MC0037 – Programação para Web

Aula 2: Hibernate / JPA

2 Q / 2018

Aula de hoje: Roteiro

- Hibernate / JPA

Persistência / ORM

- Usar o JDBC é uma das maneiras mais diretas para desenvolver um aplicativo Java que interaja com um BD
- Porém, escrever queries SQL no meio do código Java é improdutivo (tempo de programação é grande)
- Também, devido às diferenças entre as linguagens (sintaxe do SQL), em geral não é fácil trocar de um BD para outro

- Além disso, há uma diferença de paradigmas entre orientação a objetos e o esquema entidade-relacionamento (é preciso transformar objetos em linhas de tabelas e linhas de tabelas em objetos)

- Para auxiliar nesta tarefa, surgiram ferramentas de Mapeamento Objeto-Relacional (ORM) que fazem a persistência de forma automatizada e transparente

Mapeamento Objeto Relacional (ORM)

- ORM (Object/Relational Mapping): técnica para mapear dados de um modelo de objetos (POJOs) para um modelo relacional (registros do banco de dados)
- POJO (Plain Old Java Object): se refere a classes Java simples e comuns, que não estendem ou implementam alguma outra classe (ou interface)
  - Ex.: Classe ContaCorrente (da aplicação exemplo dos exercícios)
  - Não é POJO:

```
class TesteServlet extends HttpServlet { … }
```

- Vantagens do ORM
  - Agiliza o desenvolvimento e reduz a quantidade de código
  - Foca na lógica de negócios
  - Portabilidade: é independente do banco de dados (SQL específica de fabricante é gerada automaticamente)

JPA

- O JPA (Java Persistence API), é a API padrão do Java para especificação da persistência (Java SE e Java EE)

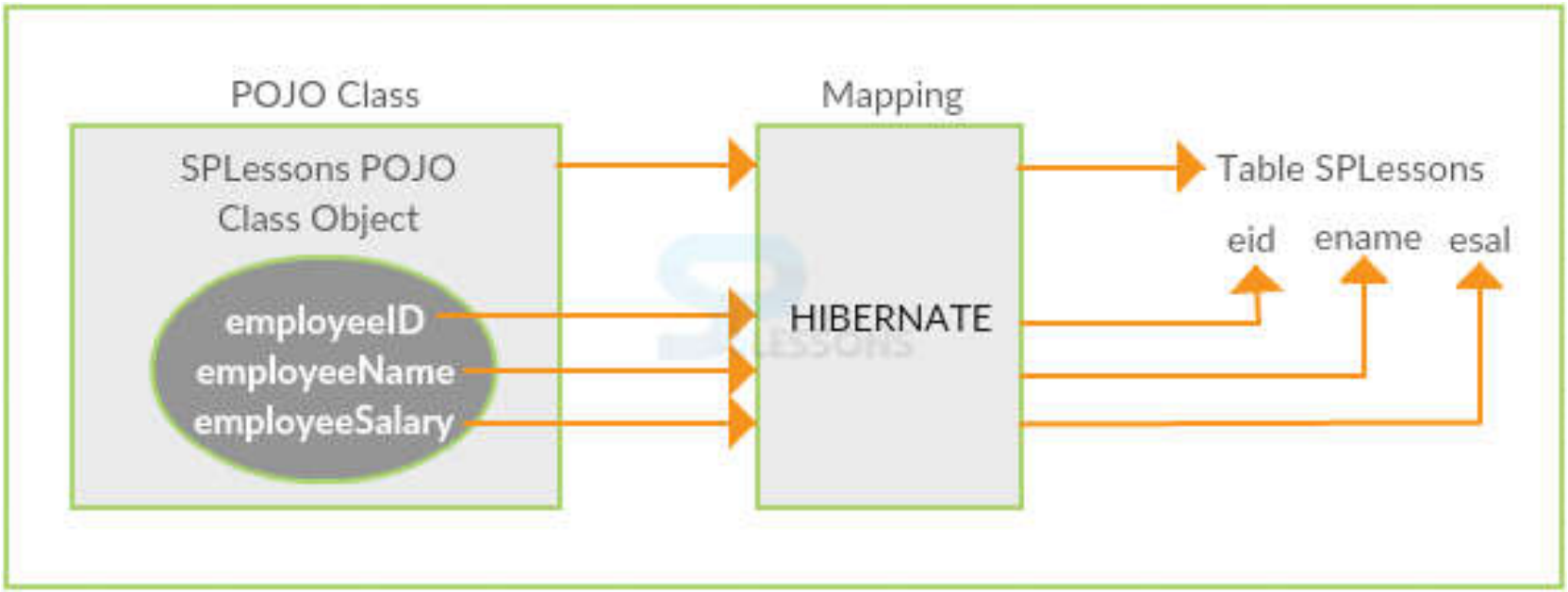
- JPA é uma especificação (independente de fabricante do BD) que permite mapear objetos Java a tabelas de bancos de dados relacionais

