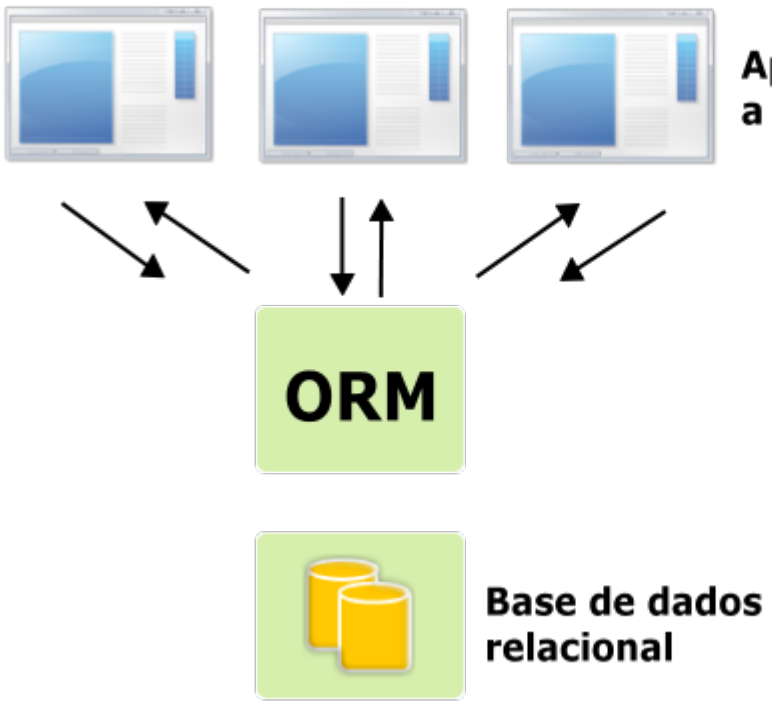


Java Persistence

[Projeto de Banco de Dados]
Aula 06: MOR e JPA

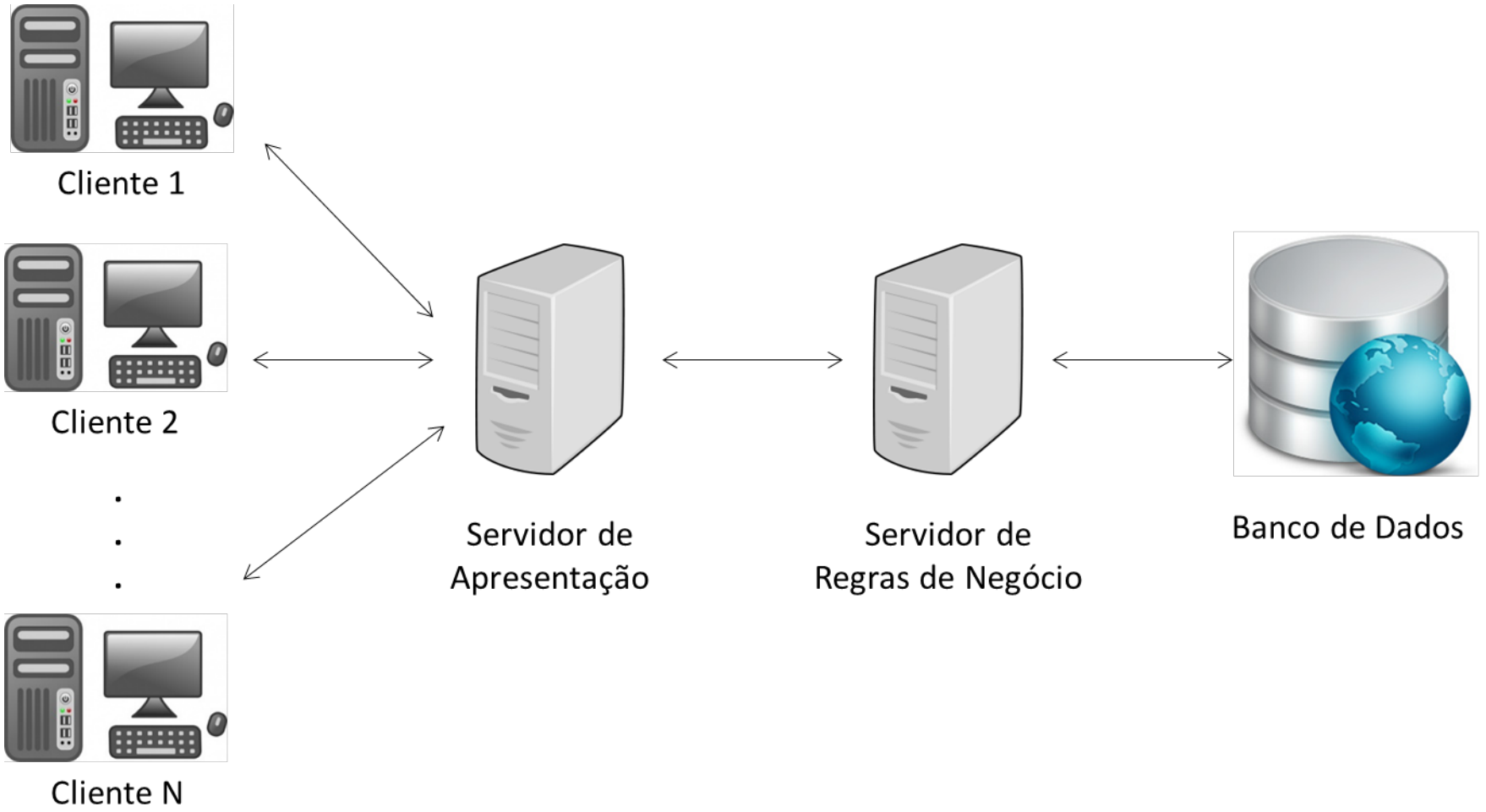
Contexto

- Java Enterprise Edition - JavaEE
  - Desenvolvimento de aplicações corporativas.
    - Aplicações distribuídas, seguras, escaláveis
    - Alta disponibilidade
    - Baixo custo de manutenção

- “[Estas] aplicações são inerentemente complexas, potencialmente com acesso a dados de uma variedade de fontes [distintas] e distribuição de aplicativos para uma variedade de clientes”
  - Java EE7
    - http://www.oracle.com/technetwork/java/javaee/overview/index.html

