Desenvolvimento para Servidores-II

Herança em JPA

Neste tópico abortaremos o uso de JPA para mapear a herança

Prof. Ciro Cirne Trindade

Herança

- Para trabalhar com herança o JPA fornece 4 alternativas:
  - Mapped superclass
  - Single table
  - Joined
  - Table per class

Mapped superclass

- Uma classe anotada como `@MappedSuperclass` pode ser mapeada de forma semelhante a uma entidade, entretanto os mapeamentos se aplicam somente a suas subclasses já que nenhuma tabela será criada para a própria superclasse
- *Mapped superclasses* não podem ser anotadas com `@Entity` ou `@Table`

AbstractEntity.java (1/2)

```
package br.fatec.financas.model;

import java.io.Serializable;
import java.util.Objects;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.MappedSuperclass;

@MappedSuperclass
public abstract class AbstractEntity implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    public Long getId() {
        return id;
    }

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

AbstractEntity.java (2/2)

```
    @Override
    public int hashCode() {
        int hash = 7;
        hash = 53 * hash +
                  Objects.hashCode(this.id != null ? this.id : 0);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null || getClass() != obj.getClass())
            return false;
        final AbstractEntity other = (AbstractEntity) obj;
        return Objects.equals(this.id, other.id);
    }

}
```

Conta.java

```
@Table(name = "tb_conta")
@Entity
public class Conta extends AbstractEntity {
   private static final long serialVersionUID = 1L;

   @Column(name = "nm_titular", length = 100)
   private String titular;
   @Column(name = "nr_agencia")
   private Integer agencia;
   @Column(name = "nm_numero", length = 10)
   private String numero;
   @Column(name = "vl_saldo")
   private Float saldo;


   public Conta() { }

    // getters e setters

}
```

Mapped superclass

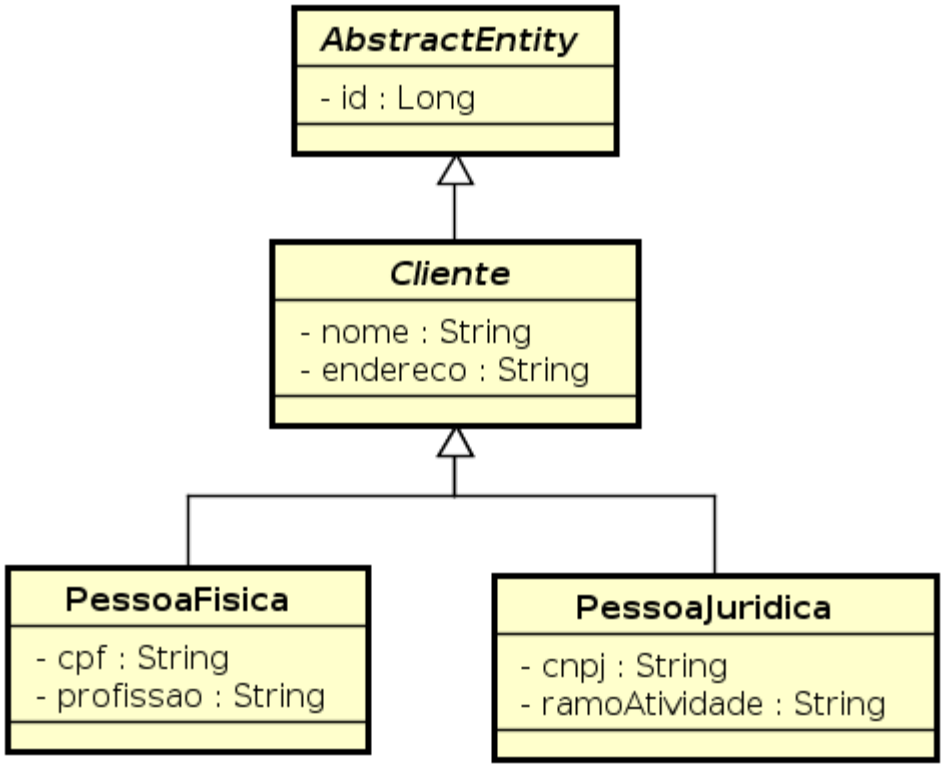
- A classe `Conta` herda da classe `AbstractEntity` os atributos e métodos que lá existirem
- Uma *Mapped superclass* não é persistida, pois não é uma entidade

Single Table

- É possível ter toda a hierarquia de classes de herança persistida em uma única tabela
- Para tal usamos a seguinte anotação na superclasse:
  - `@Inheritance(strategy = InheritanceType.SINGLE_TABLE)`

Single Table

- Vamos considerar a seguinte hierarquia de classes



Cliente.java

@DiscriminatorColumn indica o nome da coluna na tabela que determinará a "dona" da coluna no banco de dados, se não for especificado, será criada uma coluna `DTYPE` para esse fim

