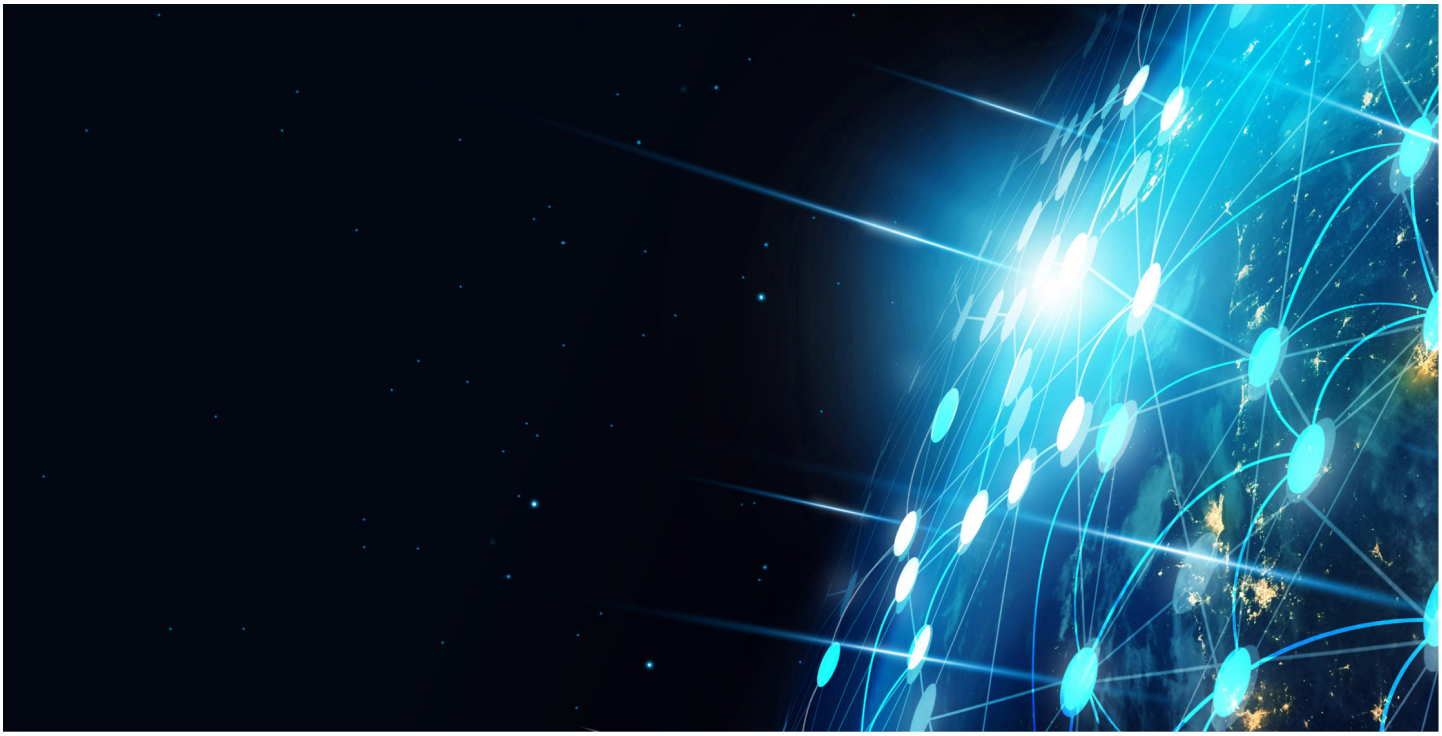


Camada de modelo – mapeamento objeto-relacional

Prof. Sérgio Assunção Monteiro

Descrição

Apresentação dos aspectos essenciais da camada de modelo da arquitetura MVC para o mapeamento objeto-relacional.

Propósito

Para padronizar o desenvolvimento de sistemas, foi criada a arquitetura MVC, um acrônimo para modelo-visão-controle. Em especial, a camada de modelo trata de aspectos essenciais sobre o acesso e persistência de dados que, sem dúvidas, é um conhecimento essencial para profissionais de tecnologia que pretendem desenvolver sistemas mais eficientes.

Preparação

Para rodar os exemplos, você vai precisar dos seguintes programas:

- JSDK versão 10 ou superior;

- Maven 3.8.4;
- Eclipse.

Os códigos fontes aqui implementados encontram-se no arquivo **projeto**.

Objetivos

Módulo 1

Criação das classes de modelo, suas associações e multiplicidades

Reconhecer a criação das classes de modelo, suas associações e multiplicidades.

Módulo 2

Aplicação das anotações JPA com Hibernate

Aplicar as anotações JPA com Hibernate.

Módulo 3

Uso do banco de dados H2 como servidor

Identificar o uso de repositórios com o framework Spring Data JPA e o banco de dados H2.

Módulo 4

Consultas personalizadas com HQL

Implementar consultas personalizadas com HQL.

Introdução

Os sistemas computacionais fazem parte da nossa vida cotidiana. Como consequência, temos um efeito de profundas mudanças sobre as nossas interações sociais, pois usamos aplicativos para realizar diversos serviços on-line que, até pouco tempo, necessitavam de nossa presença.

Parte da explicação do porquê os sistemas fizeram tanto sucesso se deve à economia de recursos financeiros e de tempo que eles proporcionam. Mas, principalmente, porque eles se tornaram mais fáceis de serem usados. Porém, a fim de proporcionar essa comodidade para os usuários, foram criadas arquiteturas de desenvolvimento que compartimentalizaram os elementos do software, reduzindo, assim, erros, tempo de entrega e que, além disso, permitiram a construção mais eficiente de cada componente do sistema. A arquitetura de maior sucesso com esses objetivos foi a MVC, que separa os elementos estruturais de um software nas camadas de Modelo, Visão e Controle.

Aqui, vamos estudar a camada de controle utilizando o framework Hibernate para realizar operações no banco de dados.

1 - Criação das classes de modelo, suas associações e multiplicidades

Ao final deste módulo, você será capaz de reconhecer a criação das classes de modelo, suas associações e multiplicidades.

Criação de classes da camada de modelo

Aspectos básicos da arquitetura MVC

Muitas aplicações modernas interagem com sistemas gerenciadores de banco de dados (SGBD), realizam processamentos desses dados e exibem resultados em gráficos e relatórios. Então, podemos distinguir elementos estruturais distintos que não devem ser tratados da mesma forma.

> **Vamos pensar em um cenário ruim: precisamos construir uma aplicação que obtenha dados financeiros em um banco de dados Oracle.**

Em seguida, esses dados precisam passar pelo processamento de regras de negócio relacionadas a pagamentos de fornecedores, prestadores de serviço e recolhimento de impostos. Por fim, a aplicação precisa exibir os resultados dos processamentos em relatórios dinâmicos com diferentes níveis de detalhamento para as áreas operacionais e gerenciais da empresa.

No cenário que descrevemos, faria sentindo que o código que trata do acesso aos dados estivesse misturado com o código que trata da visualização dos dados? Bem, sabemos que a resposta é não. Isso ocorre porque cada uma dessas partes que destacamos na aplicação precisa de um tratamento específico, no entanto elas precisam interagir em determinados momentos. Para tratar desse tipo de aplicação, foi desenvolvida a arquitetura em camadas MVC, abreviação da língua inglesa para Model, View e Controller (traduzido como modelo, visão e controle).

