

AULA 15 –MVC E HIBERNATE

GSI020 – Programação Orientada a Objetos II

Prof. Dr. Murillo G. Carneiro
mgcarneiro@ufu.br



Material baseado nos slides cedidos pelo Prof. Rafael D. Araújo (FACOM/UFU)

Objetivo da aula

- Entender o padrão arquitetural MVC.

Model-View-Controller (MVC)

- O padrão MVC (em português, Modelo-Visão-Controle) não é um padrão de projetos e, sim, um **padrão arquitetural**

“Uma arquitetura de software envolve a **descrição de elementos arquiteturais** dos quais os sistemas serão construídos, **interações** entre esses elementos, **padrões** que guiam suas composições e **restrições** sobre estes padrões”.

¹ S. L. Pfleeger, *Software Engineering: Theory and Practice*, Prentice Hall, 1998

Sistema com uma única camada



- Difícil reuso
- Código desorganizado
- Alterações em qualquer camada afetam todas as outras

Sistema com uma única camada

- Imagine um aplicativo para tocar músicas feito para ser executado em três equipamentos diferentes: computador desktop, celular e som de



Sistema com uma única camada

- Imagine um aplicativo para tocar músicas feito para ser executado em três equipamentos diferentes: computador, mp3, celular e som de

Repetição de Código!



Sistema várias camadas



- Fácil reuso
- Código organizado
- Alterações em uma camada não afeta as outras

MVC



MVC

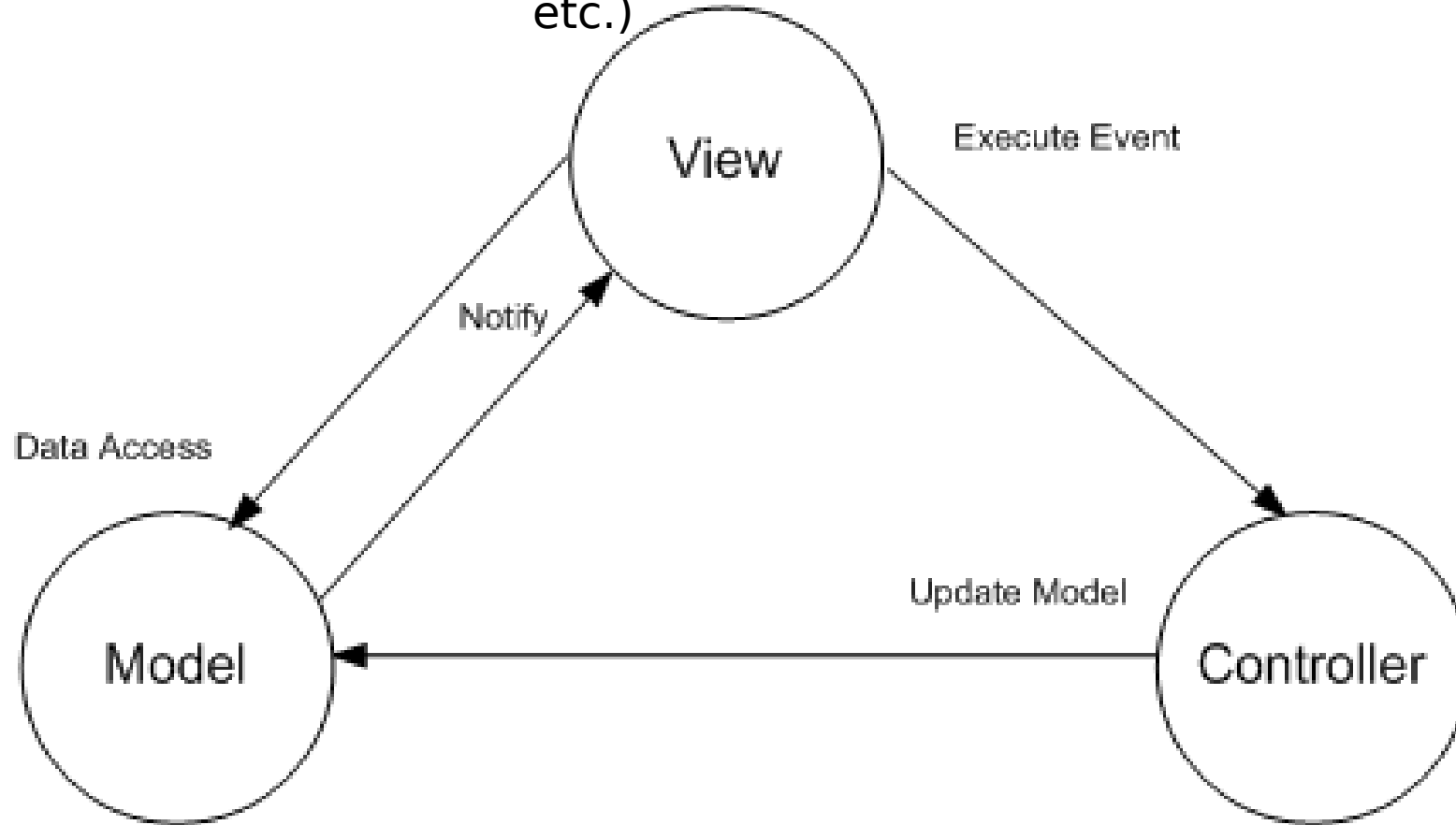
- Indica que os componentes de um sistema devem ser divididos em três camadas:
 - **Modelo:** *componentes de entidade e persistência*
 - **Visão:** *componentes de apresentação (janelas, formulários etc.)*
 - **Controle:** *componentes de processamento (processos de negócio)*

