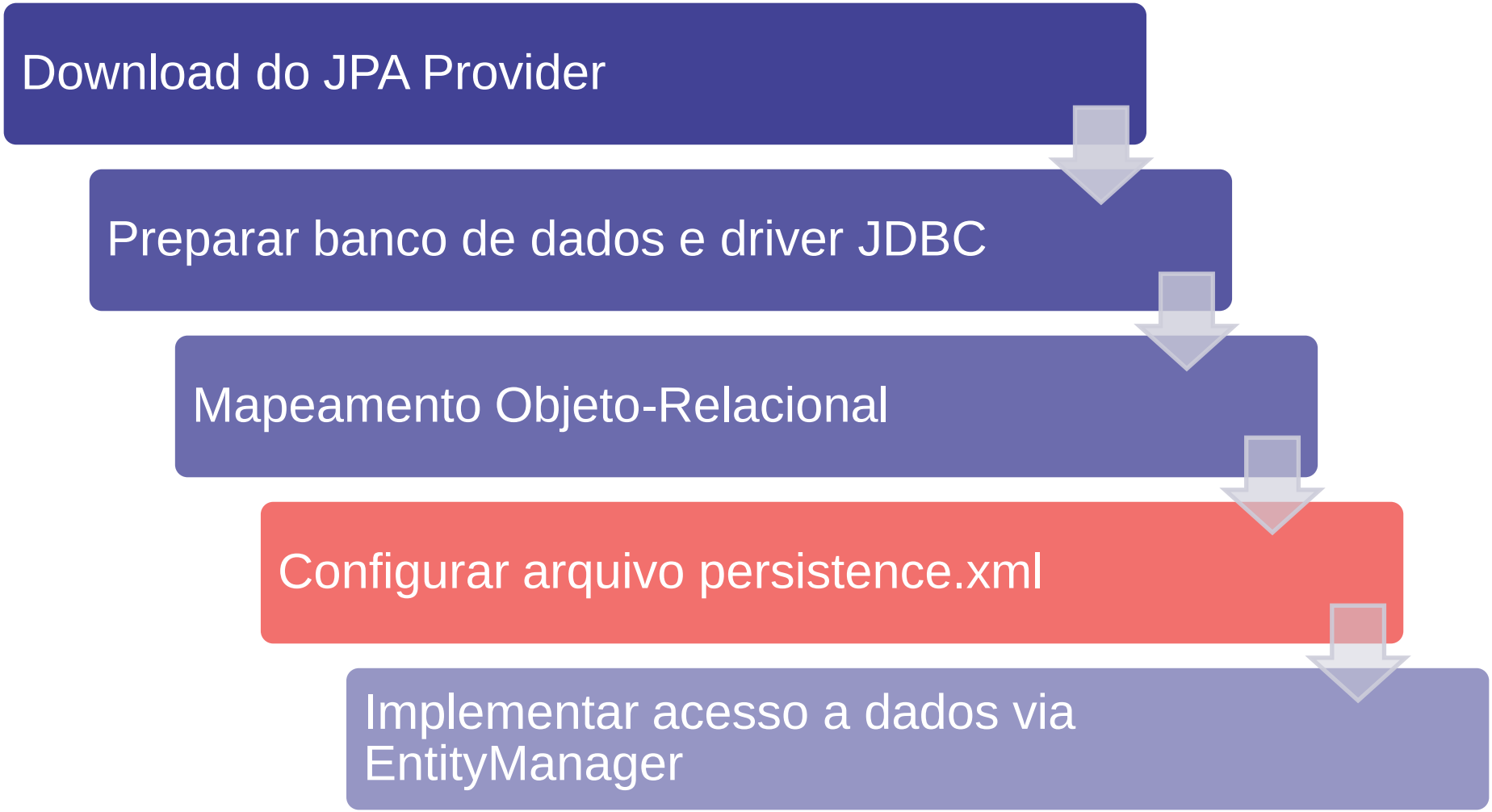
```
@NamedQuery(name = "Cliente.relatorioPedidos",
 query = "SELECT NEW " +
         "br.com.tdc.floripa.jpa.dto.RelatorioPedidosDTO(c.nome, COUNT(p.idPedido)) " +
         "FROM Cliente c LEFT JOIN c.pedidoCollection p GROUP BY c.nome"))
```

```
package br.com.tdc.floripa.jpa.dto;

public class RelatorioPedidosDTO {

    private String nomeCliente;

    private Long qtdePedidos;

    public RelatorioPedidosDTO(String nomeCliente, Long qtdePedidos) {
        this.nomeCliente = nomeCliente;
        this.qtdePedidos = qtdePedidos;
    }
```

Consultas: e muito mais...

- DISTINCT
- IN
- LIKE
- IS NULL
- IS EMPTY
- BETWEEN
- ORDER BY
- GROUP BY
- HAVING
- ...

