



# Domain Driven Design



PROF. ELIANE RODRIGUES MARION SANTA ROSA  
[Profeliane.rosa@fiap.com.br](mailto:Profeliane.rosa@fiap.com.br)

# 1

## CLASSE DAO – métodos de consulta



## A CLASSE EnderecoDao

FIAP

Agora vamos fazer os métodos de consulta, podemos fazer a consulta de um registro colocando como condição a chave primária, mas também podemos trazer todos os registros da tabela ou um conjunto de registros dependendo do critério de busca.

Vamos então trabalhar com as consultas iniciando com a busca por um único registro, a grande diferença é que quando fazemos um select no banco haverá um retorno e precisamos então saber trabalhar com este retorno.



## Método buscarPorId - EnderecoDao

```
public Endereco buscarPorId(int id){  
    Endereco endereco = new Endereco();  
    conexao = GerenciadorBD.obterConexao();  
    PreparedStatement comandoSql = null;  
    try{  
        comandoSql = conexao.prepareStatement( sql: "Select * from endereco where idEndereco = ?");  
        comandoSql.setInt( parameterIndex: 1,id);  
        ResultSet rs = comandoSql.executeQuery();  
        if(rs.next()){  
            endereco.setId(rs.getInt( columnIndex: 1));  
            endereco.setCep(rs.getString( columnIndex: 2));  
            endereco.setLogradouro(rs.getString( columnIndex: 3));  
            endereco.setComplemento(rs.getString( columnIndex: 4));  
        }  
    }  
}
```



## Método buscarPorId - EnderecoDao

FIAP

```
        endereco.setBairro(rs.getString(columnIndex: 5));  
        endereco.setLocalidade(rs.getString(columnIndex: 6));  
        endereco.setUf(rs.getString(columnIndex: 7));  
    }  
    }catch (SQLException e){  
        e.printStackTrace();  
    }  
    return endereco;  
}
```



## Entendendo o código

FIAP

Criamos um método chamado `buscarPorId` que recebe um inteiro (`id`) e retorna um endereço.

```
public Endereco buscarPorId(int id){  
}
```

Criamos um objeto chamado `endereco` do tipo `Endereco`.

```
Endereco endereco = new Endereco();
```

Obtemos a conexão através do método `obterConexao` que escrevemos na classe `GerenciadorBD`.

```
conexao = GerenciadorBD.obterConexao();
```



## Entendendo o código

FIAP

Nesta linha estamos criando um objeto chamado `comandoSQL` do tipo `PreparedStatement`. Dentro do `try` tentaremos buscar um registro no banco de dados.

```
PreparedStatement comandoSQL = null;
```



## Entendendo o código

```
comandoSQL = conexao.prepareStatement("select * from endereco where idEndereco = ?");
```

- Nesta linha fazemos o uso da conexão com o banco de dados para preparar um statement com a instrução SQL select. Devemos escrever a instrução da mesma forma que escrevemos no banco de dados e no lugar dos valores colocaremos interrogação(?).





## Entendendo o código

FIAP

- Neste caso só temos o id, portanto teremos uma única linha atribuindo valor ao parâmetro indicado no PreparedStatement.

```
comandoSQL.setInt(1, id);
```

- Agora a grande diferença do select para os comandos vistos anteriormente, quando fazemos um select no banco haverá um retorno, então precisaremos de um objeto para tratar estes dados.



- Após montarmos nosso objeto `comandoSQL` o próximo passo é de fato executar a instrução `select` no banco, isso é feito com o método `executeQuery()`.
- O método `executeQuery()` executa uma instrução SQL e retorna um objeto do tipo `ResultSet`, que é a representação dos dados de um banco de dados em forma de tabela e possui os seguintes métodos:
  - ✓ `next()`: avança uma "linha" na tabela.
  - ✓ `previous()`: retrocede uma "linha" na tabela.



## Entendendo o código

FIAP

- ✓ `getInt(int index), getBoolean(int index), getString(int index), getDouble(int index):` retorna o valor em uma determinada coluna da tabela, dada pelo índice.
- Portanto ao fazermos uma consulta devemos criar um `ResultSet` e armazenar o resultado do método `executeQuery()` e preencher o objeto, neste caso endereco com os resultados adquiridos na busca.



- Após montarmos nosso objeto comandoSQL o próximo passo é de fato executar a instrução select no banco, isso é feito com o método `executeQuery()`, e o resultado da busca estamos guardando no objeto `rs` do tipo `ResultSet`.

```
ResultSet rs = comandoSQL.executeQuery();
```

- Precisamos verificar se a busca retornou algum resultado, para isso fazemos um teste:



## Entendendo o código

FIAP

```
if (rs.next()) {  
    c.setId(rs.getInt(1));  
    c.setNome(rs.getString(2));  
    c.setCelular(rs.getString(3));  
    c.setEmail(rs.getString(4));  
    String tipo =(rs.getString(5).toString());  
    if (tipo.equals("FAMILIA"))  
        c.setTipo(TipoEnderecoEnum.FAMILIA);  
    else if (tipo.equals("AMIGOS"))  
        c.setTipo(TipoEnderecoEnum.AMIGOS);  
    else if (tipo.equals("TRABALHO"))  
        c.setTipo(TipoEnderecoEnum.TRABALHO);  
    else if (tipo.equals("COMERCIAL"))  
        c.setTipo(TipoEnderecoEnum.COMERCIAL);  
    else  
        c.setTipo(TipoEnderecoEnum.COLEGA);  
}
```

Caso haja uma linha no ResultSet, avançamos

`c.setId(rs.getInt(1));`

Pega o conteúdo da primeira coluna do resultSet e atribui no id do objeto c (endereço)

Estamos tratando neste caso a Enum, pois no banco é um varchar e no objeto é uma enum.



