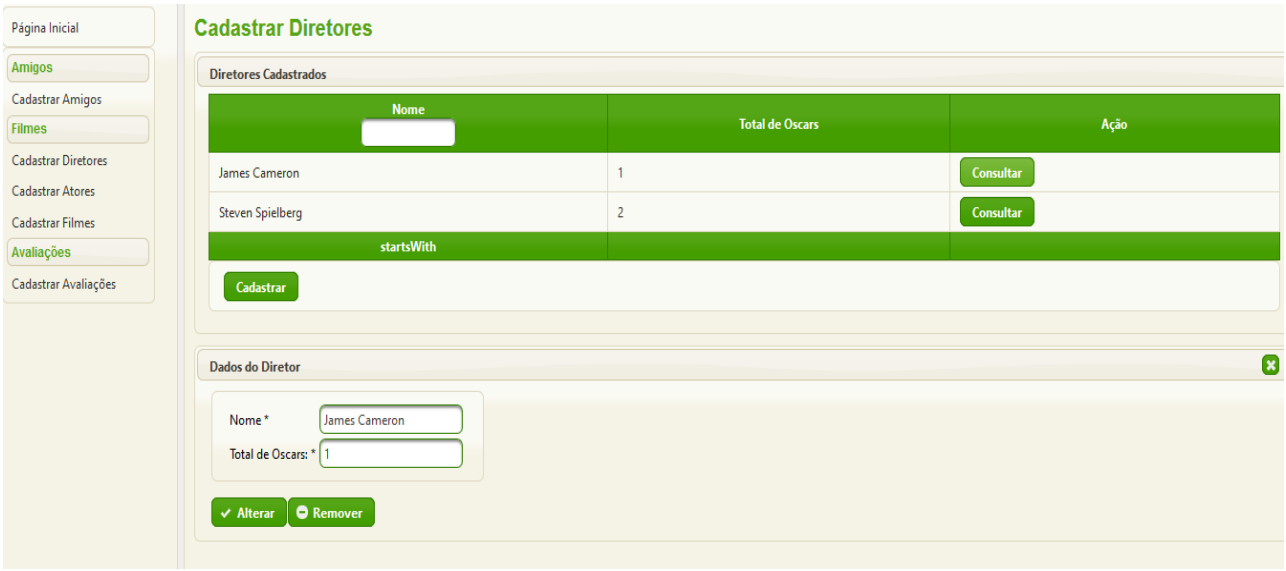
LPIII - Relacionamentos e Autocomplemento

1 – Relacionamentos, Autocomplemento e Data/Hora

Neste tutorial, serão ilustrados somente os conceitos que ainda não apareceram nos Tutoriais 2 e 3. Portanto, se for citado um conceito que já foi utilizado em um Tutorial anterior, como por exemplo um enumerado, você deverá recorrer ao Tutorial 3 para: definir um tipo enumerado; definir uma variável baseada em um tipo enumerado; e utilizar valores de um tipo enumerado para definir um filtro ou para preencher um campo de um formulário.

Nas entidades Ator e Amigo, e nas páginas cadastrarAtores e cadastrarAmigos, não houve alteração.

Na entidade Diretor o atributo boolean ganhadorOscar foi alterado para int totalOscars. Atualização da página cadastrarDiretores:



A entidade Filme é definida com o enumerado Gênero e os com seguintes atributos:

- String título
- int ano
- Gênero gênero
  - ação, aventura, comédia, drama, faroeste, ficção, guerra, infantil, musical, romance, suspense, terror;
- boolean oscarMelhorFilme

Com a criação da entidade Filme, são definidos os relacionamentos:

- Filme [n] : [1] Diretor
  - um filme é dirigido por um diretor
  - um diretor pode dirigir vários filmes
- Filme [n] : [n] Ator
  - em um filme atuam vários atores
  - um ator pode atuar em vários filmes

O relacionamento Filme [n] : [1] Diretor é representado da seguinte forma na entidade Filme:

```
@ManyToOne
 private Diretor diretor;
```

O relacionamento Filme [n] : [1] Diretor é representado da seguinte forma na entidade Diretor:

```
@OneToMany(mappedBy = "diretor")
private List<Filme> filmes;
```

O relacionamento Filme [n] : [n] Ator é representado da seguinte forma na entidade Filme:

```
@ManyToMany
private List<Ator> atores;
```

Em função desse relacionamento deverá ser inserido na classe Filme o método:

```
public void addAtor(Ator ator) { atores.add(ator); }
```

O relacionamento Filme [n] : [n] Ator é representado da seguinte forma na entidade Ator:

```
@ManyToMany (mappedBy = "atores")
private List<Filme> filmes;
```

Observe que nos relacionamentos [n] : [n] e [n] : [1], a representação na segunda entidade é denotada como mapeada previamente (mappedBy) na primeira entidade.

