ADS - ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Desenvolvimento Distribuído

Prof. Rafael Guimarães Sakurai

Universidade Metodista de São Paulo | Campus EAD

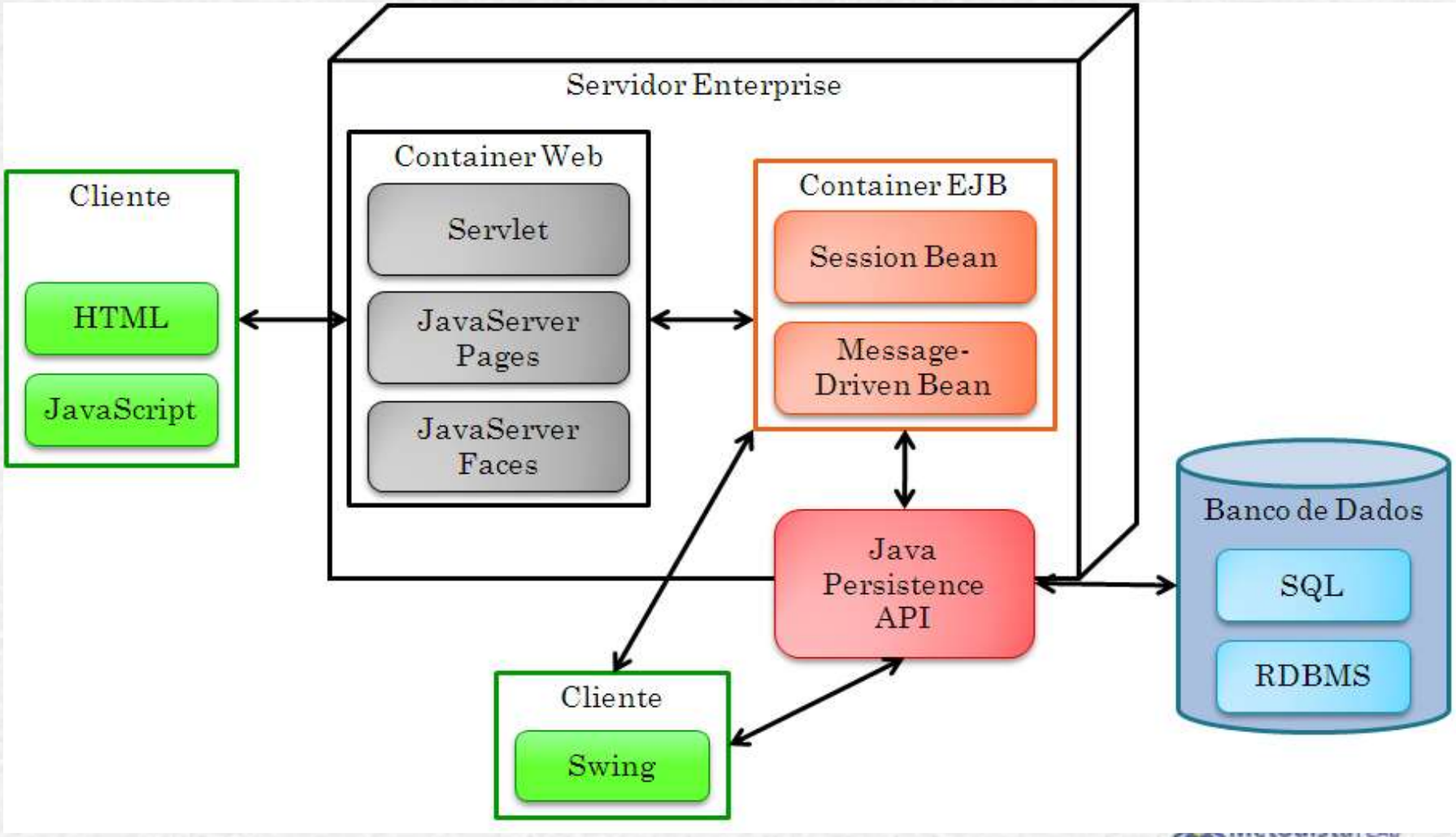
Objetivo da teleaula

Apresentar o uso de chave composta, relacionamento entre entidades e consultas com JPA.

Objetivo do módulo

- Introdução ao desenvolvimento de componente no lado do servidor para sistemas de computação distribuída
- **Conexão com o banco de dados utilizando Java Persistence API 2.0**
- Enterprise JavaBeans 3.0 (EJB)
- Web Services (JAX-WS)
- Web Services REST (JAX-RS)

Exemplo de estrutura de aplicação



Aula passada

- Introdução ao Java Persistence API
- Criação de Entity e seu ciclo de vida
- Uso do EntityManager
- Criação da Unidade de Persistência (persistence.xml)
- CRUD com JPA

Chave Composta

Quando utilizamos chave composta, precisamos separar os atributos que formam a chave composta em uma classe separada e adicionar a anotação @Embeddable.

