Desenvolvimento para Servidores-II

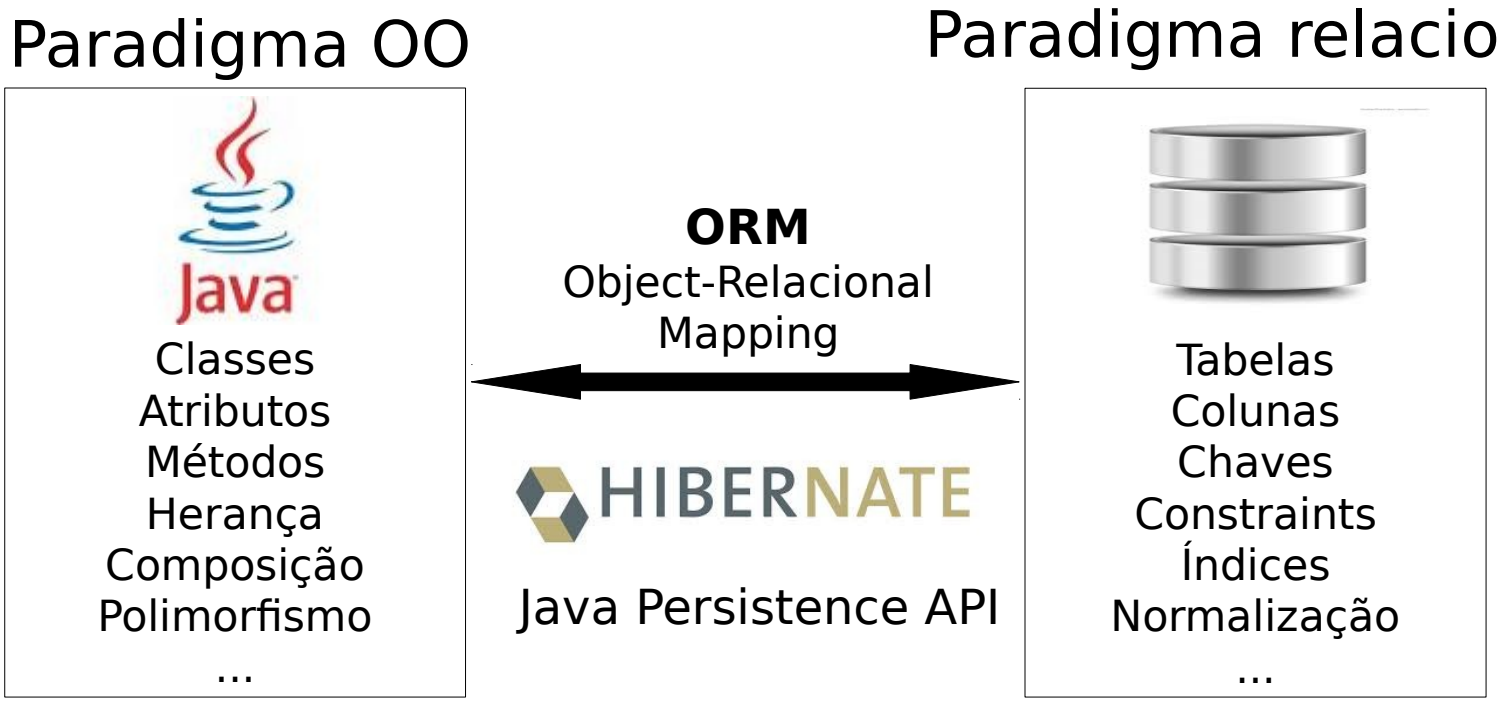
JPA (Java Persistence API)

Neste tópico abortaremos o uso de JPA para persistência de objetos em um banco de dados relacional