## Entendendo o código

FIAP

Caso haja uma linha no ResultSet, avançamos

```
if(rs.next()){  
    endereco.setId(rs.getInt(1));  
    endereco.setCep(rs.getString(2));  
    endereco.setLogradouro(rs.getString(3));  
    endereco.setComplemento(rs.getString(4));  
    endereco.setBairro(rs.getString(5));  
    endereco.setLocalidade(rs.getString(6));  
    endereco.setUf(rs.getString(7));  
}
```

endereco.setId(rs.getInt(1));

Pega o conteúdo da primeira coluna do resultSet e atribui no id do objeto endereco



## Entendendo o código

- Acabamos de atribuir os dados devolvidos pela consulta no banco ao nosso objeto do tipo endereço.

- Por fim fechamos nossa conexão

```
conexao.close();  
comandoSQL.close();
```

- E precisamos devolver um objeto do tipo endereço, por isso retornamos o endereço.

```
return endereco;
```

# 2

**Testando a busca por id  
no banco**





## Testando sua conexão

Podemos criar outra classe ou utilizarmos a classe TestaInsercao que já criamos.

```
public class TestaInsercao {  
  
    public static void main(String[] args) {  
        Scanner ent = new Scanner(System.in);  
        Endereco endereco = new Endereco();  
        EnderecoDao dao = new EnderecoDao();  
        System.out.println("Digite o id do endereco: ");  
        int id = ent.nextInt();  
        endereco = dao.buscarPorId(id);  
        System.out.println(endereco);  
    }  
}
```

Para esta última linha funcionar precisamos sobrescrever o método toString() na classe Endereco



## Reescrita do método toString

Na classe `Endereco` podemos sobrescrever o método `toString`.

```
@Override
public String toString() {
    return "Endereco\n" +
        logradouro +
        ", " + complemento +
        ", cep = " + cep +
        bairro +
        ", " + localidade +
        ", " + uf;
}
```



## Exibição dos dados

Caso optarmos por não sobrescrever o método `toString` precisaremos montar a mensagem no método `main` utilizando os getters.

```
System.out.println(endereco.getLogradouro());  
System.out.println(", " + endereco.getComplemento());
```

# 3

**Buscar todos endereços**



## Método buscarTodosEnderecos - EnderecoDao

```
public List<Endereco> buscarTodosEnderecos(){  
    List<Endereco> enderecos = new ArrayList<>();  
    conexao = GerenciadorBD.obterConexao();  
    PreparedStatement comandoSql = null;  
    try{  
        comandoSql = conexao.prepareStatement( sql: "Select * from endereco ");  
        ResultSet rs = comandoSql.executeQuery();  
        while(rs.next()){  
            Endereco endereco = new Endereco();  
            endereco.setId(rs.getInt( columnIndex: 1));  
            endereco.setCep(rs.getString( columnIndex: 2));  
            endereco.setLogradouro(rs.getString( columnIndex: 3));  
        }  
    }  
}
```



## Método buscarTodosEnderecos - EnderecoDao

FIAP

```
        endereco.setComplemento(rs.getString( columnIndex: 4));  
        endereco.setBairro(rs.getString( columnIndex: 5));  
        endereco.setLocalidade(rs.getString( columnIndex: 6));  
        endereco.setUf(rs.getString( columnIndex: 7));  
        enderecos.add(endereco);  
    }  
} catch (SQLException e){  
    e.printStackTrace();  
}  
return enderecos;  
}
```



## Entendendo o código

Criamos um método chamado `buscarTodosEnderecos` que retorna uma lista de endereço e dentro declaramos um objeto chamado `enderecos` do tipo `Lista de endereços`.

```
public ArrayList<Endereco> buscarTodosEnderecos(){  
    List<Endereco> enderecos = new ArrayList<Endereco>();  
}
```

Obtemos a conexão através do método `obterConexao` que escrevemos na classe `GerenciadorBD`.

```
conexao = GerenciadorBD.obterConexao();
```



## Entendendo o código

Nesta linha estamos criando um objeto chamado `comandoSQL` do tipo `PreparedStatement`. Dentro do `try` tentaremos buscar todos os endereços cadastrados no banco de dados.

```
PreparedStatement comandoSQL = null;
```





```
comandoSQL = conexao.prepareStatement("select * from endereco ");
```

- Nesta linha fazemos o uso da conexão com o banco de dados para preparar um statement com a instrução SQL select. Devemos escrever a instrução da mesma forma que escrevemos no banco de dados.