Na página cadastrarFilmes, são utilizados como filtros: (a) os atributos título, ano, gênero, oscarMelhorFilme da entidade Filme; e (b) o atributo nome da entidade Diretor. Na classe FilmeBean, é acrescentada:

- a variável (e seus métodos de leitura e alteração)
  List<Filme> filmesFiltrados
- e o método para gerar as opções que serão utilizadas pelo filtro baseado no atributo oscarMelhorFilme

```
public SelectItem[] getOptionsOscarMelhorFilme() {
    return new SelectItem[]{
        new SelectItem(String.valueOf(""), ""),
        new SelectItem(Boolean.TRUE.toString(), "sim"),
        new SelectItem(Boolean.FALSE.toString(), "não")
    };
}
```

Observe que no método getOptionsOscarMelhorFilme, os valores são convertidos para de boolean para String, dado que o atributo oscarMelhorFilme é do tipo boolean.

Na página cadastrarFilmes: (a) o campo no formulário para definir o atributo oscarMelhorFilme é definido da seguinte forma:

```
<p:selectBooleanCheckbox id="oscarSelectBooleanCheckbox"
                         value="#{filmeBean.value.oscarMelhorFilme}"/>
```

e (b) e o filtro baseado no atributo oscarMelhorFilme é definido da seguinte forma:

```
<p:column filterBy="oscarMelhorFilme"
          headerText="Oscar de Melhor Filme" footerText="exact"
          filterOptions="#{filmeBean.getOptionsOscarMelhorFilme()}"
          filterMatchMode="exact">
    <h:outputText value="#{filme.oscarMelhorFilme?'sim':'não'}"/>
</p:column>
```

Observe que o atributo oscarMelhorFilme é testado para retornar um string em português, para manter coerência com os demais dados da tabela de visualização de filmes.

Um recurso que torna a interface bastante amigável é o autocomplemento, que suporta a selecionação strings a partir de seus prefixos. Esse recurso será utilizado para preencher o campo diretor e o campo atores no formulário de preenchimento dos dados de um filme.

Para que seja possível preencher o campo diretor no formulário de cadastro de filmes, utilizando autocomplemento, é necessário: (a) acrescenter um método, que utiliza SQL para acessar objetos no banco de dados, na classe DiretorService;

```
public List<Diretor> filter(String prefix) {
    prefix += "%";
    Query query = em.createQuery
        ("SELECT diretor from Diretor diretor WHERE diretor.nome LIKE :prefix");
    query.setParameter("prefix", prefix);
    return query.getResultList();
}
```

e (b) acrescentar um método, que será utilizado na página cadastrarFilmes, na classe DiretorBean.

```
public List<Diretor> completaDiretores(String prefixo) { return diretorService.filter(prefixo); }
```

Para que seja possível preencher o campo atores no formulário de cadastro de filmes, utilizando autocomplemento, é necessário criar métodos equivalentes nas classes AtorService e AtorBean.

A utilização de um atributo da classe Diretor como filtro na tabela de visualização de filmes na página cadastrarFilme fica da seguinte forma:

```
<p:column filterBy="diretor" headerText="Nome do Diretor" footerText="startsWith">
    <h:outputText value="#{filme.diretor.nome}" />
</p:column>
```