> **A MVC é uma arquitetura muito utilizada na prática para organizar os códigos de um sistema. A ideia básica é que cada seção do nosso código tem uma finalidade e, portanto, precisa ser tratada de uma forma específica.**

Uma parte do nosso código é responsável por realizar operações com o SGBD, outra parte deve focar em uma interação homem-máquina com qualidade, para que o usuário tenha uma boa experiência com a aplicação e, ainda, tem a parte do código que controla como a nossa aplicação vai funcionar.

Portanto, de forma resumida, a MVC nos ajuda a organizar as funções principais do código da nossa aplicação. A consequência disso é que ganhamos com a melhoria da organização que facilita realizar manutenções, melhorias e otimizar o trabalho em equipe.

Camadas da arquitetura MVC

A arquitetura MVC é formada pelas seguintes camadas:

Modelo

Essa camada é responsável pela representação dos dados. A forma mais comum do seu uso é na construção de classes cujos campos refletem a estrutura de tabelas de um banco de dados, além de representar as relações com outras tabelas e métodos básicos de interação com os dados no padrão get e set.

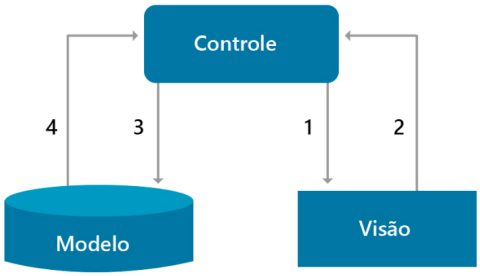
Visão

O código dessa camada é composto por todas as funções necessárias para interagir com o usuário. Ou seja, ela é responsável pela interface homem-máquina.

Controle

O código dessa camada faz a conexão entre as camadas Modelo e Visão. Ou seja, é ela que implementa as regras de negócio do sistema.

Na figura a seguir, podemos ver como as camadas da arquitetura MVC interagem.



Representação da Arquitetura MVC.
Sérgio Assunção Monteiro

Na figura, temos as seguintes interações entre as camadas:

1. A camada de controle fornece os dados processados para a camada de visão.
2. A camada de visão faz requisições para a camada de controle.
3. A camada de controle faz requisições para a camada modelo.
4. A camada modelo fornece os dados para a camada de controle.

Com essa divisão em camadas, a MVC nos ajuda a organizar a construção, manutenção e melhoria de um sistema. Agora, vamos analisar alguns aspectos sobre a construção de classes, que é a base para implementarmos os códigos.

Classes das camadas de modelo e de controle

Aspectos fundamentais das classes

Uma classe faz a modelagem de entidade para o domínio do sistema. Por exemplo, podemos pensar em uma entidade como uma pessoa, um carro ou um livro. Para modelar essas entidades em classes, precisamos descrever suas propriedades por meio de atributos e um conjunto de funções que manipularão essas propriedades.

A classe é um conceito abstrato, ou seja, é um modelo para representação das entidades. Para podermos operar com os dados, precisamos utilizar objetos que são instâncias da classe. Por meio de um objeto, podemos realizar operações de consulta e escrita dos atributos. Por exemplo, um objeto representando uma entidade livro pode ter um atributo título. O conceito de classe é o conceito fundamental da programação orientada a objetos (POO).

A POO possui três propriedades principais:

Encapsulamento

Expõe para as demais classes do programa apenas os métodos. Dessa forma, a complexidade de manipular os atributos fica restrita à própria classe. Isso significa que, na prática, quando um código precisar de alguma informação de um objeto, ele simplesmente vai obtê-lo através da chamada de um método.

Herança

É capacidade de uma classe herdar as características de uma outra classe, normalmente chamada de classe-mãe. Essa propriedade é muito útil para generalizar comportamentos. Um exemplo clássico é das classes Pessoa Física e Pessoa Jurídica, que herdam as características da classe-mãe Pessoa.

Polimorfismo

Significa que métodos de mesmo nome dentro de uma classe, ou relacionados por meio da herança, podem ter comportamentos distintos dependendo do contexto de operação. Essa propriedade é útil em muitos cenários, mas deve ser usada com muita cautela para não gerar confusão. Especialmente no processo de depuração de um programa, ela pode ser um grande desafio.

Bem, agora que vimos os conceitos fundamentais de classes, vamos entender como eles se relacionam no contexto da MVC.

Classes para mapeamento das entidades

A camada de modelo trata do mapeamento explícito das tabelas do banco de dados. A ideia básica dessas classes é de representar os campos de uma tabela através de atributos e utilizar métodos do tipo getter e setter com os seguintes objetivos:

Getter

Serve para realizar consultas. Por exemplo, no caso de uma classe livro com um atributo “titulo”, teremos um método “getTitulo()” que retorna o valor do atributo “titulo”.

Setter

É aplicado para realizar operações de escrita. Voltando ao exemplo da classe livro do item anterior, teremos um método “setTitulo(nome)” que vai atribuir o valor do parâmetro “nome” ao atributo “titulo” do livro.

Essas classes, normalmente, são bem simples. Os pontos que exigem mais atenção são os dos tipos dos atributos que devem ser compatíveis com os tipos dos respectivos campos das tabelas do banco de dados.

Objetos de Acesso aos Dados (DAO)

Agora que sabemos como mapear as tabelas do banco de dados em classes, precisamos manipular essas classes para realizar as operações no banco de dados. Para isso, utilizamos um padrão chamado de Objeto de Acesso aos Dados que é mais conhecido como DAO (acrônimo para Data Access Object).

> **O objetivo do padrão DAO é isolar a camada de negócios da camada de persistência, ou seja, do banco de dados relacional, embora possa ser qualquer outro mecanismo de persistência. O padrão DAO pode ser combinado com o Mapeamento Objeto-Relacional mais conhecido como ORM (Object-Relational Mapping).**

O ORM é uma técnica que faz a conversão de dados entre os bancos de dados relacionais e as linguagens orientadas a objetos, facilitando bastante o processo de construção de uma solução, pois utiliza frameworks que já possuem métodos, os quais nos permitem fazer as operações CRUD – inserção, leitura, modificação e exclusão – sem entrar nos detalhes do SGBD.

Comentário

Um importante framework de ORM é o Hibernate. Ele realiza as operações de persistência no banco de dados, ou seja, as operações CRUD. É comum utilizarmos o Hibernate com o JPA, que é uma camada usada para descrever interface para um framework ORM. Agora, vamos estudar sobre associações e multiplicidades das classes da camada de modelo.

Associações e multiplicidades das classes da camada de modelo

Associação

Associação é um relacionamento entre duas classes separadas. Podemos representá-la por uma linha entre essas classes com uma seta indicando a direção da navegação. Na figura a seguir, temos um exemplo de associação entre as classes Professor e Livro.

Exemplo de associação.
Sérgio Assunção Monteiro.

Na figura, podemos ver o termo “Escreve” indicando um fluxo de navegação entre as classes, no caso, o professor escreve livros. Mas, além da relação de navegação, a figura apresenta as simbologias “1..*” e “0..*”, conhecidas como multiplicidade, que é exatamente o conceito que vamos abordar na sequência.