Chave Composta

Criando uma chave composta:

```
@Embeddable
public class TelefonePK implements Serializable {
  private Short ddd;
  private String numero;

  public Short getDdd() { return ddd; }
  public void setDdd(Short ddd) { this.ddd = ddd; }

  public String getNumero() { return numero; }
  public void setNumero(String numero) {
    this.numero = numero;
  }
}
```

Chave Composta

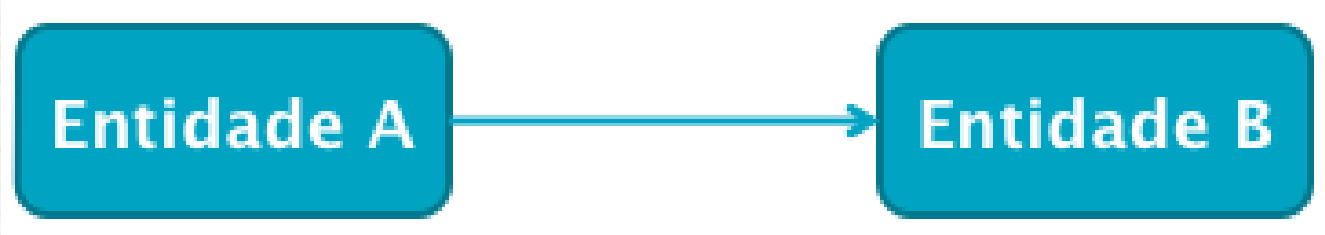
Para adicionar a chave composta na entidade, crie um atributo do tipo da chave composta e adicione a anotação @EmbeddedId nele:

```
@Entity
public class Telefone implements Serializable {

   @EmbeddedId
   private TelefonePK id;
   private String cliente;

   // get e set.
}
```

Relacionamento entre entidades

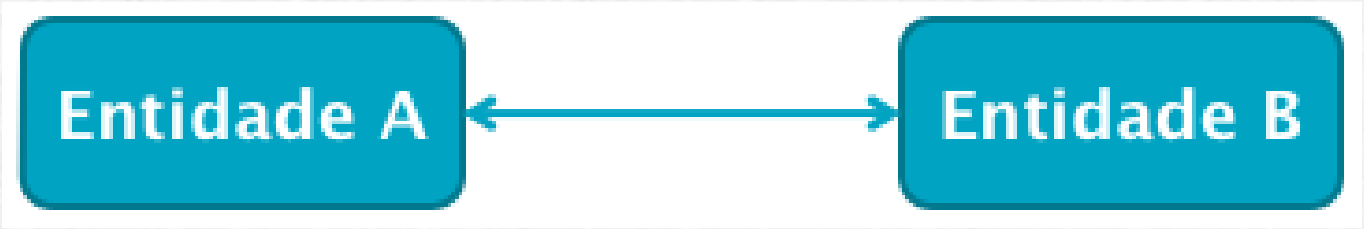
Os relacionamentos podem ser:

- **Unidirecional**
  - A partir de uma entidade é possível encontrar outra entidade, mas o contrário não acontece



Relacionamento entre entidades

- **Bidirecional**
  - Ambas entidades se conhecem



Relacionamento entre entidades

Tipos de relacionamentos:

- Um-para-Um (@OneToOne)
- Um-para-Muitos (@OneToMany)
- Muitos-para-Um (@ManyToOne)
- Muitos-para-Muitos (@ManyToMany)

