

VÍDEO 7.1: Realizando mapeamento de relacionamento entre entidades

Olá! Tudo bem?

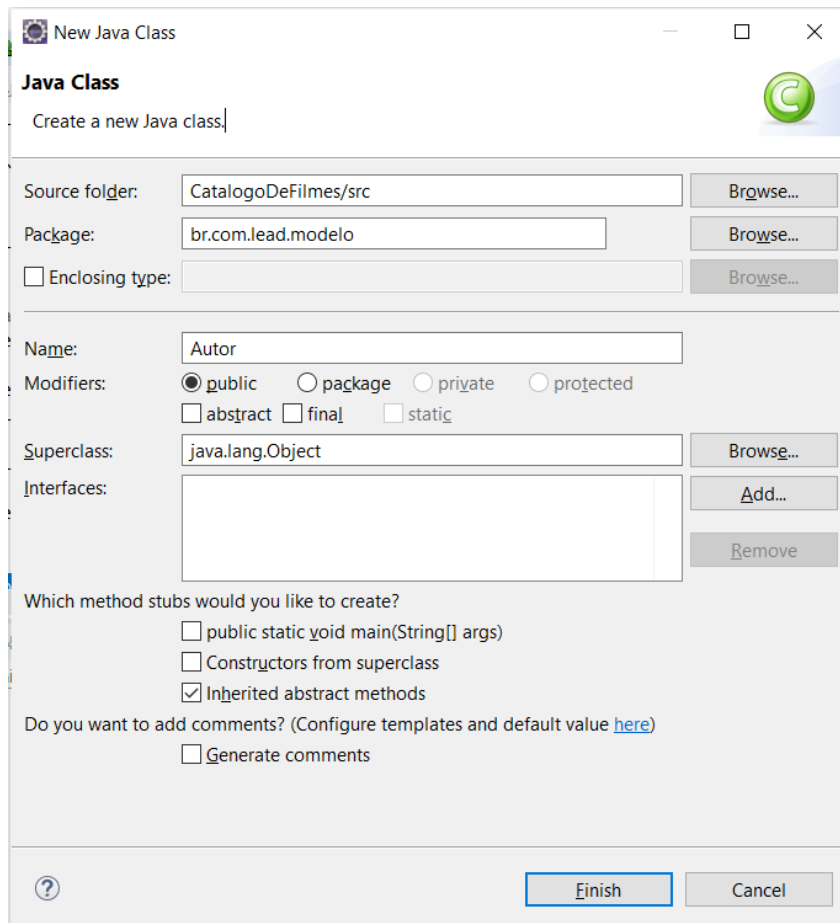
Nesta videoaula, você estudará como ocorre os relacionamentos entre entidades. Essa é a maneira de informar que duas entidades da sua aplicação possuem certa afinidade, ou seja, certa relação. Por exemplo, suponha que na sua aplicação, além de armazenar os dados de filme, você deseja também armazenar os dados do autor do filme. Ao fazer uso da orientação a objetos, essa informação sobre o autor do filme deve estar presente em uma outra entidade, denominada Autor. Entretanto, os dados de autor estão relacionados a um filme, logo essas duas entidades possuem um relacionamento.

Existem alguns tipos de relacionamentos entre entidades, como: o relacionamento muitos para um @ManyToOne; um para um @OneToOne; e o relacionamento muitos para muitos @ManyToMany. Está achando um pouco confuso? Calma! Ao longo desta aula, você estudará cada um desses relacionamentos na seguinte sequência: relacionamento muitos para um, um para um e muitos para muitos.

Como mencionado anteriormente, você aprenderá a criar uma entidade denominada Autor, que servirá para armazenar as informações referentes aos autores dos filmes do projeto. Nessa entidade, será armazenado apenas o nome do autor e a data de nascimento.

Após criar a entidade Autor, deve-se realizar o mapeamento da entidade. Em seguida, adicione o relacionamento entre as entidades Autor e Filme.

O primeiro passo para execução da atividade é acessar o seu projeto através do Eclipse. Feito isso, acesse o menu de opções, isso pode ser feito através da tecla Aplicação do teclado. Depois, navegue até a opção New, e, em seguida, até a opção Class. Como realizado em aulas anteriores, na criação de novas classes, você deve informar o nome da classe Autor e informar o pacote br.com.lead.modelo.



New Java Class

Create a new Java class

Source folder: CatalogoDeFilmes/src Browse...

Package: br.com.lead.modelo Browse...