Conectividade

A conectividade corresponde ao tipo de associação entre duas classes: “muitos para muitos”, “um para muitos” e “um para um”. A conectividade da associação entre duas classes depende dos símbolos de multiplicidade que são utilizados na associação. As representações de conectividade são:

Um-para-um

Descreve uma relação exclusiva entre dois itens. Por exemplo, uma pessoa pode ter apenas um passaporte e um passaporte pertence apenas para uma pessoa.

Um para muitos

Aqui, um item pode se relacionar a diversos outros. Por exemplo, um médico poder ter muitos pacientes.

Muitos para um

Nesse caso, muitos itens podem estar relacionados a apenas uma entidade. Por exemplo, na relação entre livros e biblioteca, temos que muitos livros estão em uma biblioteca e não podem fazer parte de outra biblioteca.

Muitos para muitos

Aqui, muitos itens podem se relacionar com muitos outros. Por exemplo, na relação entre professor e aluno, temos que muitos alunos estão associados a um professor e, também, o professor pode estar relacionado a muitos alunos.

Multiplicidade

Representam a informação dos limites inferior e superior da quantidade de objetos aos quais outro objeto pode se associar. A tabela, a seguir, ilustra as multiplicidades possíveis na UML:

Nome	Simbologia na UML
Apenas Um	1..1 (ou 1)
Zero ou Muitos	0..* (ou *)
Um ou Muitos	1..*
Zero ou Um	0..1
Intervalo Específico	$I_i..I_s$

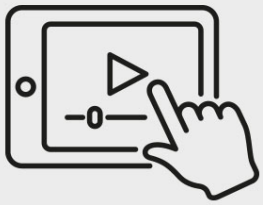
Tabela exemplar das multiplicidades possíveis na UML.
Sérgio Assunção Monteiro.

Os conceitos que acabamos de estudar são necessários para que, mais adiante, possamos trabalhar com JPA e Hibernate.

Implementação da Camada de Modelo

Assista agora a um vídeo em que são apresentados os principais conceitos sobre a implementação da Camada de Modelo.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Ao desenvolver um sistema, devemos nos preocupar com a segregação de responsabilidades de suas partes. Nesse contexto, a arquitetura MVC auxilia nessa separação e, como consequência, facilita a manutenção e atualização do sistema ao longo do tempo. Nesse sentido, selecione a alternativa correta a respeito da camada de modelo da arquitetura MVC:

A Tem como responsabilidade atuar sobre o mapeamento e persistência de dados.

B É responsável por interagir com o usuário por meio de relatórios que apresentam o estado do banco de dados.

C Trata dos aspectos de segurança dos dados.

D Tem como objetivo aplicar as regras de negócio.

E É formada pelas funções que fazem transformações dos dados.

Parabéns! A alternativa A está correta.

A arquitetura MVC organiza um sistema em três camadas: Modelo, Visão e Controle. No caso da camada de Modelo, seu objetivo é fazer a intermediação entre o sistema e o banco de dados, portanto ela trata do mapeamento de tabelas e operações de persistência.

Questão 2

A programação orientada a objetos é um importante paradigma de desenvolvimento. Quando aplicada para a arquitetura MVC, ele desempenha papéis específicos sobre o ciclo de vida dos dados. Um exemplo disso são os objetos de acesso aos dados. Nesse sentido, selecione a alternativa correta a respeito da utilização dos objetos de acesso aos dados.

A É responsável por mapear as tabelas com classes.

B Trata da aplicação de regras para manipular os dados.

C São artefatos que combinam as funcionalidades das camadas de persistência e negócio.

D Devem ser usados para melhorar o modelo de dados das tabelas.

E É uma interface que deve ser usada como referência para outras classes do projeto.

Parabéns! A alternativa B está correta.

Os objetos de acesso aos dados, ou, ainda, classes DAO, isolam as regras de negócio do sistema da camada de persistência. O objetivo é separar as responsabilidades de cada camada por meio da execução de tarefas específicas. As classes DAO tratam de operações CRUD.

2 - Aplicação das anotações JPA com Hibernate

Ao final deste módulo, você será capaz de aplicar as anotações JPA com Hibernate.

Introdução para a JPA

Conceitos básicos da JPA

A JPA é uma especificação da linguagem de programação Java. Ela é um acrônimo para Java Persistence API (JPA), traduzido para o português como API de Persistência Java. O objetivo dela é fazer a persistência de dados dos objetos implementados em Java com o banco de dados relacional. Portanto, a JPA desempenha um papel que faz a ponte entre os modelos de domínio orientados a objetos e os sistemas de banco de dados relacionais.

> **Um ponto importante que devemos ficar atentos: a JPA é apenas uma especificação, ou seja, ela não realiza nenhuma operação de persistência sozinha.**

Para isso, precisa de uma outra aplicação. É aí que entram as ferramentas de ORM, como Hibernate, que implementa especificações JPA para persistência de dados.

Arquitetura da JPA

A arquitetura da JPA é formada por classes e interfaces utilizadas para fazer a persistência de entidades em um banco de dados, ou seja, fazer o armazenamento dos dados em tabelas. Os principais componentes da arquitetura da JPA são (KONDA, 2012):

EntityManagerFactory

O nome deste componente é bem descritivo em relação ao seu objetivo, Fábrica de Entidades Gerenciadoras. Ou seja, é utilizada para criar e gerenciar várias instâncias do componente EntityManager.

EntityManager

É a gerenciadora de entidades. Ela gerencia as operações de persistência em objetos e funciona como uma fábrica de instâncias do componente Query.

Entity

Este componente representa os objetos que serão armazenados como registros no banco de dados.

EntityTransaction

Possui um relacionamento um-para-um com o EntityManager de tal forma que, para cada EntityManager, as transações são mantidas pela classe EntityTransaction.

Persistence

Este componente contém métodos estáticos para obter a instância do componente EntityManagerFactory.

Query

É uma interface que deve ser implementada por cada fornecedor JPA para obter objetos relacionais que satisfaçam aos critérios.

Esses componentes nos ajudam a desenvolver códigos, para que possamos focar no mapeamento das classes com tabelas de banco de dados. O nosso próximo passo é conhecer as anotações do JPA.

Principais anotações da JPA

Anotações da JPA

Um conceito fundamental para trabalhar com JPA é o de anotações. As anotações da JPA são usadas para mapear objetos Java com tabelas do banco de dados, colunas etc. A JPA possui muitas anotações, mas, normalmente, precisamos de apenas um subconjunto delas. Vamos apresentar algumas das principais anotações divididas nas seguintes categorias:

Definição de uma classe de entidade

Mapeamentos de colunas

Mapeamentos de Associação

Definição de uma classe de entidade

As entidades JPA não precisam implementar nenhuma interface ou herdar de uma superclasse. A única coisa que precisamos fazer é identificar uma classe como uma entidade e, em alguns casos, também adaptar o mapeamento de tabela. Para isso, usamos as anotações abaixo:

@Entity

Usada para identificar uma classe como uma classe de entidade. Um exemplo de aplicação é:

Java

@Table

Usada para mapear a classe com uma tabela. Por padrão, cada classe de entidade faz mapeamento com uma tabela de banco de dados com o mesmo nome no esquema padrão de nosso banco de dados. No entanto, podemos usar nomes diferentes para fazer esse mapeamento. Abaixo, apresentamos um exemplo de aplicação da anotação @Table:

Java

Mapeamentos de colunas

Por padrão, todas as implementações de JPA fazem o mapeamento de cada atributo de entidade para uma coluna de banco de dados com o mesmo nome e um tipo compatível. No entanto, há casos em que precisamos fazer modificações. Para realizar esse tipo de tarefa, a JPA disponibiliza as seguintes anotações:

@Column

É uma anotação opcional que permite fazer o mapeamento personalizado entre o atributo de entidade e a coluna do banco de dados. Veja um exemplo a seguir:

Java

Devemos notar que o nome do atributo é “nome” e estamos mapeando-o com o nome da coluna “titulo”. Outra coisa: o atributo não pode ser modificado, mas pode ser cadastrado.