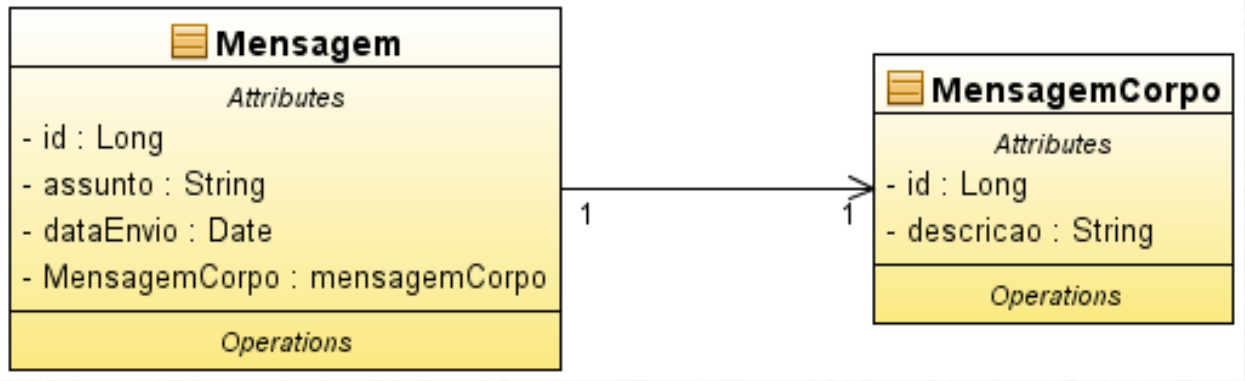
@OneToOne

Vídeo 1

@OneToOne

http://youtu.be/Sh-Y-beMxns

Um-para-Um (@OneToOne)




```
CREATE TABLE  Mensagem (
  id              number(5) NOT NULL,
  assunto     varchar(200) NOT NULL,
  dataEnvio date NOT NULL,
  mensagemcorpo_id number(5) NOT NULL,
  PRIMARY KEY (id)
);
```

```
CREATE TABLE  MensagemCorpo (
  id              number(5) NOT NULL,
  descricao varchar(200) NOT NULL,
  PRIMARY KEY (id)
);
```

Um-para-Um (@OneToOne)

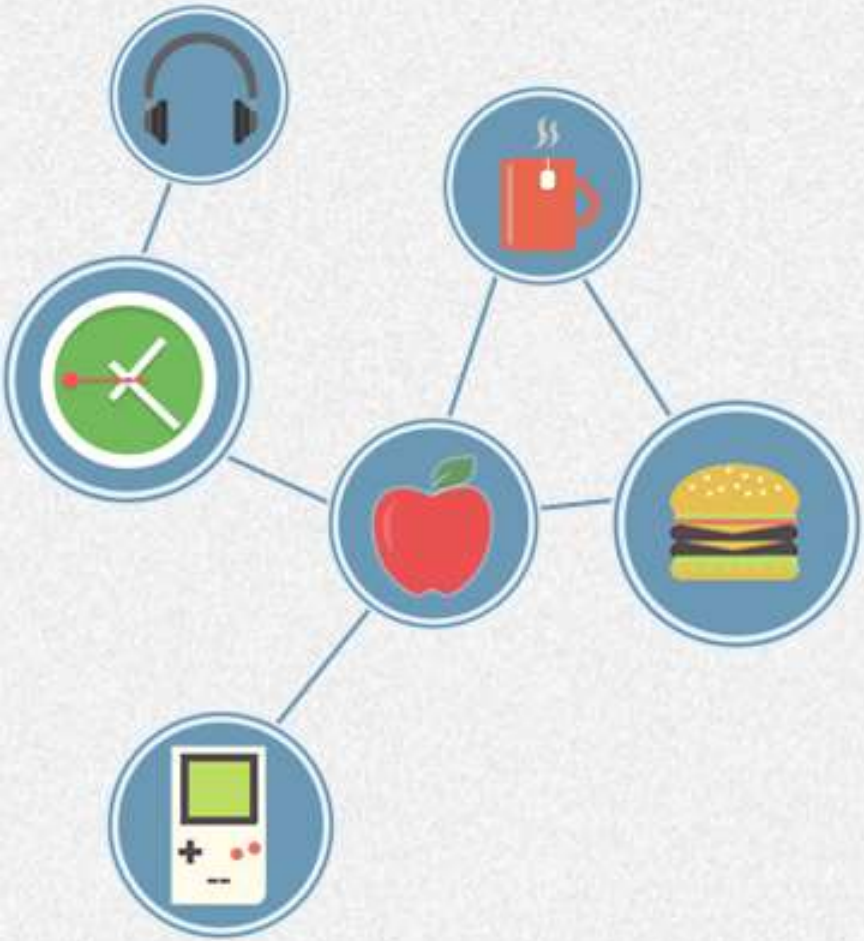
Na Mensagem tem a anotação **@OneToOne** para realizar a associação Um-para-Um com a entidade MensagemCorpo:

```
@Entity
public class Mensagem {
  @Id
  private Long id;
  private String assunto;
  @Temporal(TemporalType.DATE)
  private Date dataEnvio;

  @OneToOne(cascade=CascadeType.ALL)
  private MensagemCorpo mensagemCorpo;
}
```

Um-para-Um (@OneToOne)

A entidade MensagemCorpo não possui nenhuma relação com a entidade Mensagem, portanto temos um relacionamento unidirecional:

```
@Entity
public class MensagemCorpo {
  @Id
  private Long id;
  private String descricao;
}
```



