Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte
Campus Natal-Central
Diretoria Acadêmica de Gestão e Tecnologia da Informação

Tecnologia em Análise e Desenvolvimento de Software

Herança em JPA

Prof. Fellipe Aleixo
fellipe.aleixo@ifrn.edu.br

ORM e Herança

- Para estar completo o mecanismo de mapeamento objeto-relacional precisa suportar o conceito de herança da orientação a objetos
- A JPA é compatível com:
  - Herança de entidade
  - Relacionamentos/associações polimórficas
  - Consultas polimórficas

ORM e Herança

- São disponibilizadas três maneiras diferentes de mapear herança para o banco de dados:
  - Uma única tabela por hierarquia de classe
    - Uma única tabela terá todas as propriedades de cada classe na hierarquia
  - Uma tabela por classe concreta
    - Cada classe terá uma tabela dedicada a ela, com todas as suas propriedades e as propriedades de sua superclasse
  - Uma tabela por subclasse
    - Cada classe terá a sua própria tabela – esta tabela terá apenas as propriedades definidas para esta classe

Tabela Única por Hierarquia de Classe

- Uma tabela do banco de dados representa todas as classes de uma dada hierarquia
- Exemplo no sistema da biblioteca:

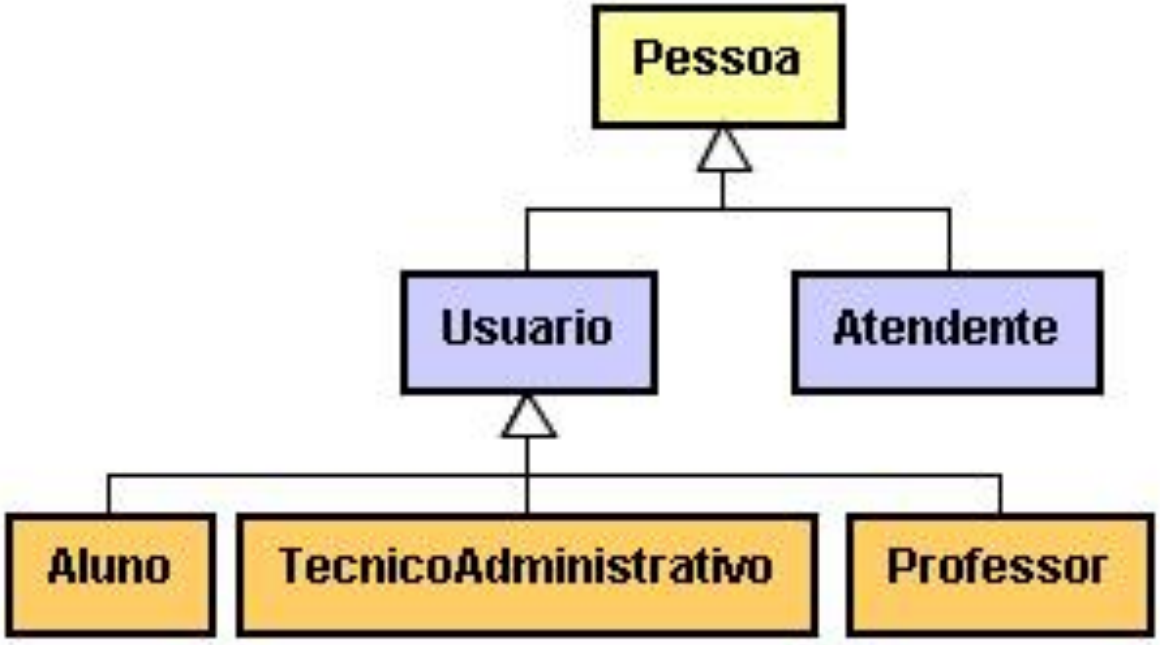


Tabela Única por Hierarquia de Classe

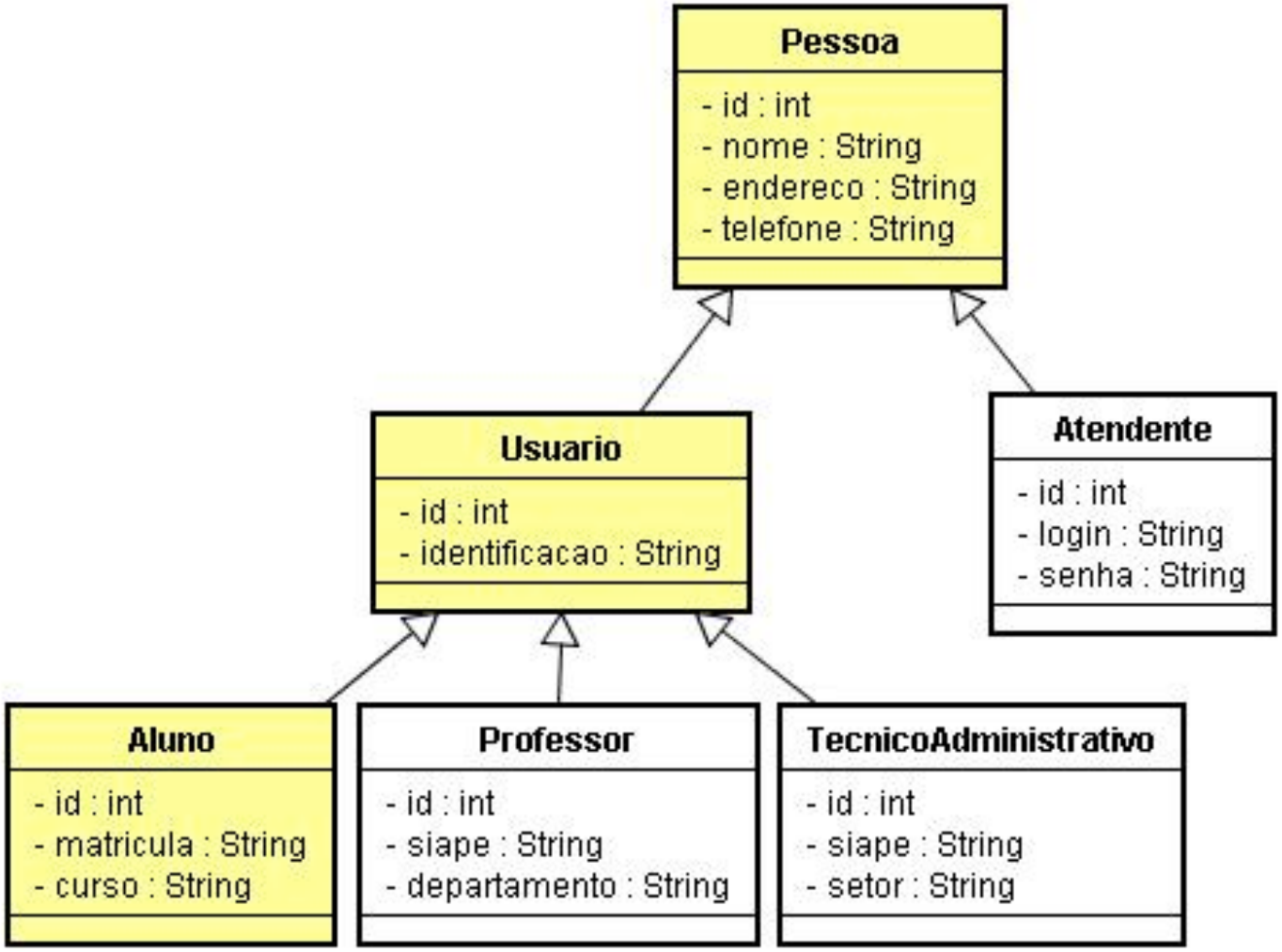
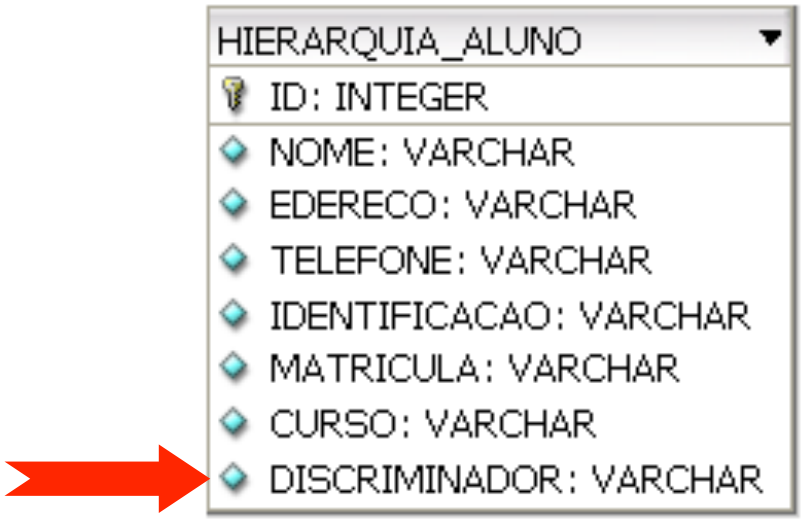