- Implementações comuns do JPA:
  - Hibernate, Jboss
  - EclipseLink, Eclipse Foundation
  - OpenJPA, Apache

- Utilizar a API do JPA possibilita a portabilidade entre diferentes implementações (ou provedores, por exemplo, do Hibernate para EclipseLink)

Hibernate

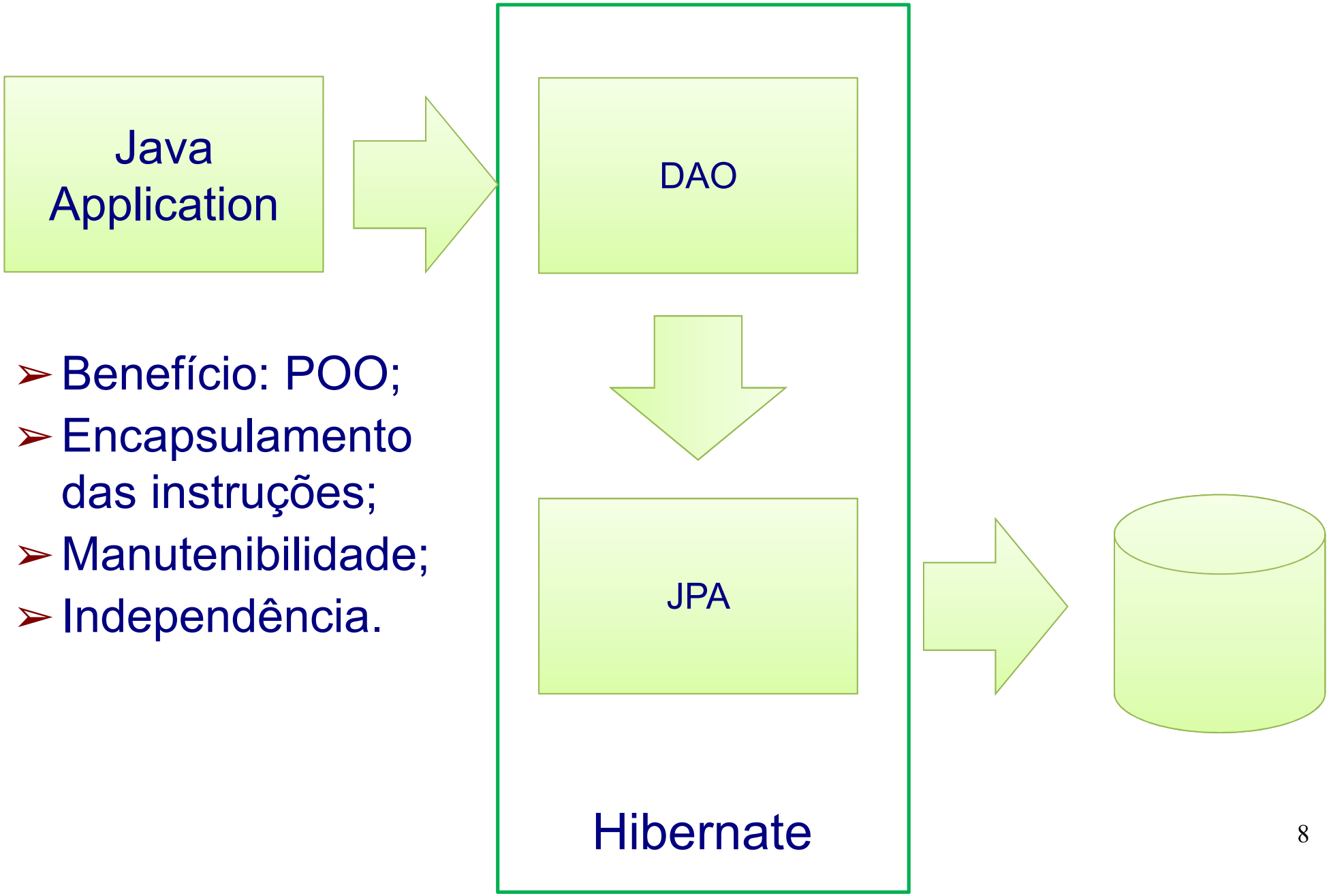
- O Hibernate é um framework objeto-relacional para aplicações Java
- É open source, além de sua API nativa, implementa a especificação JPA
- Vantagens: alta performance, escalabilidade, confiável, extensível
- http://hibernate.org/orm/releases/5.0/



