

# CRUD com JDBC e DAO

Este roteiro detalha o desenvolvimento de um CRUD em Java utilizando a API JDBC e o padrão DAO.

## 1 CRUD

Aplicação CRUD (*Create*, *Read*, *Update* e *Delete* - Criar, Ler, Atualizar e Excluir)



## 2 Banco de Dados

### 2.1 Criação do Banco de Dados

No MySQL Workbench execute os comandos abaixo para criar e selecionar o seguinte banco de dados:

```
CREATE DATABASE unoesc_crud_jdbc;  
USE unoesc_crud_jdbc;
```

### 2.2 Criação da Tabela

Crie a tabela com o SQL abaixo:

```
CREATE TABLE produto (  
    id_prod INT AUTO_INCREMENT PRIMARY KEY,  
    nome_prod VARCHAR(50),  
    data_cadastro DATE,  
    quantidade INT,  
    preco DECIMAL(10, 2)  
) engine=InnoDB;
```

Insira os dados a seguir:

```
INSERT INTO produto  
VALUES (1, 'Notebook', '2023-01-01', 5, 666),  
       (2, 'Smartphone', '2023-10-30', 15, 1234.56),  
       (3, 'TV', '2023-04-01', 2, 4999.99),  
       (4, 'Smartband', '2023-04-01', 50, 499.99);
```

Verifique se os dados foram corretamente inseridos:

```
SELECT * FROM produto;
```

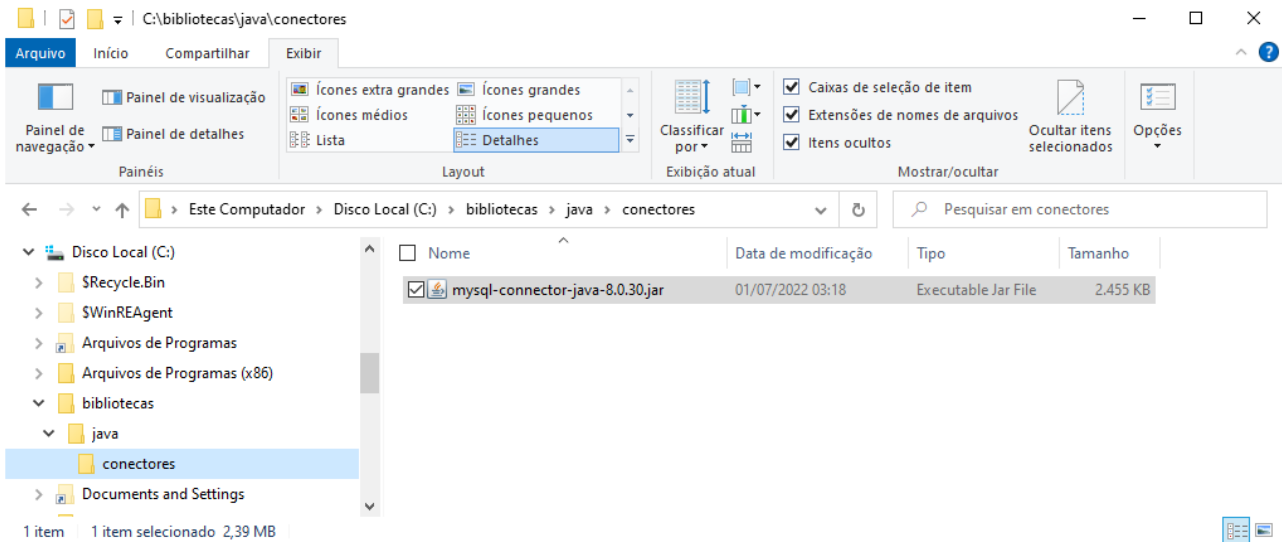
	id_prod	nome_prod	data_cadastro	quantidade	preco
▶	1	Notebook	2023-01-01	5	666.00
	2	Smartphone	2023-10-30	15	1234.56
	3	TV	2023-04-01	2	4999.99
	4	Smartband	2023-04-01	50	499.99

## 3 Projeto Inicial

### 3.1 Driver JDBC do MySQL

Siga os passos abaixo para baixar o *driver*:

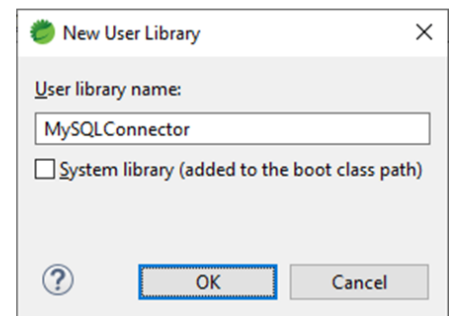
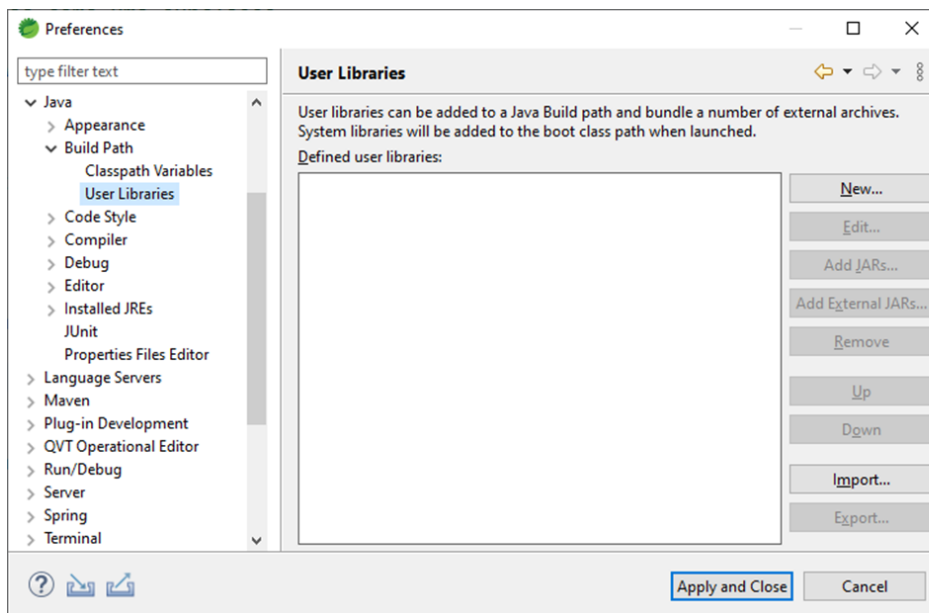
1. Baixe o conector Java para MySQL ([Connector/J](#)) do *site* do MySQL ou então na [trilha da disciplina no moodle](#)
2. No disco *C:\* crie uma pasta chamada *bibliotecas* e, dentro dela, uma pasta com o nome *java*; dentro da pasta *java* crie uma pasta chamada *conectores*
3. Abra o arquivo *.zip* baixado e arraste o arquivo *mysql-connector-java-8.0.30.jar* para dentro da pasta *conectores*



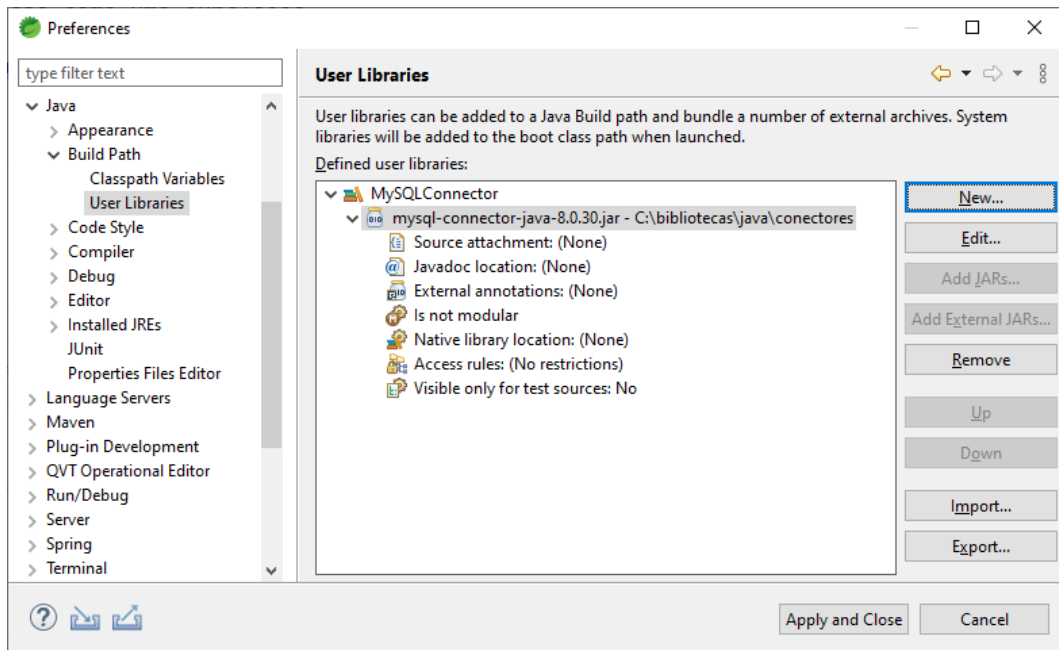
### 3.2 Configuração da Biblioteca no STS

Siga os passos abaixo para configurar uma biblioteca de usuário no STS:

1. No STS vá em *Window → Preferences*, abra a categoria *Java → Build Path → User Libraries* e então clique no botão *New ...* - a seguir informe o nome *MySQLConnector*.



2. Clique agora no botão *Add External JARs ...* e acrescente a biblioteca baixada – após isso clique em *Apply and Close*