@Id

É utilizada para especificar um atributo como chave primária de uma entidade. Por exemplo:

Java

@GeneratedValue

É aplicada para especificar se uma chave será gerada sequencialmente, ou incrementada. Adiante, mostramos um exemplo em que a coluna Id será incrementada:

Java

@Enumerated

Essa anotação permite definir como um atributo enumerável é persistido no banco de dados. Abaixo, apresentamos um exemplo:

Java

Mapeamentos de Associação

A JPA permite que possamos mapear associações entre as entidades. Quando trabalhamos no banco de dados, essas associações são modeladas como colunas de chave estrangeira. No caso da JPA, essas

associações são mapeadas como atributos do tipo da entidade associada em seu modelo de domínio. Para isso, utilizamos as anotações @ManyToMany, @ManyToOne, @OneToMany ou @OneToOne. A seguir, apresentamos essas anotações com mais detalhes:

@ManyToMany

Essa é a associação muitos-para-muitos que estudamos anteriormente. Vamos ver o exemplo Livros e Autores. Primeiro, vamos ver a classe Livro:

Java

O que acabamos de fazer foi um mapeamento unidirecional. Para fazer um mapeamento bidirecional, precisamos fazer a correspondência na classe Autor, conforme o trecho de código:

Java

Um ponto de atenção é o parâmetro “mappedBy”. Ele precisar estar com o nome do atributo que define a associação, no caso, o atributo “autores”.

@ManyToOne

Essa anotação define o lado proprietário de uma associação bidirecional. Nós a utilizamos na entidade que mapeia a tabela de banco de dados que contém a coluna de chave estrangeira. Como mostrado no exemplo:

Java

No exemplo, temos que muitos livros podem ser publicados pela mesma editora. Outro ponto de atenção é com relação ao parâmetro “fetch”. Esse parâmetro indica como os dados serão carregados na memória. Por padrão, o JPA utiliza o tipo EAGER que diz ao Hibernate para obter as entidades relacionadas com a consulta inicial. Isso pode ser muito eficiente, mas, na maioria dos casos, pode criar uma enorme sobrecarga. O outro tipo de busca é LAZY que informa para o Hibernate atrasar a inicialização do relacionamento.

@OneToMany

Usamos essa anotação para definir o lado de referência de uma associação bidirecional. Da mesma forma que definimos o lado de referência de uma associação bidirecional de muitos-para-muitos, podemos fazer a referência ao nome do atributo que possui a associação no atributo mappedBy. Confira no exemplo:

Java

Um ponto de atenção aqui é para o parâmetro “cascade” com o valor “CascadeType.ALL”. Isso indica que todas as operações de persistência serão propagadas da classe Editora para as entidades relacionadas.

@OneToOne

Essa anotação é usada para fazer o mapeamento de associação um-para-um. Podemos utilizá-lo para modelar associações unidirecionais ou bidirecionais. A seguir, apresentamos um exemplo de mapeamento unidirecional:

Java

Agora, para modelarmos como uma associação bidirecional, precisamos definir o atributo mappedBy do lado de referência da associação para o nome do atributo que possui a associação, conforme o exemplo:

Java

Aspectos fundamentais do Hibernate e JPA

Fundamentos do Hibernate

O Hibernate é um framework do Java voltado para o desenvolvimento de aplicações Java que vão interagir com o banco de dados. Trata-se de uma ferramenta de ORM (Mapeamento de Objetos Relacionais) de código aberto. Além disso, ele implementa as especificações do JPA para persistência de dados.

O objetivo de uma ferramenta de ORM é simplificar a criação, manipulação e acesso aos dados por meio do mapeamento de um objeto para os dados armazenados no banco de dados. Internamente, o Hibernate usa a API JDBC para interagir com o banco de dados. Além disso, o Hibernate tem as seguintes características:

- É uma das implementações de JPA mais usadas.
- É definido através do pacote org.hibernate.
- Ele usa a interface SessionFactory para criar instâncias de sessão.
- Usa a interface Session para criar, ler e excluir operações para instâncias de classes de entidade mapeadas.
- Trabalha com uma Linguagem de Consulta Hibernate – Hibernate Query Language – mais conhecida como HQL, que é orientada a objetos para realizar operações de banco de dados.

Configurações para usar o Hibernate no projeto

Basicamente, temos que configurar os arquivos “persistence.xml” e “pom.xml” para utilizarmos o Hibernate em um projeto. Vamos detalhar um pouco mais esse processo de configuração:

- **persistence.xml:** neste arquivo, precisamos preencher informações a respeito das propriedades de Driver, caminho para o banco de dados, usuário e senha de acesso ao banco de dados e dialeto para se comunicar com o banco de dados.
- **pom.xml:** neste arquivo, precisamos informar as dependências do projeto. No caso do Hibernate, basicamente teremos que colocar um código semelhante a este abaixo:

Xml

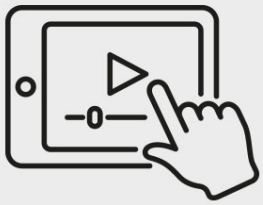
Para concluir essa introdução ao Hibernate e JPA, as classes que vamos utilizar as operações de persistência são as de Acesso Direto a Objetos através de um objeto do tipo EntityManager. Com veremos

mais adiante, ela faz uma abstração de vários detalhes de implementação, tornando o código mais legível e, portanto, mais fácil de rastrear.

Aplicação das anotações JPA com Hibernate

Assista agora a um vídeo em que são apresentados os principais conceitos para aplicação das anotações JPA com Hibernate.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

A JPA é uma importante aliada para construção de sistemas que fazem persistência de dados. Ela é formada por recursos, tais como classes e interfaces, que têm objetivos específicos. Nesse sentido, selecione a alternativa correta a respeito dos componentes da arquitetura da JPA:

A Para representar um objeto que será armazenado, a JPA utiliza a classe Persistence.

B O gerenciamento das operações de persistência é da responsabilidade do EntityManager.

C A classe Entity produz instâncias de consultas para realizar operações SQL.

D A interface Query é utilizada para realizar operações de persistência.

E As transações do sistema são da responsabilidade do EntityManager.

Parabéns! A alternativa B está correta.

Os componentes principais da arquitetura da JPA são: EntityManagerFactory, EntityManager, Entity, EntityTransaction, Persistence e Query. Cada um tem um objetivo específico em relação à representação e ao gerenciamento do ciclo de vida dos dados. Especialmente em relação ao EntityManager, ele é usado para realizar as operações de persistência dos objetos.

