

Disciplina Regular 3

Desenvolvimento Web com Java EE

Graduação em Engenharia de Software - 2020

Etapa 4 Aula 1

Java Persistence API

Para acompanhar pelo Moodle você
deve estudar a etapa 6

Etapa 4 Aula 1 - Competências

- **Competências Trabalhadas Nesta Etapa**
 - **Compreender e utilizar o Java Persistence API.**
 - **Dominar as técnicas de geração de código “bottom-up”.**
 - **Utilizar a sintaxe JQL para consultas.**
 - Implementar relacionamentos entre entidades.
 - Implementar transações declarativas.

Etapa 4 Aula 1 - Checklist

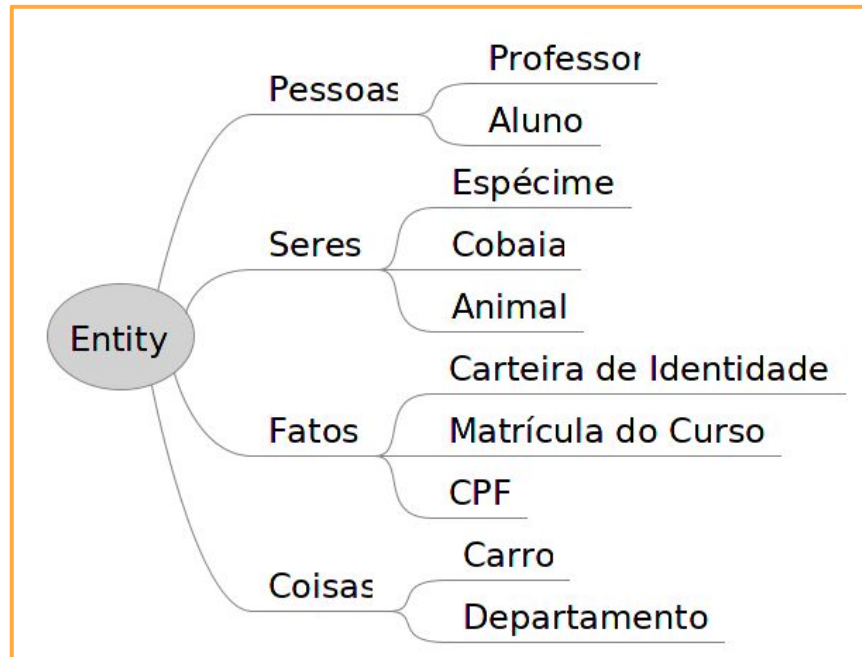
- **Checklist DR3:**

- Ter revisado os slides e **os exercícios**, principalmente **Spring MVC**.
- Ter entregue o **TP1** ontem.

- **Checklist PB:**

- Ter iniciado a construção do caso de uso “Manter Cadastro de Clientes” para o dia **11/5/2020**.
 - Inclusão, Consulta e Alteração.
- Ter iniciado os estudos sobre “Sistemas de Recomendação” e as 2 técnicas apresentadas ontem.
- Ter iniciado os ajustes na especificação do módulo, considerando a aula de ontem.