MVC

- Facilita o desenvolvimento, a manutenção e o reaproveitamento de código grandes aplicações
- Acomoda mudanças mais facilmente
 - *Alteração de interface com o usuário (visão) acontece frequentemente enquanto alterações de negócio (controle) são menos frequentes*
 - *Torna-se possível alterar apenas uma das camadas sem alterar todo o resto*

MVC

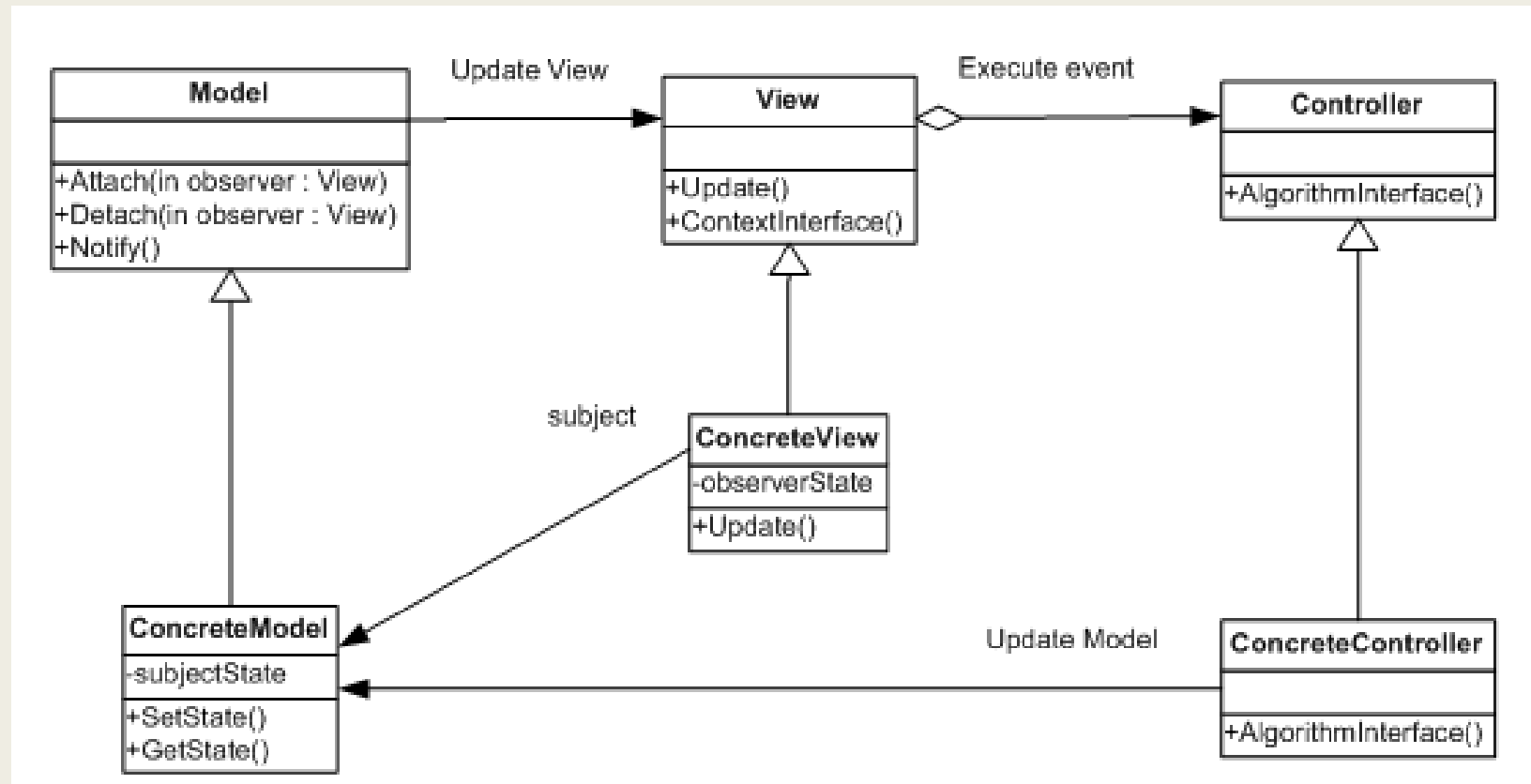
- Componentes de interação com usuário (janelas, formulários, etc.)