```
@Entity
@Table(name="tb_cliente")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="nm_pertence_a_classe", length=20)
public abstract class Cliente extends AbstractEntity {
    private static final long serialVersionUID = 1L;
    @Column(name="nm_nome", length=60)
    private String nome;
    @Column(name="ds_endereco", length=120)
    private String endereco;

    public Cliente() { }

    // getters e setters
}
```

PessoaFisica.java

@DiscrimatorValue informa o valor que irá identificar a classe na tabela, se não for definido o valor será o nome da classe

```
@Entity
@DiscriminatorValue("Pessoa Física")
public class PessoaFisica extends Cliente {
    private static final long serialVersionUID = 1L;

    @Column(name="cd_cpf", length = 11)
    private String cpf;
    @Column(name="nm_profissao", length = 30)
    private String profissao;

    public PessoaFisica() { }

    // getters e setters
}
```

PessoaJuridica.java

```
@Entity
@DiscriminatorValue("Pessoa Jurídica")
public class PessoaJuridica extends Cliente {
    private static final long serialVersionUID = 1L;

    @Column(name="cd_cnpj", length=14)
    private String cnpj;
    @Column(name="nm_ramo_atividade", length=20)
    private String ramoAtividade;

    public PessoaJuridica() {
    }

    // getters e setters

}
```

Single Table

- Seria gerada uma tabela `tb_cliente` com a seguinte estrutura:

id	BIGINT(20)
nm_pertence_a_classe	VARCHAR(20)
nm_nome	VARCHAR(60)
ds_endereco	VARCHAR(120)
cd_cpf	VARCHAR(11)
nm_profissao	VARCHAR(30)
cd_cnpj	VARCHAR(14)
nm_ramo_atividade	VARCHAR(20)

Joined

- A estratégia `JOINED` utiliza a abordagem de uma tabela para cada entidade da herança
- É especificada pela seguinte anotação na superclasse:
  - `@Inheritance(strategy = InheritanceType.JOINED)`

Exemplo de Joined

```
@Entity
@Table(name="tb_cliente")
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Cliente extends AbstractEntity
{ ... }

@Entity
@Table(name="tb_pessoa_fisica")
public class PessoaFisica extends Cliente { ... }

@Entity
@Table(name="tb_pessoa_juridica")
public class PessoaJuridica extends Cliente { ... }
```

Joined

- Gera 3 tabelas com as seguintes estruturas:
  - `tb_cliente: id, nm_nome, ds_endereco`
  - `tb_pessoa_fisica: id, cd_cpf, nm_profissao`
  - `tb_pessoa_juridica: id, cd_cnpj, nm_ramo_atividade`

A coluna `id` nas tabelas `tb_pessoa_fisica` e `tb_pessoa_juridica` são chave estrangeiras que referenciam o `id` da tabela `tb_cliente`

Table per class

- *Table per class* trabalha com a ideia de uma tabela para cada entidade representada por classe **concreta**
- É especificada pela seguinte anotação na superclasse:
  - `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)`
- É necessário alterar a estratégia de geração das chaves primárias de `IDENTITY` para **`TABLE`**

  - Garante que chaves distintas serão geradas para as subclasses

Exemplo de Table per class

```
@Entity
@Inheritance(strategy =
                InheritanceType.TABLE_PER_CLASS)
public abstract class Cliente extends AbstractEntiy
{ ... }

@Entity
@Table(name="tb_pessoa_fisica")
public class PessoaFisica extends Cliente { ... }

@Entity
@Table(name="tb_pessoa_juridica")
public class PessoaJuridica extends Cliente { ... }
```

Table per class

- Gera 2 tabelas com as seguintes estruturas:
  - `tb_pessoa_fisica: id, nm_nome, ds_endereco, cd_cpf, nm_profissao`
  - `tb_pessoa_juridica: id, nm_nome, ds_endereco, cd_cnpj, nm_ramo_atividade`

Referências

- CORDEIRO, G. *Aplicações Java para web com JSF e JPA*. São Paulo: Casa do Código, 2012.
- GEARY, David; HORSTMANN, Cay. *Core JavaServer Faces*. 3. ed., Prentice-Hall, 2010.
- ORACLE Corporation. *The Java EE 7 Tutorial*. Disponível em: https://docs.oracle.com/javaee/7/JEETT.pdf, 2014.