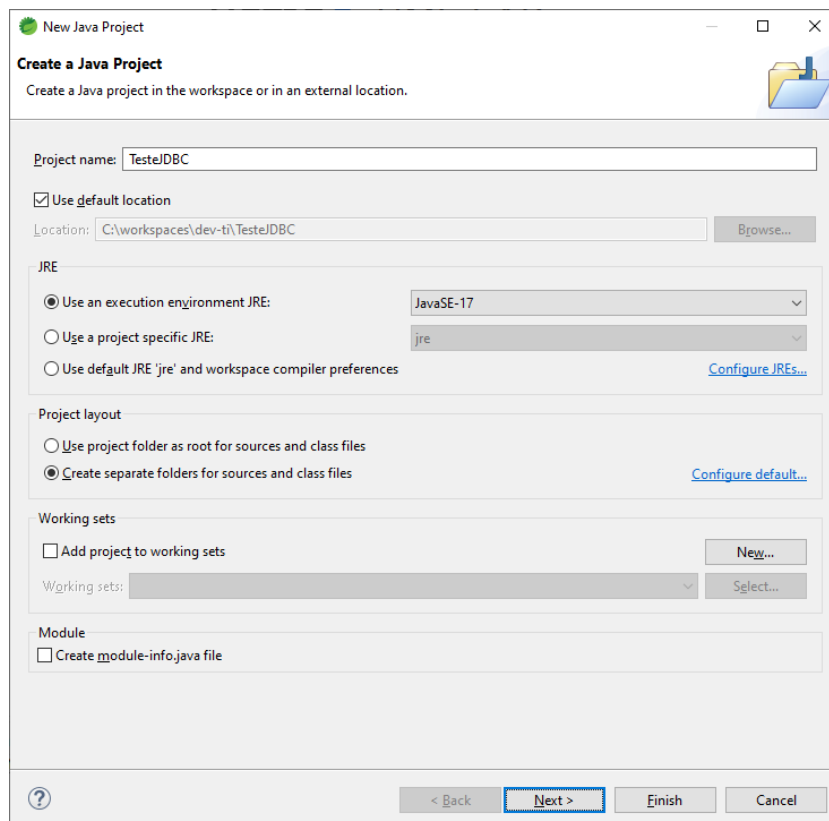


## 4 Desenvolvimento do Projeto (Versão 1)

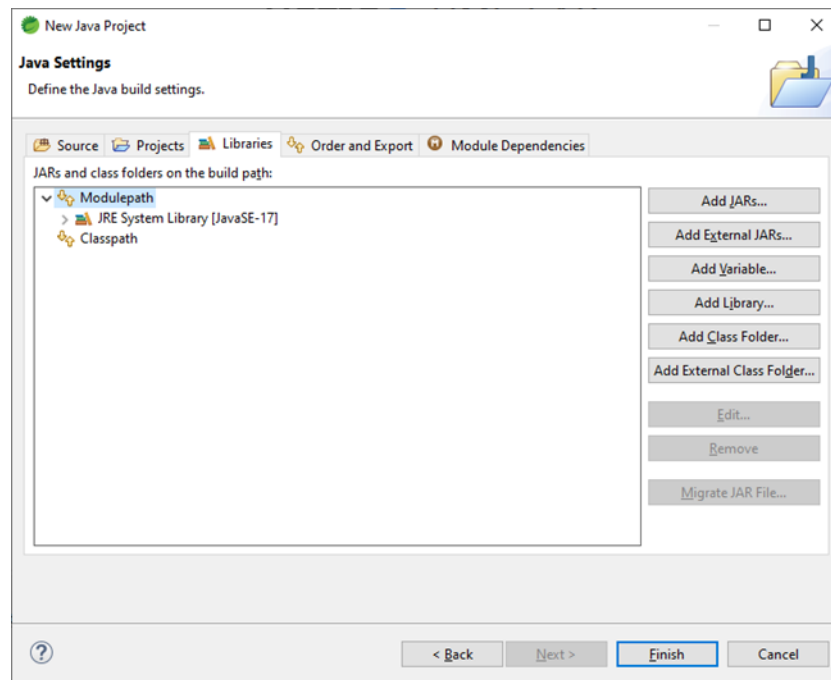
### 4.1 Criação do Projeto

Siga os passos abaixo para criar o projeto no STS:

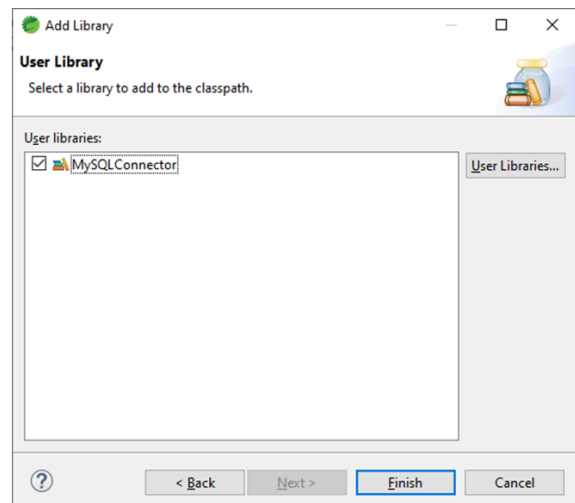
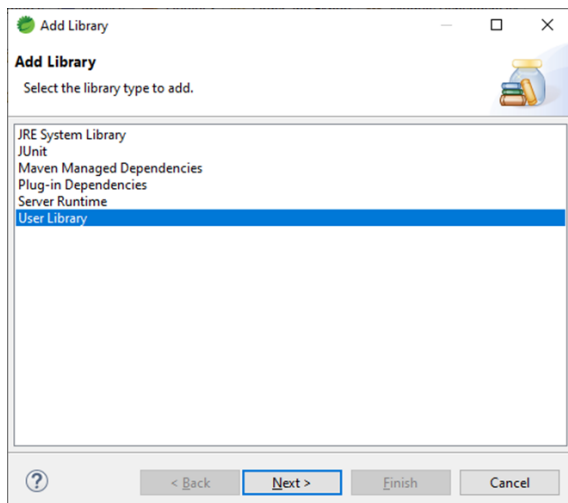
1. Crie um novo projeto Java (*File → New → Java Project*) – dê o nome de *TesteJDBC*, desmarque a opção *Create module-info.java file* e clique em *Next*



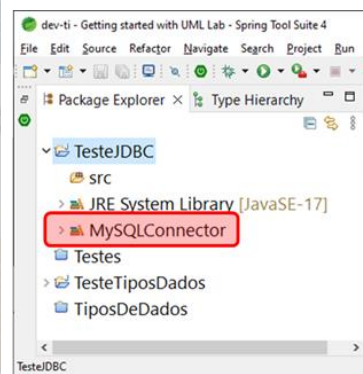
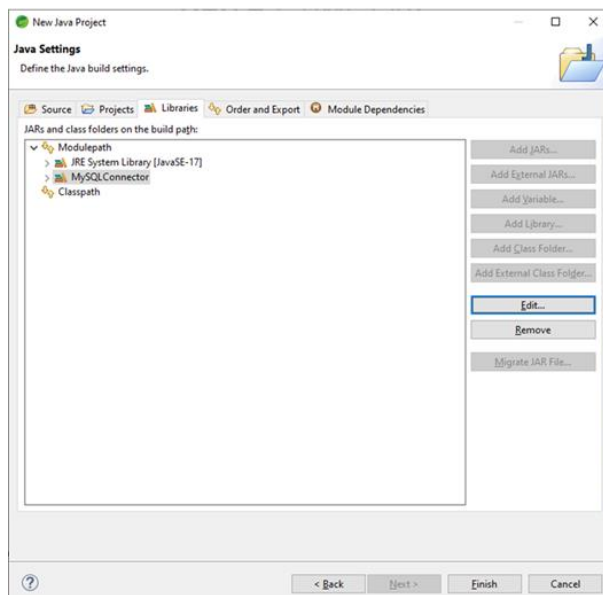
2. Clique na aba *Libraries*, selecione o item *Modulepath* e então clique no botão *Add Library ...*



3. Escolha a opção *User Library* e clique em *Next* – a seguir marque a biblioteca criada anteriormente (*MySQLConnector*) e clique no botão *Finish*



4. A biblioteca deve ter sido acrescentada no projeto – clique novamente em *Finish*



## 4.2 Criação da Conexão

Crie o pacote `db` e nele a classe `FabricaConexao`:

- A implementação devolve uma única instância (*singleton*) de uma conexão
- `getConexao()` é uma fábrica de conexões (cria novas conexões)

```
1 package db;
2 import java.sql.Connection;
3 import java.sql.DriverManager;
4 import java.sql.SQLException;
5
6 public class FabricaConexao {
7     private static Connection conexao = null;
8
9     private FabricaConexao() { }
10
11     public static Connection getConexao() {
12         try {
13             final String url = "jdbc:mysql://localhost/unoesc_crud_jdbc";
14             final String usuario = "root";
15             final String senha = "root";
16
17             conexao = DriverManager.getConnection(url, usuario, senha);
18             System.out.println("Conexão realizada com sucesso!");
19
20             return conexao;
21         } catch (SQLException e) {
22             throw new RuntimeException(e.getMessage());
23         }
24     }
25
26     public static void fechaConexao() {
27         if (conexao != null) {
28             try {
29                 conexao.close();
30                 System.out.println("Conexão fechada com sucesso!");
31             } catch (SQLException e) {
32                 throw new RuntimeException(e.getMessage());
33             }
34         }
35     }
36 }
```

Crie a classe `Principal` dentro do pacote `app` para testar a conexão:

```
1 package app;
2
3 import java.sql.Connection;
4
5 import db.FabricaConexao;
6
7 public class Principal {
8     public static void main(String[] args) {
9         Connection conexao = FabricaConexao.getConexao();
10        FabricaConexao.fechaConexao();
11    }
12 }
```

### 4.3 Classe de Domínio

A entidade de domínio (classe `Produto`) representa um produto na tabela no banco de dados no modelo relacional. Crie-a dentro do pacote `modelo`.

- Utilize a IDE para criar os construtores de forma automatizada
- Da mesma forma, crie os métodos do tipo `get` e `set` com ajuda da IDE
- Use a IDE para criar o método `toString()` – representação do objeto
- Atenção na hora de importar o pacote relativo ao objeto `dataCadastro` – deve-se importar o pacote `java.sql.Date` e não `java.util.Date`.

```
1 package modelo;
2
3 import java.math.BigDecimal;
4 import java.sql.Date;
5
6 public class Produto {
7     private Integer idProd;
8     private String nomeProd;
9     private Date dataCadastro;
10    private Integer quantidade;
11    private BigDecimal preco;
12
13    public Produto() { }
14
15    public Produto(Integer idProd, String nomeProd, Date dataCadastro, Integer quantidade, BigDecimal preco) {
16        super();
17        this.idProd = idProd;
18        this.nomeProd = nomeProd;
19        this.dataCadastro = dataCadastro;
20        this.quantidade = quantidade;
21        this.preco = preco;
22    }
23
24    public Integer getIdProd() { return idProd; }
25    public void setIdProd(Integer idProd) { this.idProd = idProd; }
26
27    public String getNomeProd() { return nomeProd; }
28    public void setNomeProd(String nomeProd) { this.nomeProd = nomeProd; }
29
30    public Date getDataCadastro() { return dataCadastro; }
31    public void setDataCadastro(Date dataCadastro) { this.dataCadastro = dataCadastro; }
32
33    public Integer getQuantidade() { return quantidade; }
34    public void setQuantidade(Integer quantidade) { this.quantidade = quantidade; }
35
36    public BigDecimal getPreco() { return preco; }
37    public void setPreco(BigDecimal preco) { this.preco = preco; }
38
39    @Override
40    public String toString() {
41        return "Produto [idProd=" + idProd + ", nomeProd=" + nomeProd + ", dataCadastro=" + dataCadastro
42            + ", quantidade=" + quantidade + ", preco=" + preco + "]\n";
43    }
44 }
```

### 4.4 Interface para a Classe DAO

Crie a interface `IProdutoDAO` dentro do pacote `dao`:

```
1 package dao;
2
3 import java.util.List;
4
5 import modelo.Produto;
6
7 public interface IProdutoDAO {
8     void adicionar(Produto p);
9     void alterar(Produto p);
10    void excluir(Integer id);
11    List<Produto> listarTodos();
12    Produto buscarPorId(Integer id);
13 }
```

## 4.5 Funcionalidade Listar

### 4.5.1 Classe DAO

Crie a classe `ProdutoDAO`, que implementa a *interface* `IProdutoDAO` e codifique o método `listarTodos()`.

Atenção: Seguindo boas práticas de programação, os objetos `Statement` e `ResultSet` também devem ser fechados. Isso não foi feito de forma a manter o tamanho do código menor.

