

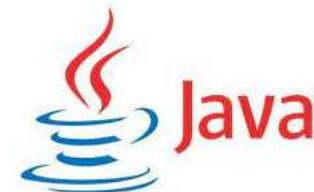


# Residência em Tecnologia da Informação e Comunicação

## JPA

Professor:

Alvaro Degas Coelho



INSTITUIÇÃO EXECUTORA



COORDENADORA



APOIO



## O problema da Impedância

- Mapeamento Objeto-Relacional
  - A estrutura de um Banco de Dados Relacional
    - Tabelas
  - A estrutura de um programa Orientado a Objetos
    - Classes
- Tornar classes em tabelas é possível
  - Num contexto geral
- Tornar classes em tabelas é difícil
  - Em contextos específicos



## Observe o código abaixo (produzido na última aula)

```
public static ArrayList<Usuario> readAll() {
    DAO dao = new DAO();
    Connection con = dao.criarConexao();
    String query = "SELECT Login, Senha, Email FROM Usuario";
    ArrayList<Usuario> lista = new ArrayList<Usuario>();
    try {
        PreparedStatement preparedStatement = con.prepareStatement(query);
        ResultSet result = preparedStatement.executeQuery();
        while (result.next()) {
            Usuario usuario = new Usuario();
            usuario.setLogin(result.getString("Login"));
            usuario.setSenha(result.getString("Senha"));
            usuario.setEmail(result.getString("Email"));
            lista.add(usuario);
        }
        return lista;
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return null;
    }
}
```

## Problemas

- Transportar dados de um objeto para uma tabela pode ser muito trabalhoso
  - Percorrer o conjunto de dados que foi trazido (ou que vai ser enviado)
  - Converter o dado (ResultSet) no atributo (Objeto)
  - Produzir a Coleção (ArrayList)

## Mais dificuldades

- SGBDR
  - Relacionamentos são implementados usando chaves
    - Atributos da Tupla
- Linguagens OO
  - Relacionamentos são implementados usando atributos DO TIPO DO OBJETO

## Exemplo

Observe as duas estruturas abaixo

```
public class Curso {  
  
    int id;  
    String nome;  
    int numSemestres;  
  
}  
  
public class Estudante {  
  
    int id;  
    String Nome;  
    String Email;  
    String Matricula;  
    Curso curso;  
  
}
```

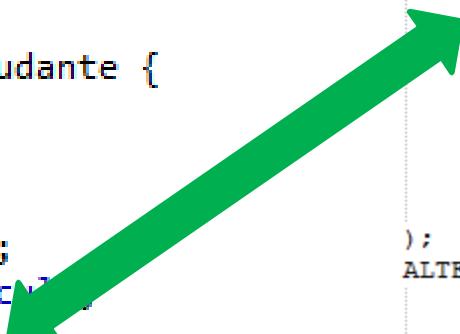
```
CREATE TABLE Curso (  
    Id INT NOT NULL AUTO_INCREMENT,  
    Nome VARCHAR(30) NOT NULL,  
    NumSemestres INT NOT NULL,  
    PRIMARY KEY (Id),  
    UNIQUE (Nome)  
);  
  
CREATE TABLE Estudante (  
    Id INT NOT NULL AUTO_INCREMENT ,  
    IdCurso INT NOT NULL ,  
    Nome VARCHAR(50) NOT NULL ,  
    Email VARCHAR(30) NOT NULL ,  
    Matricula VARCHAR(12) NOT NULL ,  
    PRIMARY KEY (Id),  
    UNIQUE (Email),  
    UNIQUE (Matricula)  
);  
ALTER TABLE Estudante  
ADD FOREIGN KEY (IdCurso)  
REFERENCES Curso(Id)  
ON DELETE RESTRICT ON UPDATE RESTRICT;
```

