SENAI

Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

Herança

Desenvolvimento de Sistemas

Prof. Me. Reneilson Santos

Abril/2024

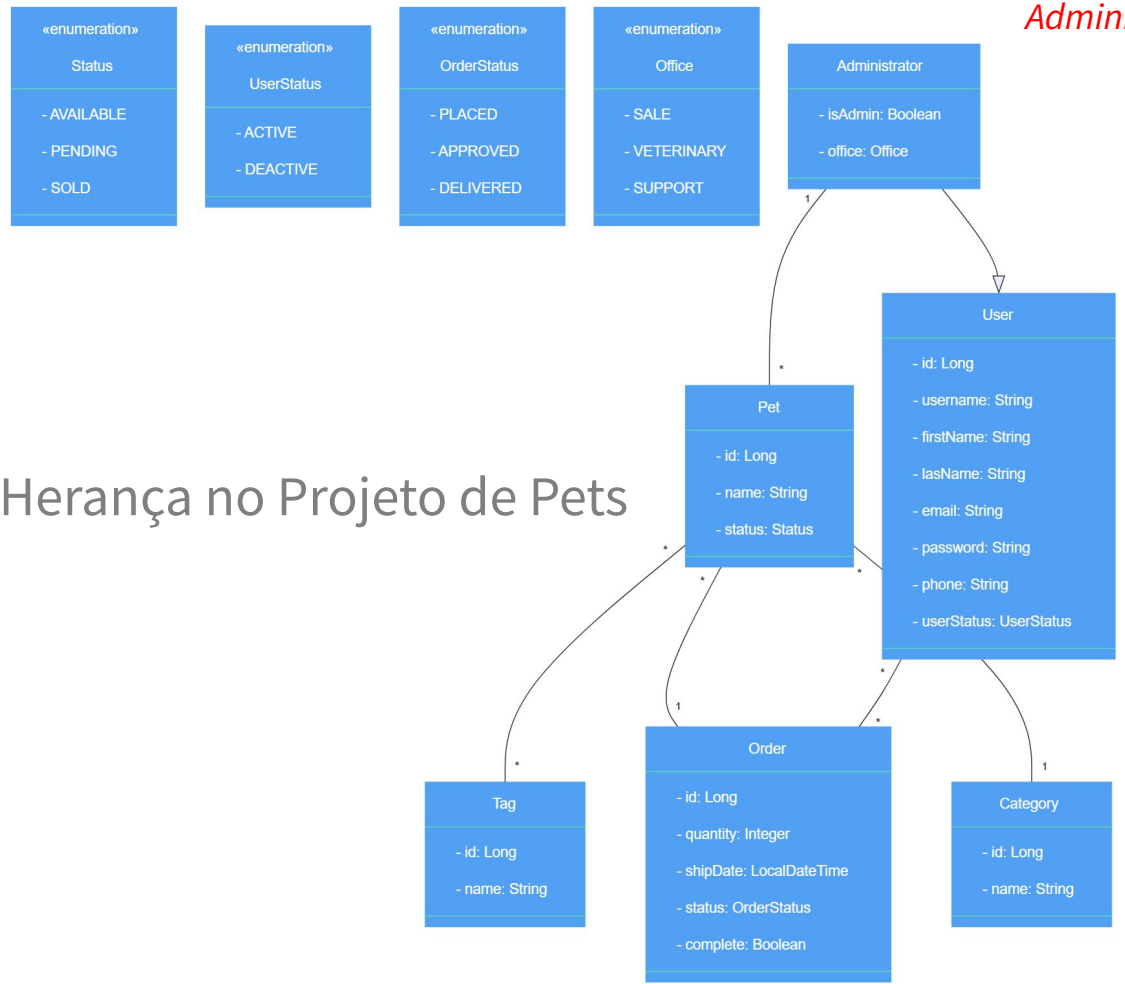
SENAI
PELO FUTURO DO TRABALHO

Agenda

- Herança
  - *MappedSuperclass*
  - *Single Table*
  - *Joined Table*
  - *Table per Class*
- Bibliografia

Administrator é uma especialização de User

Herança no Projeto de Pets



Herança

Herança

Bancos de dados relacionais não têm uma maneira direta de mapear hierarquias de classes em tabelas de banco de dados. Para resolver isso, a especificação JPA fornece várias estratégias:

MappedSuperclass – as classes pai não podem ser entidades (não geram tabela própria).

Single Table – As entidades de diferentes classes com uma superclasse comum são colocadas em **uma única tabela**.

Joined Table – Cada classe tem sua tabela, e consultar uma entidade de subclasse requer **unir as tabelas**.

Table per class – todas as propriedades de uma classe estão em sua tabela, portanto, nenhuma junção é necessária.

MappedSuperclass

MappedSuperclass

Usando a estratégia MappedSuperclass, a herança é evidente apenas na classe, mas não na tabela da entidade no banco de dados. É como se fosse feito um *embedded* dos dados da classe mãe nas classes filhas.