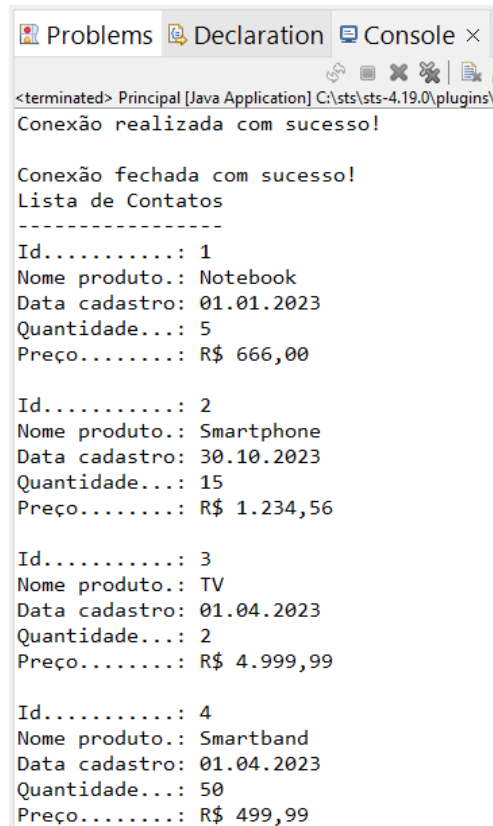
```
1 package dao;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.util.ArrayList;
8 import java.util.List;
9
10 import db.FabricaConexao;
11 import modelo.Produto;
12
13 public class ProdutoDAO implements IProdutoDAO {
14     private Connection conexao;
15
16     @Override
17     public List<Produto> listarTodos() {
18         conexao = FabricaConexao.getConexao();
19         List<Produto> lista = new ArrayList<>();
20
21         try {
22             String sql = "SELECT * FROM produto";
23             PreparedStatement stmt = this.conexao.prepareStatement(sql);
24
25             ResultSet rs = stmt.executeQuery();
26             while (rs.next()) {
27                 Produto ct = new Produto();
28
29                 ct.setIdProd(rs.getInt("id_prod"));
30                 ct.setNomeProd(rs.getString("nome_prod"));
31                 ct.setDataCadastro(rs.getDate("data_cadastro"));
32                 ct.setQuantidade(rs.getInt("quantidade"));
33                 ct.setPreco(rs.getBigDecimal("preco"));
34
35                 lista.add(ct);
36             }
37         } catch (SQLException e) {
38             throw new RuntimeException(e.getMessage());
39         } finally {
40             FabricaConexao.fechaConexao();
41         }
42
43         return lista;
44     }
45
46     @Override
47     public void adicionar(Produto p) {
48         // TODO Auto-generated method stub
49     }
50
51     @Override
52     public void alterar(Produto p) {
53         // TODO Auto-generated method stub
54     }
55
56     @Override
57     public void excluir(Integer id) {
58         // TODO Auto-generated method stub
59     }
60
61     @Override
62     public Produto buscarPorId(Integer id) {
63         // TODO Auto-generated method stub
64         return null;
65     }
66 }
67
68
```

#### 4.5.2 Teste no Programa Principal

Modifique a classe `Principal` a fim de testar o método recém-criado.

```
1 package app;
2
3 import java.text.NumberFormat;
4 import java.text.SimpleDateFormat;
5 import java.util.List;
6
7 import dao.ProdutoDAO;
8 import modelo.Produto;
9
10 public class Principal {
11     public static void main(String[] args) {
12         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
13         NumberFormat fm = NumberFormat.getCurrencyInstance();
14
15         ProdutoDAO dao = new ProdutoDAO();
16
17         List<Produto> lista = dao.listarTodos();
18
19         System.out.println("Lista de Contatos");
20         System.out.println("-----");
21
22         for (Produto produto : lista) {
23             System.out.println("Id.....: " + produto.getIdProd());
24             System.out.println("Nome produto.: " + produto.getNomeProd());
25             System.out.println("Data cadastro: " + fd.format(produto.getDataCadastro()));
26             System.out.println("Quantidade....: " + produto.getQuantidade());
27             System.out.println("Preço.....: " + fm.format(produto.getPreco()));
28             System.out.println();
29         }
30     }
31 }
```

Resultado:



```
<terminated> Principal [Java Application] C:\sts\sts-4.19.0\plugins\
Conexão realizada com sucesso!

Conexão fechada com sucesso!
Lista de Contatos
-----
Id.....: 1
Nome produto.: Notebook
Data cadastro: 01.01.2023
Quantidade....: 5
Preço.....: R$ 666,00

Id.....: 2
Nome produto.: Smartphone
Data cadastro: 30.10.2023
Quantidade....: 15
Preço.....: R$ 1.234,56

Id.....: 3
Nome produto.: TV
Data cadastro: 01.04.2023
Quantidade....: 2
Preço.....: R$ 4.999,99

Id.....: 4
Nome produto.: Smartband
Data cadastro: 01.04.2023
Quantidade....: 50
Preço.....: R$ 499,99
```



## 4.6 Funcionalidade Adicionar

### 4.6.1 Classe DAO

Na classe `ProdutoDAO`, acrescente o método `adicionar()`.

```
46 @Override
47 public void adicionar(Produto p) {
48     conexao = FabricaConexao.getConexao();
49
50     try {
51         String sql = "INSERT INTO produto (nome_prod, data_cadastro, quantidade, preco) "
52             + "VALUES (?, ?, ?, ?)";
53
54         PreparedStatement stmt = this.conexao.prepareStatement(sql);
55         stmt.setString(1, p.getNomeProd());
56         stmt.setDate(2, p.getDataCadastro());
57         stmt.setInt(3, p.getQuantidade());
58         stmt.setBigDecimal(4, p.getPreco());
59
60         stmt.execute();
61     } catch (SQLException e) {
62         throw new RuntimeException(e.getMessage());
63     } finally {
64         FabricaConexao.fechaConexao();
65     }
66 }
```

### 4.6.2 Teste no Programa Principal

Modifique a classe `Principal` a fim de testar o método recém-criado:

```
1 package app;
2
3 import java.math.BigDecimal;
4 import java.sql.Date;
5 import java.text.NumberFormat;
6 import java.text.SimpleDateFormat;
7 import java.time.LocalDate;
8 import java.util.List;
9
10 import dao.ProdutoDAO;
11 import modelo.Produto;
12
13 public class Principal {
14     public static void main(String[] args) {
15         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
16         NumberFormat fm = NumberFormat.getCurrencyInstance();
17
18         ProdutoDAO dao = new ProdutoDAO();
19
20         Produto prod = new Produto(null, "TV",
21             Date.valueOf(LocalDate.now()),
22             15,
23             new BigDecimal("20000.5"));
24         dao.adicionar(prod);
25
26         List<Produto> lista = dao.listarTodos();
27
28         System.out.println("Lista de Contatos");
29         System.out.println("-----");
30
31         for (Produto produto : lista) {
32             System.out.println("Id.....: " + produto.getIdProd());
33             System.out.println("Nome produto.: " + produto.getNomeProd());
34             System.out.println("Data cadastro: " + fd.format(produto.getDataCadastro()));
35             System.out.println("Quantidade...: " + produto.getQuantidade());
36             System.out.println("Preço.....: " + fm.format(produto.getPreco()));
37             System.out.println();
38         }
39     }
40 }
```

Um novo produto deve ter sido inserido na base e mostrado na tela. Após isso **comente** a linha 24 que chama o método `adicionar()` a fim de evitar inserir um novo registro cada vez que o programa for executado.

Resultado:

```
Lista de Contatos
-----
Id.....: 1
Nome produto.: Notebook
Data cadastro: 01.01.2023
Quantidade...: 5
Preço.....: R$ 666,00

Id.....: 2
Nome produto.: Smartphone
Data cadastro: 30.10.2023
Quantidade...: 15
Preço.....: R$ 1.234,56

Id.....: 3
Nome produto.: TV
Data cadastro: 01.04.2023
Quantidade...: 2
Preço.....: R$ 4.999,99

Id.....: 4
Nome produto.: Smartband
Data cadastro: 01.04.2023
Quantidade...: 50
Preço.....: R$ 499,99

Id.....: 5
Nome produto.: TV
Data cadastro: 31.10.2023
Quantidade...: 15
Preço.....: R$ 20.000,50
```

Anote o Id do novo produto inserido (no exemplo acima é o Id 5), para utilizar na próxima seção.

## 4.7 Funcionalidade Alterar

### 4.7.1 Classe DAO

Na classe `ProdutoDAO`, acrescente o método `alterar()`.

```
68⇒ @Override
69 public void alterar(Produto p) {
70     conexao = FabricaConexao.getConexao();
71
72     try {
73         String sql = "UPDATE produto SET nome_prod=?, data_cadastro=?, quantidade=?, preco=? " +
74             "WHERE id_prod=?";
75
76         PreparedStatement stmt = this.conexao.prepareStatement(sql);
77         stmt.setString(1, p.getNomeProd());
78         stmt.setDate(2, p.getDataCadastro());
79         stmt.setInt(3, p.getQuantidade());
80         stmt.setBigDecimal(4, p.getPreco());
81
82         stmt.setInt(5, p.getIdProd());
83
84         stmt.execute();
85     } catch (SQLException e) {
86         throw new RuntimeException(e.getMessage());
87     } finally {
88         FabricaConexao.fechaConexao();
89     }
90 }
```

#### 4.7.2 Teste no Programa Principal

Modifique a classe `Principal` a fim de testar o método recém-criado. Na linha 23 utilize o `id` anotado anteriormente, do último registro inserido.

```
1 package app;
2
3 import java.math.BigDecimal;
4 import java.sql.Date;
5 import java.text.NumberFormat;
6 import java.text.SimpleDateFormat;
7 import java.util.List;
8
9 import dao.ProdutoDAO;
10 import modelo.Produto;
11
12 public class Principal {
13     public static void main(String[] args) {
14         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
15         NumberFormat fm = NumberFormat.getCurrencyInstance();
16
17         ProdutoDAO dao = new ProdutoDAO();
18
19         Produto prod = new Produto(null, "TV modificada",
20                                     Date.valueOf("2023-10-31"),
21                                     51,
22                                     new BigDecimal("123.45"));
23         prod.setIdProd(5);
24         dao.alterar(prod);
25
26         // dao.adicionar(prod);
27
28         List<Produto> lista = dao.listarTodos();
29
30         System.out.println("Lista de Contatos");
31         System.out.println("-----");
32
33         for (Produto produto : lista) {
34             System.out.println("Id.....: " + produto.getIdProd());
35             System.out.println("Nome produto.: " + produto.getNomeProd());
36             System.out.println("Data cadastro: " + fd.format(produto.getDataCadastro()));
37             System.out.println("Quantidade...: " + produto.getQuantidade());
38             System.out.println("Preço.....: " + fm.format(produto.getPreco()));
39             System.out.println();
40         }
41     }
42 }
```

Resultado:

```
Lista de Contatos
-----
Id.....: 1
Nome produto.: Notebook
Data cadastro: 01.01.2023
Quantidade...: 5
Preço.....: R$ 666,00

Id.....: 2
Nome produto.: Smartphone
Data cadastro: 30.10.2023
Quantidade...: 15
Preço.....: R$ 1.234,56

Id.....: 3
Nome produto.: TV
Data cadastro: 01.04.2023
Quantidade...: 2
Preço.....: R$ 4.999,99

Id.....: 4
Nome produto.: Smartband
Data cadastro: 01.04.2023
Quantidade...: 50
Preço.....: R$ 499,99

Id.....: 5
Nome produto.: TV modificada
Data cadastro: 31.10.2023
Quantidade...: 51
Preço.....: R$ 123,45
```

## 4.8 Funcionalidade Excluir

### 4.8.1 Classe DAO

Na classe `ProdutoDAO`, acrescente o método `excluir()`.

```
92 @Override
93 public void excluir(Integer id) {
94     conexao = FabricaConexao.getConexao();
95
96     try {
97         String sql = "DELETE FROM produto WHERE id_prod=?";
98
99         PreparedStatement stmt = this.conexao.prepareStatement(sql);
100         stmt.setInt(1, id);
101
102         stmt.execute();
103     } catch (SQLException e) {
104         throw new RuntimeException(e.getMessage());
105     } finally {
106         FabricaConexao.fechaConexao();
107     }
108 }
109
```

