

FIAP

FIAP

# Corporate – Tokio Marine

PROGRAMAÇÃO ORIENTADA OBJETOS O.O

**Prof. Dr. Emerson R. Abraham**

# **| Agenda**

- **Revisando:**
  - ✓ **Conexão com banco de dados**
  - ✓ **Configuração de driver**
  - ✓ **Padrões Factory e DAO**
- **Padrão MVC**
- **Relacionamentos**

- Connection Factory é um padrão de criação do tipo FACTORY, utilizado para estabelecer conexão com bancos de dados, através de String de conexão (tipo de banco de dados, endereço do banco, schema, usuário e senha) implementada no método estático getConnection da classe DriverManager.

```
return DriverManager.getConnection  
(jdbc:oracle:thin:@oracle.fiap.com.br:1521:ORCL,"root","root");
```

- “O Factory Method define uma interface para criar um objeto, mais deixa que as subclasses decidam qual classe instanciar.
- O método getConnection é uma fábrica de conexões, isto é, ele cria novas conexões para nós.
- Basta invocar o método e recebemos uma conexão pronta para uso, não importando de onde elas vieram e eventuais detalhes de criação.

```
public class ConnectionFactory {  
    public Connection conectar() {  
        try {  
            return DriverManager.getConnection("jdbc:oracle:thin:@oracle.fiap.com.br:1521:ORCL", "usuario", "senha");  
        } catch (SQLException e) {  
            System.out.println("Erro ao conectar");  
            throw new RuntimeException(e);  
        }  
    }  
}
```

## JAVABEANS

- JavaBeans são classes que possuem atributos, construtor (opcional) e métodos de acesso getters e setters.
- Podem ser definidos como sendo componentes de software que permitem a geração de partes reutilizáveis.
- Seu objetivo é deixar o programador trabalhar mais nas regras de negócios.
- Cada atributo da classe representa um campo na tabela de dados e cada objeto um novo registro.
- A classe Usuario é um exemplo de JavaBeans, com variáveis e métodos get e set

## JAVABEANS

```
1 package model;
2
3 import java.sql.Date;
4
5 public class Usuario {
6
7     private long id;
8     private String nome;
9     private String email;
10    private int telefone;
11    private Date data;
12
13
14
15
16    public Usuario() {
17
18    }
19
20
21    //getters / setters
22
```



## DAO (DATA ACCESS OBJECT)

- Data Access Object é um Design Pattern, utilizado para persistência em banco de dados.
- O DAO busca dados no banco e transforma em objetos e vice versa, mantendo uniformidade no código e evitando que o acesso e manipulação aos dados fique espalhado por toda aplicação.
- A classe UsuarioDAO implementa em seu construtor um objeto do tipo Connection Factory que será utilizado junto do método PreparedStatement para execução de queries no banco de dados utilizado.

## DAO (DATA ACCESS OBJECT)

```
1 package repository;
2
3+ import java.sql.Connection;
12
13 public class UsuarioDAO {
14
15     private Connection conexao;
16
17- public UsuarioDAO() {
18         this.conexao = new ConnectionFactory().conectar();
19     }
20
21- public void insert(Usuario usuario) throws SQLException {
22         String sql = "insert into usuarios(id, nome, email, telefone, data) values (?, ?, ?, ?, ?)";
23         PreparedStatement stmt = conexao.prepareStatement(sql);
24
25         stmt.setLong(1, usuario.getId());
26         stmt.setString(2, usuario.getNome());
27         stmt.setString(3, usuario.getEmail());
28         stmt.setInt(4, usuario.getTelefone());
29         stmt.setDate(5, usuario.getData());
30
31         stmt.execute();
32         stmt.close();
33     }
34
```

