Laboratório de Engenharia de Software

JPA - Hibernate

Daniel Mendonça da Silva

danielmend@les.inf.puc-rio.br

Agenda

- Introdução
- Mapeando Relações
  - Tipos de relações
  - Relações objeto vs. relacional
    - Um para um
    - Um para muitos
    - Muitos para muitos
- Mapeando herança
  - Tabela por hierarquia
  - Tabela por classe concreta
  - Tabela por classe
  - Comparação das estratégias

Java Persistence API (JPA)

- API fornece uma abstração facíl de usar sobre o JDBC
- API define as interfaces usadas nos EntityBeans permitindo uma abstração também sobre a implementação da JPA
- Já existe várias implementações da JPA
  - Hibernate
  - TopLink
  - Kodo

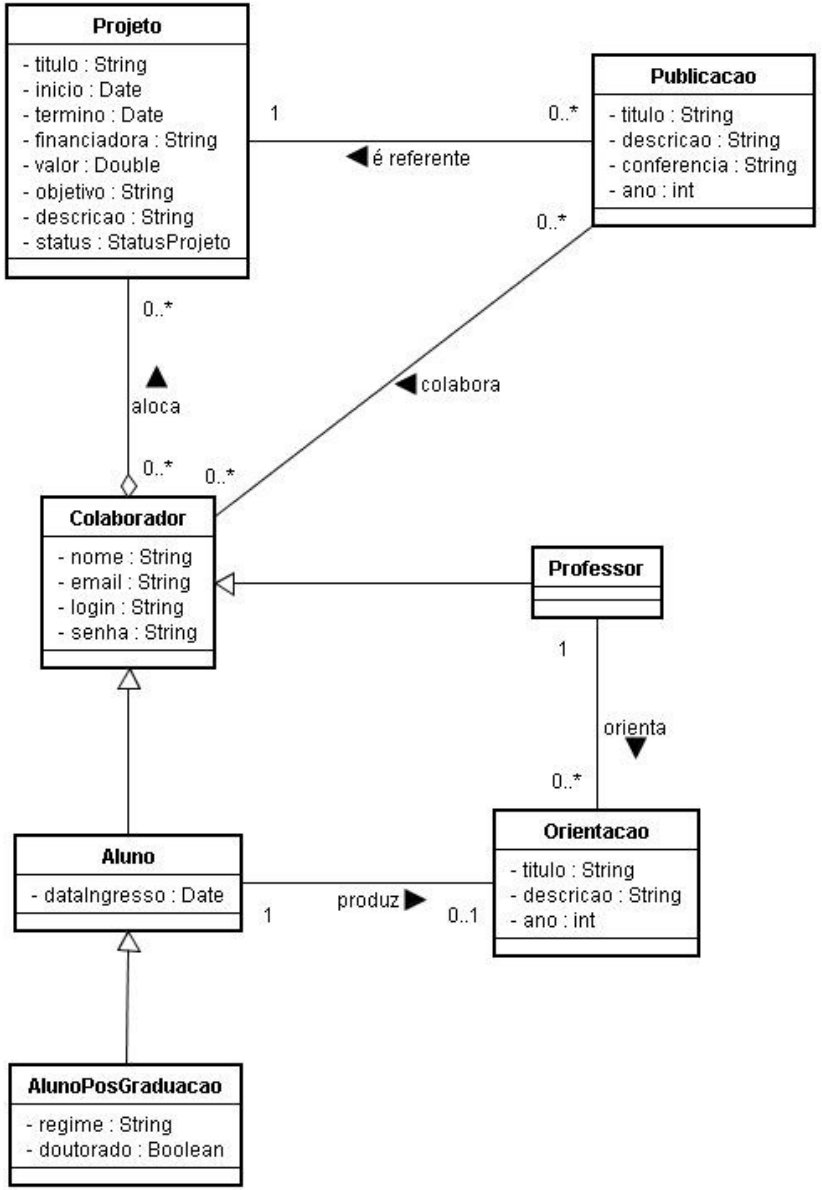
Hibernate – O que é?

- Ferramenta para mapeamento O/R em Java
  - Uma das mais difundidas
- Implementa a JPA
- Transparência
- Independência quanto ao tipo de base de dados
- Consulta de dados
  - HQL, Criteria Queries, SQL, JPA QL
- Extensível

Objetos Persistentes

- Implementam entidades lógica do negócio
- POJOs (Plain Old Java Object)
  - Construtor padrão
  - Possui métodos de acesso – gets/sets (opcional)
- Possui um atributo identificador (recomendado)
- Classes não final (recomendado)
- Recomenda-se implementar Serializable

Modelo


Projeto
- titulo : String
- inicio : Date
- termino : Date
- financiadora : String
- valor : Double
- objetivo : String
- descricao : String
- status : StatusProjeto
Publicacao
- titulo : String
- descricao : String
- conferencia : String
- ano : int
1
0..*
é referente
0..*
0..*
aloca
colabora
0..*
0..*
Colaborador
- nome : String
- email : String
- login : String
- senha : String
Professor
1
orienta
0..*
Aluno
- dataIngresso : Date
1
produz
0..1
Orientacao
- titulo : String
- descricao : String
- ano : int
AlunoPosGraduacao
- regime : String
- doutorado : Boolean