### 4.8.2 Teste no Programa Principal

Modifique a classe `Principal` a fim de testar o método recém-criado. Na linha 27 utilize o `id` anotado anteriormente, do último registro inserido. As linhas 19 a 25 podem ser comentadas pois o único dado necessário para o método `excluir()` é o `id`.

```
1 package app;
2
3 import java.math.BigDecimal;
4 import java.sql.Date;
5 import java.text.NumberFormat;
6 import java.text.SimpleDateFormat;
7 import java.util.List;
8
9 import dao.ProdutoDAO;
10 import modelo.Produto;
11
12 public class Principal {
13     public static void main(String[] args) {
14         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
15         NumberFormat fm = NumberFormat.getCurrencyInstance();
16
17         ProdutoDAO dao = new ProdutoDAO();
18
19         // Produto prod = new Produto(null, "TV modificada",
20         //                               Date.valueOf("2023-10-31"),
21         //                               51,
22         //                               new BigDecimal("123.45"));
23         // prod.setIdProd(5);
24         // dao.alterar(prod);
25         // dao.adicionar(prod);
26
27         dao.excluir(5);
28
29         List<Produto> lista = dao.listarTodos();
30
31         System.out.println("Lista de Contatos");
32         System.out.println("-----");
33
34         for (Produto produto : lista) {
35             System.out.println("Id.....: " + produto.getIdProd());
36             System.out.println("Nome produto.: " + produto.getNomeProd());
37             System.out.println("Data cadastro: " + fd.format(produto.getDataCadastro()));
38             System.out.println("Quantidade...: " + produto.getQuantidade());
39             System.out.println("Preço.....: " + fm.format(produto.getPreco()));
40             System.out.println();
41         }
42     }
43 }
```

No resultado abaixo pode-se ver que o registro com `id` 5 foi excluído da base de dados:

```
Lista de Contatos
-----
Id.....: 1
Nome produto.: Notebook
Data cadastro: 01.01.2023
Quantidade...: 5
Preço.....: R$ 666,00

Id.....: 2
Nome produto.: Smartphone
Data cadastro: 30.10.2023
Quantidade...: 15
Preço.....: R$ 1.234,56

Id.....: 3
Nome produto.: TV
Data cadastro: 01.04.2023
Quantidade...: 2
Preço.....: R$ 4.999,99

Id.....: 4
Nome produto.: Smartband
Data cadastro: 01.04.2023
Quantidade...: 50
Preço.....: R$ 499,99
```

## 4.9 Funcionalidade de Busca por Id

### 4.9.1 Classe DAO

Na classe `ProdutoDAO`, acrescente o método `buscarPorId()`.

```
110 @Override
111 public Produto buscarPorId(Integer id) {
112     conexao = FabricaConexao.getConexao();
113
114     Produto p = null;
115
116     try {
117         String sql = "SELECT * FROM produto WHERE id_prod=?";
118
119         PreparedStatement stmt = this.conexao.prepareStatement(sql);
120         stmt.setInt(1, id);
121
122         ResultSet rs = stmt.executeQuery();
123         if (rs.next()) {
124             p = new Produto();
125
126             p.setIdProd(rs.getInt("id_prod"));
127             p.setNomeProd(rs.getString("nome_prod"));
128             p.setDataCadastro(rs.getDate("data_cadastro"));
129             p.setQuantidade(rs.getInt("quantidade"));
130             p.setPreco(rs.getBigDecimal("preco"));
131         }
132     } catch (SQLException e) {
133         throw new RuntimeException(e.getMessage());
134     } finally {
135         FabricaConexao.fechaConexao();
136     }
137
138     return p;
139 }
```

#### 4.9.2 Teste no Programa Principal

Modifique a classe `Principal` a fim de testar o método recém-criado.

```
1 package app;
2
3 import java.math.BigDecimal;
4 import java.sql.Date;
5 import java.text.NumberFormat;
6 import java.text.SimpleDateFormat;
7 import java.util.List;
8
9 import dao.ProdutoDAO;
10 import modelo.Produto;
11
12 public class Principal {
13     public static void main(String[] args) {
14         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
15         NumberFormat fm = NumberFormat.getCurrencyInstance();
16
17         ProdutoDAO dao = new ProdutoDAO();
18
19         // Produto prod = new Produto(null, "TV modificada",
20         //                               Date.valueOf("2023-10-31"),
21         //                               51,
22         //                               new BigDecimal("123.45"));
23         // prod.setIdProd(5);
24         // dao.alterar(prod);
25         // dao.adicionar(prod);
26
27         // dao.excluir(5);
28
29         List<Produto> lista = dao.listarTodos();
30
31         System.out.println("Lista de Contatos");
32         System.out.println("-----");
33
34         for (Produto produto : lista) {
35             System.out.println("Id.....: " + produto.getIdProd());
36             System.out.println("Nome produto.: " + produto.getNomeProd());
37             System.out.println("Data cadastro: " + fd.format(produto.getDataCadastro()));
38             System.out.println("Quantidade...: " + produto.getQuantidade());
39             System.out.println("Preço.....: " + fm.format(produto.getPreco()));
40             System.out.println();
41         }
42
43         Produto prod = dao.buscarPorId(2);
44         if (prod != null) {
45             System.out.println("Id.....: " + prod.getIdProd());
46             System.out.println("Nome produto.: " + prod.getNomeProd());
47             System.out.println("Data cadastro: " + fd.format(prod.getDataCadastro()));
48             System.out.println("Quantidade...: " + prod.getQuantidade());
49             System.out.println("Preço.....: " + fm.format(prod.getPreco()));
50             System.out.println();
51         }
52     }
53 }
```

Resultado (parte final):

```
Id.....: 2
Nome produto.: Smartphone
Data cadastro: 30.10.2023
Quantidade...: 15
Preço.....: R$ 1.234,56
```

#### 4.10 Exercício (opcional)

- Modifique a tabela `produtos` acrescentando um novo campo chamado `observação` do tipo `VARCHAR(50)`.
- Altere a classe de domínio (`Produto`), a DAO e o programa principal de forma a suportar este novo campo.

## 5 Desenvolvimento do Projeto (Versão 2)

Melhorias:

- Testes com o SGBD PostgreSQL
- Utilização de exceção personalizada
- Externalização dos dados de conexão ao banco em um arquivo externo
- Liberação de recursos `Statement` e `ResultSet`
- Unificação dos métodos `adicionar()` e `alterar()` em um só método `salvar()`
- Adição do método `buscarPorNome()` com utilização de filtro
- Adição de método que retorna o número de registros na tabela
- Retorno da chave do registro no caso de inserção na tabela
- Indicação de quantos registros foram afetados no método `salvar()`
- Retorno *booleano* no método `excluir()` e posterior verificação de exclusão
- Controle transacional com `setAutoCommit(false)`, `commit()` e `rollback()`

### 5.1 Testes com o SGBD PostgreSQL

#### 5.1.1 Criação do Banco de Dados

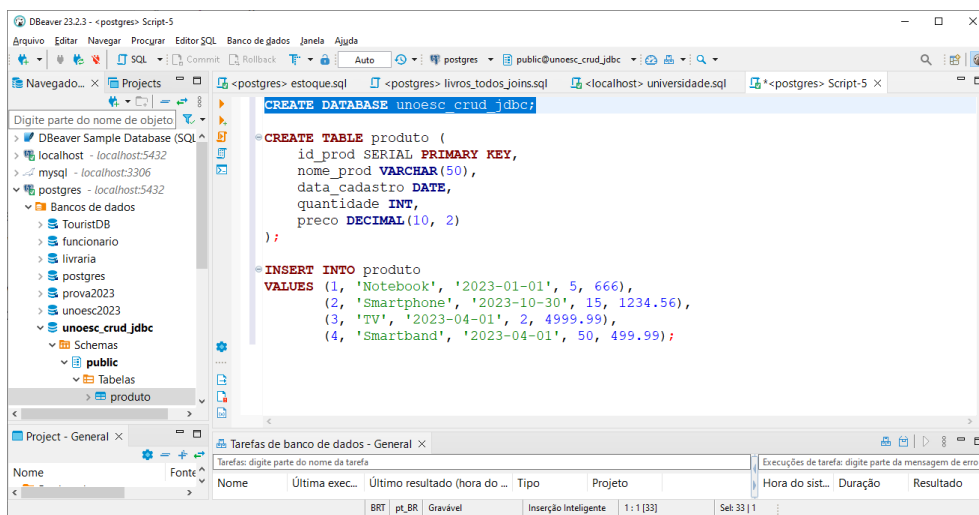
Crie o banco de dados, a tabela e insira os dados nela usando as consultas abaixo:

```
CREATE DATABASE unoesc_crud_jdbc;
```

```
CREATE TABLE produto (  
    id_prod SERIAL PRIMARY KEY,  
    nome_prod VARCHAR(50),  
    data_cadastro DATE,  
    quantidade INT,  
    preco DECIMAL(10, 2)  
);
```

```
INSERT INTO produto  
VALUES (1, 'Notebook', '2023-01-01', 5, 666),  
       (2, 'Smartphone', '2023-10-30', 15, 1234.56),  
       (3, 'TV', '2023-04-01', 2, 4999.99),  
       (4, 'Smartband', '2023-04-01', 50, 499.99);
```

DBeaver:



Dados inseridos:

	id_prod	nome_prod	data_cadastro	quantidade	preco
1	1	Notebook	2023-01-01	5	666
2	2	Smartphone	2023-10-30	15	1.234,56
3	3	TV	2023-04-01	2	4.999,99
4	4	Smartband	2023-04-01	50	499,99