Passo a passo para utilização



Configurar Persistence.xml

> Hibernate

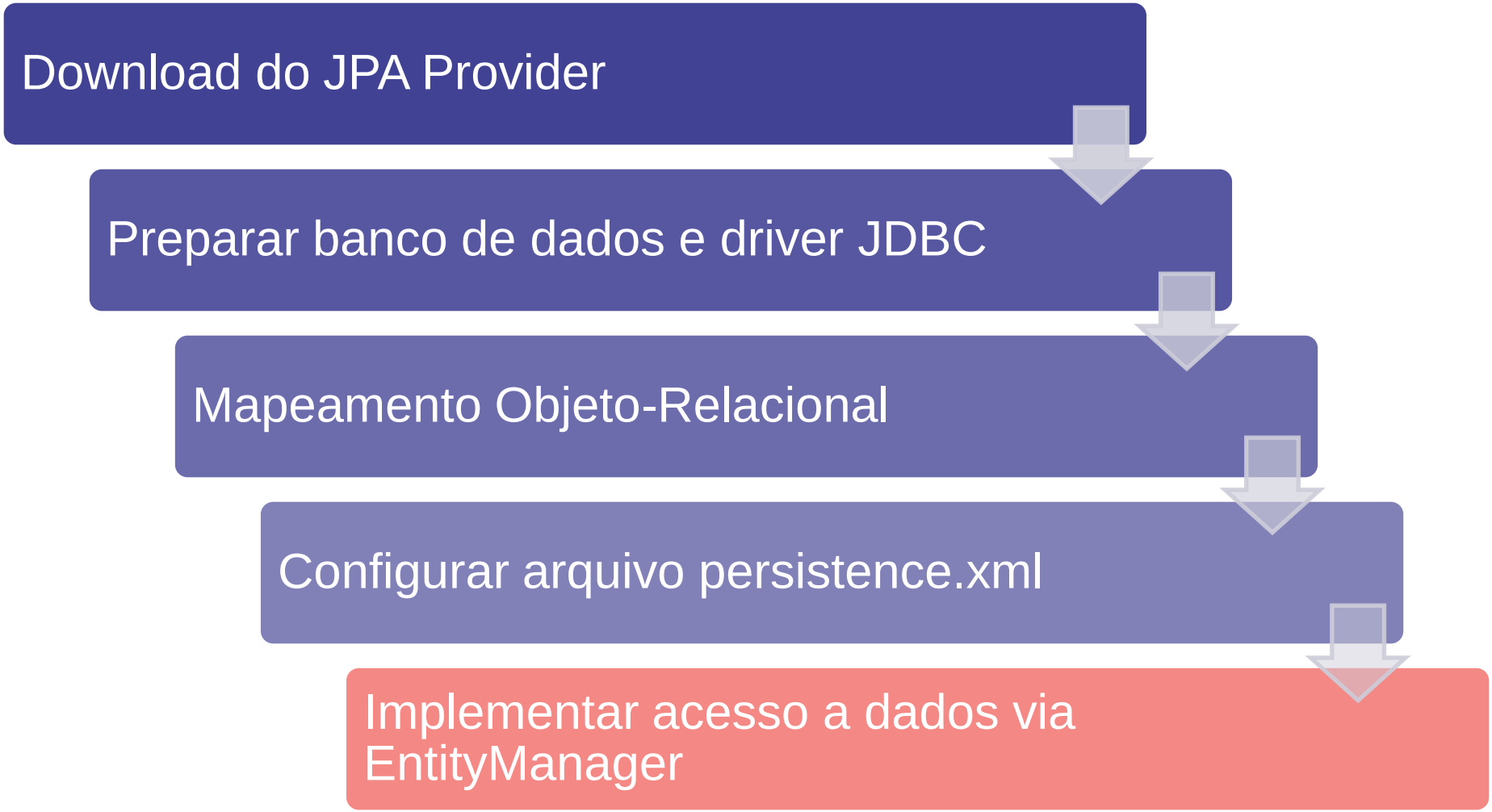
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www
    <persistence-unit name="tdc-floripa-jpa-projectPU" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.ejb.HibernatePersistence</provider>
        <class>br.com.tdc.floripa.jpa.entity.Fornecedor</class>
        <class>br.com.tdc.floripa.jpa.entity.Pedido</class>
        <class>br.com.tdc.floripa.jpa.entity.Cliente</class>
        <class>br.com.tdc.floripa.jpa.entity.ClientePessoaFisica</class>
        <class>br.com.tdc.floripa.jpa.entity.ClientePessoaJuridica</class>
        <class>br.com.tdc.floripa.jpa.entity.Produto</class>
        <class>br.com.tdc.floripa.jpa.entity.ItemPedido</class>
        <properties>
            <property name="hibernate.connection.driver_class" value="com.mysql.jdbc.Driver"/>
            <property name="hibernate.connection.url" value="jdbc:mysql://localhost:3306/jpa"/>
            <property name="hibernate.connection.username" value="root"/>
            <property name="hibernate.connection.password" value=""/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