Intervalo

Universidade Metodista de São Paulo | Campus EAD

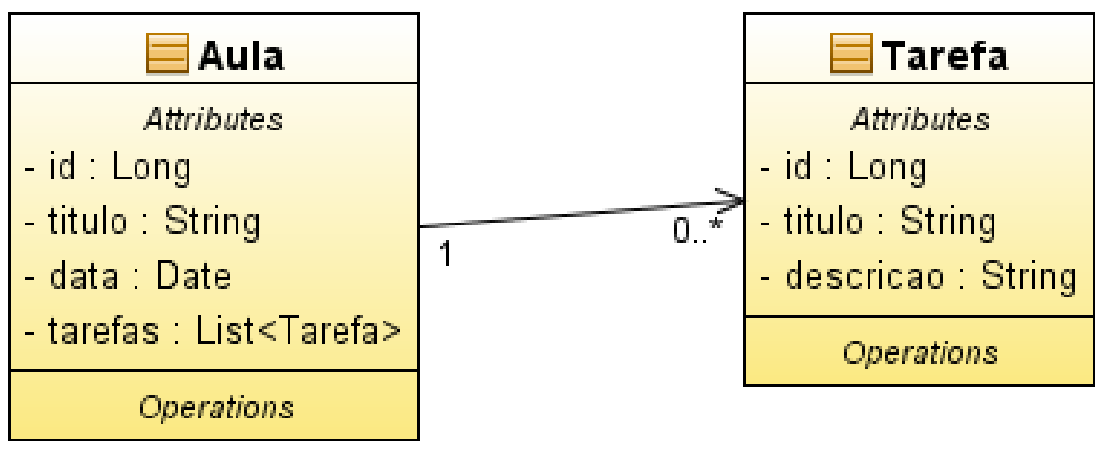
@OneToMany - @ManyToOne

Vídeo 2

@OneToMany - @ManyToOne

http://youtu.be/B5wArXmXy9M

Um-para-Muitos (@OneToMany)




```
CREATE TABLE Aula (
  id       number(5) NOT NULL,
  titulo   varchar(45),
  data     date,
  PRIMARY KEY  (id)
);
```

```
CREATE TABLE Tarefa (
  id           number(5) NOT NULL,
  titulo       varchar(45),
  descricao varchar(45),
  aula_id      number(5),
  PRIMARY KEY  (id)
);
```

Um-para-Muitos (@OneToMany)

Na entidade Aula, utilizamos a anotação **@OneToMany** para realizar a associação Um-para-Muitos com a entidade Tarefa:

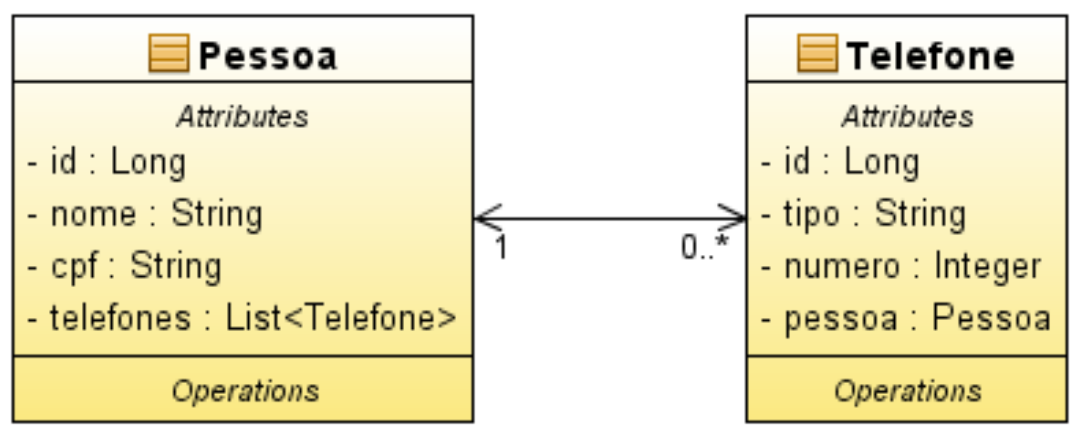
```
@Entity
public class Aula {
  @Id
  private Long id;
  private String titulo;
  @Temporal(TemporalType.DATE)
  private Date data;
  @OneToMany(cascade = CascadeType.ALL)
  @JoinColumn(name="aula_id")
  private List<Tarefa> tarefas;
}
```

Um-para-Muitos (@OneToMany)

Classe que representa a entidade Tarefa, note que ela não possui nenhum relacionamento com a entidade Aula, portanto temos um relacionamento unidirecional:

```
@Entity
public class Tarefa {
  @Id
  private Long id;
  private String titulo;
  private String descricao;
}
```