Questão 2

Um aspecto básico da JPA é o uso de anotações. São por meio delas que o desenvolvedor caracteriza as propriedades de um item. Nesse sentido, selecione a alternativa correta a respeito das anotações da JPA:

A Implementam instruções SQL para gerenciar os objetos.

B Sempre devem ser usadas da mesma forma para qualquer projeto que use JPA.

C Só devem ser usadas nas classes DAO.

D Podem ser usadas para definir estratégias de geração de valores para as colunas das tabelas.

E O único lugar onde devem ser usadas são com as classes de mapeamento de entidades.

Parabéns! A alternativa D está correta.

Para programar com JPA é necessário conhecer as anotações. É um recurso fundamental para definir classes de entidades, fazer mapeamento de colunas e de associações, além de poder ser utilizadas para implementação de consultas entre outras tantas funcionalidades.

3 - Uso do banco de dados H2 como servidor

Ao final deste módulo, você será capaz de identificar o uso de repositórios com o framework Spring Data JPA e o banco de dados H2.

Framework Spring Data JPA

Aspectos básicos do Spring Data JPA

O Spring Data JPA faz parte do framework Spring Data. O objetivo dele é tornar o processo de desenvolvimento de repositórios baseados em JPA mais simples. Uma boa fonte de referência sobre Spring Data JPA é na página oficial desse framework.

Basicamente, ela nos fornece algumas interfaces com métodos genéricos que podemos utilizar para manipular os dados persistidos nos objetos. Em especial, para aplicações em que precisamos implementar serviços REST, ela ajuda bastante, pois é muito comum precisarmos fazer consultas pelo campo identificador de um objeto, ou listar todos os objetos, para que, em seguida, possamos aplicar algum filtro.

Concretamente, podemos usar um repositório JPA através da extensão de uma interface. Podemos ver um exemplo típico de utilização no código a seguir:

Java

A classe “MinhaClasse” é a que representa a tabela do banco de dados e a “JpaRepository” é a interface de repositório do JPA. Além disso, a função “findById” já é disponibilizada pela JPA para que a utilizemos.

Configuração do Spring Data JPA

Para utilizar o Spring Data JPA em um projeto, precisamos adicionar a dependência dele no arquivo pom.xml, semelhante ao código abaixo:

Xml

O arquivo POM (acrônimo para Project Object Model), traduzido para o português Modelo de Objeto do Projeto, é a unidade mais importante de um projeto no Maven. Trata-se de um arquivo XML que contém diversas informações sobre o projeto, plugins, perfis de compilação e as configurações das dependências que são necessárias para que o projeto funcione corretamente.

Atenção!

A base de um projeto focado na camada de Modelo da arquitetura MVC é acessar o banco de dados. Com esse objetivo, nosso próximo passo será estudar um banco de dados muito útil para as etapas iniciais de construção de um projeto e execução de testes.

Banco de Dados H2

Conceitos Básicos sobre o H2

O H2 é um banco de dados Java de código aberto. Ele é bastante utilizado nas etapas iniciais para criação de um projeto, pois ele precisa de poucos recursos para funcionar bem. Ele pode ser utilizado em uma aplicação desenvolvida com Java e, ainda, pode operar no modo cliente-servidor.

> **Ainda sobre as características que tornam o H2 tão interessante, é que ele pode ser configurado para ser executado como banco de dados na memória. Isso significa que os dados não serão persistidos no disco.**

Exatamente por causa do banco de dados embutido, ele não é adequado para ser aplicado no ambiente de produção, mas, como dissemos, é uma boa escolha para desenvolvimento e teste.

Características do H2

Podemos ter uma noção errada de que o H2 oferece poucos recursos. Na verdade, ele possui diversas características que o tornam uma ótima escolha para realizar testes de programas e treinamento. Entre as principais características do banco de dados H2 estão:

- Possui um mecanismo de banco de dados de alta velocidade.
- É um software de código aberto e escrito em Java.
- Suporta SQL padrão e API JDBC, além de poder usar o driver ODBC do PostgreSQL.
- Pode operar incorporado a uma aplicação (que será o caso que faremos mais adiante) e em modo de servidor.
- Fornece suporte para clustering.
- Possui recursos de segurança.
- As tabelas do banco de dados do H2 são temporárias e não são persistentes no disco após o fechamento de uma conexão.
- Fornece suporte para transações e bloqueio de nível de tabela.
- O otimizador para consultas complexas é um algoritmo genético.
- Fornece suporte para banco de dados criptografado (AES), criptografia de senha SHA-256, funções de criptografia e SSL.

Configuração do H2

Bem, agora que conhecemos mais sobre o H2, precisamos saber como configurá-lo para utilizar em uma aplicação. Teremos que configurá-lo em dois arquivos: pom.xml e persistence.xml.

No caso do pom.xml, simplesmente, teremos que adicionar a dependência, como podemos ver no código:

Xml

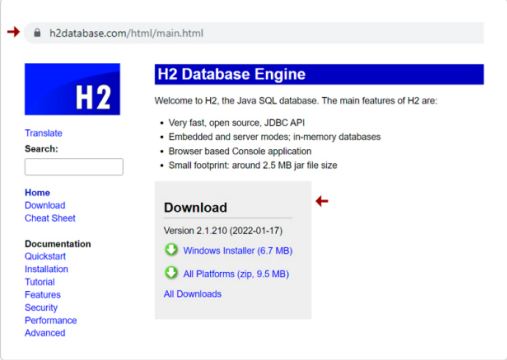
Já no arquivo persistence.xml, teremos que fornecer informações sobre o banco de dados, driver de comunicação, nome do usuário do sistema e senha, além de outras informações que veremos

detalhadamente, quando fizermos o nosso exemplo completo com o Java.

Uso do banco de dados H2 como servidor

Instalação do H2

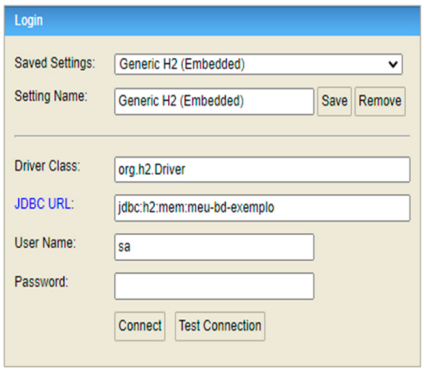
Para instalar o H2, precisamos visitar a página oficial do H2 Database Engine. Em seguida, temos que fazer o download para nossa máquina local e executar o programa. Abaixo, mostramos onde baixar o arquivo executável do site oficial.



Site oficial do H2.

A instalação é muito simples, sendo necessário apenas pressionar “Next”.

Ao final da instalação, basta executar o programa e o H2 vai abrir no navegador, conforme a figura.



H2 em execução.

Precisamos parametrizar o banco de dados agora, mas isso é bem simples. Basicamente, vamos dar um nome para o banco de dados. Observe na opção “JDBC URL”, que chamamos o banco de “meu-bd-exemplo”.

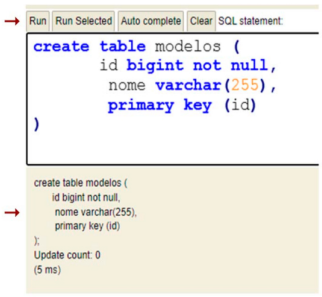
Feito isso, devemos pressionar o botão “Connect” e vamos entrar no ambiente de trabalho do H2. A seguir, vamos criar uma tabela, fazer algumas inserções e consultas.