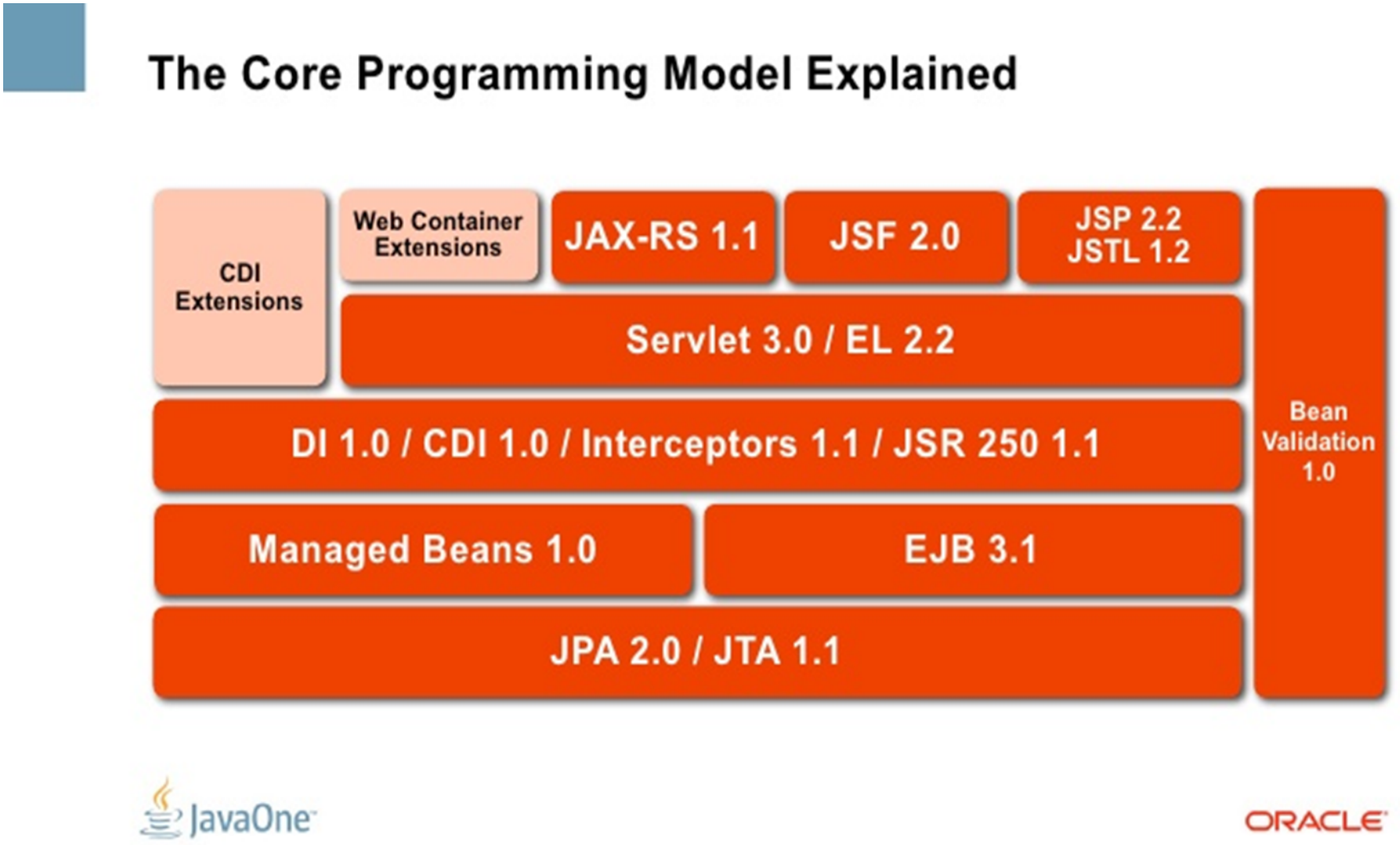
Contexto

- Java Enterprise Edition - JavaEE
  - Arquitetura Multicamada



Contexto

- Java Enterprise Edition - JavaEE



Problema

- Diferenças de Paradigmas:
  - Paradigma Relacional
  - Paradigma Orientado a Objeto.

- Solução encontrada:
  - Bancos de Dados Orientados à Objetos

- Outros problemas.

Mapeamento Objeto-Relacional

- Mapeamento objeto relacional (*object-relational mapping* ou ORM):
  - é uma técnica de programação para conversão de dados entre BDR e linguagens POO.

- Comparando:

Modelo relacional	**Modelo OO**
Tabela	Classe
Linha	Objeto
Coluna	Atributo
-	Método
Chave estrangeira	Associação

Mapeamento Objeto-Relacional

- O mapeamento ORM é feito usando metadados que descrevem a relação entre objetos e banco de dados