- Componentes de dados e persistência

- Componentes de processamento (regras de negócio)

MVC - Exemplo



Persistência

- Existem várias formas de se conectar uma aplicação com um banco de dados
- Em Java, utilizamos o JDBC (*Java Database Connectivity*)
 - *É uma API (Application Programming Interface, em português, Interface de Programação de Aplicação) para permitir a execução de consultas em qualquer BD*
 - *Conjunto de classes e interfaces parte do JSE*
- É necessário incluir um *driver* específico para ser possível efetuar uma conexão com o BD desejado

Persistência

- Para conectar a um banco de dados, recuperar e manipular informações, precisamos de pelo menos três objetos:
 - *Connection*
 - *Statement*
 - *ResultSet*

Persistência

A exceção do tipo
SQLException
precisa ser tratada!

```
Connection conexao = DriverManager.getConnection(  
    "jdbc:mysql://localhost/mydb", "usuario", "senha");
```

String de
conexão

```
System.out.println("Conectado!");
```

Persistência

```
String sql = "select * from pessoa";  
PreparedStatement pst = conexao.prepareStatement(sql);  
ResultSet rs = pst.executeQuery();  
while( rs.next() ) { //cada registro retornado  
    ... //recupera o valor de cada coluna  
}  
conexao.close();
```


Persistência – Padrão DAO (*Data Access Object*)

- É um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados
- Efetua o mapeamento objeto relacional
- Esconde todos os detalhes relativos ao armazenamento de dados do resto da aplicação
- Atua como um intermediário entre a aplicação e o banco de dados

Persistência – Padrão DAO (*Data Access Object*)

- Para cada entidade, cria-se uma classe com o nome da entidade com o final DAO para encapsular as tarefas de persistência de dados

- *Por exemplo, Cliente e ClienteDAO*

Nomenclatura não obrigatória, mas facilita a identificação

- Como utilizar?

- *ClienteDAO.adiciona(objeto_cliente);
//adicionar um cliente no BD*

Frameworks de persistência

- Um *framework* é uma abstração que une um conjunto de código para prover uma funcionalidade genérica e não é um código executável
 - *Precisa ser estendido para criar uma aplicação em si*
- Existem alguns *frameworks* para persistência de dados em Java, como EJB (Enterprise JavaBeans), JPA (Java Persistence API), Hibernate, etc.
- Um dos mais utilizados é o **Hibernate**.

Hibernate

- *Framework* de mapeamento objeto relacional (do inglês, *Object-Relational Mapping* - ORM) para aplicações Java
- *Open Source*
- Oferece suporte para:
 - *Coleções*
 - *Relacionamento entre objetos*
 - *Herança, polimorfismo e composições*
- Possui uma linguagem consulta própria
 - *HQL -Hibernate Query Language*

Benefícios

■ *Produtividade*

- *Foco no problema de negócio*

■ *Manutenção*

- *Menos linhas de código*

■ *Desempenho*

- *Permite otimizações*

■ *Independência de SGBD*

- *Portabilidade*

O Hibernate abstrai o código SQL, a camada JDBC!