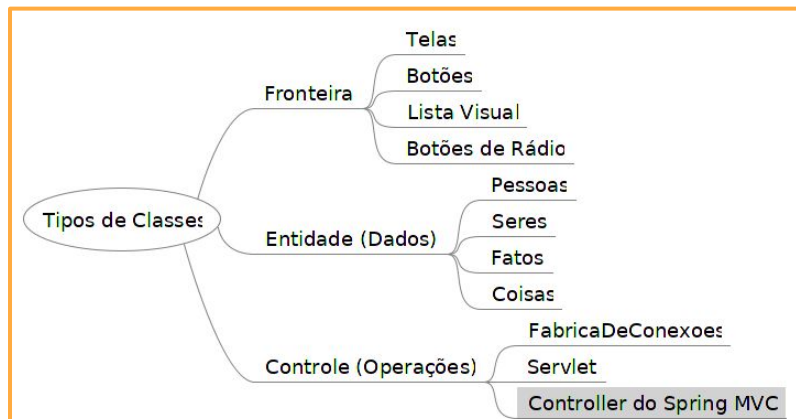
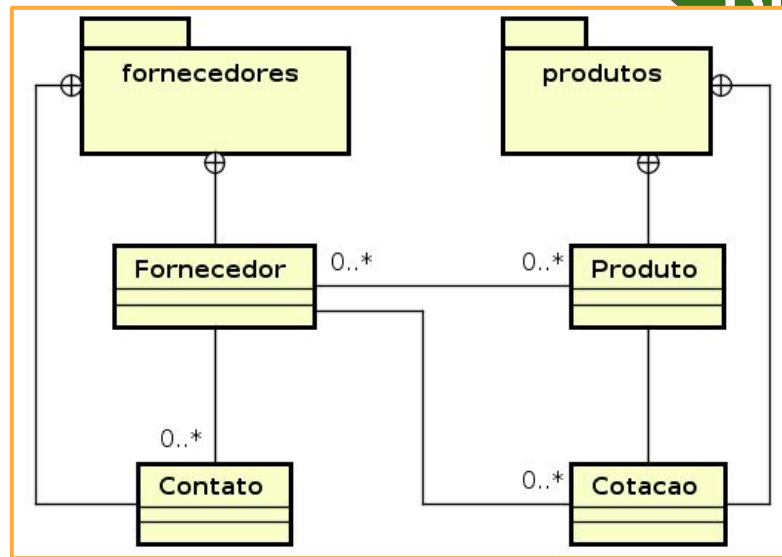
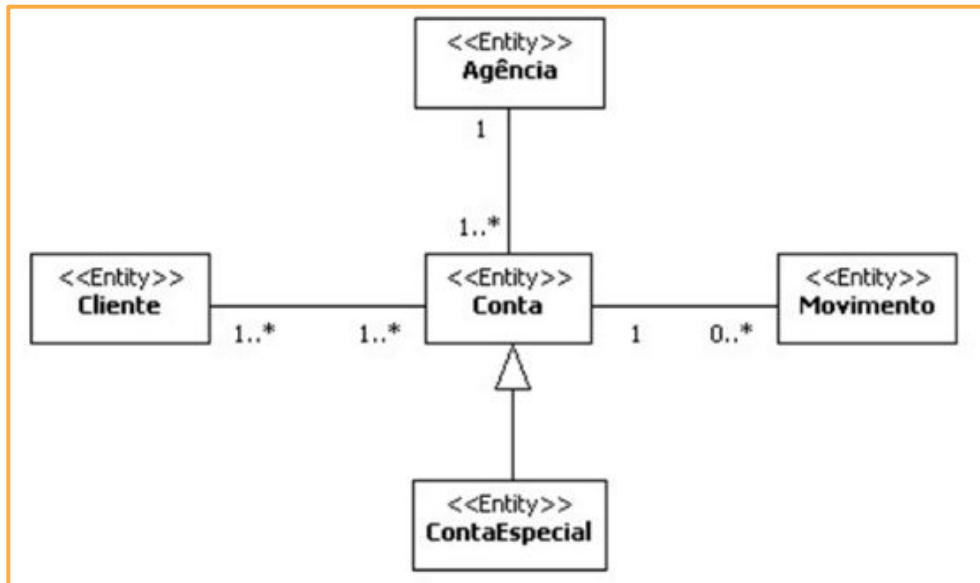
Entity / Entities