☐ Enclosing type: Browse...

Name: Autor

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

? Finish Cancel

#Audiodescricao: A imagem mostra a caixa de diálogo “New Java Class”. Nela, há o texto “Java Class. Create a new Java class”. O campo “Source folder”, está preenchido por “CatalogoDeFilmes/src”; abaixo dela, temos o campo: “Package”, preenchido por “br.com.lead.modelo”. Em seguida, o campo “Name” está preenchido por “Autor”; a caixa de seleção “Modifiers” está marcada em “public”. Em seguida, o campo “Superclass” está preenchido por: “java.lang.Object” e a caixa de seleção que aparece em seguida “Which method stubs would you like to create?”, tem a opção “Inherited abstract methods” marcada. Abaixo, há o botão “Finish”, que está selecionado e o botão “Cancel”.

Agora, você irá adicionar os atributos da sua entidade da seguinte forma: primeiro, o atributo id do tipo Integer; depois, o atributo nome do tipo String; e, por último, o atributo dataNascimento do tipo LocalDate. Todos esses atributos devem estar

marcados como private. Uma vez criados os atributos, você irá criar os métodos getters e setters para cada atributo.

Pronto! Criados os métodos, você mapeará a classe Autor para que o JPA possa entendê-la como uma entidade. Em primeiro lugar, você deve adicionar a anotação `@Entity` acima do nome da classe. Em seguida, adicione a anotação `@Id` acima do atributo `id`; e, por fim, adicione a anotação `@GeneratedValue(strategy = GenerationType.IDENTITY)` abaixo da anotação `@Id`. Depois, acima do atributo `dataNascimento`, você deve informar a anotação `@Column(name="data_nascimento")`, desse jeito, o JPA, ao criar a estrutura no banco, criará a coluna com o nome `data_nascimento`.

```
@Entity
public class Autor {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    private String nome;

    @Column(name="data_nascimento")
    private LocalDate dataNascimento;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public LocalDate getDataNascimento() {
        return dataNascimento;
    }

    public void setDataNascimento(LocalDate dataNascimento) {
        this.dataNascimento = dataNascimento;
    }
}
```

#Audiodescricao: A imagem mostra o seguinte código:

@Entity

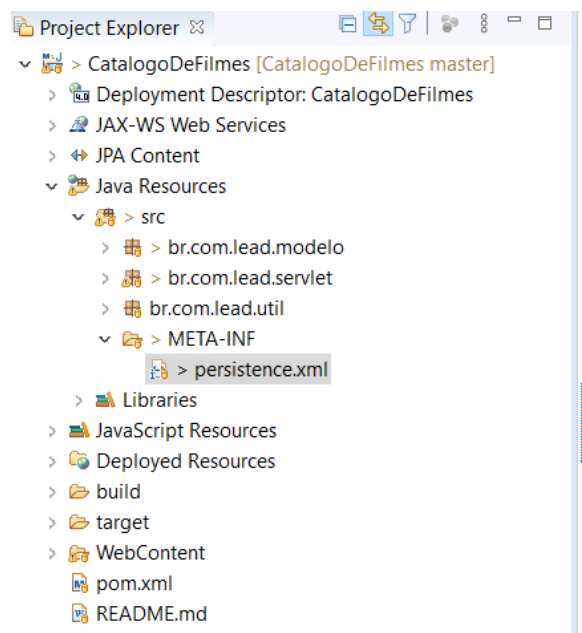
Centro de Pesquisa, Desenvolvimento e Inovação Dell

Telefone:(85)3492-1062 | www.leadfortaleza.com.br
Av. Santos Dumont, 2456 - 1906 | 60150162 - Fortaleza. CE

```
public class Autor {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Integer id;  
  
    private String nome;  
  
    @Column(name="data_nascimento")  
    private LocalDate dataNascimento;  
  
    public Integer getId() {  
        return id;  
    }  
  
    public void setId(Integer id) {  
        this.id = id;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public LocalDate getDataNascimento() {  
        return dataNascimento;  
    }  
  
    public void setDataNascimento(LocalDate dataNascimento) {  
        this.dataNascimento = dataNascimento;  
    }  
}
```

Mapeado a estrutura de Autor, adicione, ao arquivo de persistence.xml, a informação de que existe esta nova entidade que deve ser persistida no banco. Por isso, realize esse passo para que o JPA possa identificar a entidade Autor criada, para que seja