Tabela Única por Hierarquia de Classe

- Nesse exemplo, as entidades Pessoa, Usuario e Aluno são representadas na mesma tabela, como a mostrada a seguir



  - A coluna "disciminador" identifica o tipo de entidade

Tabela Única por Hierarquia de Classe

```
@Entity
@Table(name="HIERARQUIA_ALUNO")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="DISCRIMINADOR",
                     discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue("PESSOA")
public class Pessoa {
        private int id;
        private String nome;
        private String endereco;
        private String telefone;
        ...
        @Id
        @GeneratedValue
        public int getId() { return id; }
        public void setId(int novo) { id = novo; }
        ...
}
```

Tabela Única por Hierarquia de Classe

```
@Entity
@DiscriminatorValue(“USUARIO”)
public class Usuario extends Pessoa {
        private String identificacao;
        ...
        // nesse caso não precisa especificar o “id” pois é herdado
        public String getIdentificacao() {
                return identificacao;
        }
        public void setIdentificacao(String nova) {
                identificacao = nova;
        }
}
```

Tabela Única por Hierarquia de Classe

```
@Entity
// Utiliza o valor padrão do discriminador
public class Aluno extends Usuario {
        private String matricula;
        private String curso;
        ...
        // nesse caso não precisa especificar o “id” pois é herdado
        public String getMatricula() { return matricula; }
        public void setMatricula(String nova) { matricula = nova; }
        public String getCurso() { return curso; }
        public void setCurso(String novo) { curso = novo; }
}
```

Tabela Única por Hierarquia de Classe

- Anotações utilizadas:
  - @javax.persistence.Inheritance
  - @javax.persistence.DiscriminatorColumn
  - @javax.persistence.DiscriminatorValue

```
@Target(TYPE) @Retention(RUNTIME)
public @interface Inheritance {
        InheritanceType strategy() default SINGLE_TABLE;
}

public enum InheritanceType {
        SINGLE_TABLE, JOINED, TABLE_PER_CLASS
}
```

Tabela Única por Hierarquia de Classe

```
@Target(TYPE) @Retention(RUNTIME)
public @interface DescriminatorColumn {
        String name() default “DTYPE”;
        DiscriminatorType discriminatorType() default STRING;
        // o discriminatorType pode ser também CHAR ou INTEGER
        String columnDefintion() default “”;
        int length() default 10;
}
```

```
@Target(TYPE) @Retention(RUNTIME)
public @interface DiscriminatorValue {
        String value();
}
```

Tabela Única por Hierarquia de Classe

```
<entity-mappings>
        <entity class="exemplo.biblioteca.modelo.Pessoa">
                <inheritance strategy="SINGLE_TABLE"/>
                <discriminator-column name="DISCRIMINADOR"
                                      discriminatorType="STRING"/>
                <discriminator-value>PESSOA</discriminator-value>
                <attributes>
                        <id name="id">
                                <generated-value/>
                        </id>
                </attributes>
        </entity>
        <entity class="exemplo.biblioteca.modelo.Usuario">
                <discriminator-value>USUARIO</discriminator-value>
        </entity>
        <entity class="exemplo.biblioteca.modelo.Aluno"/>
</entity-mappings>
```