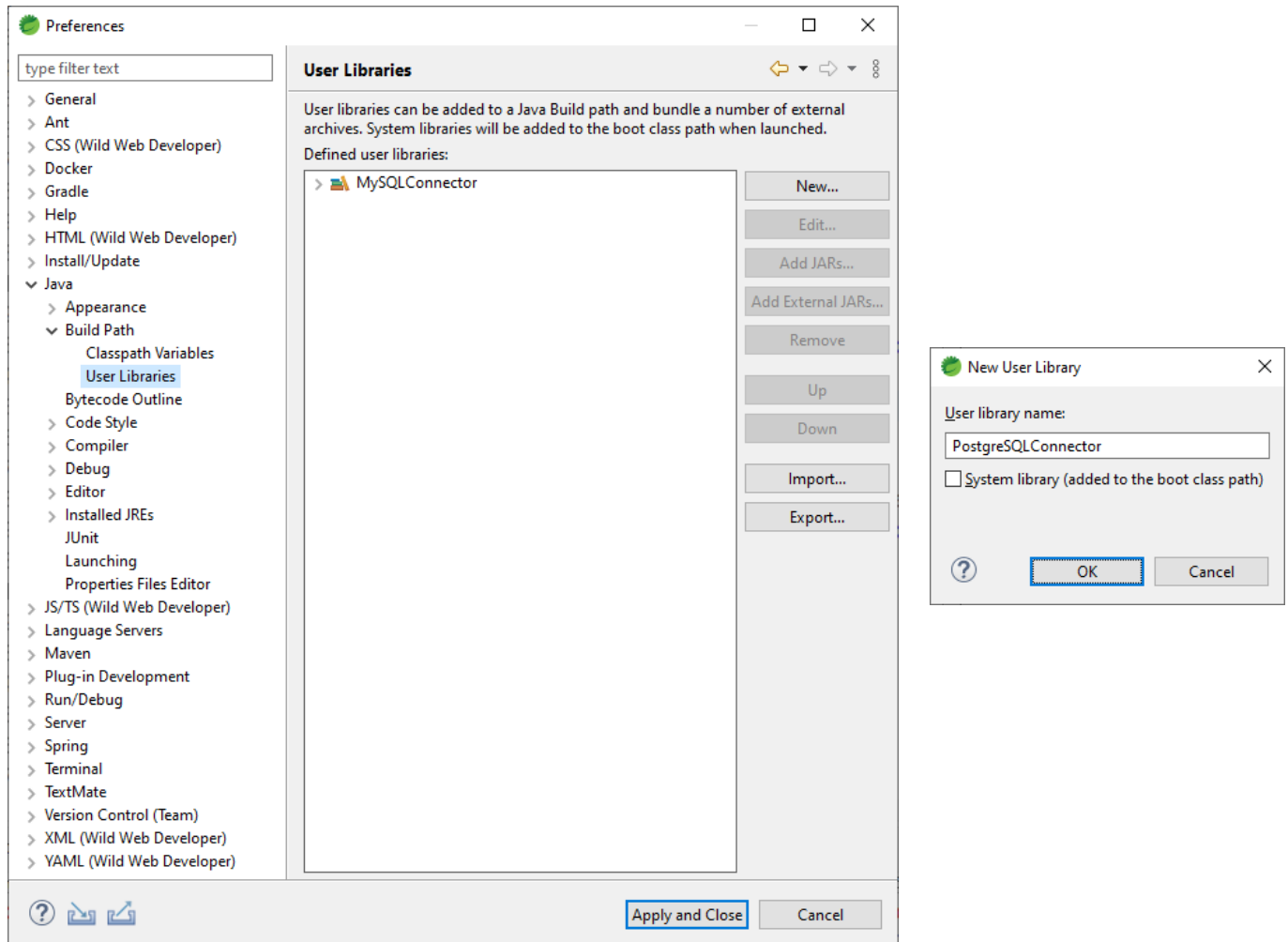


### 5.1.2 Criação da Biblioteca de Usuário

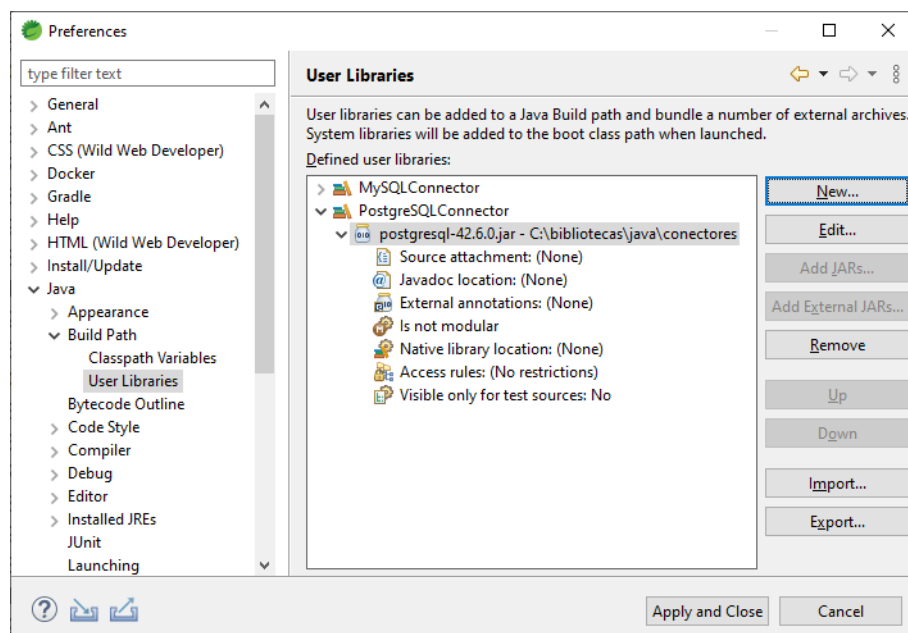
Primeiramente baixe o *driver* JDBC para o PostgreSQL [neste endereço](#). Copie o arquivo baixado para a pasta `C:\bibliotecas\java\conectores`.

Siga os passos abaixo para configurar uma biblioteca de usuário no STS:

1. No STS vá em *Window* → *Preferences*, abra a categoria *Java* → *Build Path* → *User Libraries* e então clique no botão *New ...* - a seguir informe o nome *PostgreSQLConnector*.

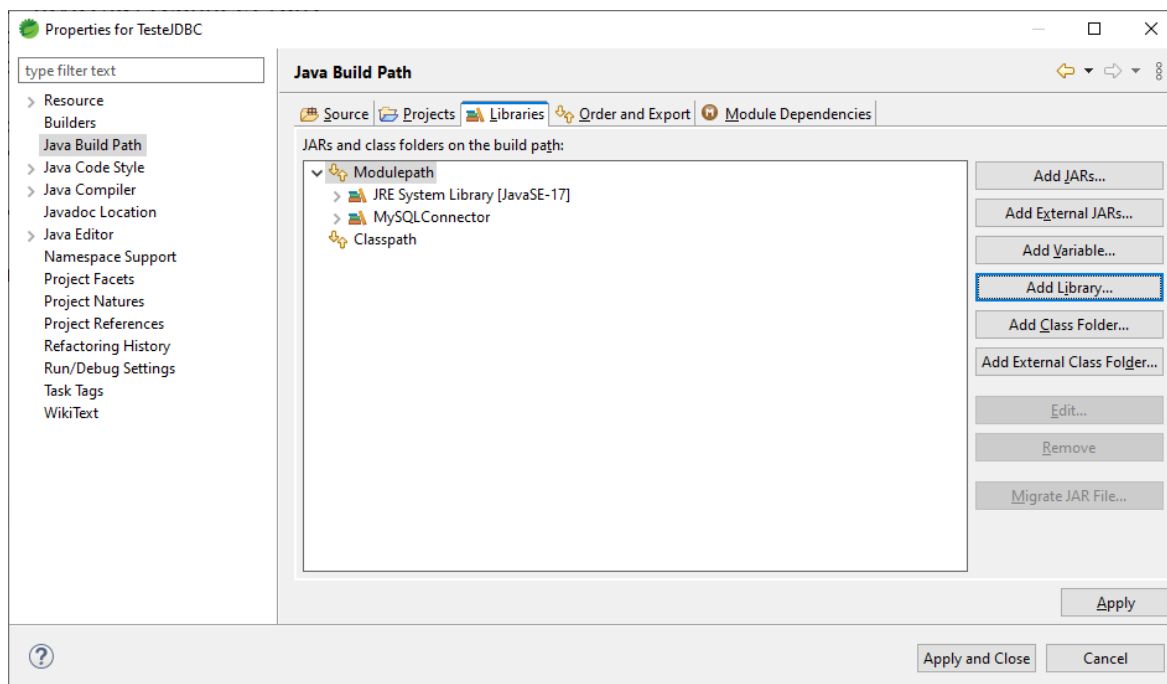


2. Clique agora no botão *Add External JARs ...* e acrescente a biblioteca baixada – após isso clique em *Apply and Close*

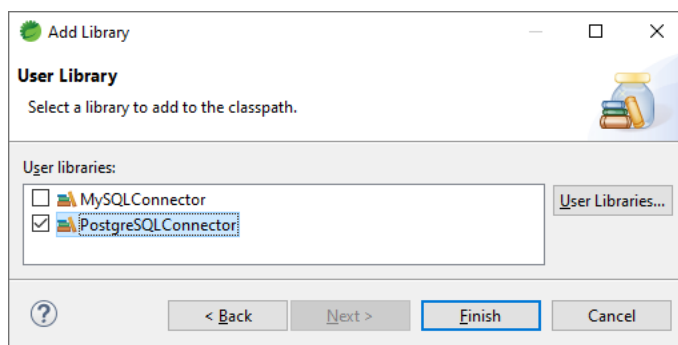
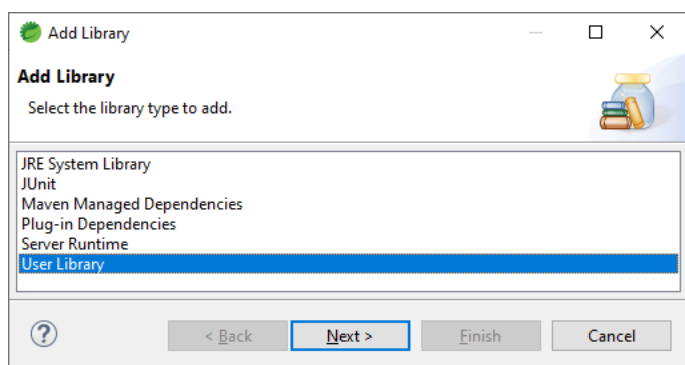




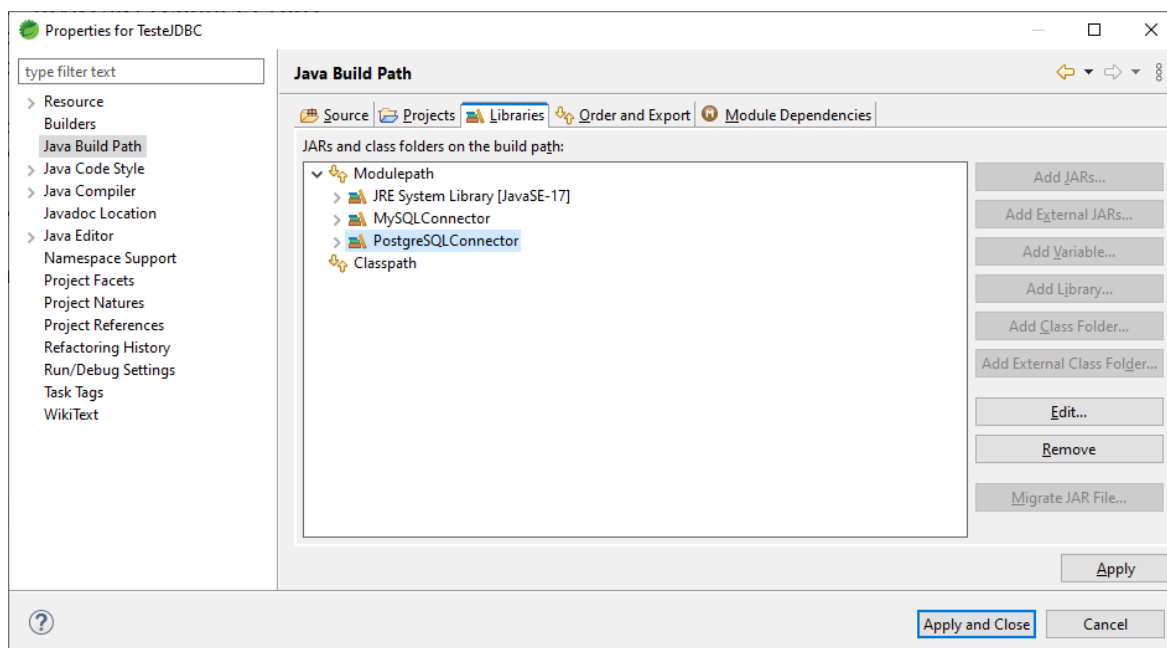
3. Clique com o botão direito sobre o projeto e escolha *Properties*. Vá em *Java Build Path*, clique em *Modulepath* e então no botão *Add Library*:



4. Na janela que irá aparecer clique em *User Library* e então em *Next*, a seguir selecione a biblioteca recém criada e clique em *Finish*:



5. Clique em *Apply and Close*.

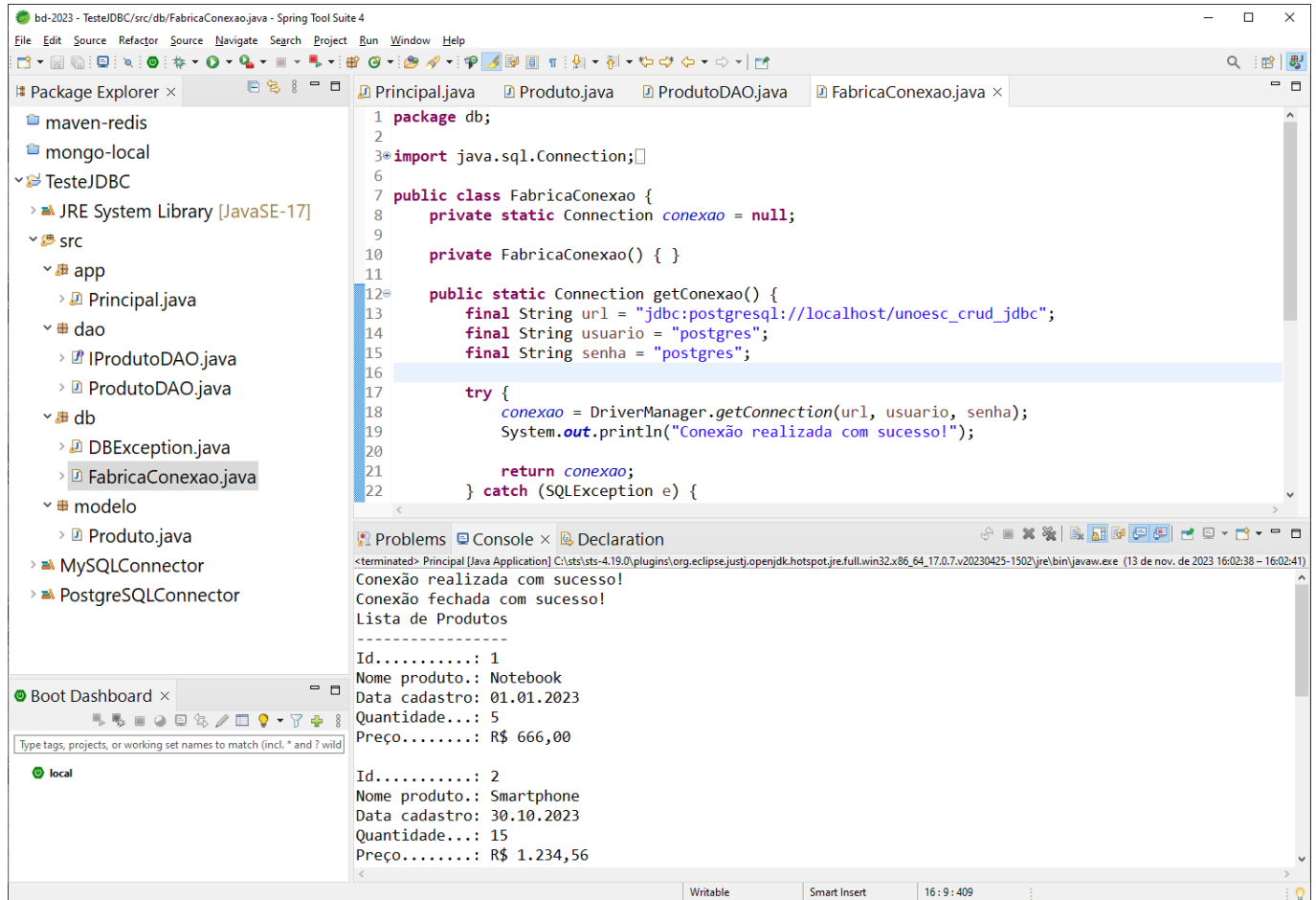


### 5.1.3 Utilização da Biblioteca de Usuário

Na classe `FabricaConexao` modifique a *string* de conexão e também o usuário e senha, caso necessário.

```
final String url = "jdbc:postgresql://localhost/unoesc_crud_jdbc";  
final String usuario = "postgres";  
final String senha = "postgres";
```

Ao executar, a conexão deverá ser feita ao SGBD PostgreSQL e a DAO deve estar funcionando como antes, não sendo necessário realizar nenhuma modificação na classe `ProdutoDAO` e nem no arquivo `Principal.java`:



Caso ocorra algum erro certifique-se que o servidor do PostgreSQL está sendo executado.

## 5.2 Utilização de Exceção Personalizada

É uma boa prática de programação criar as próprias classes de exceção caso seja necessário um tratamento personalizado para elas.

Crie a classe `DbException` dentro do pacote `db` com o seguinte código:

```
1 package db;  
2  
3 public class DbException extends RuntimeException {  
4     public DbException(String mensagem) {  
5         super("DAO: " + mensagem);  
6     }  
7 }
```

Modifique agora as linhas 23 e 33 da classe `FabricaConexao`:

```
1 package db;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class FabricaConexao {
8     private static Connection conexao = null;
9
10    private FabricaConexao() { }
11
12    public static Connection getConexao() {
13        final String url = "jdbc:mysql://localhost/unoesc_crud_jdbc";
14        final String usuario = "root";
15        final String senha = "";
16
17        try {
18            conexao = DriverManager.getConnection(url, usuario, senha);
19            System.out.println("Conexão realizada com sucesso!");
20
21            return conexao;
22        } catch (SQLException e) {
23            throw new DbException(e.getMessage());
24        }
25    }
26
27    public static void fechaConexao() {
28        if (conexao != null) {
29            try {
30                conexao.close();
31                System.out.println("Conexão fechada com sucesso!");
32            } catch (SQLException e) {
33                throw new DbException(e.getMessage());
34            }
35        }
36    }
37 }
```

Caso ocorra algum erro de senha na abertura da conexão, será mostrada agora uma mensagem semelhante à esta:

```
Exception in thread "main" db.DbException: DAO: Access denied for user 'root'@'localhost' (using password: YES)
    at db.FabricaConexao.getConexao(FabricaConexao.java:23)
    at dao.ProdutoDAO.listarTodos(ProdutoDAO.java:84)
    at app.Principal.main(Principal.java:34)
```

Ou este aqui caso haja algum problema de conexão do cliente com o servidor, por exemplo, se o servidor não estiver sendo executado:

```
Exception in thread "main" db.DbException: DAO: Communications link failure
```

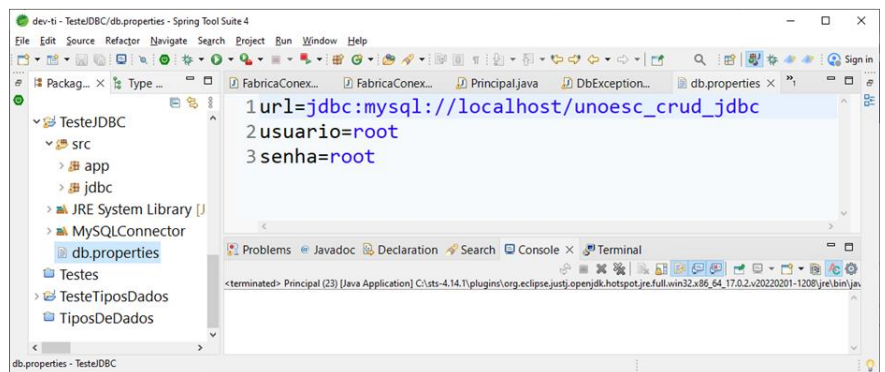
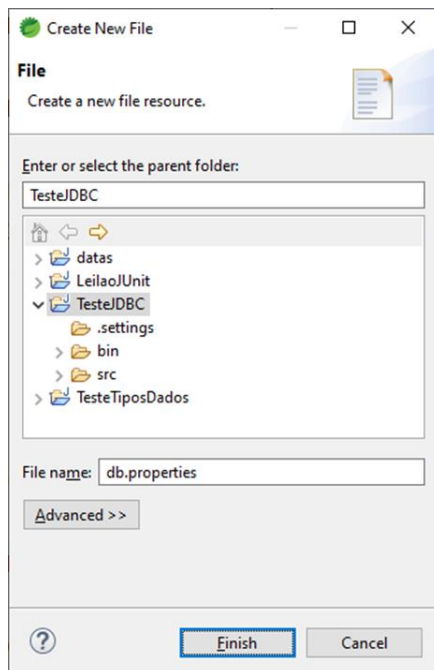
```
The last packet sent successfully to the server was 0 milliseconds ago. The driver has not received any packets from the server.
    at db.FabricaConexao.getConexao(FabricaConexao.java:23)
    at dao.ProdutoDAO.listarTodos(ProdutoDAO.java:84)
    at app.Principal.main(Principal.java:34)
```

### 5.3 Externalização dos Dados de Conexão ao Banco

Os dados da conexão, como usuário e senha, não devem estar *hard-coded* pois isso pode ser uma grande brecha de segurança. Também deve-se cuidar para que estas informações não acabem sendo transferidas para servidores como o GitHub. Uma solução para isso é armazenar os dados de conexão em um arquivo externo e excluí-lo do sistema de versionamento (por exemplo, listando-o no arquivo `.gitignore` do Git).

Para fazer este procedimento, clique com o botão direito sobre o projeto `TesteJDBC` e escolha `New → File`.

A seguir crie o arquivo chamado *db.properties* e insira nele o conteúdo mostrado abaixo:

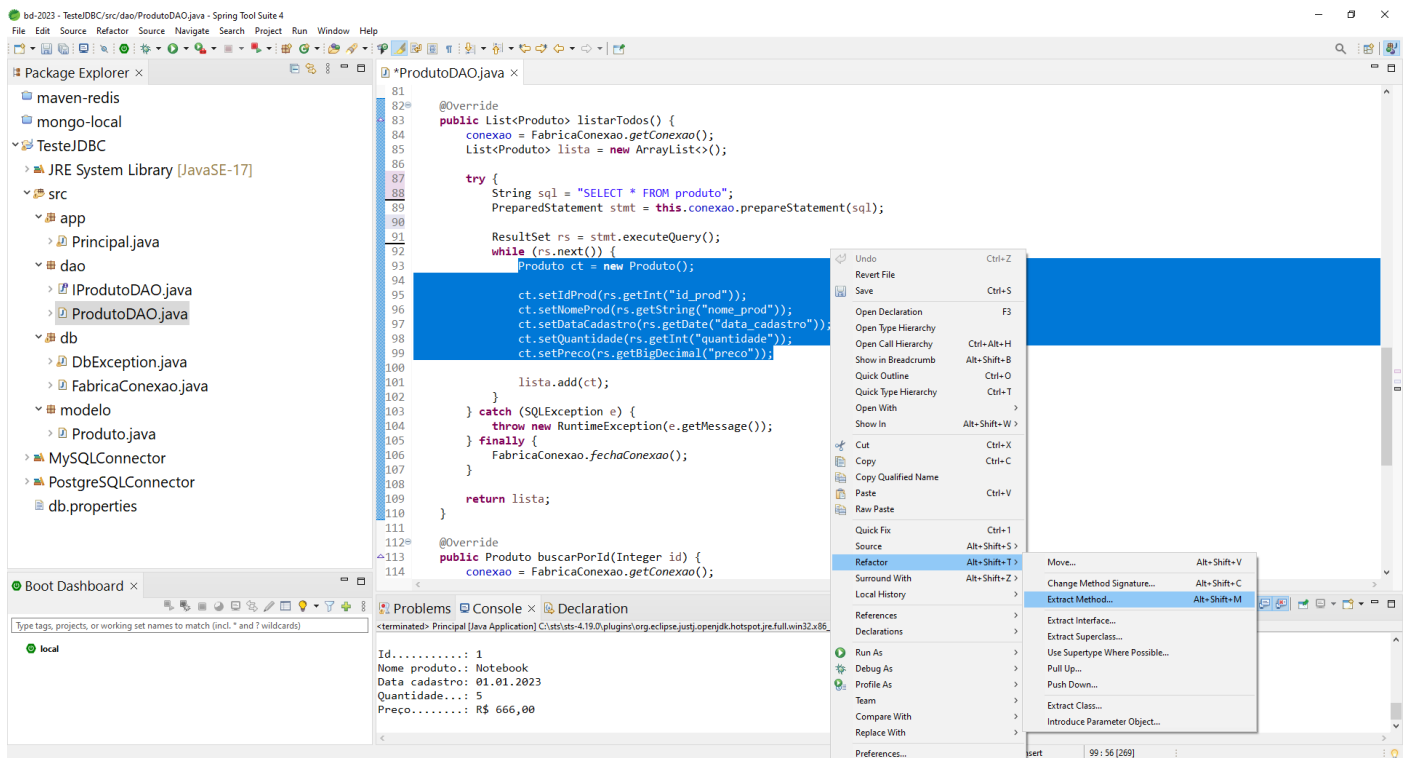


Código:

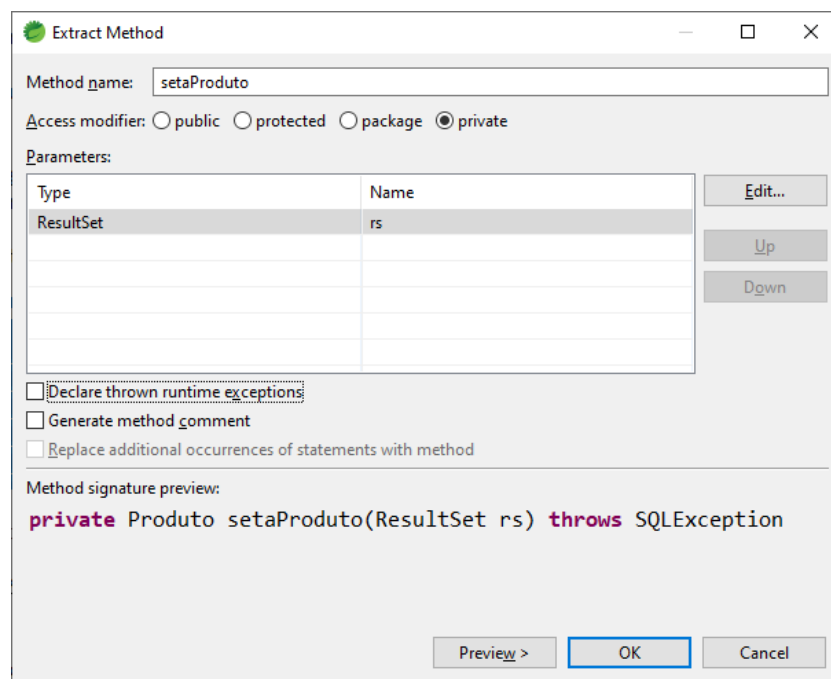
```
1 package db;
2
3 import java.io.FileInputStream;
4 import java.io.FileNotFoundException;
5 import java.io.IOException;
6 import java.sql.Connection;
7 import java.sql.DriverManager;
8 import java.sql.SQLException;
9 import java.util.Properties;
10
11 public class FabricaConexao {
12     private static Connection conexao = null;
13
14     private FabricaConexao() { }
15
16     public static Connection getConexao() {
17         try {
18             if (conexao != null && !conexao.isClosed()) {
19                 return conexao;
20             }
21
22             Properties prop = LoadProperties();
23             final String url = prop.getProperty("url");
24             final String usuario = prop.getProperty("usuario");
25             final String senha = prop.getProperty("senha");
26
27             conexao = DriverManager.getConnection(url, usuario, senha);
28             return conexao;
29         } catch (SQLException | IOException e) {
30             // Converte exceção checada em uma não checada
31             throw new DbException(e.getMessage());
32         }
33     }
34
35     public static void fechaConexao() {
36         if (conexao != null) {
37             try {
38                 conexao.close();
39             } catch (SQLException e) {
40                 throw new DbException(e.getMessage());
41             }
42         }
43     }
44
45     private static Properties loadProperties() throws FileNotFoundException, IOException {
46         try (FileInputStream fs = new FileInputStream("db.properties")) {
47             Properties prop = new Properties();
48             prop.load(fs);
49             return prop;
50         }
51     }
52 }
```

## 5.4 Refatoração setaProduto()

Selecione o trecho destacado abaixo do método `listarTodos()`, clique com o botão direito sobre ele e escolha a opção *Refactor* → *Extract Method* ...



Dê o nome de `setaProduto` e clique em *Ok*. Esta janela permite também definir o modificador de acesso (público, privado, protegido ou pacote). A janela mostra também a assinatura do método, informando, neste caso, que o valor de retorno será do tipo `Produto`. O parâmetro para o método `setaProduto()` também é mostrado, sendo, neste caso, do tipo `ResultSet`.



O método `setaProduto()` será criado e a chamada ajustada:

```
private Produto setaProduto(ResultSet rs) throws SQLException {
    Produto ct = new Produto();

    ct.setIdProd(rs.getInt("id_prod"));
    ct.setNomeProd(rs.getString("nome_prod"));
    ct.setDataCadastro(rs.getDate("data_cadastro"));
    ct.setQuantidade(rs.getInt("quantidade"));
    ct.setPreco(rs.getBigDecimal("preco"));

    return ct;
}
```

```
@Override
public List<Produto> listarTodos() {
    conexao = FabricaConexao.getConexao();
    List<Produto> lista = new ArrayList<>();

    try {
        String sql = "SELECT * FROM produto";
        PreparedStatement stmt = this.conexao.prepareStatement(sql);

        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            Produto ct = setaProduto(rs);

            lista.add(ct);
        }
    } catch (SQLException e) {
        throw new RuntimeException(e.getMessage());
    } finally {

```

## 5.5 Liberação de recursos `Statement` e `ResultSet`

A liberação dos recursos também é uma boa prática – a documentação do Java afirma que ela não é necessária, mas há relatos em fóruns que nem todos os *drivers* estariam liberando os recursos automaticamente.

### 5.5.1 Método `listarTodos()`

Neste método é usada uma versão do `try` chamada *try with resources* que libera os recursos ao finalizar o `try`.

Código:

```
public List<Produto> listarTodos() {
    conexao = FabricaConexao.getConexao();
    List<Produto> lista = new ArrayList<>();
    String sql = "SELECT * FROM produto";

    try (
        PreparedStatement stmt = this.conexao.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery();
    ) {
        while (rs.next()) {
            Produto ct = setaProduto(rs);

            lista.add(ct);
        }
    } catch (SQLException e) {
        throw new RuntimeException(e.getMessage());
    } finally {
        FabricaConexao.fechaConexao();
    }

    return lista;
}
```



### 5.5.2 Método buscarPorId()

O código refatorado será reutilizado em `buscarPorId()`, mas neste caso a liberação dos recursos terá que ser diferente, pois é possível colocar a linha 127 dentro do bloco de recursos do `try`.

```
118 @Override
119 public Produto buscarPorId(Integer id) {
120     conexao = FabricaConexao.getConexao();
121     String sql = "SELECT * FROM produto WHERE id_prod=?";
122     Produto p = null;
123
124     try (
125         PreparedStatement stmt = this.conexao.prepareStatement(sql);
126     ) {
127         stmt.setInt(1, id);
128         try (ResultSet rs = stmt.executeQuery()) {
129             if (rs.next()) {
130                 p = setaProduto(rs);
131             }
132         }
133     } catch (SQLException e) {
134         throw new RuntimeException(e.getMessage());
135     } finally {
136         FabricaConexao.fechaConexao();
137     }
138
139     return p;
140 }
```

### 5.6 Integração dos métodos `adicionar()` e `alterar()` em um só método `salvar()`

O código dos métodos `adicionar()` e `alterar()` é muito semelhante; com pouco esforço ele pode ser unificado em um só método, `salvar()`. O método `salvar()` decide se irá fazer uma inclusão ou alteração baseado no valor do campo `id`. Se o valor for `null` isso indica que será feita uma inclusão, caso contrário assume-se que o registro já existe (pois a chave não pode ser nula) e será feita uma alteração.

#### 5.6.1 Alteração dos Métodos na Classe DAO

Remova os métodos `adicionar()` e `alterar()` e crie o método `salvar()` abaixo:

```
16 @Override
17 public void salvar(Produto p) {
18     conexao = FabricaConexao.getConexao();
19     String sql;
20
21     if (p.getIdProd() == null) {
22         sql = "INSERT INTO produto (nome_prod, data_cadastro, quantidade, preco) "
23             + "VALUES (?, ?, ?, ?)";
24     } else {
25         sql = "UPDATE produto SET nome_prod=?, data_cadastro=?, quantidade=?, preco=? "
26             + "WHERE id_prod=?";
27     }
28
29     try {
30         PreparedStatement stmt = this.conexao.prepareStatement(sql);
31
32         stmt.setString(1, p.getNomeProd());
33         stmt.setDate(2, p.getDataCadastro());
34         stmt.setInt(3, p.getQuantidade());
35         stmt.setBigDecimal(4, p.getPreco());
36
37         if (p.getIdProd() != null) {
38             stmt.setInt(5, p.getIdProd());
39         }
40
41         stmt.execute();
42     } catch (SQLException e) {
43         throw new RuntimeException(e.getMessage());
44     } finally {
45         FabricaConexao.fechaConexao();
46     }
47 }
```

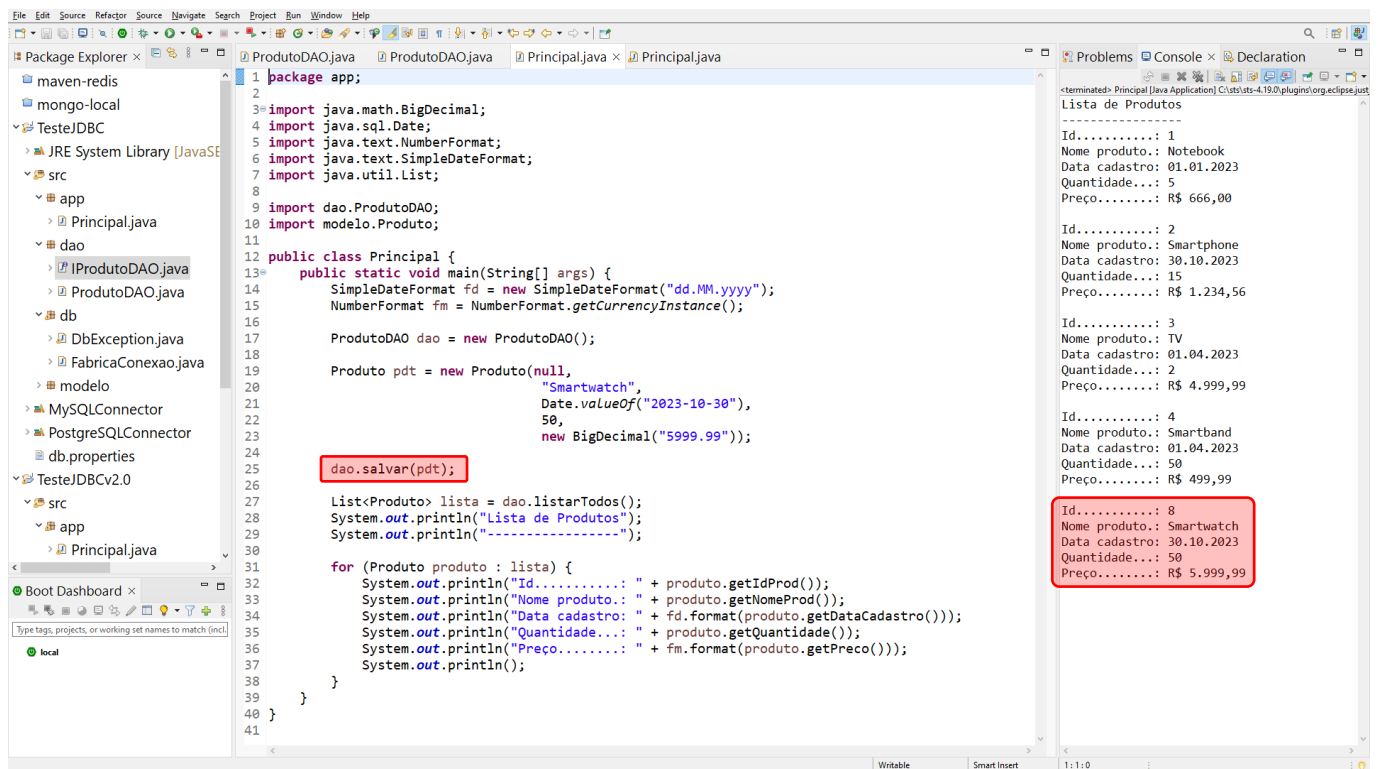
## 5.6.2 Modificação da Interface

Altere a definição do arquivo *IProdutoDAO.java* para:

```
1 package dao;
2
3 import java.util.List;
4
5 import modelo.Produto;
6
7 public interface IProdutoDAO {
8     void salvar(Produto p);
9     void excluir(Integer id);
10    List<Produto> listarTodos();
11    Produto buscarPorId(Integer id);
12 }
```

## 5.6.3 Método Principal

Código com um novo teste. Após executar a primeira vez com sucesso comente a linha 25 para evitar sucessivas inclusões na tabela.



