POO2

Encontro 04 – ORM, POJO e JPA

Professor Luiz Augusto Rodrigues

luiz.a.rodrigues@cogna.com.br

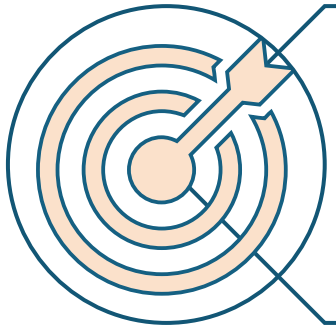
Sumário

- Conceito de ORM
- Conceitos de JPA
- Usando o H2 Database
- Referências

Informações Importantes

Nenhum material será enviado via e-mail. Os materiais serão disponibilizados no **AVA** e no Google Drive da Disciplina.

Dúvidas, questionamentos, entre outros deverão ser realizados pelo **e-mail** e pelo grupo de **Whatsapp** da disciplina.



A Disciplina de Programação Orientada a Objetos, a partir deste slide, será referenciada pela sigla **POO2**.

Informações Importantes

Grupo de Whatsapp da Disciplina:

https://tinyurl.com/mw2xdkd7

POO2

Instruções Importantíssimas

Instruções Importantíssimas

Para este semestre, na disciplina de POO, utilizaremos OBRIGATORIAMENTE as seguintes ferramentas:

- Sistema Operacional Windows 10, Linux ou MacOS.
- Linguagem de Programação Java.
- JDK 17 ou superior
  - https://download.oracle.com/java/17/archive/jdk-17.0.6_windows-x64_bin.exe
- VS Code.

Instruções Importantíssimas

- Git 2.39.2 ou superior
  - https://github.com/git-for-windows/git/releases/download/v2.39.2.windows.1/Git-2.39.2-64-bit.exe
- Github
  - https://github.com/
  - https://desktop.github.com/
- Tutorial em Git e Github
  - Tutorial Git e Github 2022 – Introdução prática para iniciantes
  - Canal DevSuperior
  - https://youtu.be/_hZf1teRFNg

Quem for utilizar Linux ou MacOSx, fale comigo após a aula, ou pelo grupo do WhatsApp, para instruções de instalação.

Instruções Importantíssimas

- Os relatórios não serão utilizados nessa disciplina, por se tratar de uma disciplina 90% prática.
- No lugar, teremos Listas de Atividades Formativas (Atividade do Professor), que deverão ser respondidas e justificadas, para fixação do conteúdo discutido nas aulas.
- As atividades de aula estarão disponíveis Google Drive da turma.
- O tutorial do Git e Github é obrigatório.

POO2

ORM, POJO e JPA

Conceito de ORM

Conceitos de ORM

Mapeamento objeto-relacional (*Object-Relational Mapping*) é uma técnica de desenvolvimento utilizada para reduzir a impedância da programação orientada aos objetos utilizando bancos de dados relacionais.

As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes.

Com esta técnica, o programador não precisa se preocupar com os comandos em linguagem SQL; ele irá usar uma interface de programação simples que faz todo o trabalho de persistência.

Conceitos de ORM

Não é necessária uma correspondência direta entre as tabelas de dados e as classes do programa.

A relação entre as tabelas onde originam os dados e o objeto que os disponibiliza é configurada pelo programador, isolando o código do programa das alterações à organização dos dados nas tabelas do banco de dados.