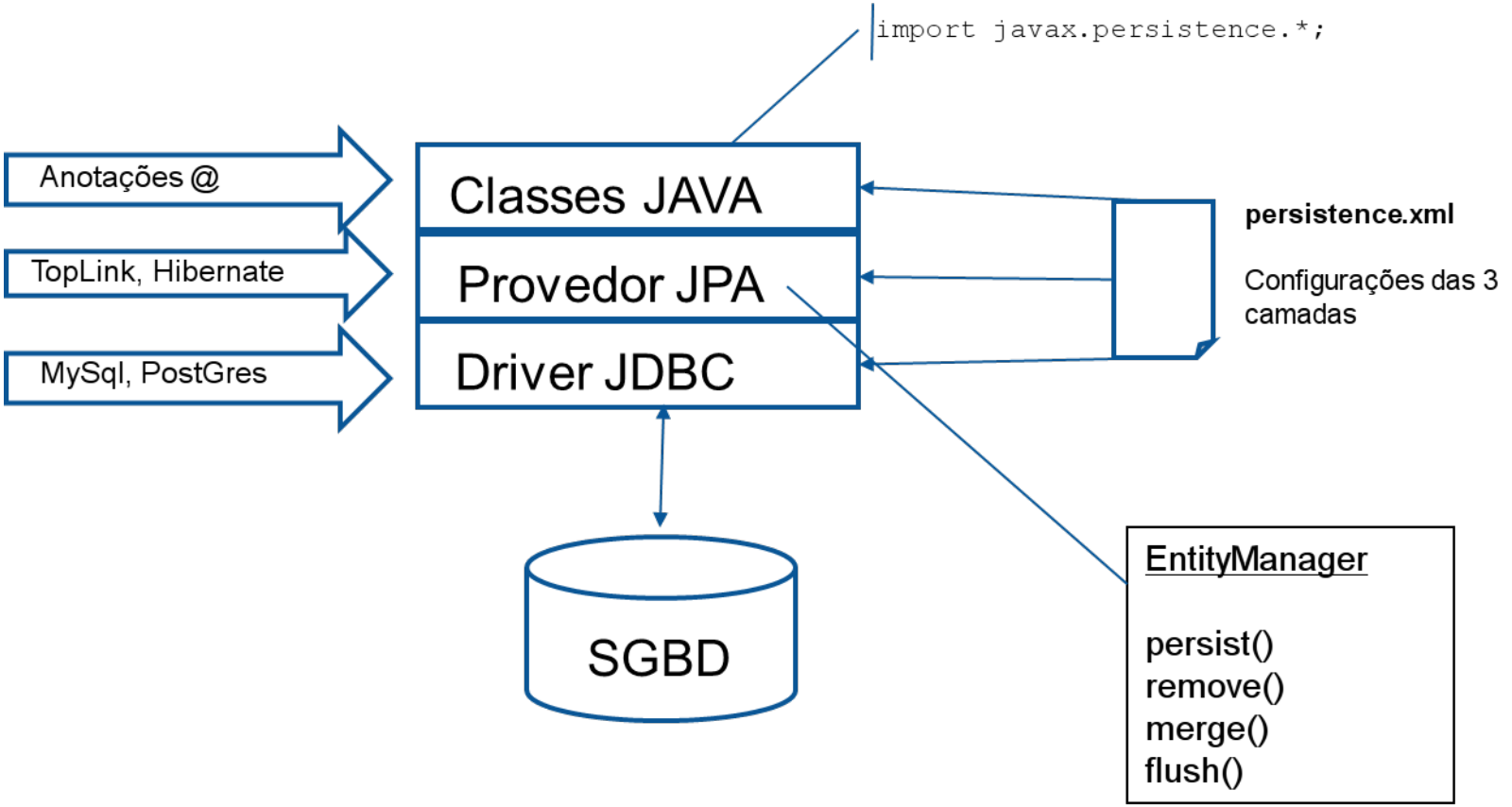
- Vantagens:
  - interação entre modelo de classes de domínio e o BD.
  - comandos SQL automaticamente gerados a partir dos metadados.
  - os códigos SQL/JDBC são abstraídos.
  - diminuição de esforço de codificação
  - aumento de produtividade
  - concentração na lógica de negócio,
  - aumento de manutenibilidade
  - integração com BD's distintos, entre outros.

Java Persistence API (JPA)

- Solução em Java para a persistência de dados no modelo OO em Banco de Dados Relacionais.
- Abordagem para mapeamento objeto-relacional (ORM)
  - Classes que serão “persistidas” recebem anotações especiais.
  - Especificação padrão de objetos Java -> BD relacional.
  - Pode inclusive ser usado em ambientes JSE, fora de containers EJB.

Java Persistence API (JPA)

- Arquitetura:



JPA vs. JDBC

- Persistência com JDBC:
  - mapeamento manual do bando de dados
  - mapeamento manual das consultas SQL
  - mapeamento dos objetos para a consulta, e da consulta
  - maior trabalho braçal
- Persistência com ORM/JPA:
  - uso de anotações para classes persistentes
  - utilização de contextos de persistência
  - utilização do POO em alto-nível
  - mais trabalho lógico que braçal

JPA e JDBC

- Comparação:

Suporte	Serialização	JDBC	ORM	ODB	EJB 2	JDO	JPA
Objetos Java	Sim	Não	Sim	Sim	Sim	Sim	Sim
Conceitos avançados de OO	Sim	Não	Sim	Sim	Não	Sim	Sim
Integridade Transacional	Não	Sim	Sim	Sim	Sim	Sim	Sim
Concorrência	Não	Sim	Sim	Sim	Sim	Sim	Sim
Conjunto Largo de Dados	Não	Sim	Sim	Sim	Sim	Sim	Sim
Existência de Esquemas	Não	Sim	Sim	Não	Sim	Sim	Sim
Relacional e Não-Relacional	Não	Não	Não	Não	Sim	Sim	Não
Consultas	Não	Sim	Sim	Sim	Sim	Sim	Sim
Padrões Estritos / Portabilidade	Sim	Não	Não	Não	Sim	Sim	Sim
Simplicidade	Sim	Sim	Sim	Sim	Não	Sim	Sim

JPA: Entidades

- Um entidade é:
  - Um “objeto” do mundo real (classe java)
  - Um objeto de domínio persistente.
  - Deve ser sinalizada com a anotação *@Entity*

- Principais características de uma entidade:
  - Persistibilidade:
    - são entidades podem ser representadas em um BD (objetos persistentes)
  - Identidade:
    - identidade de objeto (SW), e uma persistente (uma instância única de identificação no BD e nas outras entidades)
  - Transacionabilidade:
    - alterações nos dados necessitam de transações abertas entre o provedor e o BD
  - Granularidade:
    - conjunto agregado de estados ou informações armazenados num único lugar (linha). Valor mais para o BD que para a semântica da aplicação.

JPA: Entidades

- Exemplo:

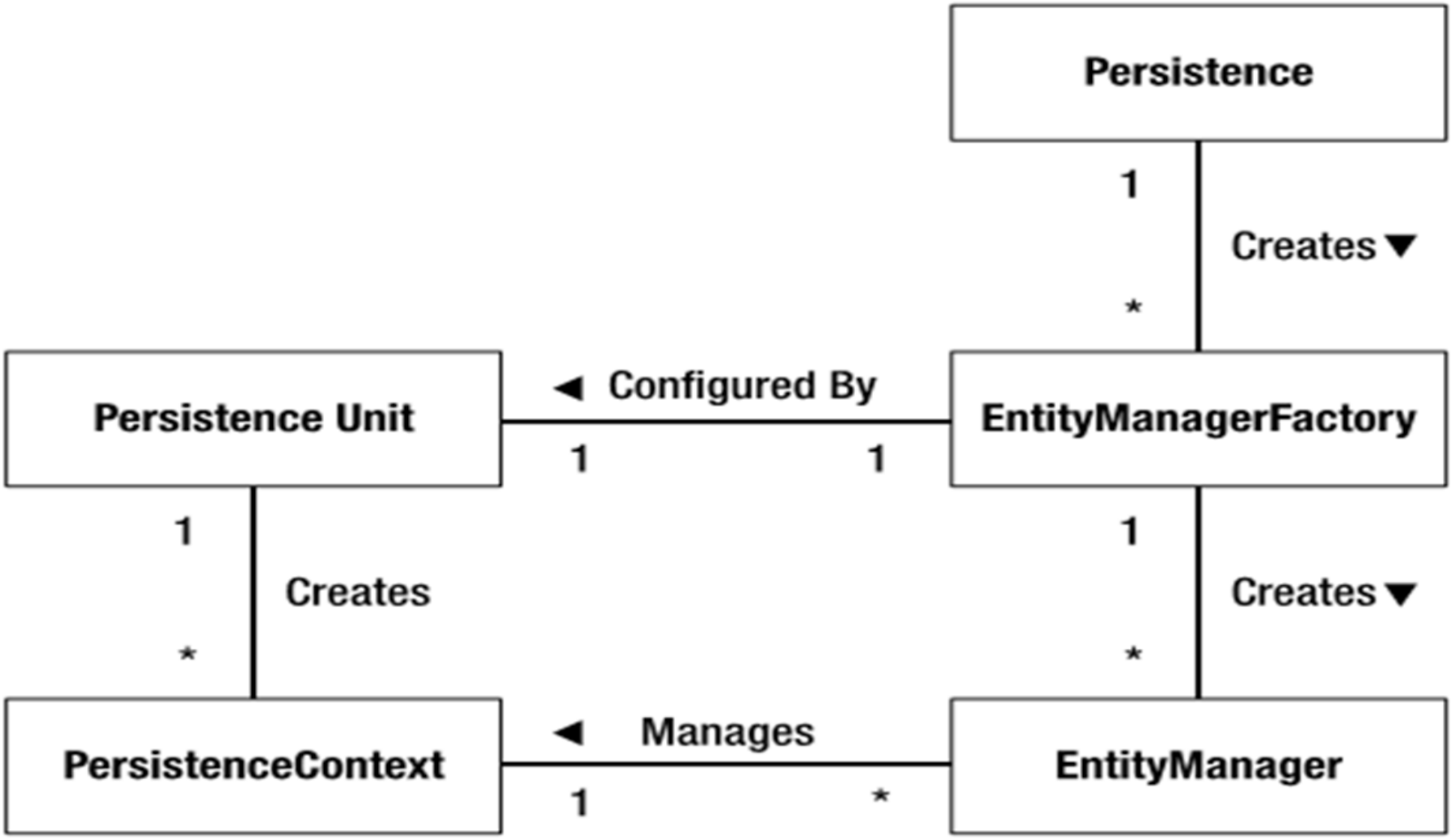
```
@Entity
public class Pessoa{
   @Id
   private int id;
   private String nome;
   private int idade;

   //getters e setters
}
```