Muitos-para-Um (@ManyToOne)




```
CREATE TABLE Pessoa (
  id      number(5) NOT NULL,
  nome varchar(200),
  cpf     varchar(11),
  PRIMARY KEY (id)
);
```

```
CREATE TABLE Telefone (
  id             number(5) NOT NULL,
  tipo           varchar(200),
  numero      number(5),
  pessoa_id  number(5),
  PRIMARY KEY (id)
);
```

Muitos-para-Um (@ManyToOne)

Na entidade Pessoa utilizamos a anotação **@OneToMany** para realizar a associação Um-para-Muitos com a entidade Telefone:

```
@Entity
public class Pessoa {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String nome;
   private String cpf;
   @OneToMany(mappedBy = "pessoa", cascade = CascadeType.ALL)
   private List<Telefone> telefones;
}
```

Muitos-para-Um (@ManyToOne)

Na entidade Telefone utilizamos a anotação **@ManyToOne** para realizar a associação Muitos-para-Um com a entidade Pessoa:

```
@Entity
public class Telefone {
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String tipo;
   private Integer numero;
   @ManyToOne
   private Pessoa pessoa;
}
```

Tabela de associação

Vídeo 3

Tabela de associação

http://youtu.be/6LqBB2cV28Y

Tabela de associação

Podemos também utilizar uma tabela para realizar a associação um-para-muitos e muitos-para-muitos:

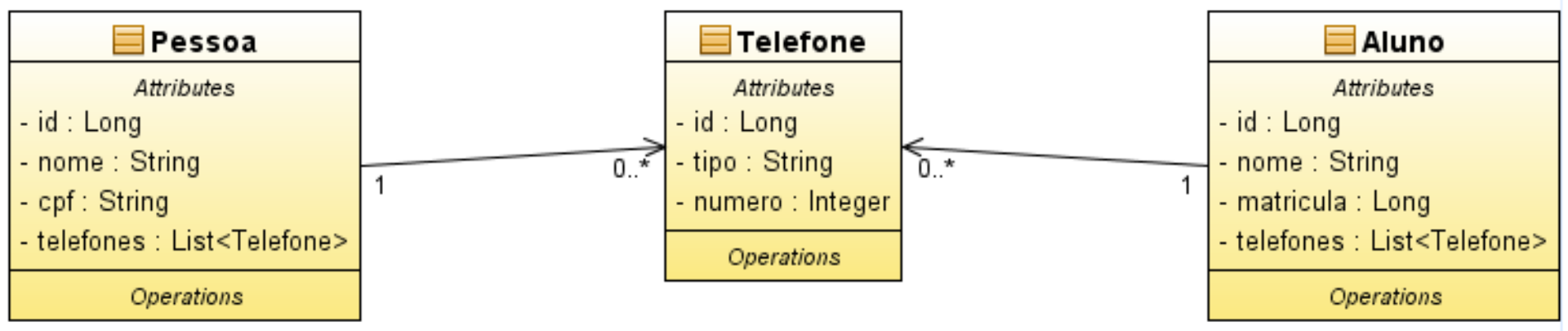


Tabela de associação

```
CREATE TABLE Aluno(
  id            number(5) NOT NULL,
  nome          varchar(200),
  matricula     number(5),
  PRIMARY KEY (id)
);
```

```
CREATE TABLE Telefone (
  id          number(5) NOT NULL,
  tipo        varchar(200),
  numero number(5),
  PRIMARY KEY (id)
);
```

```
CREATE TABLE Aluno_Telefone (
  aluno_id number(5),
  telefone_id number(5)
);
```