Prof. Ciro Cirne Trindade

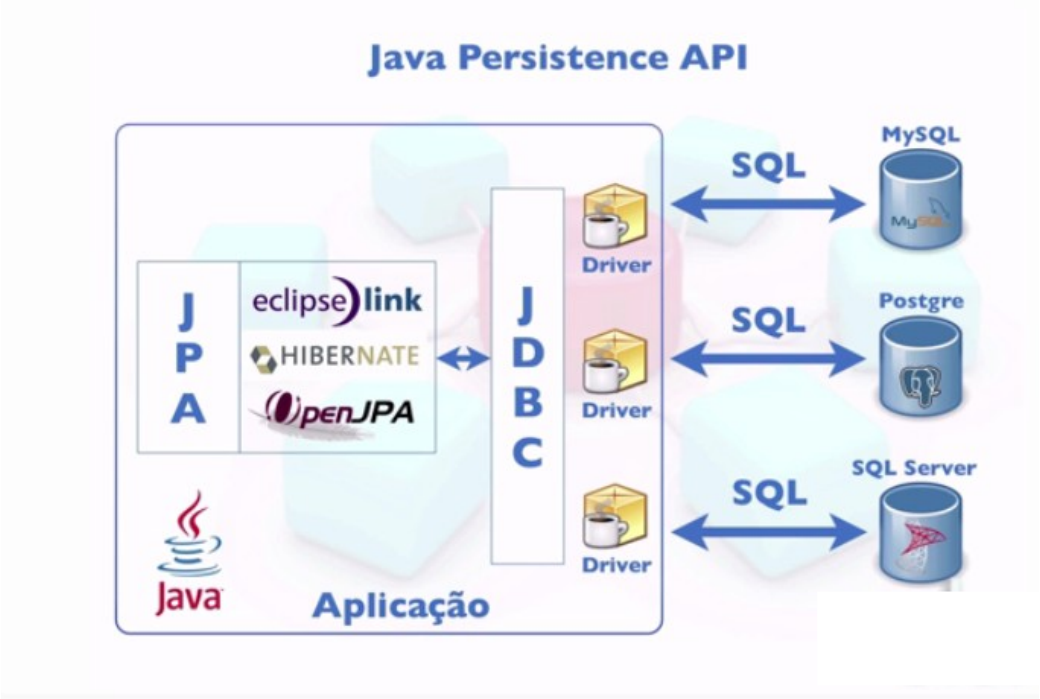
Introdução (1/4)

- JPA provê aos desenvolvedores Java facilidades de um mapeamento objeto/relacional para manipular dados relacionais em aplicações Java



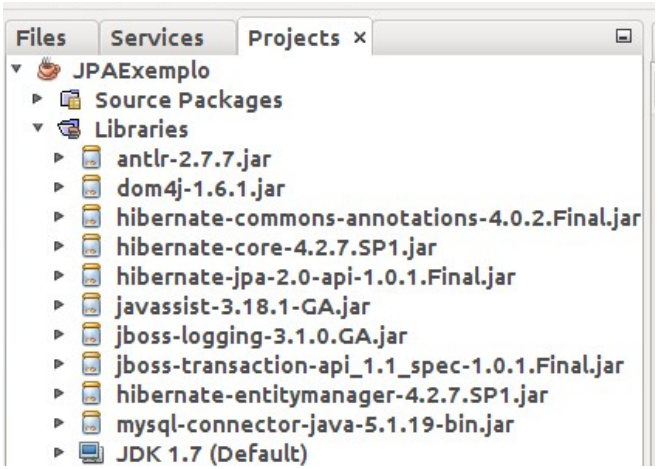
Introdução (2/4)

- Existem diferentes implementação de JPA, todas utilizam JDBC



Introdução (3/4)

- Vamos usar o JPA com Hibernate, ou seja, precisamos baixar os JARs no site do Hibernate (hibernate.org)
- JPA abstrai do desenvolvedor a camada SQL, mesmo assim é preciso copiar o JAR do driver JDBC



Introdução (4/4)

- Usando JPA a aplicação nunca manipula o banco de dados diretamente
- JPA usa anotações para marcar as classes que devem ser armazenadas no banco de dados
- Essas classes são chamadas de **entidades**

Entidades (1/3)

- Tipicamente uma entidade representa uma tabela num banco de dados relacional
- Cada instância de uma entidade representa um registro nesta tabela

Entidades (2/3)

- Requisitos para que uma classe seja uma entidade:
  - A classe deve ser anotada com `@Entity`
  - Cada classe deve possuir um identificador único, marcado com a anotação `@Id`
  - A classe deve possuir um construtor padrão (sem argumentos)
  - Se a classe for armazenada na sessão, deve implementar a interface `Serializable`
  - Os atributos da classe não devem ser públicos e só podem ser acessados por métodos da classe