Como usar

- Primeiro é necessário efetuar o download dos JARs no site do Hibernate (ou por meio dos menus do IDE)
- É necessário colocar os JARs no classpath do seu projeto
- Também é necessário baixar o driver JDBC do SGBD

Objetos

- Configuration: usado para configurar informações para inicialização do Hibernate
- Session: possibilita a comunicação entre a aplicação e a persistência
- SessionFactory: fábrica de objetos do tipo Session (específico de um BD)

Configurações do Hibernate

- Podem ser feitas via arquivo XML ou via programação
- Geralmente, utiliza-se um arquivo XML com nome hibernate.cfg.xml

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!-- Database connection settings -->
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/meubanco</property>
    <property name="connection.username">usuario</property>
    <property name="connection.password">senha</property>

  </session-factory>
</hibernate-configuration>
```

Estas são as configurações básicas, mas existem muitas outras!

HibernateUtil.java

- Geralmente criamos uma classe com um método estático para ler o arquivo de configurações e instanciar um objeto `SessionFactory`

```
public class HibernateUtil {  
    private static final SessionFactory sessionFactory;  
  
    static  
    {  
        try{  
            Configuration cfg = new Configuration();  
            cfg.configure("hibernate.cfg.xml");  
            sessionFactory = cfg.buildSessionFactory();  
  
        }catch (Throwable e){  
            System.out.println("Erro:" + e);  
            throw new ExceptionInInitializerError(e);  
        }  
    }  
  
    public static SessionFactory getSessionFactory(){  
        return sessionFactory;  
    }  
}
```

Mapeamento

■ Atributos de uma classe correspondem a colunas de uma tabela

- *Podem existir atributos na classe que não possuem mapeamento para uma coluna*

Tabela veiculo			
codigo	modelo	ano	cor



Veiculo	
-	codigo : Integer
-	modelo: String
-	ano: integer
-	cor: String

Anotações

- É um recurso usado para anotar classes, atributos e métodos, de tal maneira que essas marcações (precedidas por @) podem ser tratadas pelo compilador, ferramentas de desenvolvimento e bibliotecas
- @Entity: indica que objetos dessa classe se tornem "persistível" no banco de dados
- @Id: indica que o atributo é a chave primária
- @GeneratedValue: indica que o atributo é AUTO INCREMENT

Anotações

- Por padrão, o nome da classe é o nome da tabela e o nome de cada atributo é o nome de cada coluna da tabela
- Para definir nomes diferentes, basta utilizar outras anotações:

@Entity

@Table(name="tarefas")

```
public class Tarefa { }
```

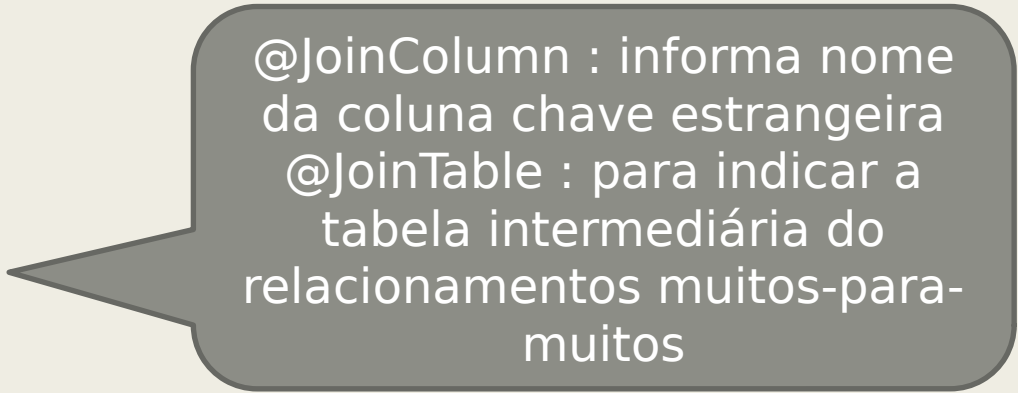
@Column(name = "nomefantasia", nullable = true)