Tabela de associação

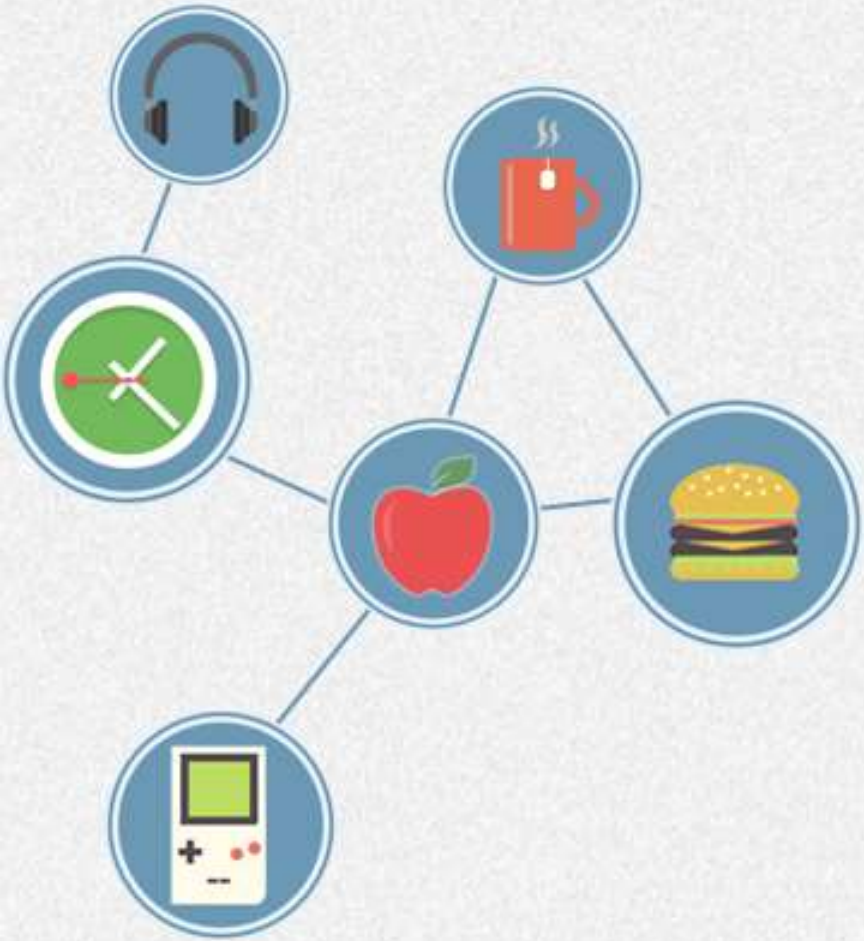
Na entidade Aluno utilizamos a anotação **@JoinColumn** para realizar a tabela de associação ALUNO_TELEFONE:

```
@Entity
public class Aluno {
  @Id
  private Long id;
  private String nome;
  private Long matricula;
  @OneToMany
  @JoinTable(name="ALUNO_TELEFONE",
    joinColumns={@JoinColumn(name = "ALUNO_ID")},
    inverseJoinColumns={@JoinColumn(name = "TELEFONE_ID")})
  private List<Telefone> telefones;
}
```

Tabela de associação

Classe que representa a entidade Telefone:

```
@Entity
public class Telefone {
  @Id
  private Long id;
  private String tipo;
  private Integer numero;
}
```



Intervalo

Universidade Metodista de São Paulo | Campus EAD

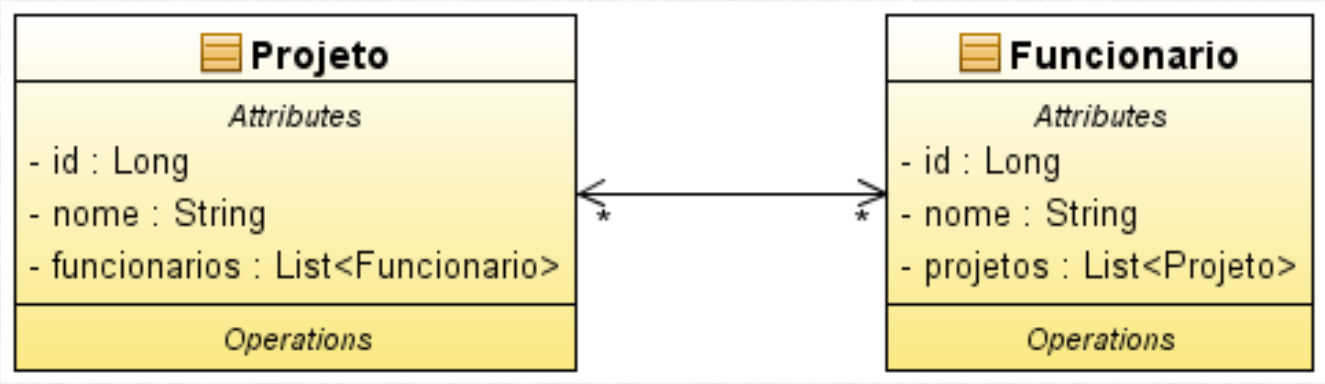
@ManyToMany

Vídeo 4

@ManyToMany

http://youtu.be/GRyNWIEZ6MQ

Muitos-para-Muitos(@ManyToMany)




```
CREATE TABLE Projeto (
  id      number(5) NOT NULL,
  nome varchar(200),
  PRIMARY KEY (id)
);
```

```
CREATE TABLE Funcionario (
  id      number(5) NOT NULL,
  nome varchar(200),
  PRIMARY KEY (id)
);
```

```
CREATE TABLE Projeto_Funcionario (
  projeto_id number(5),
  funcionario_id number(5)
);
```

Muitos-para-Muitos(@ManyToMany)

Na entidade **Projeto** utilizamos a anotação **@ManyToMany** para realizar a associação de muitos-para-muitos com a entidade **Funcionario**.