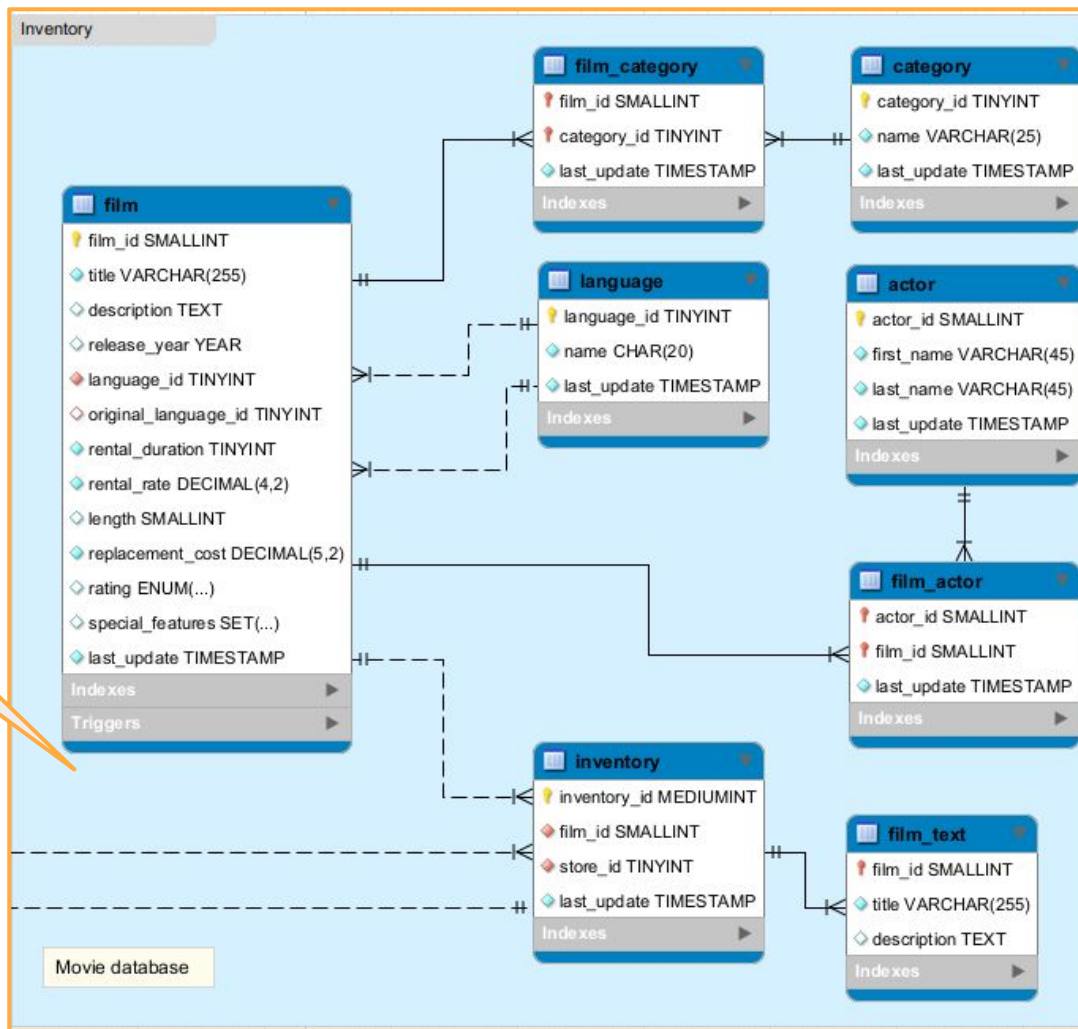
O que são Entities?

- São objetos Java comuns (POJO) que representam estados de persistência.
- Uma classe de entidade pode representar uma tabela e cada instância pode ser uma linha da tabela.
- Extenso uso de anotações dentro da classe.
- Facilidade de codificação.