Entidades (3/3)

- A anotação `@GeneratedValue` é opcional, mas é muito comum usar com chaves auxiliares
- Com ela indicamos que o banco deve atribuir o valor da chave, e não a aplicação
- Ao inserir uma conta no banco de dados, automaticamente será alocada uma `ID`
- Como usaremos MySQL deixamos a estratégia como `Identity`
  - `@GeneratedValue(strategy= GenerationType.IDENTITY)`

Exemplo de uma entidade (1/2)

```
package financas.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import java.io.Serializable;

@Entity
public class Conta implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private String titular;
    private String numero;
    private String agencia;
    private String banco;

    public Conta() { }
```

Exemplo de uma entidade (2/2)

```
    public Long getId() { return id; }

    public void setId(Long id) {
        this.id = id;
    }

    public String getAgencia() { return agencia; }

    public void setAgencia(String agencia) {
        this.agencia = agencia;
    }

    public String getBanco() { return banco; }

    public void setBanco(String banco) {
        this.banco = banco;
    }
    ...
}
```

Unidade de persistência (1/2)

- Para definir os dados de conexão, o JPA possui um arquivo de configuração, o `persistence.xml`
- Pela especificação esse arquivo deve ficar dentro da pasta `META-INF`
- No `persistence.xml` toda configuração fica dentro de um elemento `persistence-unit`

Unidade de persistência (2/2)

- Exemplo do `persistence.xml` para `financas`

```
<persistence 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" version="2.0">
   <persistence-unit name="financas">
      <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <class>financas.modelo.Conta</class>
      <properties>
         <property name="javax.persistence.jdbc.driver"
                   value="com.mysql.jdbc.Driver" />
         <property name="javax.persistence.jdbc.url"
                   value="jdbc:mysql://localhost/financas" />
         <property name="javax.persistence.jdbc.user" value="root" />
         <property name="javax.persistence.jdbc.password" value="root" />
         <property name="hibernate.dialect"
                   value="org.hibernate.dialect.MySQL5InnoDBDialect" />
         <property name="hibernate.hbm2ddl.auto" value="update" />
         <property name="hibernate.show_sql" value="true" />
         <property name="hibernate.format_sql" value="true" />
      </properties>
   </persistence-unit>
</persistence>
```

Classes de persistência

Implementação do JPA do Hibernate

Propriedades do JDBC

Configurações específicas do Hibernate

Gerenciando entidades (1/5)

- Entidades são gerenciadas pelo gerenciador de entidades, que é representado por instâncias da interface `javax.persistence.EntityManager`
- Cada instância de `EntityManager` é associada a um contexto de persistência: um conjunto de entidades de um banco de dados particular

Gerenciando entidades (2/5)

- A primeira coisa a fazer é carregar a configuração contida no arquivo `persistence.xml`
- O JPA possui uma classe com o mesmo nome: `Persistence`
- Usaremos ela para criar uma `EntityManagerFactory` baseada na unidade de persistência `financas`
  - `EntityManagerFactory emf = Persistence.createEntityManagerFactory ("financas");`

Gerenciando entidades (3/5)

- Um `EntityManager` é instanciado através do método `createEntityManager()` da classe `EntityManagerFactory`
  - `EntityManager manager = emf.createEntityManager();`
- O `EntityManager` possui os principais métodos do JPA
- Através do `EntityManager` podemos, por exemplo, persistir uma entidade

Gerenciando entidades (4/5)

- Para executar uma operação no banco de dados através do `EntityManager` é necessário iniciar e “comitar” uma transação
- O `EntityManager` possui o método `getTransaction()`, que devolve um objeto que representa a transação
  - `manager.getTransaction().begin();`
  - `// operação no banco`
  - `manager.getTransaction().commit();`

Gerenciando entidades (5/5)

- Para persistir uma entidade no banco de dados utilizamos o método `persist()` do `EntityManager`
- Este método espera como parâmetro a entidade que deve ser persistida no banco

Persistindo entidades

- O JPA cria as tabelas, mas é necessário criar o banco de dados
  - `$>mysql -u root -p`
  - `mysql>create database financas;`