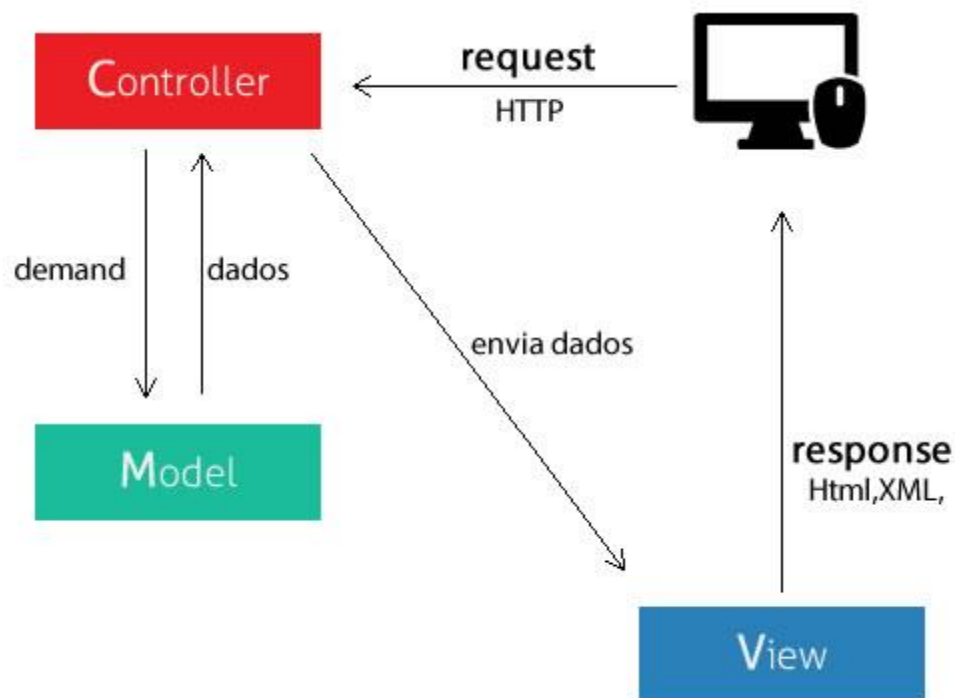
- 
- 
-

## Model View Control MVC

- **Model–view–controller (MVC)** é um padrão de arquitetura que divide a aplicação em três partes conectadas. A interação dos usuários (front-end) são separadas dos métodos que interagem com o banco de dados (back-end).
- **Model** – camada que representa as entidades no banco de dados. Armazena dados e notifica os **controllers** associados quando há uma mudança em seu estado
- **View** - camada de visualização e interação (interface com o usuário)
- **Controller** - envia comandos para o **model** para atualizar o seu estado, recebe as notificações e envia para **view** associada.

Fonte: <https://en.wikipedia.org/wiki/Model-view-controller>

## Model View Control MVC



Extraído de : <https://tableless.com.br/mvc-afinal-e-o-que/>

## Relacionamentos

- Quando temos mais tabelas associadas, temos um relacionamento entre elas. Vejamos o exemplo de parte de um sistema de contratação de funcionários.

Candidato						
id_candidato	nome	data_nasc	genero	tempo_experiencia	...	id_area

Area de atuação		
id_area	nome	List<Candidato>

- A tabela candidato possui uma chave estrangeira para se relacionar com a tabela área de atuação, por meio da chave primaria desta.
- As classes recebem os atributos convencionais, os atributos para as chaves primárias e estrangeiras e uma variável (apenas na classe) List<Candidato> que guarda os candidatos relacionados a determinada área de atuação.

# I Dúvidas ou curiosidades



















Extraído de: <https://www.latrobe.edu.au>

## **Exercícios**

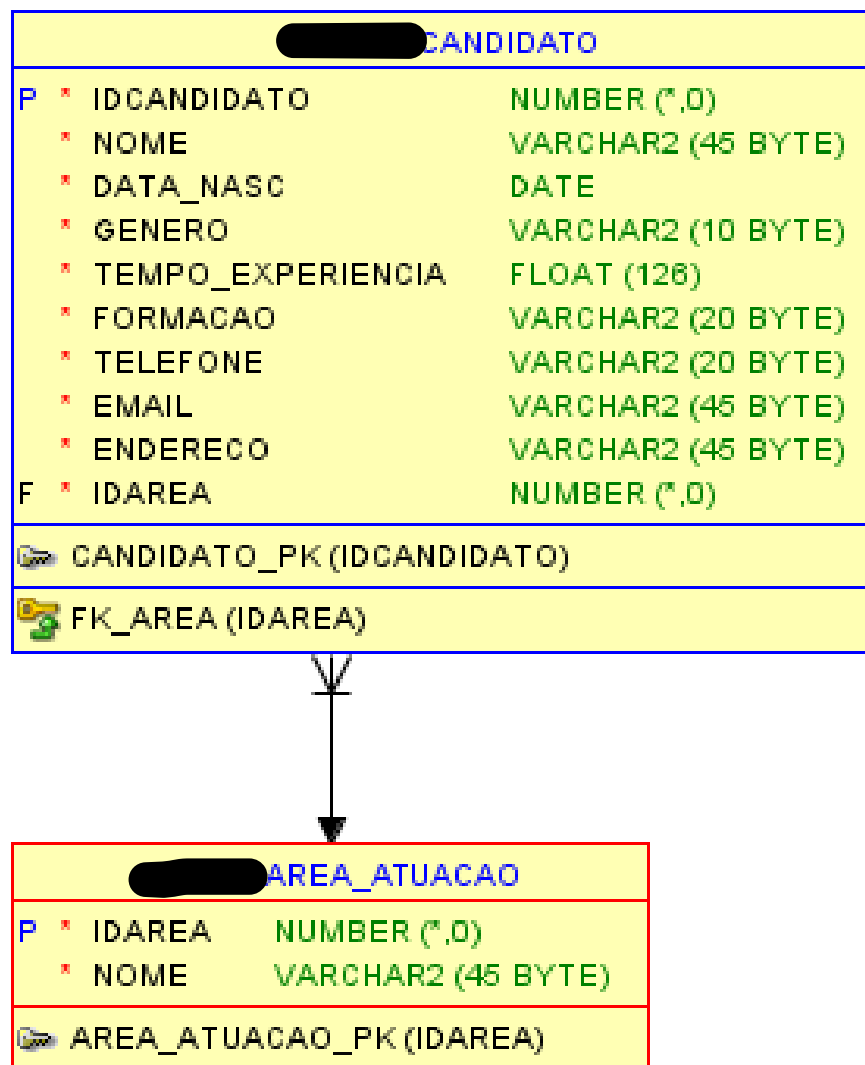
- **Crie um novo projeto denominado HRSystem**
- **Separe em camadas adotando o padrão MVC**
- **Elabore a classe de conexão**
- **Elabore as classes model**