Criação de tabela

Vamos criar uma tabela bem simples. Para isso, usaremos o código sql abaixo:

Sql

Agora, basta pressionar o botão Run para criar a tabela. Na figura abaixo, mostramos como funciona esse processo.



Criação de tabela no H2.

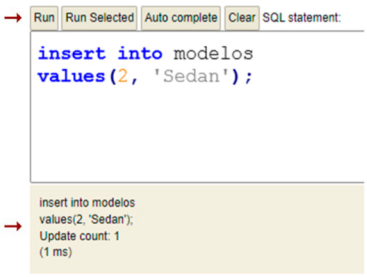
Com isso, nossa tabela está pronta para fazermos as operações de cadastro e consulta de dados.

Inserção de dados

Agora, vamos popular a nossa tabela. Para isso, precisamos executar os comando sql apresentados a seguir:

Sql

Para executá-los, basta realizar o mesmo processo que fizemos anteriormente: digitar o script na área de texto e pressionar o botão Run. A seguir, mostramos como o resultado dessa operação.



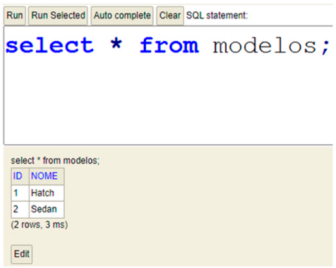
Execução do comando de inserção.

Consulta aos dados

Por fim, vamos executar uma consulta. Queremos listar todos os itens da tabela modelos. Então, precisamos usar o script:

Sql

Para executá-lo, digite o código na área de texto do H2 e pressione o botão Run. Veja o resultado desse processo.



Execução de uma consulta no H2.

Mais adiante, vamos desenvolver uma aplicação Java e usaremos o H2 para realizar nossos testes. Como vimos, é uma ferramenta muito útil como recurso didático e para realizar testes sem a necessidade de fazer nenhuma configuração mais elaborada.

Uso de repositórios com framework SpringData, JPA e H2

Assista agora a um vídeo em que são apresentados os principais conceitos sobre uso de repositórios com framework SpringData, JPA e H2.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

É uma boa prática utilizar recursos que padronizem a programação. Este é o caso do Spring Data JPA, bastante utilizado para construir serviços. Em relação ao Spring Data JPA, selecione a alternativa correta:

A Deve ser aplicado para implementar métodos complexos de consulta de dados.

B Trabalha com a classe JpaRepository para obter informações dos dados.

C Utiliza interfaces de repositórios que já possuem alguns métodos padrões.

D É uma interface de serviços utilizada para fazer persistência dos dados.

E É uma classe vinculada ao mecanismo de gerenciamento de dados do Hibernate.

Parabéns! A alternativa C está correta.

A Spring Data JPA integra o framework Spring Data. Ela é utilizada como uma interface de repositório de objetos que disponibiliza consultas padrões.

Questão 2

O tempo sempre é um fator importante no processo de desenvolvimento de software. Então, utilizar de ferramentas que auxiliem no processo de testes reduz a quantidade de problemas que podem atrapalhar para cumprir os prazos de entrega. Neste sentido, o mecanismo de banco de dados H2 ajuda a testar as funcionalidades da camada de Modelo. Em relação ao H2, selecione a alternativa correta:

A Pode ser utilizado quando o sistema estiver em produção.

B Só pode ser aplicado para instruções SQL muito simples.

C É de software livre para estudo acadêmico.

D É possível utilizá-lo para realizar consultas complexas com instruções SQL.

E Faz parte do framework Spring Data para realizar operações CRUD.

Parabéns! A alternativa D está correta.

O H2 é um software livre que possui um elaborado mecanismo de banco de dados. Ele pode ser utilizado tanto em aplicações acadêmicas, como comerciais para a fase inicial de desenvolvimento das funcionalidades do sistema e execução de testes. Todavia, é inadequado utilizá-lo quando o sistema está operando em produção.

4 - Consultas personalizadas com HQL

Ao final deste módulo, você será capaz de implementar consultas personalizadas com HQL.

Introdução às linguagens de consulta HQL e JPQL

Aspectos básicos do HQL

Atualmente, utilizamos sistemas computacionais para tratar dos mais variados assuntos. Boa parte dessas aplicações tem em comum a necessidade de gerenciar dados persistentes. Por persistência, devemos entender que os dados são armazenados de tal forma que podemos recuperá-los quando, assim, desejarmos. Uma forma de trabalhar com dados persistentes é por meio de uma interface entre o banco de dados e a aplicação. É nesse contexto que o Hibernate se encaixa.

Como já vimos anteriormente, o Hibernate é um framework de ORM (mapeamento objeto-relacional) implementado em Java que nos fornece recursos para que possamos trabalhar com objetos em Java e persisti-los no banco de dados. Ele possui uma linguagem de consulta orientada a objetos, parecida com a linguagem SQL, conhecida como HQL. HQL é um acrônimo para *Hibernate Query Language*, que traduzido para o português fica Linguagem de Consulta do Hibernate.

> **A HQL trabalha com objetos persistentes e suas propriedades, enquanto o SQL trabalha com tabelas e colunas. A ideia é que possamos utilizar a HQL para programar consultas no Java, que são traduzidas pelo Hibernate, para consultas padrões do SQL.**

Cabe observamos aqui que são essas consultas que executam ações no banco de dados. Outra coisa, para utilizar a HQL, precisamos utilizar um objeto do tipo Session que utiliza diversas informações sobre a configuração do Hibernate.

A HQL nos permite escrever comandos com as cláusulas “SELECT” e “WHERE”, por exemplo. Ela tem uma peculiaridade que precisa de atenção: ela não faz a distinção entre comandos escritos com letras maiúsculas ou minúsculas, por outro lado, ela é sensível para propriedades como nomes de tabelas e colunas.

Agora, vamos ver alguns exemplos do uso de HQL.

Cláusula SELECT

Tem o mesmo objetivo do comando SQL, ou seja, fazer uma consulta nos dados. Só precisamos ficar atentos ao fato de que com a HQL trabalhamos com objetos e não com tabelas. A seguir, mostramos um exemplo que utiliza a cláusula SELECT para obter apenas o campo nome do objeto Modelo:

Java

Cláusula WHERE

Usamos essa cláusula para restringir os objetos de uma consulta, conforme exemplo:

Java

Existem muitas outras cláusulas que a HQL nos fornece para trabalhar. Podemos encontrar uma boa fonte de estudos para a HQL na documentação do jboss: veja Hibernate Community Documentation.

Fundamentos do JPQL

A HQL é a linguagem de consulta específica para o Hibernate. Uma desvantagem em utilizá-la em um projeto é que ele vai ficar vinculado ao Hibernate e será bem difícil realizar uma migração para outro

framework, caso seja necessário. Nesse cenário, foi criada a JPQL, a linguagem de consulta padrão da JPA. Um fato interessante é que toda consulta JPQL é um comando válido para HQL, mas o contrário não é verdadeiro.

