

Classes de Domínio

Classes de Domínio

- Por default, todos os campos de uma classe de domínio são persistidas no banco de dados
- Para tipos simples como Strings e Integers, cada propriedade da classe será mapeada para uma coluna de uma tabela
- Para tipos complexos serão necessários tabelas
- Toda classe mapeada para uma tabela receberá um campo adicional, o id, que será a chave primária

Eduardo Mendes de Oliveira - edumendes@gmail.com

Validando classes de domínio

- Toda aplicação tem regras de negócios que restringem os valores válidos de uma propriedade de classe
 - Idade não pode ser < 0
 - Email deve possuir @ e .
 - O número de um cartão de crédito deve seguir um padrão
- Regras como estas deve estar em um local específico

Eduardo Mendes de Oliveira - edumendes@gmail.com

Validando as propriedades da Musica

```
class Musica {
    String titulo
    String artista
    Integer duracao
    Date lancamento

    static constraints = {
        titulo(blank: false)
        artista(blank: false)
        duracao(min:1)
    }
}
```

Eduardo Mendes de Oliveira - edumendes@gmail.com

Constraints

- | | |
|--------------|-------------|
| ■ blank | ■ notEqual |
| ■ creditCard | ■ nullable |
| ■ email | ■ range |
| ■ inList | ■ scale |
| ■ matches | ■ size |
| ■ max | ■ unique |
| ■ maxSize | ■ url |
| ■ min | ■ validator |
| ■ minSize | |

OBS: A validação ocorre quando o método save de um objeto é chamado

Eduardo Mendes de Oliveira - edumendes@gmail.com

Propriedades Transientes

- Por default, toda propriedade de uma classe de domínio é persistida no banco de dados
- E quando existirem propriedades que não necessitam ser persistidas?
- **static transients**

Eduardo Mendes de Oliveira - edumendes@gmail.com

Exemplos

```
package jctunes

class Album {
    String titulo
    String nomeFalso

    static transients = ['nomeFalso']

    ...
}
```

Outro exemplo

```
class Pessoa {
    BigDecimal saldo
    BigDecimal limiteEspecial

    BigDecimal getSaldoDisponivel () {
        saldo+limiteEspecial
    }
    static transients = ['saldoDisponivel']
    ...
}
```

Relacionamentos

```
class Carro {
    Motor motor
}

class Motor {
    Carro carro
}
```

Não há relação de pertença

Relacionamentos

```
class Carro {
    Motor motor
}

class Motor {
    static belongsTo = [carro:Carro]
}
```

Outra forma

```
class Carro {
    Motor motor
}

class Motor {
    static belongsTo = Carro
}
```

Qual a diferença

- `static belongsTo = [carro:Carro]`
 - A classe Motor tem sua própria referência para carro
- `static belongsTo = Carro`
 - A classe Motor não possui referência para carro

belongsTo

- A propriedade que pertence sofrerá exclusões em cascata
- Grails saberá que o Motor pertence a um Carro, então sempre que um Carro for excluído do banco, juntamente será excluído o Motor

Outra classe

- Artista
 - Propriedades
 - nome
 - albuns
- Album pertencerá a um Artista
 - Com referência
- Música pertencerá a um Album
 - Sem referencia

Classe Artista

```
class Artista {
    String nome

    static hasMany = [albuns:Album]
}
```

Classe Album

```
class Album {
    String titulo

    static hasMany = [musicas:Musica]

    static belongsTo = [artista:Artista]
}
```

Musica

```
class Musica{
    String titulo
    Integer duracao
    Date lancamento

    static belongsTo = Album
}
```

Classe Artista

```
class Artista {
    String nome

    static hasMany =
        [albuns:Album, instrumentos:Instrumento]
}
```