```
private String nome;
```

Ao incluir um mapeamento para uma entidade, deve-se adicionar uma configuração no XML para que o Hibernate a reconheça:
`<mapping class="nomedaclassa" />`

Anotações

- Mapeamento de atributos não persistentes
 - *@Transient*
- Mapeamento de atributos do tipo de data e/ou hora
 - *@Temporal*
- Relacionamentos
 - *@OneToMany*
 - *@ManyToOne*
 - *@ManyToMany*
 - *@OneToOne*



@JoinColumn : informa nome da coluna chave estrangeira
@JoinTable : para indicar a tabela intermediária do relacionamentos muitos-para-muitos

Alguns métodos

- A classe Session possui assinaturas de métodos para fazer operações padrão no BD, como:
 - ***save(objeto)*** : salvar no BD (insert)
 - ***saveOrUpdate(objeto)*** : salva ou atualiza um registro (insert ou update)
 - ***get(classe, id)*** : busca um objeto pela chave primária (select)
 - ***update(objeto)*** : atualiza um registro no BD (update)
 - ***delete(objeto)*** : exclui um registro do BD (delete)
 - ***createQuery("")*** : crie sua própria consulta
 - ***list()*** retorna uma lista de resultados da consulta

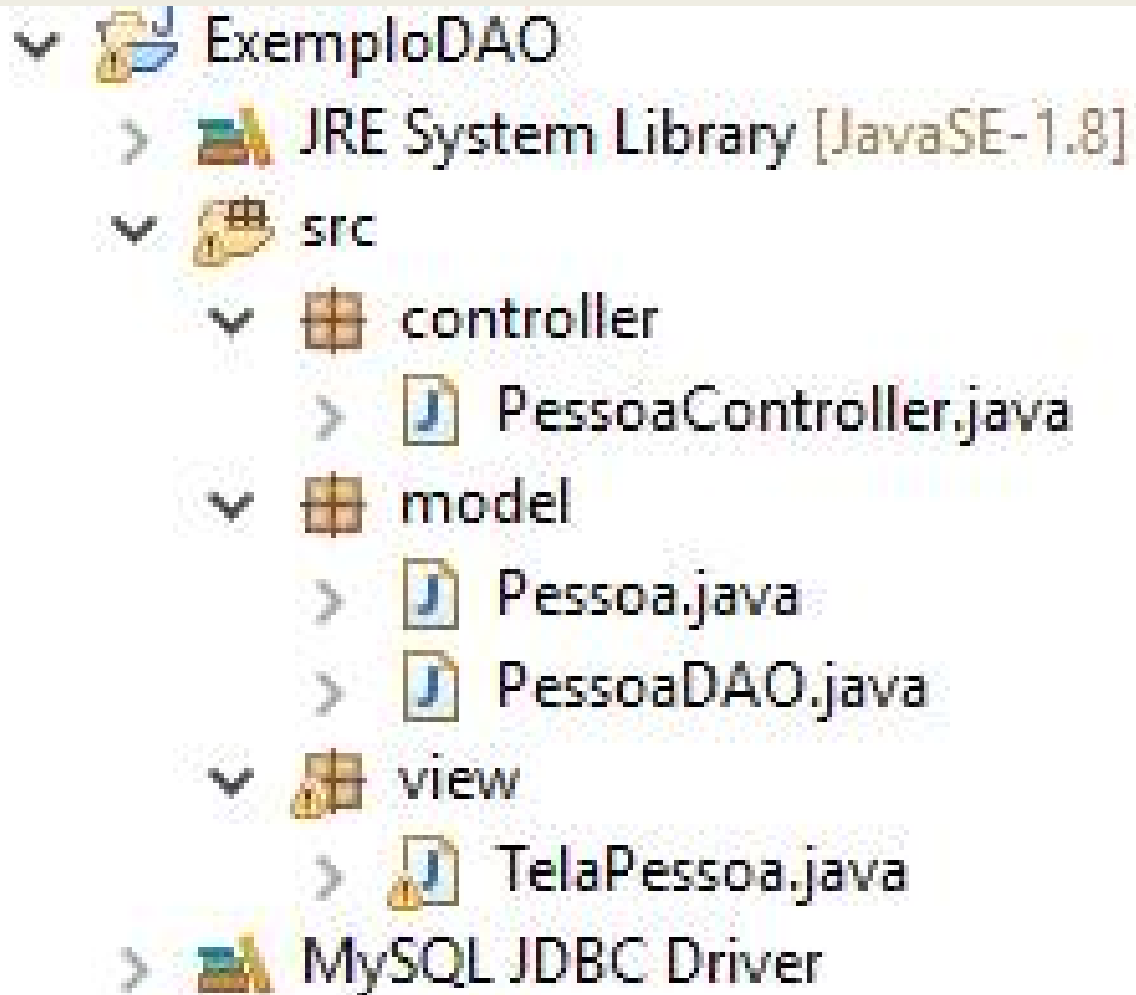
Tutorial para estudar e praticar

https://netbeans.apache.org/kb/docs/java/hibernate-java-se_pt_BR.html

Exercício

- Criar o banco “meubanco” com a tabela “pessoa”.
- Criar um projeto com 3 pacotes chamados model, view e controller.
- Criar as classes para salvar e recuperar dados da tabela de pessoas.

Exercício



Referências

- DEITEL, H. M. Java: como programar, 8. ed. São Paulo: Prentice Hall, 2010. Capítulo 24.
- BAUER, C.; KING, G. Java Persistence with Hibernate. Revised edition, New York: Manning, 2007.