De um modo geral, é mais fácil trabalhar com a JPQL do que com a HQL, pois as configurações dela são mais simples. Contudo, os objetivos das duas é o mesmo: trabalhar com ORM. Para poder utilizá-la, precisamos utilizar o gerenciador de entidades “EntityManager” que é usado para fazer a persistência dos objetos. Adiante, mostramos um exemplo de um trecho de código com a JPQL:

Java

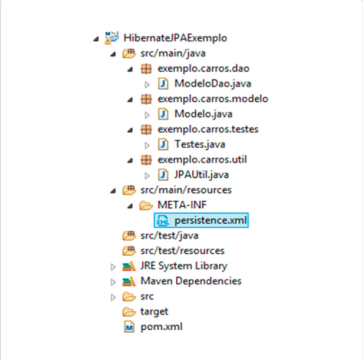
No trecho de código acima, temos o objeto “em” que é o EntityManager. Uma boa fonte de referência sobre a JPQL pode ser encontrada em Tudose (2022).

Construção da infraestrutura do projeto

Estrutura do projeto

Agora, vamos desenvolver um projeto completo com JPA. O objetivo é cadastrar dados e realizar consultas em uma tabela que representa os modelos de carros. Essa tabela tem apenas um identificador e uma coluna com o nome do modelo do carro.

Vamos precisar criar as classes de entidade, DAO, teste e uma classe para o gerenciador de entidades (EntityManager). Além disso, precisamos criar e configurar os arquivos de persistência e pom.xml. Na figura a seguir, apresentamos a estrutura do projeto final.



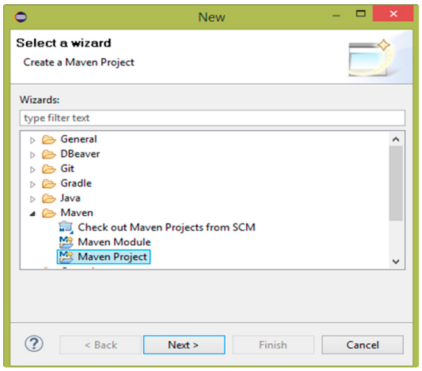
Estrutura do projeto.

Criação do projeto

O primeiro passo é criar o projeto. Para isso, vamos utilizar o Eclipse. Então, devemos acessar a barra de menu e seguir a sequência:

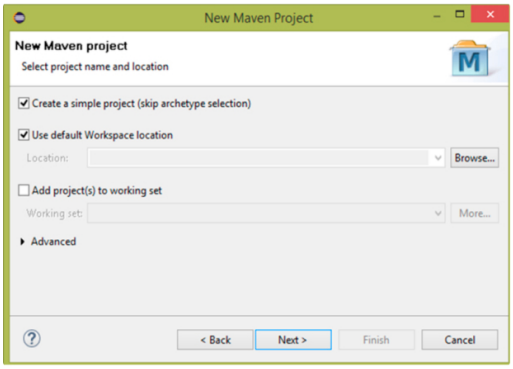
> **File → New → Other → Maven → Maven Project**

Em seguida, aparecerá um formulário. Devemos escolher a opção “Maven Project” e pressionar o botão “Next”, conforme apresentamos na figura.



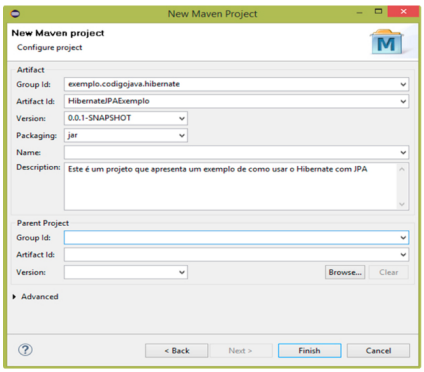
Seleção de um projeto Maven.

Na sequência, aparecerá um novo formulário para escolhermos o tipo de projeto Maven que vamos trabalhar. No caso, devemos selecionar a opção “Create a simple Project (skip archetype selection)” e pressionar o botão “Next”, conforme podemos ver na figura abaixo.



Criação de um projeto Maven.

O próximo passo é preencher o formulário com as informações a respeito do projeto Maven. Na figura, podemos ver como esses campos devem ser preenchidos.



Preenchimento de informações do projeto Maven.

Por fim, devemos pressionar o botão “Finish”. Com isso, cumprimos a etapa inicial do processo de criação do projeto. Agora, precisamos configurar o arquivo pom.xml com as dependências que vamos utilizar no projeto.

Configuração de dependências do Maven

A parte fundamental de um projeto Maven é o arquivo pom.xml, pois o utilizamos para configurar as dependências necessárias. No caso do exemplo, vamos precisar de duas dependências:

- O gerenciador de entidades do Hibernate que é o artefato “hibernate-entitymanager”;
- E o mecanismo de banco de dados H2, cujo artefato é “h2”.

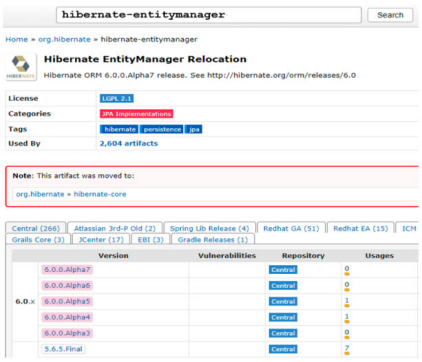
A seguir, apresentamos a versão final do arquivo pom.xml:

Xml

Ainda precisamos fazer mais alguns comentários sobre o arquivo pom.xml:

- Para obter informações sobre os artefatos do Maven, como o número das versões, precisamos ir ao “repositório do Maven". Para isso, utilize seu buscador preferido.

  Na área de pesquisa, digitamos o nome do artefato e escolhemos a versão mais adequada para nosso projeto. Na figura, mostramos um exemplo de consulta:



Exemplo de consulta no repositório do Maven.

- Uma dica que é bastante útil no desenvolvimento desse tipo de projeto é sobre a versão do Maven não está atualizada. Caso isso aconteça, devemos clicar com o botão direito do mouse sobre o projeto e realizar a sequência:

  “**Maven**” → “**Update Project** ...”

Criar arquivo de configuração JPA

A próxima etapa é criar o arquivo de configurações do JPA. Para isso, devemos criar uma pasta “META-INF” no caminho (veja estrutura do projeto):

> **src/main/resources**

Em seguida, precisamos criar o arquivo “persistence.xml”. Nesse arquivo, vamos configurar as propriedades do banco de dados, driver de comunicação e mecanismos de execução. A seguir, apresentamos a versão final do arquivo “persistence.xml”:

Xml

Mais adiante, faremos referência a esse arquivo como Unidade de Persistência. Nesse caso, o nome que escolhemos foi “carros-pu”. Terminamos a etapa de configurações, os próximos passos serão para construir as classes de entidade, DAO e de utilitários.

Implementação das classes

Classe Entidade

Agora, vamos criar a classe de entidade. No caso, vamos chamá-la de “Modelo”. Essa classe deve refletir a tabela que estamos gerenciando. Ela possui apenas dois campos: o identificador e um nome. É nela também que devemos usar as anotações do JPA, tais como “@Entity”, “@Table”, “@Id” e “@GeneratedValue”. Abaixo, mostramos o código da classe entidade (oriente-se pela estrutura do projeto ilustrada anteriormente):

Java

Classe DAO

O próximo passo é criar a classe DAO (Objeto de Acesso a Dados). Ela é responsável por realizar as operações de persistência dos objetos da classes de entidades. A seguir, mostramos a versão final da classe ModeloDao (oriente-se pela estrutura do projeto ilustrada anteriormente):

Java

Um ponto importante para destacarmos nesse código é a utilização de objeto “EntitityManager” que, de fato, é responsável pela persistência dos objetos. Além disso, no método “listarTodos”, utilizamos a JPQL para listar todos os modelos. Devemos perceber que o nome próximo da cláusula “FROM” é o do objeto e não da tabela.