## Exercícios

- ▼  src
  - ▼  br.com.fiap.jdbc.controller
    - >  AreaAtuacaoController.java
    - >  CandidatoController.java
  - ▼  br.com.fiap.jdbc.dao
    - >  AreaAtuacaoDAO.java
    - >  CandidatoDAO.java
  - ▼  br.com.fiap.jdbc.factory
    - >  ConnectionFactory.java
  - ▼  br.com.fiap.jdbc.model
    - >  AreaAtuacao.java
    - >  Candidato.java
  - ▼  br.com.fiap.jdbc.teste
    - >  Teste.java

## Exercícios



## Exercícios – Implemente as classes Controller

```
1 package br.com.fiap.jdbc.controller;
2
3 import java.sql.Connection;
4
5
6
7
8
9
10 public class CandidatoController {
11
12     private CandidatoDAO candidatoDAO;
13
14     public CandidatoController() {
15         Connection connection = new ConnectionFactory().conectarOracle();
16         this.candidatoDAO = new CandidatoDAO(connection);
17     }
18
19     public void deletar(Integer idCandidato) {
20         this.candidatoDAO.deletar(idCandidato);
21     }
22
23     public void salvarComCategoria(Candidato candidato) {
24         this.candidatoDAO.salvarComArea(candidato);
25     }
26
27     public List<Candidato> listar() {
28         return this.candidatoDAO.listar();
29     }
30
31     public void alterar(float tempoExperiencia, String formacao, String telefone, String email, String endereco,
32                         int idCandidato) {
33         this.candidatoDAO.alterar(tempoExperiencia, formacao, telefone, email, endereco, idCandidato);
34     }
35 }
36
```

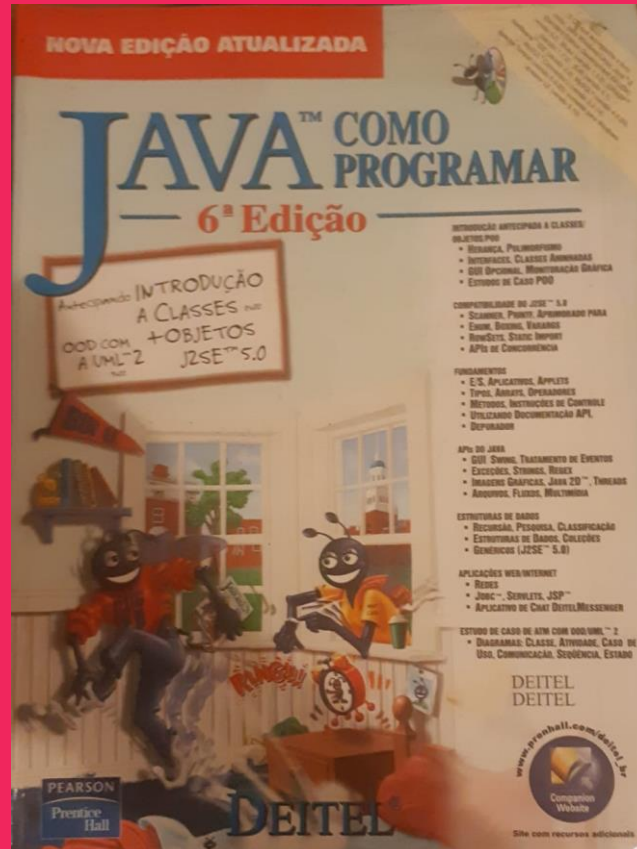
## Exercícios – Implemente as classes Controller

```
1 package br.com.fiap.jdbc.controller;
2
3 import java.sql.Connection;
4
5
6
7
8
9
10
11 public class AreaAtuacaoController {
12
13     private AreaAtuacaoDAO areaDAO;
14
15     public AreaAtuacaoController() {
16         Connection connection = new ConnectionFactory().conectarOracle();
17         areaDAO = new AreaAtuacaoDAO(connection);
18     }
19
20     public List<AreaAtuacao> listarComDAO() throws SQLException {
21         return this.areaDAO.listar();
22     }
23 }
24
```

## **| Exercícios**

- **Implemente as classes Data Access Object (DAO)**
- **Faça uma classe de testes e realize as simulações**
- **Desafio: Implemente um pacote view e elabore telas para servir de interface com o usuário.**

## Referências:



Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).