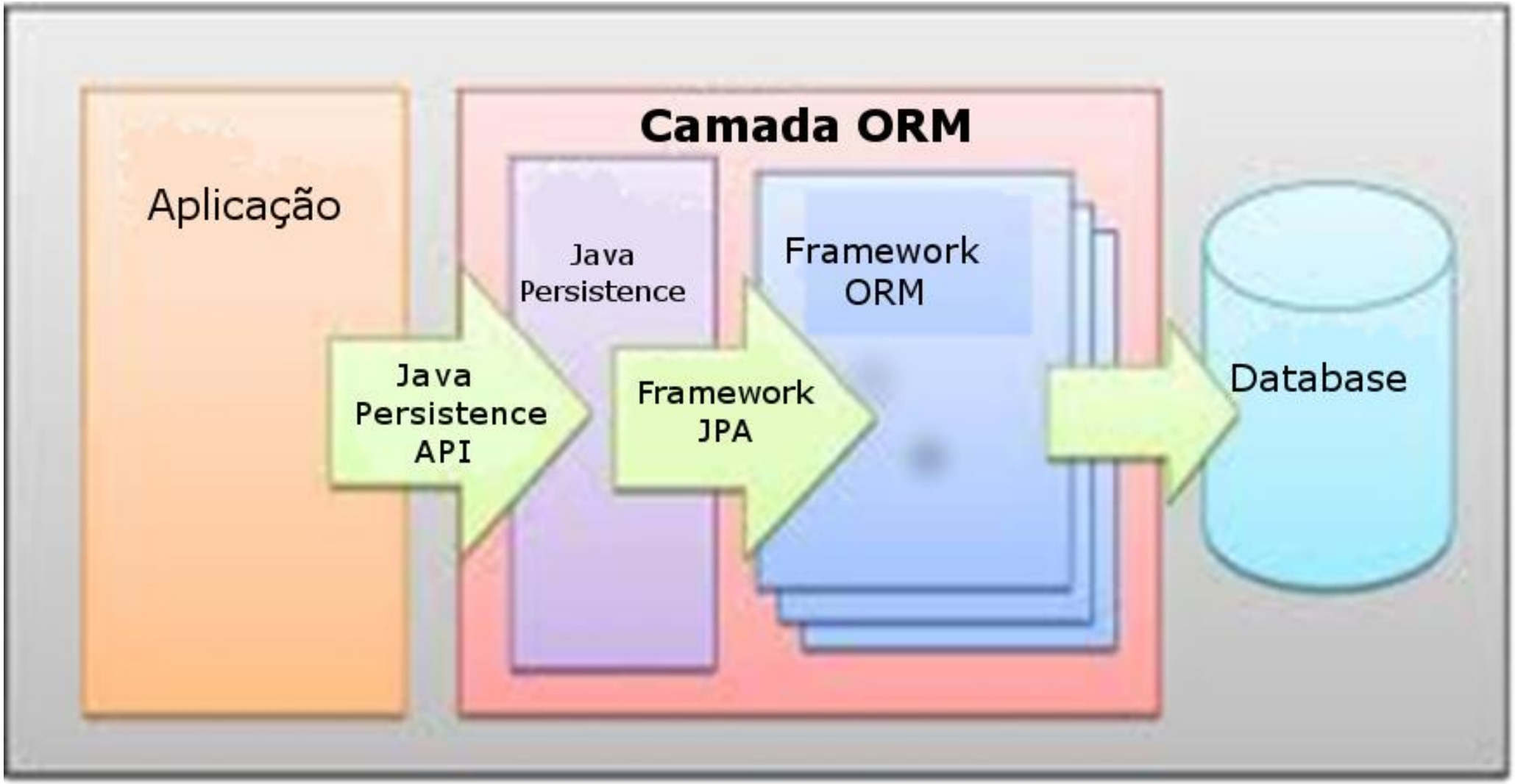
A forma como este mapeamento é configurado depende da ferramenta que estamos a usar.

Conceitos de ORM

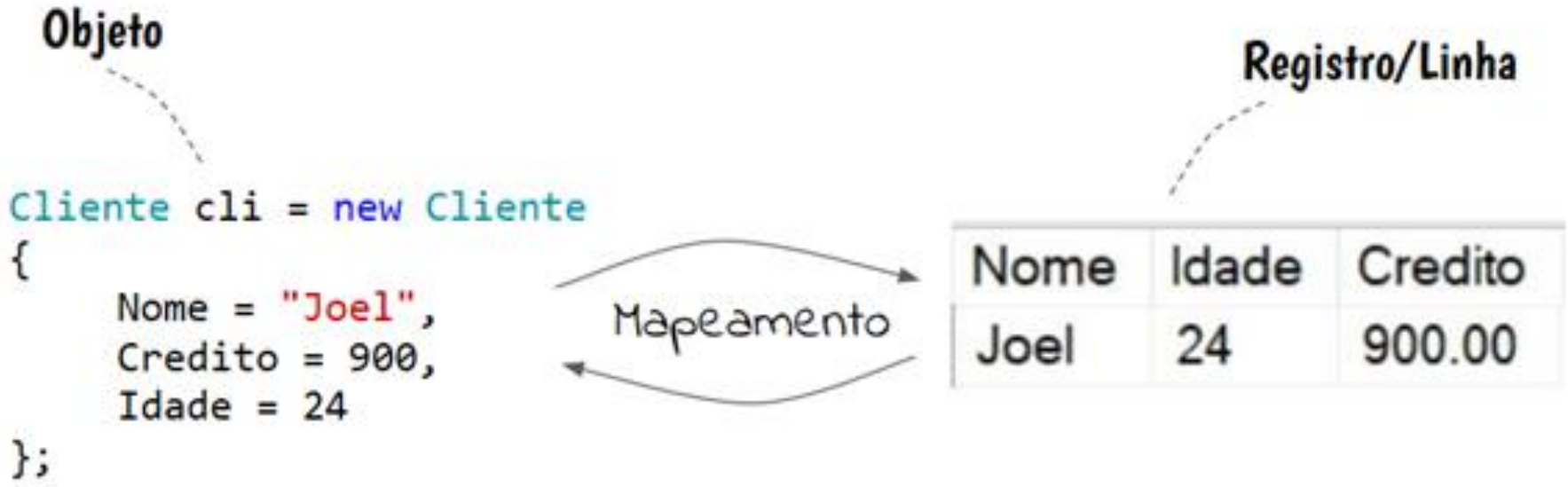
No mercado existem centenas de opções conhecidas de ORM para os mais diversos bancos de dados e linguagem de programação, sendo os mais conhecidos o