Exemplos



Banco de
Dados
Sakila
(parcial)



Implementação

- Uma classe de entidade (entity class) deve:
 - Ser anotado com `@Entity`.
 - Ter um construtor padrão (default).
 - Implementar `Serializable` para acesso remoto.
 - Ter seus atributos acessados por `get` e `set`.
 - Não pode ter atributos públicos.
 - Não pode ser usado o modificador “final”.
 - **Se a tabela tiver um nome diferente do nome da classe usar a anotação `@Table(name=“nome”)` antes da classe.**

Atributos (bd) e Propriedades (pojo)

- O estado de persistência pode ser um atributo ou propriedade JavaBean (get/set).
- Campos não persistentes devem ser anotados com `@Transient`.
- Propriedades ou atributos que representem conjuntos devem ser de alguma classe que implemente a interface `Collection` – deve-se usar `List` preferencialmente devido a sua robustez.
- Podem ser alterados pela anotação `@Column`

Chave Primária

- Toda entidade deve ter uma chave primária (chave viva = tem significado ou **surrogate key = autonumeração**).
- O atributo/propriedade deve ser anotado com @Id. Se for do tipo auto-numerável também deve ser anotado com @GeneratedValue.
- Chaves compostas (tipo de chave viva) devem ser definidas em suas próprias classes – se possível devemos evitá-las.
- O atributo/propriedade que representa uma chave composta deve ser anotado com @Id ou @EmbeddedId.

Exemplo

@Entity

```
public class Cliente implements Serializable {
```

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

```
private Long id;
```

```
private String nome;
```

```
public Long getId() {return this.id; }
```

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

```
public String getNome() {return nome; }
```

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

```
}
```

Exemplo

@Embeddable

```
public class ContaPK implements Serializable {
```

```
    private long idAgencia;
```

```
    private long idConta;
```

```
    public long getIdAgencia() { return idAgencia; }
```

```
    public void setIdAgencia(long idAgencia) { this.idAgencia =  
idAgencia; }
```

```
    public long getIdConta() { return idConta; }
```

```
    public void setIdConta(long idConta) { this.idConta = idConta; }
```

```
    @Override
```

```
    public int hashCode() ...
```

```
    @Override
```

```
    public boolean equals(Object object) ...
```

```
}
```

Relacionamentos

- Existem quatro tipos de multiplicidade e uma anotação para cada um:
 - **1 para 1**: anotação @OneToOne.
 - **1 para muitos**: anotação @OneToMany.
 - **Muitos para 1**: anotação @ManyToOne.
 - **Muitos para muitos**: anotação @ManyToMany.
- Possuem o atributo **mappedBy** que indica o campo da entidade que é “dona” do relacionamento (exceto ManyToOne).
- Para OneToOne e OneToMany existe o atributo **cascade** para propagar uma operação para os filhos: REMOVE, PERSIST, MERGE, REFRESH, ALL.

Exemplo

- Agência – Conta: 1 para N e N para 1.

- Na classe Agência:

```
@OneToMany(mappedBy="agencia", cascade=CascadeType.ALL)  
private Collection<Conta> contas = new ArrayList<>();
```

- Na classe Conta:

```
@ManyToOne  
private Agencia agencia;
```

Exemplo

- Cliente – Conta: N para N.

- Na classe Cliente:

```
@ManyToMany(mappedBy="clientes")  
private Collection<Conta> contas = new ArrayList<>();
```

- Na classe Conta:

```
@ManyToMany(mappedBy="contas")  
private Collection<Cliente> clientes = new ArrayList<>();
```


Exemplo

- Cliente – DetalhesCliente: 1 para 1.

- Na classe Cliente

```
@OneToOne
```

```
@PrimaryKeyJoinColumn
```

```
DetalhesCliente detalhes;
```

- Na classe DetalhesCliente

```
@Id
```

```
private Long id;
```

Persistência

EntityManager (interface)

- Gerenciador de Entities: EntityManager.
- Possui um contexto para criar, persistir e remover Entities denominado **Persistence Context**.
- Dois tipos: **container-managed** (é propagado para todos os componentes de uma transação) e **application-managed**.
- Usando o Spring implementamos o **container-managed**.