Classe Utilitária

Como já estudamos, precisamos utilizar o EntityManager para realizar as operações de persistência dos objetos. O código tem duas partes principais: a criação do gerenciador de entidades e o método para obter o gerenciador de entidades. Adiante, mostramos a versão final do código:

Java

Um ponto que merece a nossa atenção é na passagem de parâmetro para a criação do gerenciador de entidades: “carros-pu”. Esse parâmetro é a unidade de persistência que definimos no arquivo “persistence.xml”.

Projeto em execução

Classe de Testes

Nos passos anteriores, tivemos que criar as classes de entidade, DAO e de utilitários. Além disso, tivemos que criar os arquivos de configuração. Agora, vamos criar a nossa classe de testes. Escolhemos o nome “Testes” para classe. Vamos utilizá-la para fazer o cadastro e consultas de modelos no banco do H2. A seguir, apresentamos a versão final do código:

Java

Basicamente, temos dois métodos:

- **cadastrarModelos()**: este método utiliza o “modeloDao.cadastrar” para fazer a persistência dos dados. Precisamos ficar atentos às chamadas de “em.getTransaction().begin()” e “em.getTransaction().commit()”, pois elas iniciam e persistem os dados no banco, respectivamente.
- **ListarTodosModelos()**: neste método, utilizamos a chamada “modeloDao.listarTodos()” para executar a instrução “jpql”. O retorno é uma lista de objetos que listamos no laço “for”.

O próximo passo é analisar as saídas do programa.

Inserção de dados

Logo que o programa inicia a execução, ele chama o método “cadastrarModelos()”. Como nós configuramos as propriedades "hibernate.show_sql" e "hibernate.format_sql" com valor verdadeiro, o Java vai nos mostrar detalhes da execução das instruções SQL. No caso de cadastro, veremos a seguinte saída:

Java

Ou seja, a saída nos mostra que a tabela “modelos” foi criada e que foram feitas duas inserções nela.

Consulta dos dados

Na sequência, o programa faz a execução do método “listarTodosModelos()”, que tem como objetivo retornar todos os elementos da tabela. Abaixo, mostramos a saída que é exibida no console do Eclipse.

Java

Ou seja, no início da execução, temos a instrução de consulta do SQL e, logo abaixo, temos a listagem dos dados que cadastramos na tabela “modelos”. Com isso, fizemos a nossa aplicação Hibernate com JPA. Bons estudos!

Consultas personalizadas com HQL

Assista agora a um vídeo em que são apresentados os conceitos e as práticas para desenvolver uma aplicação com HQL.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Um projeto Maven auxilia na padronização do desenvolvimento de um projeto MVC. O principal elemento estrutural desse tipo de projeto é o arquivo pom.xml. Nesse sentido, selecione a alternativa correta a respeito do pom.xml:

A É responsável por gerenciar as dependências do projeto.

B É usado para configurar o banco de dados.

C Faz a configuração de artefatos que serão utilizados no projeto.

D É um arquivo semiestruturado com informações a respeito da configuração das tabelas do banco de dados.

E Ao trabalhar com um projeto Maven, o arquivo pom.xml é configurado automaticamente.

Parabéns! A alternativa A está correta.

O arquivo pom.xml é o principal elemento estrutural de um projeto Maven. É nesse arquivo que informamos explicitamente para o projeto quais são as dependências que vamos utilizar. Por exemplo, é no pom.xml que indicamos que vamos trabalhar com o mecanismo de banco de dados H2. Posteriormente, o Maven vai baixar essa dependência automaticamente do repositório da Internet e disponibilizar para o nosso projeto.

Questão 2

Sistemas que operam com bancos de dados basicamente são compostos por operações CRUD. Durante o processo de desenvolvimento, é comum que ocorram alguns problemas. Por isso, é essencial ter uma rotina de testes que auxiliem na identificação desses problemas, para que possam ser corrigidos rapidamente. Nesse sentido, selecione a alternativa correta a respeito da classe de testes de um projeto Maven com JPA.

A Para executar uma consulta aos dados, é necessário fornecer informações do banco de dados.

B Para cada operação de persistência dos dados, é necessário instanciar um objeto DAO.

C As transações de consulta devem ser executadas com uma operação de fechamento de conexão.

D As operações de persistência de dados devem ter um commit para que sejam efetuadas.

E A classe de testes só deve ser utilizada para realizar operações de cadastro e consulta de objetos.

Parabéns! A alternativa D está correta.

A classe de testes é muito útil para detectar erros em tempo de execução. Ela pode ser usada para testar qualquer uma das operações CRUD. Em especial, no caso de operações de persistência de dados, é necessário fazer um commit na sequência de execução delas, para que sejam efetuadas no banco de dados.

Considerações finais

Ao longo desse conteúdo, estudamos sobre os conceitos e tecnologias para construir a camada de modelo de uma aplicação. Especialmente, demos destaque para o Java, H2, Hibernate e JPA. Além, é claro, das linguagens de consulta HQL e JPQL.

A camada de modelo faz parte da arquitetura MVC, que separa uma aplicação em três camadas: modelo, visão e controle. A ideia é segmentar as funções de um projeto de tal forma que seja mais simples fazer o gerenciamento do ciclo de vida. Isso implica maior velocidade de desenvolvimento, redução de erros e facilidade de fazer manutenções e melhorias no projeto.

O conhecimento sobre os fundamentos da camada de modelo junto com o das tecnologias de desenvolvimento, em especial, com os frameworks Hibernate e JPA, são importantes aliados para implementar projetos mais seguros e com melhor desempenho. Além disso, existe uma importante demanda de mercado que precisa de bons profissionais para desenvolver, dar manutenção e implementar melhorias em muitos sistemas que utilizam essas tecnologias.

Podcast

Ouça agora um podcast no qual são apresentados os principais os aspectos fundamentais para camada de modelo – Mapeamento Objeto Relacional.

Para ouvir o *áudio*, acesse a versão online deste conteúdo.

Explore +

Acesse o site oficial do Spring, pesquise por “spring data JPA” e aprenda mais sobre os recursos desse framework por meio de muitos exemplos práticos.

Visite o site oficial do Hibernate e aprofunde seus conhecimentos sobre o que vimos no conteúdo. Além disso, você terá acesso a muitos exemplos.

Referências

HIBERNATE. **Hibernate Communit Documentation**. Consultado na internet em: 9 fev. 2022.

KONDA, M. **Just Spring Data Access**. Califórnia: O'Reilly, 2012.

TUDOSE, C.; BAUER, C.; KING, G.; GREGORY, G. **Java Persistence with Spring Data and Hibernate**. Shelter Island, NY: Manning, 2022.

Material para download

Clique no botão abaixo para fazer o download do conteúdo completo em formato PDF.

Download material

O que você achou do conteúdo?

☆ ☆ ☆ ☆ ☆

Relatar problema