Modelo - Exemplo

```
package br.puc.rio.inf.les.prds.sgpa.modelo;

import java.io.Serializable;

public class Orientacao implements Serializable {
	private Long id;
	private String descricao;
	private String titulo;
	private int ano;
	private Professor professor;
	private Aluno aluno;

	public String getDescricao() {
		return descricao;
	}
	public void setDescricao(String descricao) {
		this.descricao = descricao;
	}
	public String getTitulo() {
		return titulo;
	}
	public void setTitulo(String titulo) {
		this.titulo = titulo;
	}
	public int getAno() {
		return ano;
	}
	public void setAno(int ano) {
		this.ano = ano;
	}
	public Professor getProfessor() {
		return professor;
	}
	public void setProfessor(Professor professor){
		this.professor = professor;
	}
	public Aluno getAluno() {
		return aluno;
	}
	public void setAluno(Aluno aluno) {
		this.aluno = aluno;
	}
	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
}
```

Mapeamento

- Devemos informar ao Hibernate como relacionar o modelo de objetos com o modelo relacional

- Arquivos de mapeamento vs. Annotations

Mapeando as classes

- Declaração de entidade
  - Anotação @Entity
  - Indicação do nome da table através do @Table (opcional)

```
@Entity
@Table(name = "orientacao")
public class Orientacao implements Serializable {
...
}
```

Mapeamento de Identificadores

- Indicação da chave primária
  - @Id
- Escolha do gerador da Chave
  - @GeneratedValue
  - Estratégias: AUTO, SEQUENCE, IDENTITY e TABLE

```
@Entity
@Table(name = "orientacao")
public class Orientacao implements Serializable {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;
}
```

Mapeamento de Propriedades

- Mapeamento de atributos primitivos não são necessários
- Opcionalmente todo atributo pode ser anotado com @Column
  - name
  - lenght
  - nullable
  - unique
  - ...

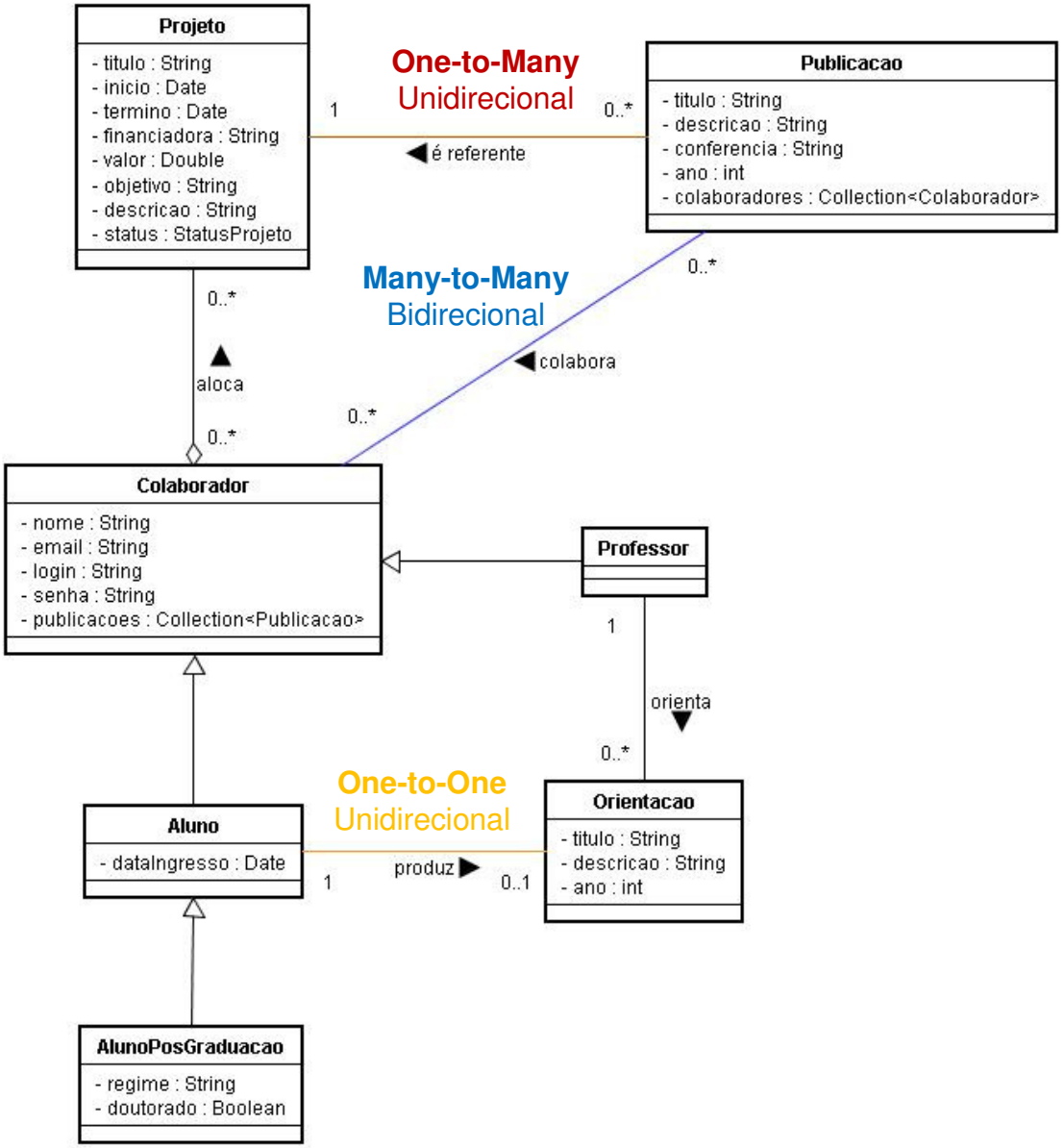
```
@Entity
@Table(name = "orientacao")
public class Orientacao implements Serializable {
        @Column(name = "descricao", length = 250, nullable = false, unique = true)
        private String descricao;
}
```