- Hibernate/NHibernate , para Java e .NET respectivamente;
- O Dapper, micro-ORM open-source,criado e utilizado pelo pessoal do Stackoverflow;
- O Entity Framework, também chamado de EF, para o MS .NET e .NET Core.

Conceitos de ORM



Conceitos de ORM



POO2

ORM, POJO e JPA

Conceitos de POJO

Conceitos de POJO

Plain Old Java Object ou POJO, são objetos Java, ou simplesmente **Javabeans**, que seguem um desenho extremamente simplificado, ou até mesmo anêmico, que possuem uma construção simples, sem qualquer ligação ou associação com outras classes.

JavaBeans são quaisquer componentes de software escritos na linguagem de programação Java, que utilização a notação padrão da linguagem.

Segundo a especificação da **Sun Microsystems** (oficialmente a criadora da linguagem, na década de 1990), os JavaBeans são "componentes reutilizáveis de software que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento".

Conceitos de POJO

Os objetos POJO seguem definições rígidas de estrutura, sendo elas:

- Possui um construtor default (padrão - sem argumentos) ou não declaram construtor (assim o compilador Java criará um construtor default automaticamente).
- Possui todos os seus atributos, sejam eles tipos primitivos ou objetos, com a visibilidade privada.
- Não possui métodos específicos, exceto aqueles que seguem o padrão de ***getter*** e ***setter*** para seus respectivos atributos.

Conceitos de POJO

Este padrão é baseado na idéia de que quanto mais simples o projeto, melhor.

O termo foi inventado por Martin Fowler, Rebecca Parsons e Josh MacKenzie em Setembro de 2000.

Segundo os autores:

> *"Nós queríamos saber por que as pessoas eram contra o uso de objetos regulares em seus sistemas e concluímos que era devido à falta de um nome extravagante para eles. Assim nós demos-lhes um, e funcionou muito bem."*

O termo segue o padrão de atribuir um nome para tecnologias que não possuem nenhuma característica nova.

Conceitos de POJO

O termo ganhou aceitação por causa da necessidade de um termo comum e facilmente inteligível que contrasta com os complicados frameworks de objetos.

É mais atrativo do que o termo ***bean*** do Java devido à confusão gerada pela semelhança dos termos ***JavaBeans*** e dos ***EJB*** (Enterprise JavaBeans).

POO2

ORM, POJO e JPA

Conceitos de JPA

Conceitos de JPA

A **Java Persistence API**, mais conhecida por sua sigla **JPA**, é uma especificação que se preocupa com a persistência, o que significa vagamente qualquer mecanismo pelo qual os objetos Java sobrevivam ao processo do aplicativo que os criou. Nem todos os objetos Java precisam ser persistidos, mas a maioria dos aplicativos persiste os principais objetos de negócios.

JPA é um framework leve, baseado em POJO's (Plain Old Java Objects) para persistir objetos Java. Diferente do que muitos imaginam, JPA não é apenas um framework para Mapeamento Objeto-Relacional (ORM - Object-Relational Mapping).

Também oferece diversas funcionalidades essenciais em qualquer aplicação corporativa.

Conceitos de JPA

Na atualidade, praticamente todas as aplicações de grande porte utilizam JPA para persistir objetos Java, provendo diversas funcionalidades para os programadores.