Obtendo um EntityManager

- Para obter o gerenciador em um container-managed entity:

```
@PersistenceContext  
EntityManager manager;
```



- Para obter o gerenciador em um application-managed entity:

```
@PersistenceUnit  
EntityManagerFactory fabrica;  
  
EntityManager m = fabrica.createEntityManager();
```

Operações

- **Busca pela chave primária:**

```
Livro livro = manager.find(Livro.class, isbn);
```

- **Persistência (criação):**

```
Livro livro = new Livro(...);  
manager.persist(livro);
```

- **Persistência (atualização):**

```
manager.merge(livro);
```

- **Remoção:**

```
manager.remove(livro);
```

- **Sincronização:**

```
manager.flush();
```

Consultas

- Método createQuery (JPA Query Language):

```
public List findWithNome(String parteDoNome) {  
  
    return manager.createQuery(  
        "SELECT a FROM Livro a WHERE a.nome LIKE  
:parametro")  
        .setParameter("parametro", parteDoNome)  
        .setMaxResults(50)  
        .getResultList();  
  
}
```

Consultas

- Método `createNamedQuery` (queries nomeadas):

```
@NamedQuery(  
    name="findAllWithTitulo",  
    query="SELECT a FROM Livro a WHERE  
           a.titulo LIKE ?1" )  
  
...  
List livros =  
    manager.createNamedQuery("findAllWithTitulo")  
        .setParameter(1, "Java")  
        .getResultList();
```

ContaRepository

```
@PersistenceContext
private EntityManager em;

public void inserir(Conta conta) { em.persist(conta); }

public void atualizar(Conta conta) { em.merge(conta); }

public void remover(Conta conta) {
    em.merge(conta);
    em.remove(conta);
}

public Conta obter(Object pk) { return (Conta) em.find(Conta.class, pk); }

public List listar() {
    return em.createQuery("select object(o) from Conta as o")
        .getResultList();
}
```



Receita para o Spring MVC com JPA - 1 de 2

- Criar o Projeto:
 - Criar o projeto web+maven normalmente: JEE 6 + JSE 8.
 - Lembrar de colocar os projetos em um diretório "raso" (com nome curto) para facilitar os backups.
- Configurar a Aplicação Web:
 - Criar a pasta WEB-INF e o deployment descriptor (web.xml).
 - Copiar e colar os arquivos de configuração do Spring.
 - **applicationContext.xml**
 - dispatcher-servlet.xml
 - Configurar o Dispatcher Servlet no web.xml.

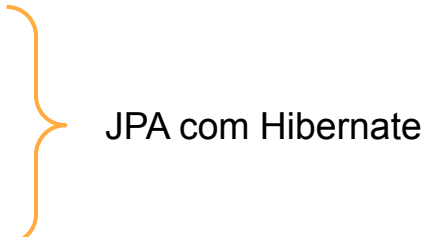


```
11 <!-- ===== -->
12 <!-- Configurações do JDBC DataSource -->
13 <!-- ===== -->
14 <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
15
16     <property name="driverClassName" value="com.mysql.cj.jdbc.Driver" />
17     <property name="url" value="jdbc:mysql://localhost:3306/sakila" />
18     <property name="username" value="root" />
19     <property name="password" value="root" />
20
21 </bean>
22 <!-- ===== -->
23 <!-- Configurações do Gerenciador de Entidades -->
24 <!-- ===== -->
25 <bean id="entityManagerFactory"
26     class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
27     p:packagesToScan="br.edu.infnet"
28     p:dataSource-ref="dataSource">
29
30     <property name="jpaVendorAdapter">
31         <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
32             <!-- Aqui vão as configurações do Hibernate -->
33             <property name="generateDdl" value="false" />
34             <property name="showSql" value="true" />
35             <property name="databasePlatform" value="org.hibernate.dialect.MySQL5Dialect"/>
36         </bean>
37     </property>
38
39 </bean>
```

Receita para o Spring MVC com JPA - 2 de 2

- Dependências de Bibliotecas do Maven:
 - "org.springframework:**spring-webmvc**"
 - "org.hibernate:**hibernate-validator**"
 - "javax.servlet:**jstl**"

Spring MVC

 - "org.hibernate:**hibernate-entitymanager**"
 - "mysql:**mysql-connector-java**"
 - "org.springframework:**spring-orm**"

JPA com Hibernate