Exemplo de persistência (1/2)

```
package financas.teste;

import financas.model.Conta;
import java.util.Scanner;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class Teste {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Conta c = new Conta();
        System.out.print("Titular: ");
        c.setTitular(in.nextLine());
        System.out.print("Número: ");
        c.setNumero(in.nextLine());
        System.out.print("Agência: ");
        c.setAgencia(in.nextLine());
        System.out.print("Banco: ");
        c.setBanco(in.nextLine());
```

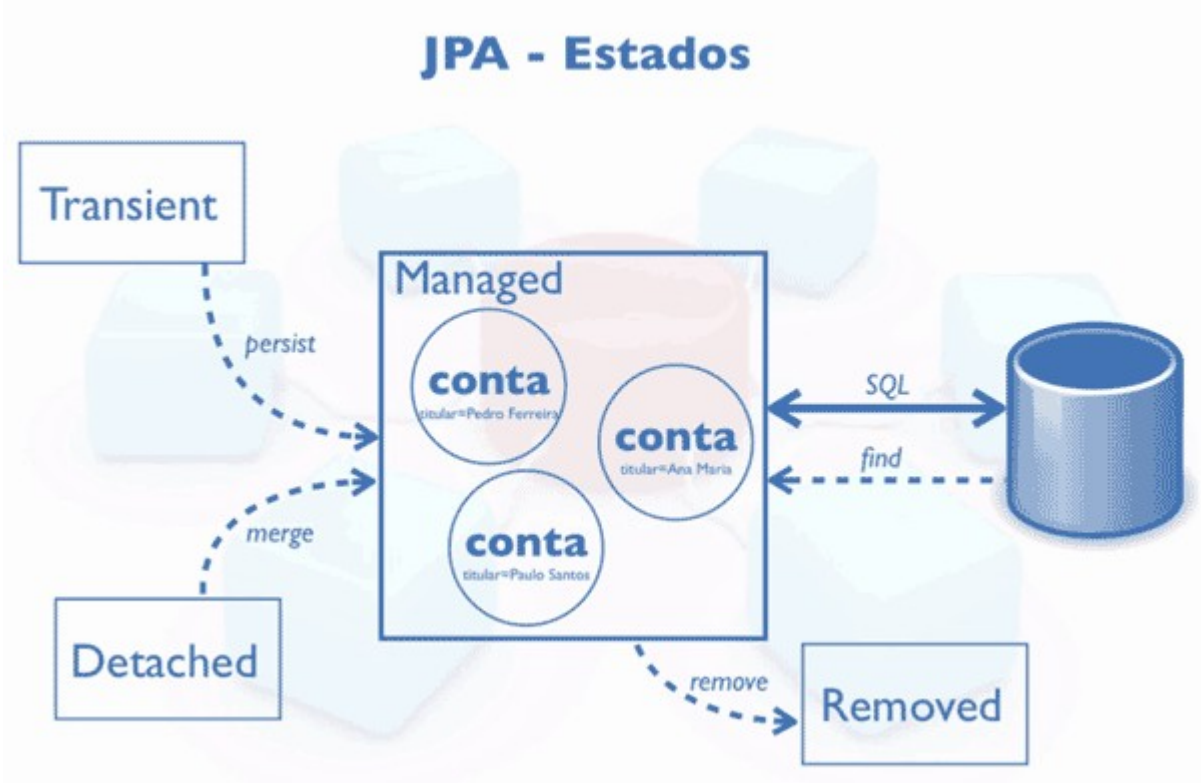
Exemplo de persistência (2/2)

```
        EntityManagerFactory emf =
Persistence.createEntityManagerFactory("financas");
        EntityManager manager =
emf.createEntityManager();

        manager.getTransaction().begin();

        manager.persist(c);

        manager.getTransaction().commit();
        manager.close();
    }
}
```

Localizando, atualizando e removendo

- Métodos de `EntityManager`
  - `find(Classe.class, <id>)`
    - Não é necessário abrir uma transação
  - `merge(objeto)`
  - `remove(objeto)`

Estados JPA


JPA - Estados
Transient
persist
Managed
conta
titular=Pedro Ferreira
conta
titular=Ana Maria
conta
titular=Paulo Santos
SQL
find
merge
Detached
remove
Removed