## Exemplo

Observe as duas estruturas abaixo

```
public class Curso {  
  
    int id;  
    String nome;  
    int numSemestres;  
  
}  
  
public class Estudante {  
  
    int id;  
    String Nome;  
    String Email;  
    String Matricula;  
    Curso curso;  
  
}
```

```
CREATE TABLE Curso (  
    Id INT NOT NULL AUTO_INCREMENT,  
    Nome VARCHAR(30) NOT NULL,  
    NumSemestres INT NOT NULL,  
    PRIMARY KEY (Id),  
    UNIQUE (Nome)  
);  
  
CREATE TABLE Estudante (  
    Id INT NOT NULL AUTO_INCREMENT ,  
    IdCurso INT NOT NULL ,  
    Nome VARCHAR(50) NOT NULL ,  
    Email VARCHAR(30) NOT NULL ,  
    Matricula VARCHAR(12) NOT NULL ,  
    PRIMARY KEY (Id),  
    UNIQUE (Email),  
    UNIQUE (Matricula)  
);  
ALTER TABLE Estudante  
ADD FOREIGN KEY (IdCurso)  
REFERENCES Curso(Id)  
ON DELETE RESTRICT ON UPDATE RESTRICT;
```



## Exemplo

Observe as duas estruturas abaixo

```
public class Curso {  
  
    int id;  
    String nome;  
    int numSemestres;  
}
```

Dado Primitivo  
(Coluna)

```
public class Estudante {  
  
    int id;  
    String Nome;  
    String Email;  
    String Matricula;  
    Curso curso;  
}
```

Dado  
Complexo  
(Objeto)

```
CREATE TABLE Curso (  
    Id INT NOT NULL AUTO_INCREMENT,  
    Nome VARCHAR(30) NOT NULL,  
    NumSemestres INT NOT NULL,  
    PRIMARY KEY (Id),  
    UNIQUE (Nome)  
);  
  
CREATE TABLE Estudante (  
    Id INT NOT NULL AUTO_INCREMENT ,  
    IdCurso INT NOT NULL ,  
    Nome VARCHAR(50) NOT NULL ,  
    Email VARCHAR(30) NOT NULL ,  
    Matricula VARCHAR(12) NOT NULL ,  
    PRIMARY KEY (Id),  
    UNIQUE (Email),  
    UNIQUE (Matricula)  
);  
ALTER TABLE Estudante  
ADD FOREIGN KEY (IdCurso)  
REFERENCES Curso(Id)  
ON DELETE RESTRICT ON UPDATE RESTRICT;
```

## Mais dificuldades

- Herança
  - Uma operação trivial em linguagens OO
    - Extends (Java)
- Uma estrutura complexa em Bancos de Dados Relacionais
  - Uma única tabela agregando todos os dados
  - Uma tabela mãe exportando chave para tabelas filhas
  - Tabelas filhas exportando chave para a tabela mãe

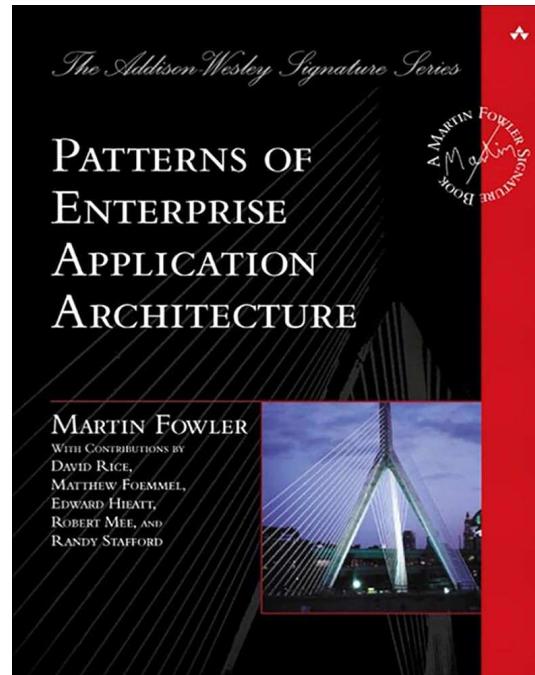
## JDBC: apenas uma conexão

- Crítica ao JDBC: provê a conexão
  - Permite que dados e instruções (SQL) viagem do programa para o SGBD
  - Permite que dados sejam trazidos do SGBD e fiquem disponíveis o programa
- É excelente. Mas é pouco
  - Necessidade de construir as sentenças SQL a partir dos objetos
  - Necessidade de construir os objetos a partir do resultado das sentenças
- MUITO esforço de programação