ID	Nome	Idade
1	Mariana de Freitas	10
2	Tarcísio Meira	12
3	Martin Valverde	25

JPA: Entity Manager

- O gerenciador de entidades, o ponto de acesso da aplicação ao gerenciamento e acesso às entidades monitoradas pela aplicação.



JPA: Entity Manager

- O gerenciador de entidades, o ponto de acesso da aplicação ao gerenciamento e acesso às entidades monitoradas pela aplicação.
  - EM é o responsável por gerenciar de fato as transações e as entidades.
  - Um EM gerencia várias entidades
  - Conjunto de persistência: conjunto de entidades gerenciadas
  - O provedor de persistência fornece a implementação da persistência através da configuração
  - Fábricas: responsável por criar vários EM necessários, que gerenciam os contextos de persistência.

JPA: Entity Manager

- Operações básicas:
  - Consulta:
    - find() e getReference()
      - localizar entidades no BD pela chave primária.
        - find: retorna *null* se a entity não for encontrada.
        - getReference: retorna uma *EntityNotFoundExeption()*
    - createQuery (CriteriaQuery ou String)
      - encontra entidades baseadas em consultas QueryCriteria ou String.

    - Todas as consultas encontradas são gerenciadas pelo Contexto de persistência ate q ele seja fechado.

- Atualização no banco (persist, delete, merge)
  - persist: recebe uma instância no estado transiente e a torna persistente.
  - merge: atualiza a instância no banco de dados, transferindo seu estado para o banco
  - refresh: atualiza o estado da instância a partir do banco de dados
  - remove: remove a instância do banco de dados.

JPA: Mapeamento de entidades

- Para que as entidades possam ser armazenadas e gerenciadas é necessário um mapeamento de seus atributos e relacionamentos (se existirem)
- O mapeamento pode ser realizado com propriedades simples como outros objetos que relacionam-se com a entidade.

- Exemplos:
  - Mapeando atributos simples
    - @Id
    - @Temporal
    - @Enumerated
    - @Transient

JPA: Mapeamento de entidades

- Para que as entidades possam ser armazenadas e gerenciadas é necessário um mapeamento de seus atributos e relacionamentos (se existirem)
- O mapeamento pode ser realizado com propriedades simples como outros objetos que relacionam-se com a entidade.

- Exemplos:
  - Mapeando relacionamentos (veremos mais detalhes depois)
    - @OneToOne
    - @OneToMany
    - @ManyToOne
    - @ManyToMany

Veremos mais adiante...

Construindo um Exemplo

- Entidades:
  - Obrigatório uso de construtor sem argumentos
  - Cada atributo deve possuir get/set
  - Obrigatório uso de chave primária.
  - Colunas das tabelas podem ter o mesmo nome dos atributos.

Construindo um Exemplo

- Configurar o ambiente
  - BuildPath

Construindo um Exemplo

- Configurar o ambiente
  - Unidade de Persistência
    - Obrigatório: src/META-INF/persistence.xml
    - Define uma ou mais unidades de persistência
    - Cada unidade de persistência tem uma identidade, um nome
    - As classes que pertencem a uma unidade de persistência pode ser selecionado ou especificado
    - Procura automática por classes anotadas com @javax.persistence.Entity
    - Cada unidade de persistência é associada a apenas 1 origem de dados.