Configurar Persistence.xml

> Toplink

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="htt
    <persistence-unit name="tdc-floripa-jpa-projectPU" transaction-type="RESOURCE_LOCAL'
        <provider>oracle.toplink.essentials.PersistenceProvider</provider>
        <class>br.com.tdc.floripa.jpa.entity.Fornecedor</class>
        <class>br.com.tdc.floripa.jpa.entity.Pedido</class>
        <class>br.com.tdc.floripa.jpa.entity.Cliente</class>
        <class>br.com.tdc.floripa.jpa.entity.ClientePessoaFisica</class>
        <class>br.com.tdc.floripa.jpa.entity.ClientePessoaJuridica</class>
        <class>br.com.tdc.floripa.jpa.entity.Produto</class>
        <class>br.com.tdc.floripa.jpa.entity.ItemPedido</class>
        <properties>
            <property name="toplink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
            <property name="toplink.jdbc.url" value="jdbc:mysql://localhost:3306/jpa"/>
            <property name="toplink.jdbc.user" value="root"/>
            <property name="toplink.jdbc.password" value=""/>
            <property name="toplink.logging.level" value="FINE"/>
        </properties>
    </persistence-unit>
</persistence>
```

Passo a passo para utilização

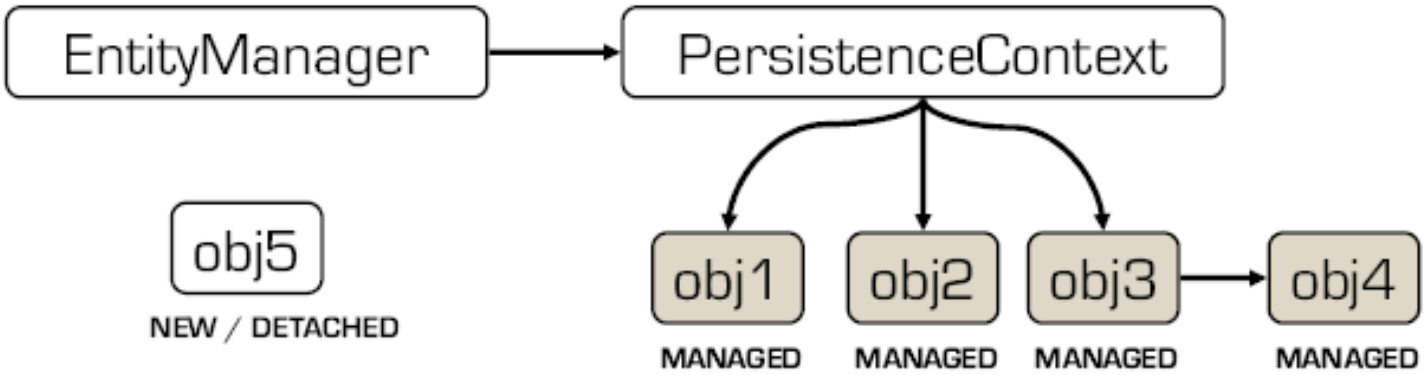


Acesso a Dados

- javax.persistence.EntityManager
  - Gerencia o ciclo de vida das entidades
    - NEW, MANAGED, DETACHED, REMOVED
  - Utilizado para criar e remover entidades, buscar entidades pela chave primária e fazer consultas
  - O conjunto de entidades que podem ser gerenciados por um EntityManager é definido dentro da Persistence Unit

Acesso a Dados

- javax.persistence.PersistenceContext
  - Conjunto de entidades associadas a um EntityManager



Acesso a Dados via JavaSE


```
public class BaseDAO {

    private EntityManagerFactory emf = null;

    public BaseDAO() {
        emf = Persistence.createEntityManagerFactory("tdc-floripa-jpa-projectPU");
    }

    protected EntityManager getEntityManager() {
        return emf.createEntityManager();
    }

    public Object create(Object object) {
        EntityManager em = getEntityManager();
        try {
            em.getTransaction().begin();
            em.persist(object);
            em.getTransaction().commit();
            return object;
        } finally {
            em.close();
        }
    }
```

Acesso a Dados via JavaEE

- EntityManager é injetado pelo container JEE
  - Depois injete o DAO nos seus EJB's de negócio

```
@Stateless
public class BaseDAO implements DAO {

    @PersistenceContext(unitName="tdc-floripa-jpa-projectPU")
    private EntityManager entityManager;

    public Object create(Object object) {
        entityManager.persist(object);
        return object;
    }
```

JPA 2.0

- Java Persistence API 2.0
  - Especificação iniciada em Julho de 2007
  - Draft publicado em Junho de 2008
- Objetivos:
  - Expandir as opções de mapeamentos e aumentar flexibilidade para modelagem de objetos
  - Adição de suporte a coleções de tipos básicos
  - Expansão da JP-QL
  - Suporte para validação
- Mais informações: JSR 317 - http://jcp.org/en/jsr/detail?id=317

Conclusões

- JPA provê uma API simples e padronizada de persistência para Java SE e Java EE
- Padronização torna possível o uso de JPA provider de sua escolha
- Uso de Annotations simplifica a configuração das entidades
- JP-QL permite a construção de consultas complexas
  - Produtividade é o ponto chave!!

Dúvidas?

- Apresentação e código fonte disponíveis em:
- http://www.thedevelopersconference.com.br
- http://code.google.com/p/vofficejava

- Contato:
- henrique@voffice.com.br