Relacionamentos

- Os relacionamentos em JPA devem ser marcados com as seguintes anotações:
  - `@OneToOne`: um para um
  - `@OneToMany`: um para muitos
  - `@ManyToOne`: muitos para um
  - `@ManyToMany`: muitos para muitos
- Os atributos que representam relacionamentos `@OneToMany` e `@ManyToMany` devem ser `Collections`

Exemplo: movimentação da Conta (1/2)

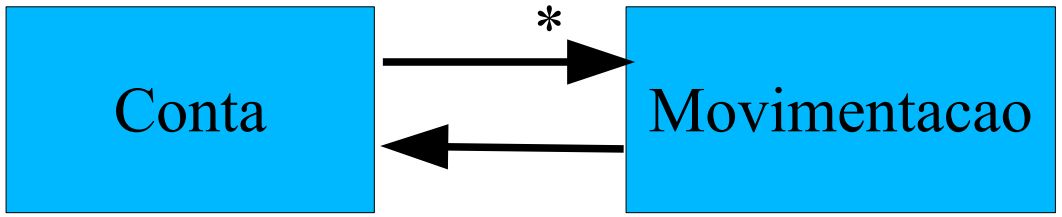
```
package model.entity;

import java.util.Date;
import javax.persistence.*;
import java.io.Serializable;

@Entity
public class Movimentacao implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private Long id;
    private double valor;
    @Enumerated(EnumType.STRING)
    private TipoMovimentacao tipo;
    private String descricao;
    @Temporal(TemporalType.DATE)
    private Date data;
    @ManyToOne
    private Conta conta;
```

Exemplo: movimentação da Conta (2/2)

```
    public Conta getConta() {
        return conta;
    }

    public void setConta(Conta conta) {
        this.conta = conta;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    ...
}
```

Relacionamentos bidirecionais

- Em relacionamentos bidirecionais é necessário informar o “dono” da relação
- Isso é feito através do atributo `mappedBy` da anotação



Exemplo: uma conta associada a várias movimentações (1/2)

```
package model.entity;

import java.util.List;
import javax.persistence.*;
import java.io.Serializable;

@Entity
public class Conta implements Serializable {
    @Id
    @GeneratedValue(strategy= GenerationType.IDENTITY)
    private Long id;
    private String titular;
    private String numero;
    private String agencia;
    private String banco;
    @OneToMany(mappedBy="conta")
    private List<Movimentacao> movimentacoes;

    public Conta() {
    }
```

Exemplo: uma conta associada a várias movimentações (2/2)

```
    public List<Movimentacao> getMovimentacoes() {
        return movimentacoes;
    }

    public void setMovimentações(List<Movimentacao>
movimentações) {
        this.movimentacoes = movimentacoes;
    }

    public String getAgencia() {
        return agencia;
    }

    public void setAgencia(String agencia) {
        this.agencia = agencia;
    }
    ...
}
```

DAO Genérico (1/2)

```
package model.dao;

import javax.persistence.EntityManager;
import util.jpa.JPAEntityManager;

public class DAO<T> {
    private final Class<T> classe;
    private EntityManager manager;

    public DAO(Class<T> classe) {
        this.classe = classe;
    }

    public void adicionar(T t) {
        manager = JPAEntityManager.getEntityManager();
        manager.getTransaction().begin();
        manager.persist(t);
        manager.getTransaction().commit();
        manager.close();
    }
```

DAO Genérico (2/2)

```
    public T consultar(Long id) {
        manager = JPAEntityManager.getEntityManager();
        T instancia = manager.find(classe, id);
        manager.close();
        return instancia;
    }

    public void alterar(T t) {
        manager = JPAEntityManager.getEntityManager();
        manager.getTransaction().begin();
        manager.merge(t);
        manager.getTransaction().commit();
        manager.close();
    }
}
```

Referências

- GEARY, David; HORSTMANN, Cay. *Core JavaServer Faces*. 3. ed., Prentice-Hall, 2010.
- ORACLE Corporation. *The Java EE 7 Tutorial*. Disponível em: https://docs.oracle.com/javaee/7/JEETT.pdf, 2014.