Residência  
em Software

## 30% do trabalho é gasto no mapeamento Objeto-Relacional



## Problemas mais concretos

- Contexto de Persistência
  - E se um objeto não estiver no contexto de salvar/alterar no SGBD (veremos em detalhes)?
- Mapa de Identidade
  - Quais objetos de memória estão persistidos corretamente no SGBD? Quais não estão?
- Carregamento Tardio
  - E se eu quiser pegar o Usuário, mas não as postagens dele?
    - Um DAO só pra usuário e outro para Usuários e Postagens?



Residência  
em Software

## **Orm – Object Relational Mapping Mapeamento (automático) Objeto Relacional**



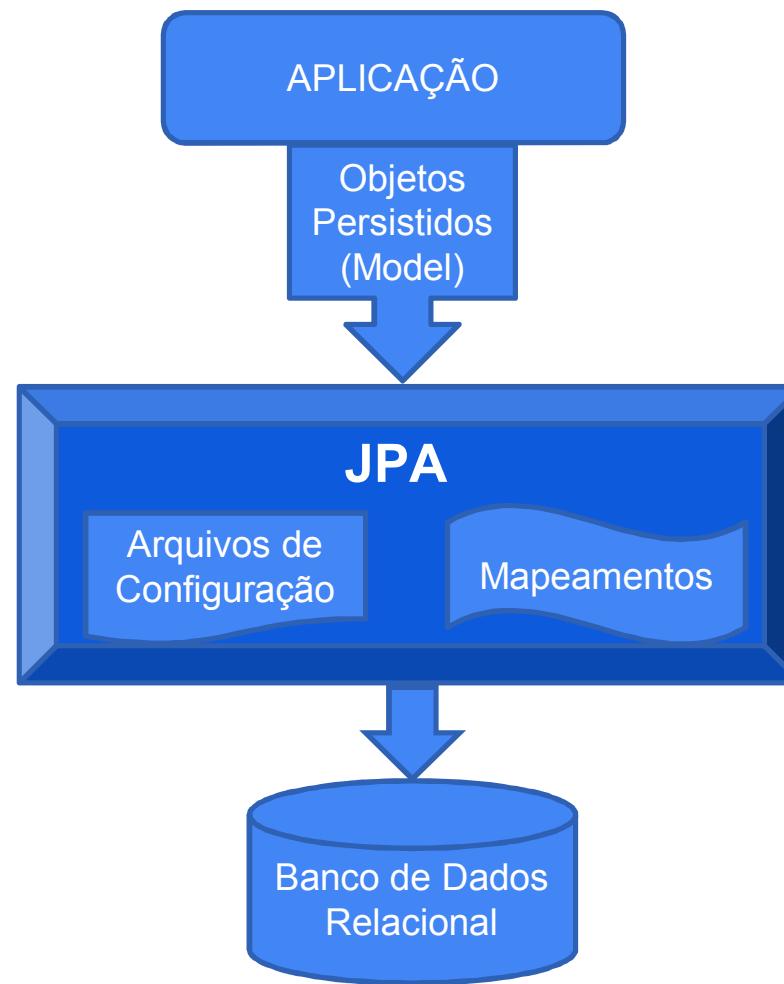
## JPA – Java Persistence API

- Especificação para mapeamento Objeto Relacional
  - Pacote javax.persistence
- Especificação ≠ Implementação
  - Interfaces
  - Padrão JSR 338
- Para usar é necessário uma IMPLEMENTAÇÃO
  - Hibernate (existem outras... Dizem. Ninguém nunca viu!)