Construindo um Exemplo

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
     xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
     xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
     http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

     <!-- UNIDADE DE PERSISTÊNCIA -->
     <persistence-unit name="testeJPA-UP" transaction-type="RESOURCE_LOCAL">
          <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

          <properties>
               <!-- Detectar automaticamente as classes-tabelas -->
               <property name="hibernate.archive.autodetection" value="class"/>

               <!-- conexão com o banco: dialeto e driver -->
               <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect" />
               <property name="javax.persistence.jdbc.driver_class" value="org.postgresql.Driver" />
               <!-- conexão com o banco: usuário, senha e banco -->
               <property name="javax.persistence.jdbc.url"
value="jdbc:postgresql://localhost:5432/testes" />
               <property name="javax.persistence.jdbc.user" value="postgres" />
               <property name="javax.persistence.jdbc.password" value="chewbacca" />

               <!-- conexão com o banco: forma de atualização: create, update... -->
               <property name="hibernate.hbm2ddl.auto" value="update" />

               <!-- Opcional: DEBUG -->
               <property name="hibernate.show_sql" value="true"/>
               <property name="hibernate.format_sql" value="true"/>
               <property name="use_sql_comments" value="true"/>

          </properties>
     </persistence-unit>
</persistence>
```

Construindo um Exemplo

- Configurar o ambiente
  - Unidade de Persistência
    - Propriedade: “hibernate.hbm2ddl.auto”
      - Cria, dropa ou atualiza as tabelas no banco
        - Valores: create, create-drop, update
    - Propriedade: “hibernate.dialect”
    - Valores:
      - org.hibernate.dialect.PostgreSQLDialect
      - org.hibernate.dialect.HSQLDialect
      - org.hibernate.dialect.Oracle9Dialect
      - org.hibernate.dialect.MySQLDialect
      - org.hibernate.dialect.SQLServerDialect
      - Depende do SGBD

Construindo um Exemplo

- Configurar o ambiente
  - Unidade de Persistência
    - Propriedade: “hibernate.connection.driver_class”
      - Valores: driver JDBC do Banco
    - Propriedade: “hibernate.connection.user”
      - Valores: Usuário do Banco
    - Propriedade: “hibernate.connection.password”
      - Valores: Senha do usuário do Banco
    - Propriedade: “hibernate.connection.url”
      - Valores: URL do Banco

Construindo um Exemplo

- Criar classes de entidade
- @javax.persistence.Entity
  - Define uma classe como Bean de Entidade
- @javax.persistence.Table
  - Associa uma classe a uma tabela no banco de dados
- @javax.persistence.Id
  - Define um campo como sendo uma chave primária
- @javax.persistence.GeneratedValue
  - Define um campo como sendo de valor auto incrementável
    - AUTO: O provedor de persistência será responsável por gerar a chave
    - IDENTITY: Utiliza um tipo especial de coluna para gerar a chave. Existem em alguns BD
    - TABLE: Designa uma tabela relacional, definida pelo usuário, onde as chaves numéricas serão geradas
    - SEQUENCE: Gera Ids seqüencialmente via um sequence gerado no BD

Construindo um Exemplo

- Criar classes de entidade
- @javax.persistence.Column
  - name="columnName"; // o nome da coluna (padrão é o nome do atributo)
  - boolean unique() default false; // determina se o valor do campo é único ou não (padrão falso)
  - boolean nullable() default true; // determinal se a coluna aceita valores nulos (padrão falso)
  - boolean insertable() default true; // se a coluna irá fazer parte da setença SQL de inserção
  - boolean updatable() default true; // se a coluna irá fazer parte da setença SQL de alteração
  - String columnDefinition() default ""; // sobrescreve o fragmento de DDL para esta coluna
  - String table() default ""; // define a tabela alvo (padrão a tabela primária)
  - int length() default 255; // tamanho da coluna (padrão 255)
  - int precision() default 0; // precisão da coluna decimal (total de dígitos)
  - int scale() default 0; // escala da coluna decimal (depois da vírgula)