Hibernate

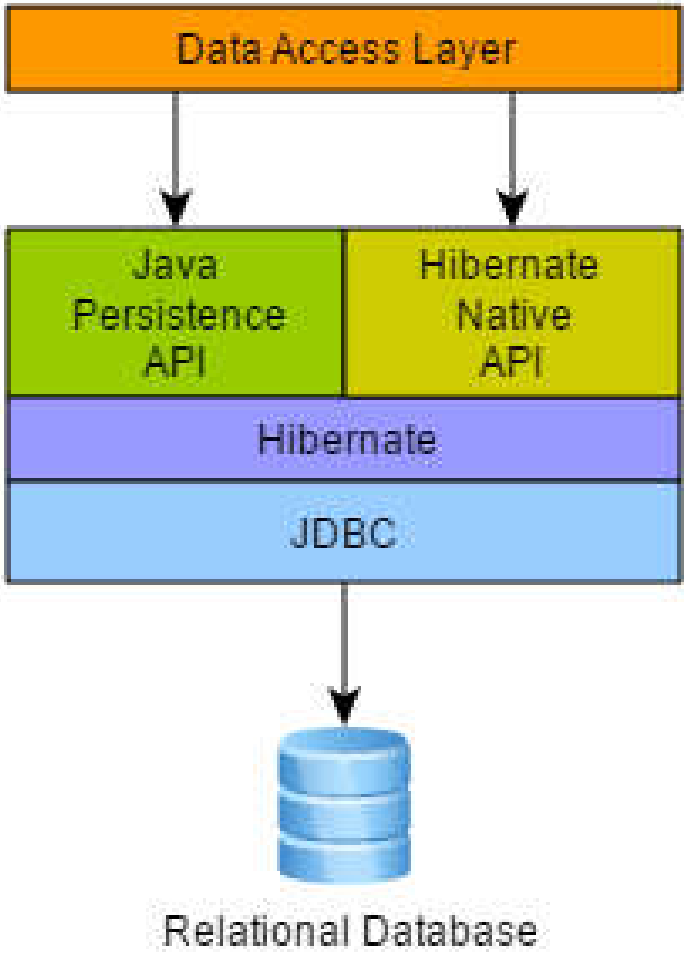
- Criar as instruções SQL na mão;
- Dependência do banco de dados (Postgresql, MySQL, etc);
- Desenvolvimento “perde” o potencial da POO;
- Difícil manutenção;
- Escalabilidade limitada;
- Etc.