Residência  
em Software

# Arquitetura padrão de uma Implementação JPA



## Da Aplicação ao SGBD

- A aplicação possui dados a serem persistidos
  - Este é o trabalho executado na JPA (Hibernate)
- Aqui a JPA atua de duas formas diferentes
  - Objetos são convertidos para dados em RDBMS
    - E salvos no BD
  - Dados RDBMS são convertidos em objetos
    - E levados para a aplicação

## Mapeamentos e Configurações

- O padrão JPA define dois aspectos diferentes de seu funcionamento
- Configuração
  - Definir quais APIs deverão ser usadas e aspectos da conexão com o RDBMS
    - Sim. Aqui usaremos o velho e bom driver JDBC
- Mapeamento
  - Definir quais objetos estarão mapeados no RDBMS, e como

# Configurações

- Maven
  - Gerenciador de dependências
    - Funciona bem para JPA e SpringBoot
- Arquivos XML
  - pom.xml
    - Dados de dependências que devem estar baixadas no projeto. Drivers e versões
  - persistence.xml
    - Dados necessários para persistir objetos do projeto
      - Persistence Unit, Credenciais de BD, Endereço do SGBD, etc.

## Mapeamentos

- Mapear objetos Java no SGBD
- Objetos Model
  - Parão MVC
- Objetos devem implementar a interface **Serializable**
  - Para permitir o transporte dos dados através do stream
- Stream
  - Neste caso: conexão TCP entre a aplicação e o SGBD

## Annotations

- Ou *Decorators*
- São tipicamente utilizadas em Frameworks
- Geram modificações no código compilado
  - Ou interpretado
- Por exemplo: `@test`
  - Informa que o método é executado pela função main do pacote Junit
- Muito utilizado em ORM (JPA é um ORM)
  - Para linkar códigos específicos de mapeamento entre objetos e BD



## Annotations do JPA

- São muitas. Usamos algumas. Sabemos poucas. Consultamos TODAS
- Nosso catálogo
  - @Table, @Entity
  - @Column
  - @Id (@Generated Value)
  - @OneToOne, @ManyToOne, @OneToMany, @ManyToMany

## @Entity

- Define que uma classe é, também, uma entidade do BD
  - Mapeia em uma tabela
- Exemplo

```
@Entity  
public class Carro implements Serializable{  
    private String placa;  
    private String renavam;  
}
```

## @Entity

- Objetos marcados com @Entity são persistidos no BD
- Sob comando de um objeto EntityManager (Gerenciador de Entities)
  - Veremos EntityManager em breve
- Objetos marcados com @Entity terão, por definição, o mesmo nome da tabela correspondente no BD
- Exceto se informado em contrário com a Annotation@Table

## @Table

- Permite desconectar o nome do objeto ao nome da tabela no BD
  - Questionável: pode dificultar compreensão do código
- Exemplo

```
@Entity  
@Table(name="Tb_Carro")  
public class Carro implements Serializable{  
    private String placa;  
    private String renavam;  
}
```



## @Column

- Similar ao @Table, permite desconectar o nome da coluna da tabela do nome do atributo do objeto
- Exemplo

```
@Entity
@Table(name="Tb_Carro")
public class Carro implements Serializable{
    private String placa;
    @Column(name="Nu_Renavan")
    private String renavam;
}
```

## @Id

- Annotation para definir a chave primária de uma tabela
  - E ligar ela a um atributo do objeto
- Indispensável em termos práticos
  - Permite a execução do método (padrão do JPA) findById()
- Qualquer atributo pode ser definido como chave primária
- Um conjunto de atributos pode ser definido como chave primária
  - Veremos