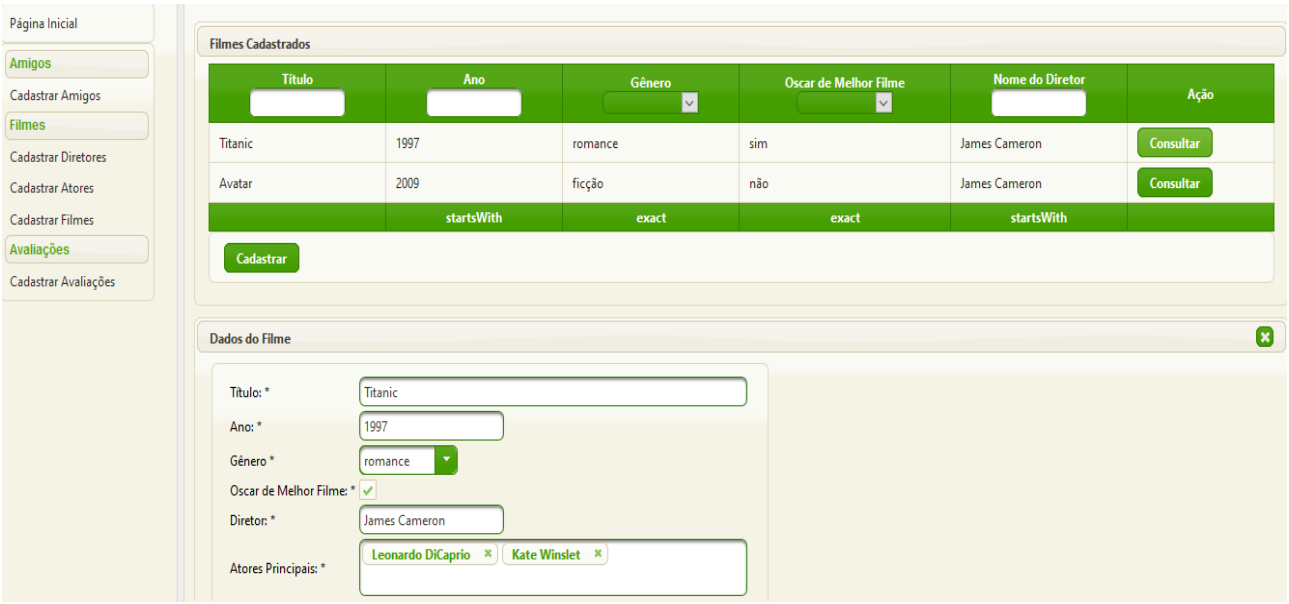
O preenchimento do campo diretor com autocomplemento, na página cadastrarFilmes, fica da seguinte forma:

```
<p:autoComplete id="diretorAutoComplete" value="#{filmeBean.value.diretor}"
                completeMethod="#{diretorBean.completaDiretores}"
                converter="#{diretorConverter}"
                var="diretor" itemValue="#{diretor}" itemLabel="#{diretor.nome}" />
```

O preenchimento do campo atores com autocomplemento, na página cadastrarFilmes, fica da seguinte forma (parâmetro adicional: multiple) :

```
<p:autoComplete id="atorAutoComplete" value="#{filmeBean.value.atores}"
                completeMethod="#{atorBean.completaAtores}" multiple="true"
                converter="#{atorConverter}"
                var="ator" itemValue="#{ator}" itemLabel="#{ator.nome}" />
```

A seguir uma ilustração da página cadastrarFilmes:



Para finalizar a aplicação web ClubeCinemaOR, será acrescentada a entidade Avaliação, com o enumerado Classificação e os seguintes atributos:

- @Temporal(TemporalType.TIME) private Date dataHora = new Date()
- Classificação classificaçãoTrama, classificaçãoDireção, classificaçãoAtuação
  - péssima, fraca, regular, boa, excelente
- boolean entreMeusDezPreferidos

Com a criação da entidade Avaliação, são definidos os relacionamentos:

- Avaliação [n] : [1] Amigo
  - uma avaliação é realizada por um amigo
  - um amigo pode realizar várias avaliações
- Avaliação [n] : [1] Filme
  - uma avaliação é sobre um filme
  - um filme pode ter tido várias avaliações

Na classe Avaliação aparecem dois relacionamentos:
@ManyToOne private Amigo amigo;
@ManyToOne private Filme filme;

Na classe Amigo o relacionamento complementar é:
@OneToMany(mappedBy = "amigo")
private List<Avaliação> avaliações;

De forma similar, na entidade Filme:
@OneToMany(mappedBy = "filme")
private List<Avaliação> avaliações;

Na página cadastrarAvaliações são utilizados filtros baseados nos atributos: (a) nome da entidade Amigo; (b) título e gênero da entidade Filme; e (c) classificaçãoTrama e entreMeusDezPreferidos da entidade Avaliação.