Hibernate

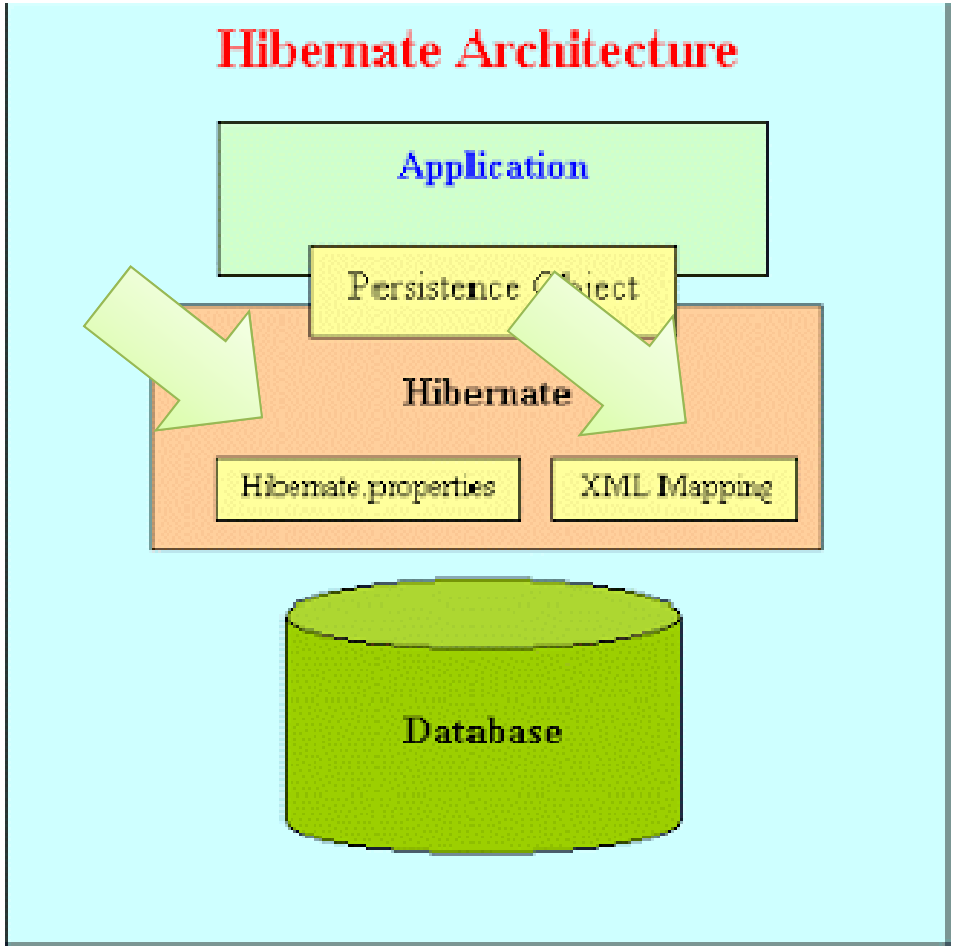


- Benefício: POO;
- Encapsulamento das instruções;
- Manutenibilidade;
- Independência.