A versão atual da JPA nasceu através das necessidades dos profissionais da área e das soluções proprietárias que já existiam para resolver os problemas com persistência.

Com a ajuda dos desenvolvedores e de profissionais experientes que criaram outras ferramentas de persistência, chegou a uma versão muito melhor que é a que os desenvolvedores Java conhecem atualmente.

Histórico de JPA

No momento em que a primeira versão do JPA foi iniciada (JPA 1.0), outros modelos de persistência ORM já haviam evoluído, implementando diversas características as quais os programadores já haviam requisitado.

Mesmo assim muitas características foram adicionadas nesta versão, em releases adicionais, e outras foram deixadas para uma próxima versão.

A versão JPA 2.0 incluiu um grande número de características que não estavam na primeira versão, especialmente as mais requisitadas pelos usuários, entre elas:

- A capacidade adicional de mapeamento;
- Expansões para a Java Persistence Query Language (JPQL);
- *API Criteria*, para criação de consultas dinâmicas, entre outras características.

Características de JPA

POJOS Persistentes

Talvez o aspecto mais importante da JPA seja o fato que os objetos são POJO's, significando que os objetos possuem design simples que não dependem da herança de interfaces ou classes de frameworks externos.

Qualquer objeto com um construtor default pode ser feito persistente sem nenhuma alteração numa linha de código.

Mapeamento Objeto-Relacional com JPA é inteiramente dirigido a metadados. Isto pode ser feito através de anotações no código ou através de um XML definido externamente.

Características de JPA

Consultas em Objetos

As consultas podem ser realizadas através da **Java Persistence Query Language** (JPQL), uma linguagem de consulta que é derivada do EJB QL e transformada depois para SQL.

As consultas usam um esquema abstraído que é baseado no modelo de entidade como oposto às colunas na qual a entidade é armazenada.

Características de JPA

Integração e Teste

Atualmente as aplicações normalmente rodam num Servidor de aplicação, sendo um padrão do mercado hoje.

Testes em servidores de aplicação são um grande desafio e normalmente impraticáveis. Efetuar teste de unidade e teste caixa branca em servidores de aplicação não é uma tarefa tão trivial.

Porém, isto é resolvido com uma API que trabalha fora do servidor de aplicação. Isto permite que a JPA possa ser utilizada sem a existência de um servidor de aplicação.

Dessa forma, testes unitários podem ser executados mais facilmente.

Componentes de JPA

EntityManagerFactory

Interface usada para interagir com a fábrica do gerenciador de entidades para a unidade de persistência.

Quando o aplicativo terminar de usar a fábrica do gerenciador de entidades e/ou no encerramento do aplicativo, o aplicativo deve fechar a fábrica do gerenciador de entidades.

Depois que um EntityManagerFactory é fechado, todos os seus gerenciadores de entidade são considerados no estado fechado.

Componentes de JPA

EntityManager

Interface usada para interagir com o contexto de persistência.

Uma instância de EntityManager está associada a um contexto de persistência.

Um contexto de persistência é um conjunto de instâncias de entidade em que, para qualquer identidade de entidade persistente, existe uma instância de entidade única. Dentro do contexto de persistência, as instâncias de entidade e seu ciclo de vida são gerenciados.

A API EntityManager é usada para criar e remover instâncias de entidade persistentes, para localizar entidades por sua chave primária e para consultar entidades.

Componentes de JPA

EntityManager

O conjunto de entidades que podem ser gerenciadas por uma determinada instância EntityManager é definido por uma unidade de persistência. Uma unidade de persistência define o conjunto de todas as classes relacionadas ou agrupadas pelo aplicativo e que devem ser colocadas em seu mapeamento em um único banco de dados.

Relembrando

Annotations

Anotações, usando a tradução, são atributos personalizados que os usuários (os programadores que usam Java) podem criar para usar em situações que eles podem ser úteis para dar mais informações ao código que não seja possível de outra forma, e elas carregam informações que podem usar usadas por ferramentas ou mesmo dentro do código.