## @Id

- Um exemplo

```
@Entity  
@Table(name="Tb_Carro")  
public class Carro implements Serializable{  
    @Id  
    private String placa;  
    @Column(name="Nu_Renavan")  
    private String renavam;  
}
```

## @Id geradas no BD

- É comum (e útil) deixar que o SGBD gerencie a geração de chaves primárias
- Neste caso uma segunda annotation é necessária, de acordo com a estratégia a ser adotada pelo SGBD
- Duas maneiras de se fazer isso
- `@Generated Value`
  - Deixa que o SGBD gere usando sua abordagem default
- `@Generated Value (strategy = GenerationType.<estratégia>)`
  - Define qual estratégia vai ser utilizada

## @Generated Value

- **@Generated Value (strategy=GenerationType.AUTO)**
  - Default. O JPA escolhe de acordo com o BD
- **@Generated Value (strategy=GenerationType.IDENTITY)**
  - Valores do identificador gerados pela coluna de auto incremento do banco de dados. Alguns bancos de dados podem não suportar essa opção.
- **@Generated Value (strategy=GenerationType.SEQUENCE)**
  - Informamos ao provedor de persistência que os valores serão gerados a partir de uma sequence. Pod ser uma global ou específica da tabela. Alguns bancos de dados podem não suportar essa opção.
- **@Generated Value (strategy=GenerationType.TABLE)**
  - Cria-se uma tabela para gerenciar as chaves primárias. Por causa da sobrecarga de consultas necessárias para manter a tabela atualizada, essa opção é pouco recomendada.



## @Id

- Exemplo

```
@Entity
@Table(name="Tb_Carro")
public class Carro implements Serializable{
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private int id;
    private String placa;
    @Column(name="Nu_Renavan")
    private String renavam;
}
```

## **@OneToOne, @ManyToOne, @OneToMany, @ManyToMany**

- Indica ao JPA que um determinado atributo tem uma relação definida no Banco de Dados
  - O JPA vai considerar esta relação no momento de persistir ou recuperar o dado
- **@OneToOne**
  - relação 1 para 1
- **@OneToMany e @ManyToOne**
  - definição bidirecional de relacionamento -> exportação de chave
- **@ManyToMany**
  - Definição da “tabela do meio” que vai receber as duas chaves



```
@Entity
public class Proprietario
    implements Serializable {
    @Id
    String CPF;
    ...
    @OneToMany
    (mappedBy="proprietario")
    private ArrayList<Carro>
    carros;
    ...
}
```

## Exemplo

```
@Entity
@Table(name="Tb_Carro")
public class Carro implements
    Serializable{
    @Id
    @GeneratedValue
    (strategy=GenerationType.AUTO
    )
    private int id;
    ...
    @ManyToOne
    @JoinColumn(name = "cpf")
    public Proprietario
    getProprietario() {
        return proprietario;
    }
}
```



```
@Entity
public class Proprietario
    implements Serializable {
    @Id
    String CPF;
    ...
    @OneToMany
    (mappedBy="proprietario")
    private ArrayList<Carro>
    carros;
    ...
}
```

## Exemplo

```
@Entity
@Table(name="Tb_Carro")
public class Carro implements
    Serializable{
    @Id
    @GeneratedValue
    (strategy=GenerationType.AUTO
    )
    private int id;
    ...
    @ManyToOne
    @JoinColumn(name = "cpf")
    public Proprietario
    getProprietario() {
        return proprietario;
    }
}
```



Residência  
em Software

**Veremos mais das Annotations na prática em breve**

## Entity Manager

- Objeto para gerenciar entidades
- Na prática: Gerencia o ciclo de vida das entidades de BD
- Ciclo de Vida?
  - C\_UD (Criar, Atualizar, Remover)
- EntityManager deve ser criado pela EntityManagerFactory classe Persistence (javax.persistence)
  - O nome da EntityManager Unit deve ser definido nas configurações



Residência  
em Software

**Veremos mais sobre EntityManager na prática em breve**