- Após montarmos nosso objeto comandoSQL o próximo passo é de fato executar a instrução select no banco, isso é feito com o método `executeQuery()`, e o resultado da busca estamos guardando no objeto `rs` do tipo `ResultSet`.

```
ResultSet rs = comandoSQL.executeQuery();
```

- Diferente da busca por id, neste caso precisamos verificar se a busca retornou algum resultado, e percorrer o `resultSet` já que o esperado são várias linhas de retorno.



## Entendendo o código

FIAP

```
while(rs.next()){  
    Endereco endereco = new Endereco();  
    endereco.setId(rs.getInt( columnIndex: 1));  
    endereco.setCep(rs.getString( columnIndex: 2));  
    endereco.setLogradouro(rs.getString( columnIndex: 3));  
    endereco.setComplemento(rs.getString( columnIndex: 4));  
    endereco.setBairro(rs.getString( columnIndex: 5));  
    endereco.setLocalidade(rs.getString( columnIndex: 6));  
    endereco.setUf(rs.getString( columnIndex: 7));  
    enderecos.add(endereco);  
}
```

Criamos um objeto endereco do tipo Endereço e atribuímos o resultado da linha do resultSet no objeto



## Entendendo o código

```
while(rs.next()){  
    Endereco endereco = new Endereco();  
    endereco.setId(rs.getInt( columnIndex: 1));  
    endereco.setCep(rs.getString( columnIndex: 2));  
    endereco.setLogradouro(rs.getString( columnIndex: 3));  
    endereco.setComplemento(rs.getString( columnIndex: 4));  
    endereco.setBairro(rs.getString( columnIndex: 5));  
    endereco.setLocalidade(rs.getString( columnIndex: 6));  
    endereco.setUf(rs.getString( columnIndex: 7));  
    enderecos.add(endereco);  
}
```

Enquanto existir linha  
no ResultSet, avançamos



## Entendendo o código

```
Endereco endereco = new Endereco();
```

Criamos um objeto  
endereco do tipo  
Endereço e atribuímos  
o resultado da linha  
do resultSet no  
objeto

```
endereco.setId(rs.getInt(1));
```

endereco

id: 1  
cep:  
logradouro:  
complemento:  
bairro:  
localidade:  
uf:

```
select * from endereco;
```

Saída do Script x

Resultado da Consulta x

Todas as Linhas Extraídas: 1 em 0,058 segundos

IDENDereco	CEP	RUA	COMPLEMENTO	BAIRRO	CIDADE	UF
1	101310914	AV. PAULISTA, 1106 4 ANDAR		BELA VISTA	SÃO PAULO	SP



## Entendendo o código

FIAP

```
while(rs.next()){  
    Endereco endereco = new Endereco();  
    endereco.setId(rs.getInt( columnIndex: 1));  
    endereco.setCep(rs.getString( columnIndex: 2));  
    endereco.setLogradouro(rs.getString( columnIndex: 3));  
    endereco.setComplemento(rs.getString( columnIndex: 4));  
    endereco.setBairro(rs.getString( columnIndex: 5));  
    endereco.setLocalidade(rs.getString( columnIndex: 6));  
    endereco.setUf(rs.getString( columnIndex: 7));  
    enderecos.add(endereco)  
}
```

Adicionamos objeto  
endereco na lista



## Entendendo o código

FIAP

- Acabamos de atribuir os dados devolvidos pela consulta no banco à lista de endereços (enderecos).

- Por fim fechamos nossa conexão

```
conexao.close();  
comandoSQL.close();
```

- E precisamos devolver a lista que geramos, por isso retornamos o enderecos.

```
return enderecos;
```

# 3

**Testando a busca de  
todos enderecos**





## Testando sua conexão

Podemos criar outra classe ou utilizarmos a classe TestaInsercao que já criamos.

```
public static void main(String[] args) {  
    Scanner ent = new Scanner(System.in);  
    Endereco endereco = new Endereco();  
    EnderecoDao dao = new EnderecoDao();  
  
    List<Endereco> listaEnderecos = new ArrayList<Endereco>();  
    String dados = "";  
    listaEnderecos = dao.buscarTodosEnderecos();  
}
```



## Testando sua conexão

Continuação.....

```
for (Endereco endereco2 : listaEnderecos) {  
    dados += "=====\n";  
    dados += "Id: " + endereco2.getId() + "\n";  
    dados += "Endereço: " + endereco2.getLogradouro() + "\n";  
    dados += "Complemento: " + endereco2.getComplemento() + "\n";  
    dados += "CEP: " + endereco2.getCep() + "\n";  
    dados += "Bairro: " + endereco2.getBairro() + "\n";  
    dados += "Cidade: " + endereco2.getLocalidade() + "\n";  
    dados += "Estado: " + endereco2.getUf() + "\n";  
    dados += "=====\n";  
}  
System.out.println(dados);  
}
```