No nosso caso (do projeto de Pet) não podemos criar uma estrutura com a anotação MappedSuperclass em Usuario pois essa classe possui relacionamento com Order, logo, precisa existir uma entidade no banco para gerar o relacionamento.

Single Table

Single Table

Usando a estratégia Single Table, uma única tabela será gerada (a tabela da superclasse) e todos os atributos das subclasses serão adicionadas na tabela da superclasse.

É criado automaticamente um tipo (DTYPE) que identificará aquela entidade como uma das subclasses.

```java
import jakarta.persistence.Inheritance;
import jakarta.persistence.InheritanceType;

@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class User {
   @Id @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String name;
   ...
}
```

```java
@Entity
public class Administrator extends User{
    private Boolean isAdmin;
    private Office office;
   ...
}
```

Nesse caso, Administrator não precisa mais da anotação “Table” já que não gerará tabela alguma e **nem do id**.

Single Table

Caso queiramos propor os próprios valores para o DTYPE com um nome diferente e um tipo diferente (inteiro ou string) podemos usar a anotação **@DiscriminatorColumn** passando os parâmetros **name** e **discriminatorType** que pode ser do INTEGER ou STRING do enumerador *DiscriminatorType* e nas subclasses colocar a anotação **DiscriminatorValue** com o valor entre parênteses e aspas duplas (mesmo se for inteiro).

```
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="user_type", discriminatorType = DiscriminatorType.STRING)
public class User { ... }
```

```
@Entity
@DiscriminatorValue("administrator")
public class Administrator extends User { ... }
```

Single Table - Problemas

O problema da estratégia do Single Table é que gerará uma tabela com vários campos que serão nulos no banco de dados.

Por exemplo, se tivéssemos um exemplo no qual a herança fosse de pessoa física e jurídica para usuário, teríamos casos em que poderíamos ter uma tabela única com vários valores nulos.

A medida que aumenta o número de subclasses, a tabela vai aumentando, o que geralmente não é uma boa estratégia.

A Single Table é recomendada quando o número de subclasses for fixo e a quantidade de parâmetros neles não for muito grande.

Joined Table

Joined Table

Usando a estratégia Joined Table os únicos campos que serão replicados em todas as tabelas será o id, que será usado para realizar o join das tabelas quando for necessário capturar os dados de uma subclasse.

Só é possível adicionar um Administrator se já tiver o ID do Usuario devido a obrigatoriedade da chave estrangeira idUser.

```java
@Entity
@Table(name="usuarios")
@Inheritance(strategy = InheritanceType.JOINED)
public class User{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    ...
}
```

```java
@Entity
public class Administrator extends User {
    private Boolean isAdmin;
    private Office office;
    ...
}
```

Joined Table - Problemas

O problema da estratégia Joined Table é que ela sempre precisará de um Join na busca pelos dados das subclasses.

Sempre que for buscar por Administrator, no nosso exemplo, será preciso fazer o join com a tabela de User, o que pode deteriorar a performance do nosso sistema.

Table per Class

Table per Class

A estratégia ***Table per Class*** mapeia cada entidade para sua tabela, que contém todas as propriedades da entidade, inclusive as herdadas.

O esquema resultante é semelhante ao que usa @MappedSuperclass. Mas Table per Class de fato definirá entidades para classes pai, permitindo associações.

```java
@Entity
@Table(name="usuarios")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class User{
    @Id @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;
    ...
}
```

A estratégia IDENTITY para a opção TABLE_PER_CLASS não é possível, pois, na TABLE_PER_CLASS o id de todas as classes que fazem parte da herança deve ser sequencial, logo, não dá para usar o IDENTITY na classe pai. Pode-se usar o TABLE ou SEQUENCE, sendo a segunda opção com melhor performance.

```java
@Entity
public class Administrator extends User {
    private Boolean isAdmin;
    ...
}
```

Table per Class - Problemas

O problema da estratégia Table per Class acontece devido a cada tabela ter todos os dados, logo, ao capturar os dados da tabela gerada pela classe pai, será necessário fazer a união de todas as tabelas da classe filho, que não é performático, muito pior que a realização dos joins, portanto, se tiver relacionamentos existentes no projeto que acontecem com a classe pai, deve-se evitar a estratégia Table per Class, a não ser que a quantidade de dados previstos para as subclasses sejam pequenas, de forma a não impactar a execução de um UNION no SQL.

Atividade

Enviar screenshot das tabelas User e Administrator (quando existirem) para cada uma das anotações de herança:

- @SingleTable
- @JoinedTable
- @TablePerClass

Bibliografia

Bibliografia

- Hibernate Inheritance. Disponível em: https://www.baeldung.com/hibernate-inheritance. Acessado em: 14 de março de 2023.