Exemplos que já utilizamos:

@RestController

@Service

@MappedSuperClass

Anotações Importantes de JPA

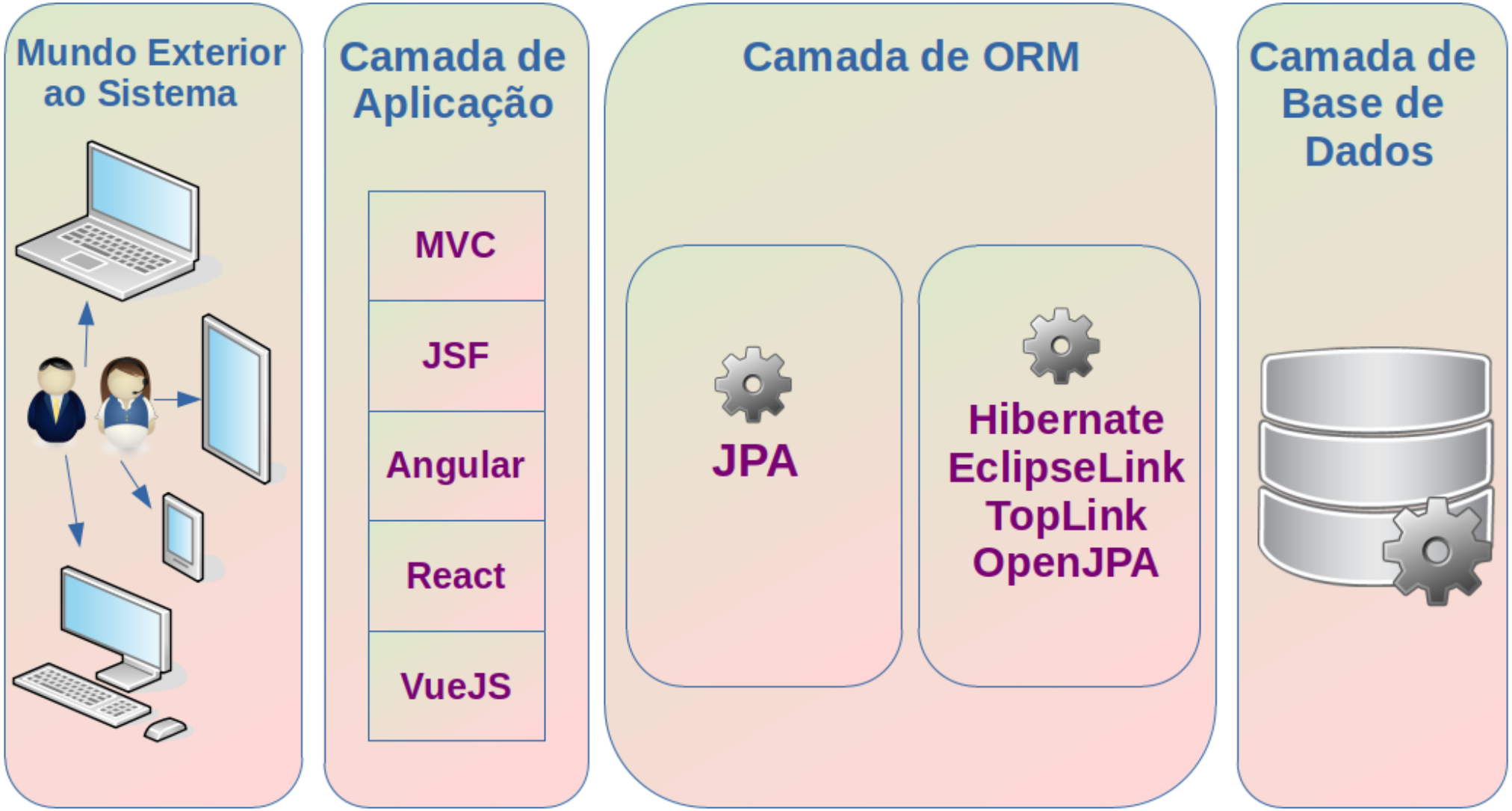
@Entity

Serve para a API JPA saber que a classe anotada com @Entity corresponde a uma tabela da base de dados. Uma entidade corresponde a uma tabela.

@Table

É usada em conjunto com a anotação @Entity, e serve para a API JPA saber o nome da tabela, em que a classe anotada com @Table, tem um atributo “name”, que contém o nome da tabela.