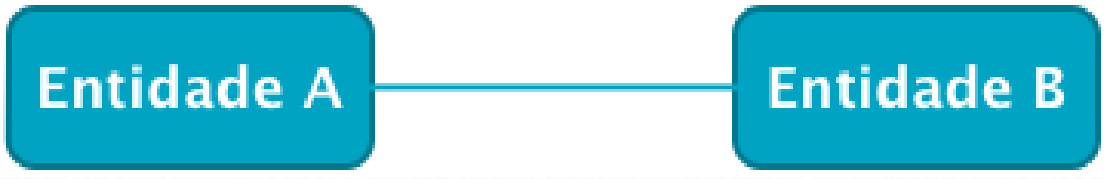
```
@Entity
public class Projeto {
  @Id
  private Long id;
  private String nome;
  @ManyToMany(mappedBy="projetos", cascade = CascadeType.ALL)
  private List<Funcionario> desenvolvedores;
}
```

Muitos-para-Muitos(@ManyToMany)

Na entidade **Funcionario** utilizamos a anotação **@ManyToMany** para realizar a associação de Muitos-para-Muitos com a entidade **Projeto**.

```
@Entity
public class Funcionario {
  @Id
  private Long id;
  private String nome;
  @ManyToMany(cascade = CascadeType.ALL)
  @JoinTable(name="PROJETO_FUNCIONARIO",
    joinColumns={@JoinColumn(name="PROJETO_ID")},
    inverseJoinColumns={@JoinColumn(name="FUNCIONARIO_ID")})
  private List<Projeto> projetos;
}
```

javax.persistence.CascadeType



PERSIST

- Quando salvar a Entidade A, também será salvo todas as Entidades B associadas.

MERGE

- Quando atualizar as informações da Entidade A, também será atualizado no banco de dados todas as informações das Entidades B associadas.

REMOVE

- Quando remover a Entidade A, também será removida todas as Entidades B associadas.

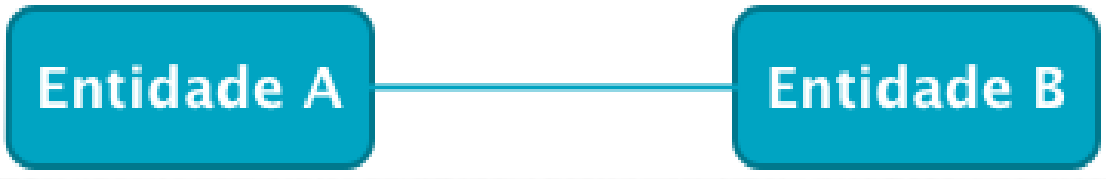
REFRESH

- Quando houver atualização no banco de dados na Entidade A, todas as Entidades B associadas serão atualizadas.

ALL

- Corresponde a todas as operações acima (PERSIST, MERGE, REMOVE e REFRESH).

javax.persistence.FetchType



EAGER

- Traz todas as entidades que estão relacionadas, então se a Entidade A possui um relacionamento com a Entidade B, quando consultar a Entidade A, também será consultado suas referencias na Entidade B.

LAZY

- Não traz as entidades que estão relacionadas, então se a Entidade A possui um relacionamento com a Entidade B, então quando consultar a Entidade A só serão retornadas as informações referentes a Entidade A.

javax.persistence.FetchType

Exemplo:

```
@Entity
public class Mensagem {
  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;
  private String assunto;
  @Temporal(TemporalType.DATE)
  private Date dataEnvio;

  @OneToOne(cascade=CascadeType.ALL, fetch=FetchType.LAZY)
  private MensagemCorpo mensagemCorpo;
}
```

Consultas com JPA

Vídeo 5

Consultas com JPA

http://youtu.be/sPC-oL-5ifQ

Consulta (JPA-QL)

LIVRO

id	titulo	autor	isbn	paginas
1	Almoçando com Java	Sakurai	111-11-1111-111-1	325
2	Classes Java em fila indiana	Cristiano	222-22-2222-222-2	120
3	Java em todo lugar	Sakurai	333-33-3333-333-3	543
4	Viajando no Java	Cristiano	444-44-4444-444-4	210

EMPRESTIMO

id	livro_id	cliente_id	dataEmprestimo	dataDevolucao
1	1	1	10/08/2009	20/08/2009
2	3	2	15/08/2009	30/08/2009
3	3	1	01/09/2009	