Hibernate



Relação Hibernate X JPA



Elementos Principais Hibernate

Criando uma entidade

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="contacorrente")
public class ContaCorrente {
        @Id
        @GeneratedValue
        private Long id;

        @Column(name="numero", nullable=false)
        private String numero;

        @Column(name="descricao")
        private String descricao;

        @Column(name="ativa")
        private boolean ativa;
        ...
}
```

Annotations

Anotações

- A anotação `@Entity` indica que a classe ContaCorrente é uma entidade (que será persistida pelo Hibernate). Toda classe (POJO – Plain Old Java Object) que representa uma entidade precisa ter essa anotação

- A anotação `@Table` permite definir o nome da tabela em que a entidade será persistida. É opcional, se não estiver definida, o nome da tabela é o mesmo nome da classe da entidade

- A anotação `@Id` indica que o campo id é a chave primária da tabela e `@GeneratedValue` significa que o id será gerado pelo Hibernate (por exemplo, AUTO_INCREMENT do MySQL)

Anotações

- A anotação `@Column` é usada para mapear uma coluna da tabela a um campo da classe. É opcional, se não estiver definida, por padrão o nome do campo é o nome da coluna
  Outros possíveis atributos: nullable, length, unique, etc.

- Os imports deve ser do pacote javax.persistence

- Veja maiores detalhes sobre as anotações em: https://docs.oracle.com/javaee/7/api/, pacote javax.persistence (Annotation Types)

Configuração do JPA/Hibernate

- Arquivo persistence.xml e deve estar em src/META-INF/:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence 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"
         version="2.1">

         <persistence-unit name="sistemabancario">

                  <!-- implementacao do JPA (provedor) -->
                  <provider>org.hibernate.ejb.HibernatePersistence</provider>

                  <!-- entidade mapeada -->
                  <class>br.com.bb.sistemabancario.entity.ContaCorrente</class>

                  <properties>
                           <!-- conexao -->
                           <property name="javax.persistence.jdbc.driver"
                                    value="org.postgresql.Driver" />
                           <property name="javax.persistence.jdbc.url"
                                    value="jdbc:postgresql://localhost/progwebjpa" />
                           <property name="javax.persistence.jdbc.user"
                                    value="root" />
                           <property name="javax.persistence.jdbc.password"
                                    value="root" />
                           <property name="hibernate.dialect"
                                    value="org.hibernate.dialect.PostgreSQLDialect" />
```

Configuração do JPA/Hibernate

- persistence.xml (cont.):

```
                        <!-- conexao H2 -->
                        <!-- <property name="javax.persistence.jdbc.driver"
                                value="org.h2.Driver" />
                        <property name="javax.persistence.jdbc.url"
                                value="jdbc:h2:tcp://localhost/~/progwebjpa" />
                        <property name="javax.persistence.jdbc.user"
                                value="admin" />
                        <property name="javax.persistence.jdbc.password"
                                value="admin" />
                        <property name="hibernate.dialect"
                                value="org.hibernate.dialect.H2Dialect"/> -->

                        <!-- imprime as queries SQL no console -->
                        <property name="hibernate.show_sql" value="true" />
                        <property name="hibernate.format_sql" value="true" />
                        <!-- gera as tabelas se necessario -->
                        <property name="hibernate.hbm2ddl.auto" value="update" />
                </properties>
        </persistence-unit>
</persistence>
```

Gerando tabelas com o JPA

- Baseado nas anotações (da entidade) e no arquivo persistence.xml, o JPA é capaz de gerar automaticamente a tabela no banco de dados
- Exemplo: para gerar a tabela “contacorrente” no banco de dados

```
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class GeradorTabelas {
        public static void main(String[] args) {
                EntityManagerFactory factory = Persistence
                        .createEntityManagerFactory("sistemabancario");
                System.out.println("Tabela gerada!");
                factory.close();
        }
}
```

Nome da unidade de persistência definida em persistence.xml

Acessando o banco de dados

- Para o acesso ao banco de dados, o JPA oferece as seguintes classes:
  - **EntityManagerFactory**: faz a conexão com o banco de dados com as propriedades definidas no arquivo persistence.xml; também gerencia os objetos **EntityManager**
  - **EntityManager:** oferece métodos para executar as operações (inserção, alteração, remoção e consulta) no banco de dados e uma instância é obtida através do objeto **EntityManagerFactory**
  - **EntityTransaction:** gerencia as transações (toda operação que modifica o banco de dados (inserção, remoção, alteração) devem ser executadas dentro de uma transação)