Estrutura do JPA



Exemplo de JPA

Considere o script a seguir.

```
CREATE TABLE Pessoa
(
    codigo NOT NULL IDENTITY(1,1) PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    dataNascimento DATETIME NOT NULL,
    email VARCHAR(MAX)
)
```

Exemplo de JPA

Em uma classe de domínio, ou uma classe POJO, teríamos:

```
@Entity
public class Pessoa {
	private Long codigo;
	private String nome;
	private Date dataNascimento;
	private String email;
	//Métodos Gets e Sets
}
```

Exemplo de JPA

```
@Entity
@Table(name = "Pessoa")
public class Pessoa {
	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long codigo;
	@Column(nullable = false)
	private String nome;
	@Temporal(TemporalType.DATE)
	@Column(name = "dataNascimento", nullable = false)
	private Date dataNascimento;
	private String email;
	//Métodos Gets e Sets...
}
```

POO2

ORM, POJO e JPA

Usando o H2 Database

Conceitos do H2 Database

H2 Database é um sistema de gerenciamento de banco de dados relacional (RDBMS) escrito em Java.

Ele é conhecido por ser um banco de dados leve, embutido e de código aberto, adequado para aplicações pequenas e para uso em desenvolvimento e testes.

O H2 é um banco de dados que pode ser usado em modo embutido (dentro da mesma JVM da aplicação) ou em modo servidor (como um banco de dados acessível pela rede).

Utilização

Desenvolvimento e Testes:

H2 é frequentemente utilizado durante o desenvolvimento e testes de aplicações devido à sua facilidade de configuração e ao fato de ser leve.

Ele permite criar e testar bancos de dados sem a necessidade de instalar e configurar um servidor de banco de dados mais complexo.

Aplicações Leves:

Ideal para aplicações com requisitos de banco de dados que não necessitam de um RDBMS mais robusto. Pode ser incorporado diretamente na aplicação, evitando a necessidade de uma instalação separada.

Utilização

Prototipagem:

Usado para criar protótipos rápidos de banco de dados e validar o design de esquemas e consultas antes de migrar para um banco de dados mais pesado.

Características

Leve e Rápido:

Ideal para aplicações pequenas e desenvolvimento, com baixo consumo de memória e tempo de inicialização rápido.

In-Memory e File-Based:

Suporta bancos de dados em memória (voláteis) e em arquivos (persistentes).

Suporte a SQL:

Suporta a maioria dos recursos SQL padrão, facilitando a portabilidade e a escrita de consultas.

Interface Web:

Oferece uma interface web para administração e visualização de dados.

Considerações

O H2 é ideal para desenvolvimento e testes, mas pode não ser adequado para aplicações de produção de grande escala devido à sua natureza leve e suas limitações em comparação com bancos de dados mais robustos como PostgreSQL, MySQL ou Oracle.

Ao usar H2 em produção, é importante considerar as características específicas do ambiente de produção e a capacidade de suportar as necessidades de desempenho e escalabilidade da aplicação.

Configurações

Dependências e Configuração Inicial:

Para usar o H2 em um projeto Java, adicione a dependência do H2 no seu gerenciador de dependências. No caso, utilizamos o Maven.

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.1.214</version>
    <scope>runtime</scope>
</dependency>
```

* Já temos configurado em nosso projeto.

Configurações

Para importar dados para as tabelas criadas automaticamente, crie um arquivo denominado import.sql, na pasta resources, contendo o script de inserção na tabela ClasseProduto.

Os scripts a seguir podem ser encontrados no Google Drive da turma.

Na pasta Materiais/Configs:

application.properties

import.sql

Configurações

Configuração no Spring Boot:

Para integrar o H2 com uma aplicação Spring Boot, para uso em modo Embutido, podemos usar a dependência do H2 e configurar as propriedades de conexão no arquivo ***application.properties***:

```
# Configs
server.port=8080
server.servlet.session.timeout=30s

# H2 Connection
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=Senha123
```

Configurações

Configuração no Spring Boot:

```
# H2 Client
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# Show SQL
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Configurações

Já no Domínio, altere a classe BaseIdentificador, com as seguintes notações:

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
protected Long codigo;

@DateTimeFormat(pattern="yyy-MM-dd-HH.mm.ss")
protected LocalDate dataDeInclusao;

@DateTimeFormat(pattern="yyy-MM-dd-HH.mm.ss")
protected LocalDate dataDeAlteracao;
```

POO2

ORM, POJO e JPA

Referências

Referências

H2 Database

https://www.h2database.com/html/main.html

Java JPA Reference

https://www.objectdb.com/api/java/jpa

Spring Data JPA Reference

https://docs.spring.io/spring-data/jpa/reference/jpa.html

Dúvidas?

Dúvidas, críticas e sugestões, entre em contato através do e-mail do professor:

luiz.a.rodrigues@cogna.com.br