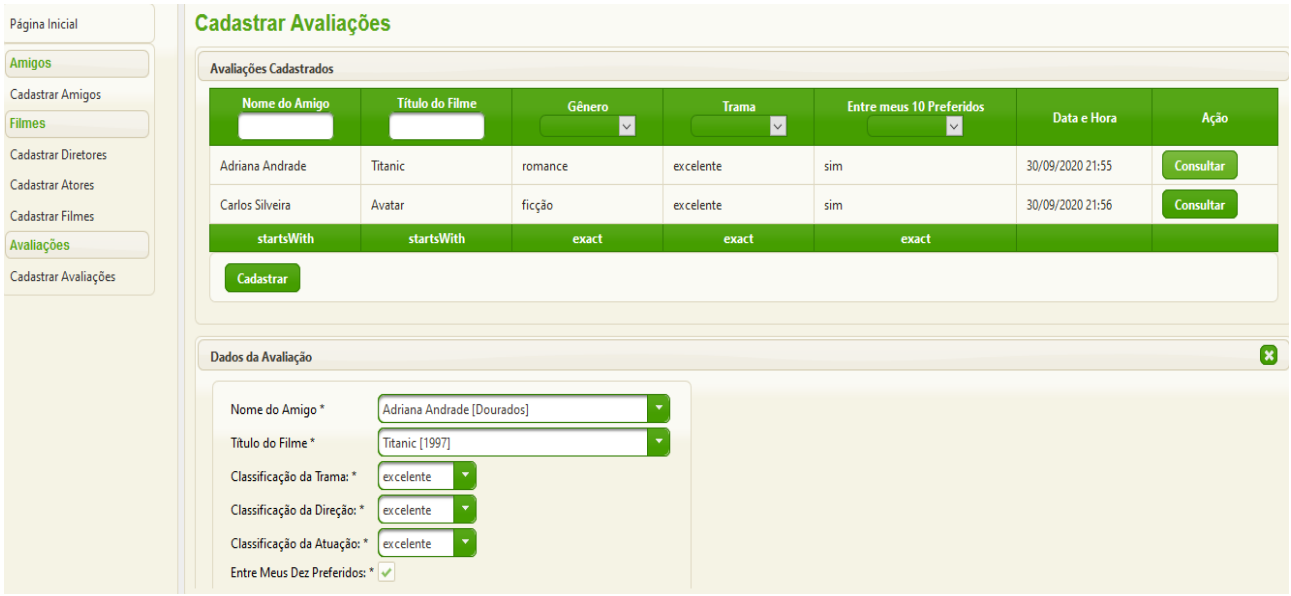
Adicionalmente, na tabela de visualização de avaliações é acrescentada a coluna com data e hora:

```
<p:column headerText="Data e Hora">
    <h:outputText value="#{avaliação.dataHora}" >
        <f:convertDateTime pattern="dd/MM/yyyy HH:mm" locale="pt_BR"/>
    </h:outputText>
</p:column>
```

A escolha do nome do amigo (e título do filme idem) é da seguinte forma:

```
<p:selectOneMenu id="amigoSelectOneMenu" value="#{avaliaçãoBean.value.amigo}"
                 required="false" label="Cliente" style="width:300px">
    <f:selectItems value="#{amigoBean.all}" />
</p:selectOneMenu>
```

A seguir, uma ilustração da página cadastrarAvaliações:



O horário obtido automaticamente, no momento do cadastro está adiantado em cerca de 4 horas.

Embora, nessa aplicação não esteja sendo utilizada a classe Endereço, é relevante comentar sobre relacionamento com essa classe, porque ela é muito genérica e, portanto, muito utilizada.

Suponha que nessa aplicação, no cadastro do amigo, fosse incluída um aba, com campos do seu endereço. Os campos seriam agrupados em uma entidade Endereço: logradouro, número, complemento, bairro, cidade, cep.

Somente a classe EndereçoConverter é criada em decorrência da criação da entidade Endereço. As classes associadas nos pacotes beans e services não são criadas. Na entidade Endereço não é incluído nenhum relacionamento.

Na entidade Amigo, é incluído o relacionamento:
@OneToOne(cascade = CascadeType.ALL)
private Endereço endereço;

No formulário da página cadastrarAmigos, o campo logradouro será utilizado para ilustrar como o valor preenchido será armazenado na classe Endereço:

```
<p:inputText id="logradouroInputText" required="false" label="Logradouro"
             value="#{amigoBean.value.endereço.logradouro}" />
```

O método getValue do objeto amigoBean retorna um objeto da entidade Amigo. A partir desse objeto é chamado o método de leitura da variável endereço: getEndereço. Esse método precisa retornar um objeto da entidade Endereço para ser preenchido e, portanto, deve ser definido da seguinte forma:

```
public Endereço getEndereço() {
    if (endereço == null) endereço = new Endereço();
    return endereço;
}
```