Tabela Única por Hierarquia de Classe

- Vantagens:
  - É a mais simples de implementar no banco de dados
  - Possui um melhor desempenho (não usa junções)
- Desvantagens:
  - Todas as colunas de propriedade de subclasse precisam permitir valores nulos
  - Em alguns casos as colunas das propriedades de subclasses não são utilizadas – desperdício
  - A estratégia SINGLE_TABLE não é normalizada

Tabela por Classe Concreta

- Uma tabela é definida para cada classe concreta, as colunas representam:
  - Propriedades específicas e das superclasses

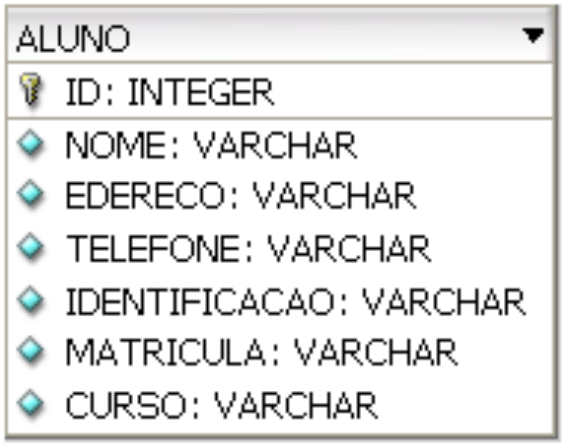


Tabela por Classe Concreta

- Não é necessária uma coluna discriminadora

```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Pessoa {
        ...
}

@Entity
public class Usuario extends Pessoa {
        ...
}

@Entity
public class Aluno extends Usuario {
        ...
}
```

Tabela por Classe Concreta

- Mapeamento equivalente em XML:

```
<entity-mappings>
        <entity class="exemplo.biblioteca.modelo.Pessoa">
                <inheritance strategy="TABLE_PER_CLASS"/>
                <attributes>
                        <id name="id">
                                <generated-value/>
                        </id>
                </attributes>
        </entity>
        <entity class="exemplo.biblioteca.modelo.Usuario"/>
        <entity class="exemplo.biblioteca.modelo.Aluno"/>
</entity-mappings>
```

Tabela por Classe Concreta

- Vantagens:
  - Podem ser definidas restrições nas propriedades das subclasses
  - Mais fácil mapear um sistema legado preexistente
- Desvantagens:
  - Estratégia não normalizada (redundância)
  - São necessárias múltiplas consultas ao carregar um relacionamento polimórfico (piora o desempenho)
  - Contraindicada pela especificação

Tabela por Subclasse

- Cada subclasse tem a sua própria tabela
  - Sem propriedades das superclasses e subclasses
  - Acontece a junção das tabelas (JOINED)