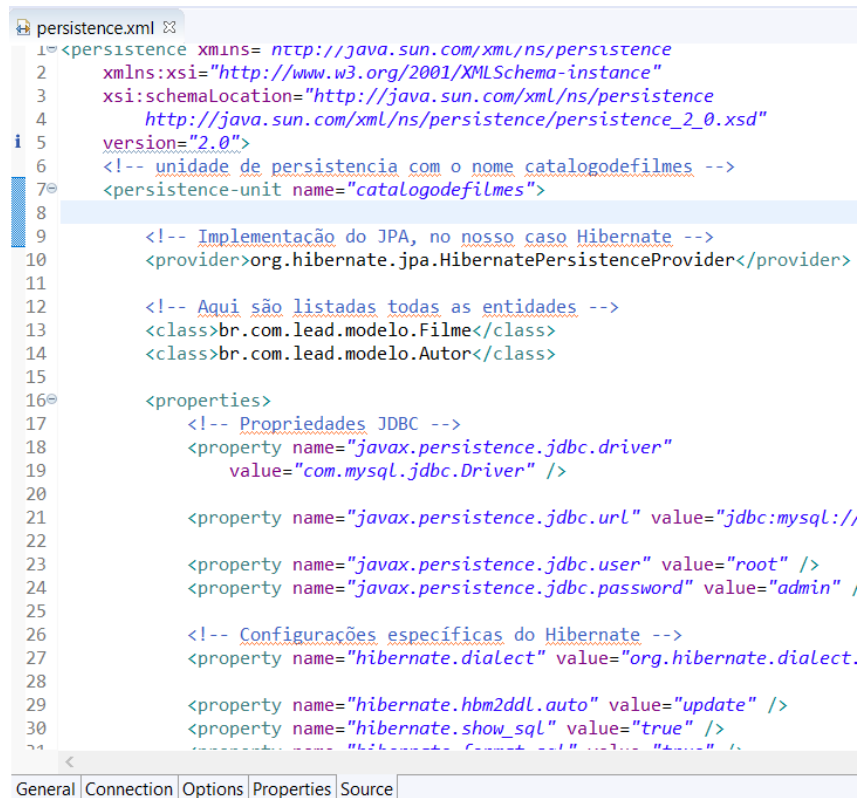
possível realizar o devido mapeamento com a estrutura do banco de dados. Feito isso, navegue até o seu projeto no Eclipse e expanda as subpastas do projeto, utilizando a seta de direção da Direita do teclado. Feito isso, localize a subpasta Java Resources para expandir, novamente, as subpastas do seu projeto. Nesse momento, você deve localizar a subpasta src e expandir as subpastas mais uma vez. Em seguida, localize a subpasta META-INF e expanda as subpastas outra vez. Na subpasta META-INF, você encontrará o arquivo persistence.xml, aquele que criado na aula 5, lembra?



#Audiodescriçao: A imagem mostra um recorte da aba “Project Explorer” do Eclipse. Nela, as pastas “CatalogoDeFilmes” e “Java Resources” estão expandidas. Em “Java Resources” a subpasta “src” está expandida. A subpasta contém os arquivos: “br.com.lead.modelo”; “br.com.lead.servlet” e “br.com.lead.util”. Em destaque, temos a subpasta “META-INF” que também está expandida revelando o arquivo “>persistence.xml”, que está selecionado.

Até aqui, tudo certo? Bem, você deve, então, acessar o arquivo persistence.xml, e, com o arquivo aberto no Eclipse, localize a aba Source para que se tenha acesso ao código fonte do arquivo. Acessando essa aba, você deverá localizar a informação `<class>br.com.lead.modelo.Filme</class>`, que foi o mapeamento realizado para a