Mapeamento de Propriedades

- Para datas (Date ou Calendar) é necessário a anotação @Temporal para definir precisão:
  - TemporalType. DATE
  - TemporalType. TIME
  - TemporalType. TIMESTAMP

```
@Entity
public class Aluno implements Serializable {
        @Temporal(TemporalType.DATE)
        @Column(name = "data_ingresso")
        private Date dataIngresso;
}
```

Mapeamento de Relacionamentos



Mapeamento – One-to-One

- Três formas
  - Relação com chave estrangeira (Código 1)
  - Relação por chave primária (Código 2)
  - Com tabela associativa (Código 3)

```
@Entity
@Table(name = "orientacao")
public class Orientacao implements Serializable {
                @OneToOne
                @JoinColumn(name = "aluno_id", unique = true)
                private Aluno aluno;
}// Código 1
```

```
@Entity
@Table(name = "orientacao")
public class Orientacao implements Serializable {
                @OneToOne
                @PrimaryKeyJoinColumn(name = "aluno_id")
                private Aluno aluno;
}// Código 2
```

```
@Entity
@Table(name = "orientacao")
public class Orientacao implements Serializable {
                @OneToOne
                @JoinTable(name="orientacao_aluno",
                                joinColumns = @JoinColumn(name = "orientacao_id"),
                                inverseJoinColumns = @JoinColumn(name = "aluno_id")
                )
                private Aluno aluno;
}// Código 3
```

Mapeamento – One-to-Many

```
@Entity
@Table(name = "projeto")
public class Projeto implements Serializable {
        @OneToMany
        @JoinColumn(name = "projeto_id", unique = true)
        private Collection<Publicacao> publicacoes;
}
```

Mapeamento – Many-to-One


```
@Entity
@Table(name = "orientacao")
public class Orientacao implements Serializable {
        @ManyToOne
        @JoinColumn(name = "professor_id", unique = true)
        private Professor professor;
}
```

Mapeamento – Many-to-Many

```
@Entity
@Table(name = "projeto")
public class Projeto implements Serializable {
        @ManyToMany
        @JoinTable(name = "projeto_colaborador",
                joinColumns = { @JoinColumn(name = "projeto_id") },
                inverseJoinColumns = { @JoinColumn(name = "colaborador_id") }
        )
        private Collection<Colaborador> colaboradores;
}
```

Mapeamento – Bidirecional


```
@Entity
@Table(name = "colaborador")
public class Colaborador implements Serializable {
        @ManyToMany(mappedBy = "colaboradores")
        private Collection<Projeto> projetos;
}
```

Herança – Tabela por hierarquia

```
@Entity
@Table(name = "colaborador")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(discriminatorType = DiscriminatorType.STRING)
@DiscriminatorValue(value = "Colaborador")
public class Colaborador implements Serializable {
...
}

@Entity
@DiscriminatorValue(value = "Aluno")
public class Aluno extends Colaborador implements Serializable {
...
}
```

Herança – Tabela por classe concreta


```
@Entity
@Table(name = "colaborador")
@Inheritance(strategy = Inheritance.TABLE_PER_CLASS)
public class Colaborador implements Serializable {
...
}

@Entity
@Table(name = "Aluno")
public class Aluno extends Colaborador implements Serializable {
...
}
```

Herança – Tabela por subclasse

```
@Entity
@Table(name = "colaborador")
@Inheritance(strategy = Inheritance. JOINED)
public class Colaborador implements Serializable {
...
}

@Entity
@Table(name = "Aluno")
public class Aluno extends Colaborador implements Serializable {
…
}
```

Cascata

- Pode ser definido nos relacionamentos (OneToOne, OneToMany, ManyToOne e ManyToMany)

- Tipos:
  - ALL
  - PERSIST
  - MERGE
  - REMOVE
  - REFRESH

```
@Entity
@Table(name = "colaborador")
public class Colaborador implements Serializable {
        @ManyToMany(cascade = {CascadeType.PERSIST, CascadeType.REMOVE})
        private Collection<Projeto> projetos;
}
```

Perguntas

Perguntas? Dúvidas?