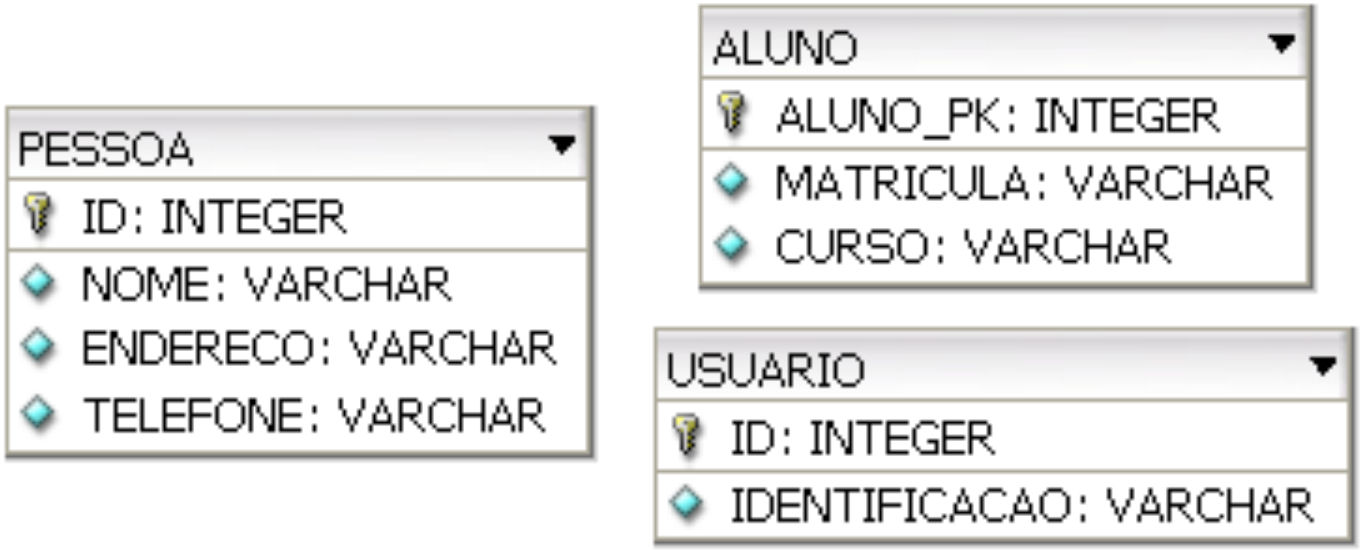


Tabela por Subclasse

- Quando o gerenciador de persistência carrega uma entidade que é uma subclasse ou atravessa um relacionamento polimórfico:
  - Ele faz uma junção SQL em todas as tabelas na hierarquia
  - Deve haver uma coluna em cada tabela que possa ser utilizada para unir cada tabela
  - No exemplo: ALUNO, USUARIO e PESSOA compartilham o mesmo valor de chave primária

Tabela por Subclasse

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
public class Pessoa {
        ...
}

@Entity
public class Usuario extends Pessoa {
        ...
}

@Entity
@PrimaryKeyJoinColumn(name="ALUNO_PK")
public class Aluno extends Usuario {
        ...
}
```

Tabela por Subclasse

- A anotação que define a chave primária a ser utilizada para a junção é a @PrimaryKeyJoinComlumn

```
@Target({TYPE, METHOD, FIELD})
public @interface PrimaryKeyJoinColumn
{
        String name() default “”;
        String referencedColumnName() default “”;
        String columnDefinition() default “”;
}
```

  - Se os nomes da coluna de chave primária forem idênticos entre a classe base e subclasses, essa anotação não é necessária

Tabela por Subclasse

- Mapeamento equivalente em XML:

```
<entity-mappings>
        <entity class="exemplo.biblioteca.modelo.Pessoa">
                <inheritance strategy="JOINED"/>
                <attributes>
                        <id name="id">
                                <generated-value/>
                        </id>
                </attributes>
        </entity>
        <entity class="exemplo.biblioteca.modelo.Usuario"/>
        <entity class="exemplo.biblioteca.modelo.Aluno">
                <primary-key-join-column name="ALUNO_PK"/>
        </entity>
</entity-mappings>
```

Tabela por Subclasse

- Vantagens:
  - É possível a definição de restrições em qualquer coluna de qualquer tabela (ex.: propriedades que podem ser nulas)
  - Modelo normalizado
- Desvantagens:
  - Desempenho pior que SIGLE_TABLE

Herança de Classes Não-entidades

- JPA permite que possa haver herança a partir de classes que não são entidades
  - A superclasse pode ser uma classe do modelo de domínio que não se deseja transformar em entidade
  - Utilizada a anotação @MappedSuperClass
  - Para ilustrar, vamos alterar a hierarquia exemplo e transformar a classe Pessoa em não-entidade

Herança de Classes Não-entidades