- Operações:
  - **persist**(objeto): salva um objeto no banco de dados
  - **remove**(objeto): remove o objeto (registro) do banco de dados
  - **merge**(objeto): altera um registro no banco de dados
  - **find**(class, id): busca um objeto do banco de dados de acordo com o ID e classe do objeto

Salvando um objeto ContaCorrente

```
public void insere(@Valid ContaCorrente cc, BindingResult result) {

        EntityManagerFactory factory = Persistence
                        .createEntityManagerFactory("sistemabancario");
        EntityManager manager = factory.createEntityManager();
        try {
                manager.getTransaction().begin();
                manager.persist(cc);
                manager.getTransaction().commit();
        }
        finally {
                if(manager.getTransaction().isActive())
                        manager.getTransaction().rollback();
        }
        manager.close();        }
```

Removendo um objeto ContaCorrente

```
public void remove(ContaCorrente cc) {
        EntityManagerFactory factory = Persistence
                .createEntityManagerFactory("sistemabancario");
        EntityManager manager = factory.createEntityManager();
        manager.getTransaction().begin();
        cc = manager.find(ContaCorrente.class, cc.getId());
        try {
                manager.getTransaction().begin();
                manager.remove(cc);
                manager.getTransaction().commit();
        }
        finally {
                if(manager.getTransaction().isActive())
                        manager.getTransaction().rollback();
        }
        manager.close();

}
```

Buscando um ContaCorrente

```
public ContaCorrente exibe(Long id) {
        EntityManagerFactory factory = Persistence
                .createEntityManagerFactory(“sistemabancario");
        EntityManager manager = factory.createEntityManager();
        ContaCorrente cc = manager.find(ContaCorrente.class, id);
        manager.close();
        return cc;
}
```

Alterando um objeto ContaCorrente

```
public void altera(@Valid ContaCorrente cc) {
        EntityManagerFactory factory = Persistence
                .createEntityManagerFactory("sistemabancario");
        EntityManager manager = factory.createEntityManager();
        try {
                manager.getTransaction().begin();
                manager.merge(cc);
                manager.getTransaction().commit();
        }
        finally {
                if(manager.getTransaction().isActive())
                        manager.getTransaction().rollback();
        }
        manager.close();
}
```

Listando ContaCorrente

```
public List<ContaCorrente> lista() {
        EntityManagerFactory factory = Persistence
                .createEntityManagerFactory(“sistemabancario");
        EntityManager manager = factory.createEntityManager();

        @SuppressWarnings("unchecked")
        List<ContaCorrente> ccs = manager.createQuery("select cc
                        from ContaCorrente cc").getResultList();

        manager.close();
        return ccs;
}
```

Migrando Banco de Dados

- Caso queiramos migrar de Postgres para outro banco (por ex., H2 ou MySQL) são necessários apenas 2 (DOIS!) passos:

  - Modificar o persitence.xml
  - Acrescentar o JDBC apropriado.

Migrando Banco de Dados - MySQL

```
<!-- conexao -->
<property name="javax.persistence.jdbc.driver"
      value="com.mysql.jdbc.Driver" />
<property name="javax.persistence.jdbc.url"
      value="jdbc:mysql://localhost/progwebjpa" />
<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" />
```

Migrando Banco de Dados – H2

```
<!-- conexao -->
<property name="javax.persistence.jdbc.driver"
        value="org.h2.Driver"/>
<property name="javax.persistence.jdbc.url"
        value="jdbc:h2:~/sistemabancario" />
<property name="javax.persistence.jdbc.user"
        value="sa" />
<property name="javax.persistence.jdbc.password"
        value="" />
<property name="hibernate.dialect"
        value="org.hibernate.dialect.H2Dialect"/>
```