entidade Filme já existente em seu projeto. Feito isso, você deve agora adicionar, logo abaixo dessa informação, o mapeamento para a classe Autor, ou seja, `<class>br.com.lead.modelo.Autor</class>`. Fique atento, pois esse passo deve sempre ser realizado toda vez que for adicionada uma nova entidade, entendido? Portanto, isso ocorre para que o JPA possa ter visão da classe. Dessa forma, coloque o caminho completo da classe e não apenas o nome.



```

1 <persistence xmlns="http://java.sun.com/xml/ns/persistence"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
4     http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
5   version="2.0">
6   <!-- unidade de persistencia com o nome catalogodefилmes -->
7   <persistence-unit name="catalogodefилmes">
8
9     <!-- Implementação do JPA, no nosso caso Hibernate -->
10    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
11
12    <!-- Aqui são listadas todas as entidades -->
13    <class>br.com.lead.modelo.Filme</class>
14    <class>br.com.lead.modelo.Autor</class>
15
16    <properties>
17      <!-- Propriedades JDBC -->
18      <property name="javax.persistence.jdbc.driver"
19        value="com.mysql.jdbc.Driver" />
20
21      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://
22
23      <property name="javax.persistence.jdbc.user" value="root" />
24      <property name="javax.persistence.jdbc.password" value="admin" />
25
26      <!-- Configurações específicas do Hibernate -->
27      <property name="hibernate.dialect" value="org.hibernate.dialect.
28
29      <property name="hibernate.hbm2ddl.auto" value="update" />
30      <property name="hibernate.show_sql" value="true" />
31      <property name="hibernate.format_sql" value="true" />
32    </properties>
33  </persistence-unit>
34 </persistence>
  
```

#Audiodescricao: A imagem mostra o seguinte código:

```

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">
  <!-- unidade de persistencia com o nome catalogodefилmes -->
  <persistence-unit name="catalogodefилmes">
  
```

```

    <!-- Implementação do JPA, no nosso caso Hibernate -->
  
```

Centro de Pesquisa, Desenvolvimento e Inovação Dell

Telefone:(85)3492-1062 | www.leadfortaleza.com.br
 Av. Santos Dumont, 2456 - 1906 | 60150162 - Fortaleza. CE

```
<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

<!-- Aqui são listadas todas as entidades -->
<class>br.com.lead.modelo.Filme</class>
<class>br.com.lead.modelo.Autor</class>

<properties>
  <!-- Propriedades JDBC -->
  <property name="javax.persistence.jdbc.driver"
    value="com.mysql.jdbc.Driver" />

    <property                                name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/catalogodefيلمes?useTimezone=true&server
Timezone=UTC" />

  <property name="javax.persistence.jdbc.user" value="root" />
  <property name="javax.persistence.jdbc.password" value="admin" />

  <!-- Configurações específicas do Hibernate -->
  <property                                name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect" />

  <property name="hibernate.hbm2ddl.auto" value="update" />
  <property name="hibernate.show_sql" value="true" />
  <property name="hibernate.format_sql" value="true" />
</properties>
</persistence-unit>
</persistence>
```

Feito essa configuração, o JPA irá conseguir ter acesso ao mapeamento realizado na entidade Autor, e, depois, conseguirá manipular a estrutura do banco, de acordo com o mapeamento realizado na entidade.

Até aqui, você realizou o mapeamento da entidade Autor. Portanto, o passo seguinte é informar que as duas classes, Filme e Autor, possuem um relacionamento. Nessa perspectiva, o relacionamento que caracteriza Filme e Autor é do tipo muitos para um, ou seja, para cada registro da entidade Autor no seu banco, podemos ter vários registros da entidade Filme associados a ele. Assim, como ocorre no mundo

cinematográfico, onde um autor pode criar vários filmes, isso também ocorrerá em sua aplicação! Legal, não é mesmo?

A próxima etapa é criar esse relacionamento. Na classe Filme, você deve criar um atributo do tipo Autor, nomeado autor. Esse atributo deve ser private. Em seguida, crie o método getter e setter desse atributo. Logo acima do atributo criado, informe o tipo de relacionamento. Ah! Como o relacionamento que você está utilizando é do tipo muitos para um, você deve utilizar a anotação @ManyToOne do javax.persistence, combinado?

```
@ManyToOne
private Autor autor;

public Autor getAutor() {
    return autor;
}

public void setAutor(Autor autor) {
    this.autor = autor;
}
```

#Audiodescricao: A imagem mostra o seguinte código:

```
@ManyToOne
private Autor autor;

public Autor getAutor() {
    return autor;
}

public void setAutor(Autor autor) {
    this.autor = autor;
}
```

Feito esse mapeamento, na próxima vez que você executar a sua aplicação, o JPA irá se encarregar de mudar a estrutura do banco, já que foi criada uma nova entidade, e que, agora, a entidade Filme se relaciona com a entidade Autor.

Deu para entender o que foi explanado na introdução desta aula, depois dessas explicações e exemplos? Ótimo! No próximo vídeo, iremos alterar nossa aplicação para persistir também as informações referentes aos autores dos filmes e verificar como o relacionamento, entre as entidades Filme e Autor, funcionará.