```
@MappedSuperClass
public class Pessoa {
        private int id;
        private String nome;
        private String endereco;
        private String telefone;
        ...
        @Id
        @GeneratedValue
        public int getId() { return id; }
        public void setId(int novo) { id = novo; }
        ...
}
```

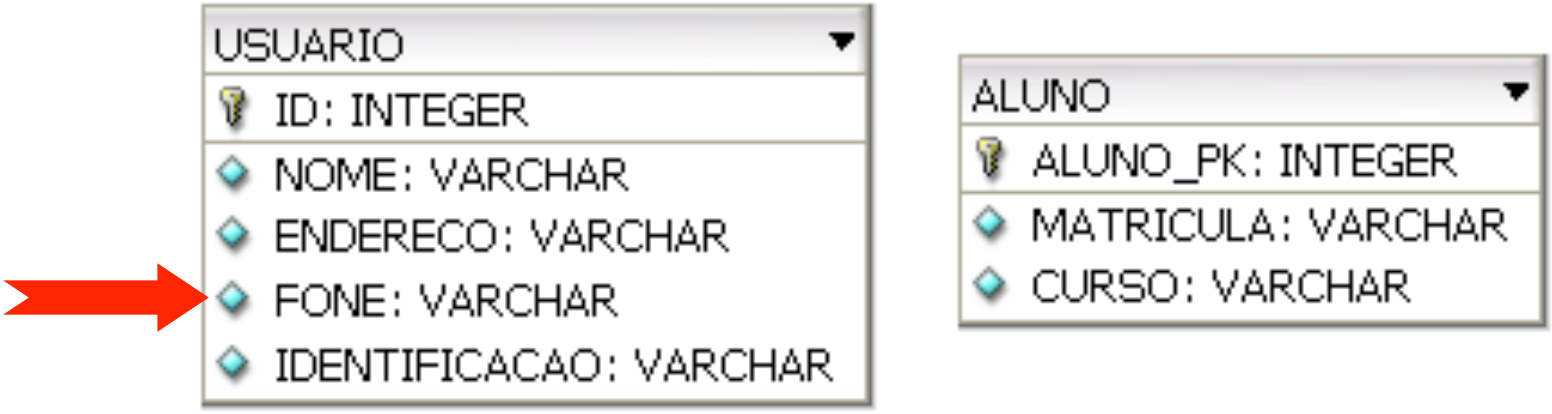
Herança de Classes Não-entidades

```
@Entity
@Table(name="USUARIO")
@Inheritance(strategy=InheritanceType.JOINED)
@AttributeOverride(name="telefone", column=@Column(name="FONE"))
public class Usuario extends Pessoa {
        private String identificacao;
        ...
        // nesse caso não precisa especificar o "id" pois é herdado
        public String getIdentificacao() {
                return identificacao;
        }
        public void setIdentificacao(String nova) {
                identificacao = nova;
        }
}
```

Herança de Classes Não-entidades

```
@Entity
@Table(name="ALUNO")
@PrimaryKeyJoinColumn(name="ALUNO_PK")
public class Aluno extends Usuario {
        ...
}
```

- Resultado:



Herança de Classes Não-entidades

- Como Pessoa não é uma entidade – não é mapeada para tabela do banco de dados
  - Estratégia usada quando se tem uma classe "base" e não se deseja forçar que esta seja uma entidade
- Qualquer propriedade de uma classe base pode ser sobrescrita através da anotação @AttributeOverride

Herança de Classes Não-entidades

```
<entity-mappings>
        <mapped-superclass class="exemplo.biblioteca.Pessoa">
                <attributes>
                        <id name="id">
                                <generated-value/>
                        </id>
                </attributes>
        </mapped-superclass>
        <entity class="exemplo.biblioteca.Usuario">
                <inheritance strategy="JOINED"/>
                <attribute-override name="telefone">
                        <column name="FONE"/>
                </attribute-override>
        </entity>
        <entity class="exemplo.biblioteca.Aluno">
                <primary-key-join-column name="ALUNO_PK"/>
        </entity>
</entity-mappings>
```