CLIENTE

id	nome	cpf	telefone
1	Marcelo	333.333.333-33	9999-8888
2	Ana	222.222.222-22	7777-6666

Consulta (JPA-QL)

Selecionar todos os empréstimos:

```
SELECT e FROM Emprestimo e
```

Consultar a quantidade de empréstimos por livro:

```
SELECT count(e) FROM Emprestimo e WHERE e.livro.id = :id
```

Consultar empréstimos por titulo do livro:

```
SELECT e FROM Emprestimo e, Livro l
WHERE e.livro.id = l.id
AND   l.titulo LIKE :titulo
```

Onde escrever as consultas?

As consultas podem estar fixas nas entidades:

```
@NamedQueries({
    @NamedQuery(name = "Cliente.consultarTodosClientes",
    query = "SELECT c FROM Cliente c")
})
@Entity
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String nome;
    private String cpf;
    private String telefone;
}
```

Interface javax.persistence.Query

A interface Query é responsável por:

- setParameter() – Passar parâmetros para a consulta
- getSingleResult() – Pegar um simples resultado
- getResultList() – Pegar uma lista de resultados
- executeUpdate() – Executar os updates
- setFistResult(int) – Define a posição do primeiro resultado
- setMaxResult(int) – Define a quantidade máxima de resultados

Onde escrever as consultas?

A consulta também pode ser criada dinamicamente:

```
Query query = em.createQuery("SELECT m FROM mensagem m");
```

Também podemos criar uma consulta nativa:

```
Query query = em.createNativeQuery("SELECT * FROM Mensagem",
Mensagem.class);

List<Mensagem> msgs = (List<Mensagem>) query.getResultList();
```

Executando uma consulta

Consulta que retorna uma lista:

```
Query query =
em.createNamedQuery("Cliente.consultarTodosClientes");

(List<Pessoa>) query.getResultList();
```

Consulta que retorna um valor:

```
Query query =
em.createNamedQuery("Emprestimo.qtdEmprestimosPorLivro");

Long qtdEmprestimos = (Long) query.getSingleResult();
```

Passando parâmetros para a consulta

Para declarar que a consulta deve receber um parâmetro:

```
SELECT count(e)
FROM Emprestimo e
WHERE e.livro.id = :id
GROUP BY e.livro.id
```

Passando o parâmetro para a consulta:

```
Query query =
em.createNamedQuery("Emprestimo.qtdEmprestimosPorLivro");
query.setParameter("id", idLivro);
Long qtdEmprestimos = (Long) query.getSingleResult();
```

Exemplo

```
@Entity
public class Venda implements Serializable {
   private static final long serialVersionUID = -86830108460433753l3L;

   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Integer id;
   @ManyToOne
   private Funcionario funcionario;
   @OneToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
   @JoinColumn(name="produto_id", referencedColumnName="id")
   private List<Produto> produtos;

   @Temporal(javax.persistence.TemporalType.DATE)
   private Date dataVenda;

   //get e set
}
```

Exemplo

```
@NamedQueries({
   @NamedQuery(name="Venda.consultarPorPeriodo",
      query="SELECT v FROM Venda v WHERE v.dataVenda >= :inicio AND v.dataVenda
<= :fim"),
   @NamedQuery(name="Venda.consultarPorFuncionario”,
      query="SELECT v FROM Venda v WHERE v.funcionario.id = :funcionarioId")
})
@Entity
public class Venda implements Serializable {

   ...

   //get e set
}
```

Exemplo

```
public class VendaDAO {
  private EntityManager em;

  public VendaDAO(EntityManager em) { this.em = em; }

  public List<Venda> consultarPorPeriodo(Date inicio, Date fim) {
    Query q = em.createNamedQuery("Venda.consultarPorPeriodo");
    q.setParameter("inicio", inicio);
    q.setParameter("fim", fim);
    return q.getResultList();
  }

 public List<Venda> consultarPorFuncionario(Long id) {
    Query q = em.createNamedQuery("Venda.consultarPorFuncionario");
    q.setParameter("funcionarioId", id);
    return q.getResultList();
  }

  //métodos para salvar, alterar, remover, consultar, etc.
}
```

Exercícios

- Venda de piões de madeira.

Bibliografia

- Enterprise JavaBeans 3.0 - Bill Burke, Richard Monson-Haefel - 2006 - O'Reilly
- Especificação EJB 3.0 e JPA 1.0 - http://java.sun.com/products/ejb
- Java EE 5 Tutorial - http://java.sun.com/javaee/5/docs/tutorial/doc