## 5.6.4 Liberação de Recursos no Método salvar()

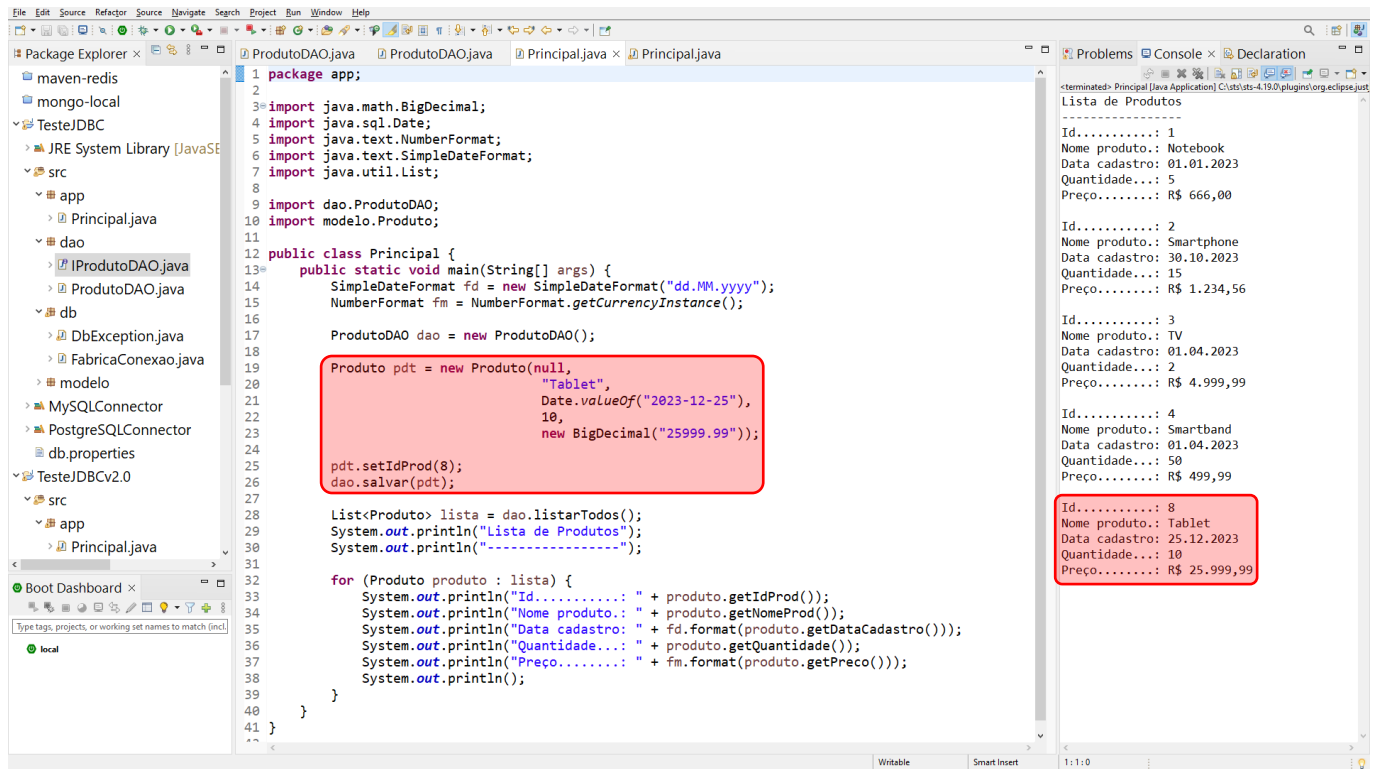
Volte a modificar o método `salvar()` a fim de liberar os recursos utilizados.

```
17 @Override
18 public void salvar(Produto p) {
19     conexao = FabricaConexao.getConexao();
20     String sql;
21
22     if (p.getIdProd() == null) {
23         sql = "INSERT INTO produto (nome_prod, data_cadastro, quantidade, preco) "
24             + "VALUES (?, ?, ?, ?)";
25     } else {
26         sql = "UPDATE produto SET nome_prod=?, data_cadastro=?, quantidade=?, preco=? "
27             + "WHERE id_prod=?";
28     }
29
30     try (PreparedStatement stmt = this.conexao.prepareStatement(sql);) {
31         stmt.setString(1, p.getNomeProd());
32         stmt.setDate(2, p.getDataCadastro());
33         stmt.setInt(3, p.getQuantidade());
34         stmt.setBigDecimal(4, p.getPreco());
35     }
36 }
```



## 5.6.5 Teste da Alteração no Método `salvar()`

Modifique novamente o método `main()` a fim de testar a alteração de registros. Use o `id` do último registro criado.



## 5.7 Método que Retorna o Número de Registros na Tabela

### 5.7.1 Modificação na Interface

Código:

```
1 package dao;
2
3 import java.util.List;
4
5 import modelo.Produto;
6
7 public interface IProdutoDAO {
8     void salvar(Produto p);
9     boolean excluir(Integer id);
10    List<Produto> listarTodos();
11    Produto buscarPorId(Integer id);
12    List<Produto> buscarPorNome(String nome);
13    int getNumeroRegistros();
14 }
```

### 5.7.2 Adição na Classe DAO

Adicione o método `getNumeroRegistros()` na classe DAO. Este método será testado posteriormente.

```
129 @Override
130 public int getNumeroRegistros() {
131     conexao = FabricaConexao.getConexao();
132     String sql = "SELECT count(*) FROM produto";
133     int numRegistros = 0;
134
135     try (
136         PreparedStatement stmt = this.conexao.prepareStatement(sql);
137         ResultSet rs = stmt.executeQuery();
138     ) {
139         if (rs.next()) {
140             numRegistros = rs.getInt(1);
141         }
142     } catch (SQLException e) {
143         throw new RuntimeException(e.getMessage());
144     } finally {
145         FabricaConexao.fechaConexao();
146     }
147
148     return numRegistros;
149 }
```

## 5.8 Adição do Método `buscarPorNome()` com Utilização de Filtro

### 5.8.1 Modificação na *Interface*

Código:

```
1 package dao;
2
3 import java.util.List;
4
5 import modelo.Produto;
6
7 public interface IProdutoDAO {
8     void salvar(Produto p);
9     void excluir(Integer id);
10    List<Produto> listarTodos();
11    Produto buscarPorId(Integer id);
12    List<Produto> buscarPorNome(String nome);
13    int getNumeroRegistros();
14 }
```

### 5.8.2 Adição na Classe DAO

Adicione o método `buscarPorNome()` na classe DAO.

```
129 @Override
130 public List<Produto> buscarPorNome(String nome) {
131     conexao = FabricaConexao.getConexao();
132     List<Produto> lista = new ArrayList<>();
133     String sql = "SELECT * FROM produto WHERE nome_prod LIKE ?";
134
135     try (
136         PreparedStatement stmt = this.conexao.prepareStatement(sql);
137     ) {
138         stmt.setString(1, "%" + nome + "%");
139         try (ResultSet rs = stmt.executeQuery()) {
140             while (rs.next()) {
141                 Produto p = setaProduto(rs);
142                 lista.add(p);
143             }
144         }
145     } catch (SQLException e) {
146         throw new RuntimeException(e.getMessage());
147     } finally {
148         FabricaConexao.fechaConexao();
149     }
150
151     return lista;
152 }
```

### 5.8.3 Teste no Programa Principal

Código e teste:

The screenshot shows an IDE with the following components:

- Package Explorer:** Shows the project structure with packages like `maven-redis`, `mongo-local`, `TesteJDBC`, `JRE System Library [JavaSE-17]`, `src`, `app`, `dao`, `db`, `modelo`, `MySQLConnector`, `PostgreSQLConnector`, `db.properties`, `TesteJDBCv2.0`, and `Principal.java`.
- Source Editor:** Displays the `Principal.java` file. The `main` method is highlighted with a red box. It shows the creation of a `Produto` object, saving it to the database, and then calling `dao.buscarPorNome(nome)` to search for products by name. The results are printed to the console.
- Console:** Shows the output of the program. It displays the message "Informe o nome do produto a pesquisar: smart", followed by a list of products found: "Lista de Produtos - # Total de registros: 5". The list includes details for two products: "Smartphone" and "Smartband".

```
1 package app;
2
3 import java.math.BigDecimal;
4 import java.sql.Date;
5 import java.text.NumberFormat;
6 import java.text.SimpleDateFormat;
7 import java.util.List;
8 import java.util.Scanner;
9 import dao.IProdutoDAO;
10 import modelo.Produto;
11
12 public class Principal {
13     public static void main(String[] args) {
14         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
15         NumberFormat fm = NumberFormat.getCurrencyInstance();
16
17         ProdutoDAO dao = new ProdutoDAO();
18
19         Produto pdt = new Produto(null,
20             "Tablet",
21             Date.valueOf("2023-12-25"),
22             10,
23             new BigDecimal("25999.99"));
24
25         pdt.setIdProd(8);
26         dao.salvar(pdt);
27
28         System.out.println("Informe o nome do produto a pesquisar: ");
29         Scanner sc = new Scanner(System.in);
30         String nome = sc.nextLine();
31         List<Produto> lista = dao.buscarPorNome(nome);
32
33         if (lista.isEmpty()) {
34             System.out.println("A tabela está vazia!");
35         } else {
36             System.out.println("Lista de Produtos - ");
37             System.out.println("# Total de registros: " + dao.getNumeroRegistros());
38             String msg = "# registros coincidindo com o filtro de busca: " + lista.size();
39             System.out.println(msg.repeat(msg.length()));
40
41             for (Produto produto : lista) {
42                 System.out.println("Id.....: " + produto.getIdProd());
43                 System.out.println("Nome produto.: " + produto.getNomeProd());
44                 System.out.println("Data cadastro.: " + fd.format(produto.getDataCadastro()));
45                 System.out.println("Quantidade....: " + produto.getQuantidade());
46                 System.out.println("Preço.....: " + fm.format(produto.getPreco()));
47             }
48
49             System.out.println(msg.repeat(msg.length()));
50             System.out.println(msg);
51         }
52     }
53 }
```

Informe o nome do produto a pesquisar: smart

Lista de Produtos - # Total de registros: 5

Id.....	Nome produto..	Data cadastro..	Quantidade....	Preço.....
2	Smartphone	30.10.2023	15	R\$ 1.234,56
4	Smartband	01.04.2023	50	R\$ 499,99

# registros coincidindo com o filtro de busca: 2

## 5.9 Indicação de Quantos Registros foram Afetados no Método `salvar()`

Esta seção e a próxima modificam o método `salvar()` de forma que ele retorne algumas informações. Diferente de linguagens como o Python, o Java não permite, diretamente, retornar mais de um valor em uma função/método. Algumas formas de contornar este problema são:

- Retornar um *array*
- Retornar um par/tupla usando *generics* (recurso de programação genérica)
- Retornar um `AbstractMap.SimpleEntry` ou uma lista desses objetos
- Retornar um registro (*record*), disponível a partir do Java 14
- Retornar uma coleção, como uma lista ou mapa (como o `HashMap`)
- Criar uma classe contêiner (POJO) que envolva os valores desejados

A solução usada será a de retornar uma coleção. Com isso será fácil implementar também o retorno de chave.

### 5.9.1 Modificação na Interface

Código:

```
1 package dao;
2
3 import java.util.List;
4 import java.util.Map;
5
6 import modelo.Produto;
7
8 public interface IProdutoDAO {
9     Map<String, Integer> salvar(Produto p);
10    void excluir(Integer id);
11    List<Produto> listarTodos();
12    Produto buscarPorId(Integer id);
13    List<Produto> buscarPorNome(String nome);
14    int getNumeroRegistros();
15 }
```

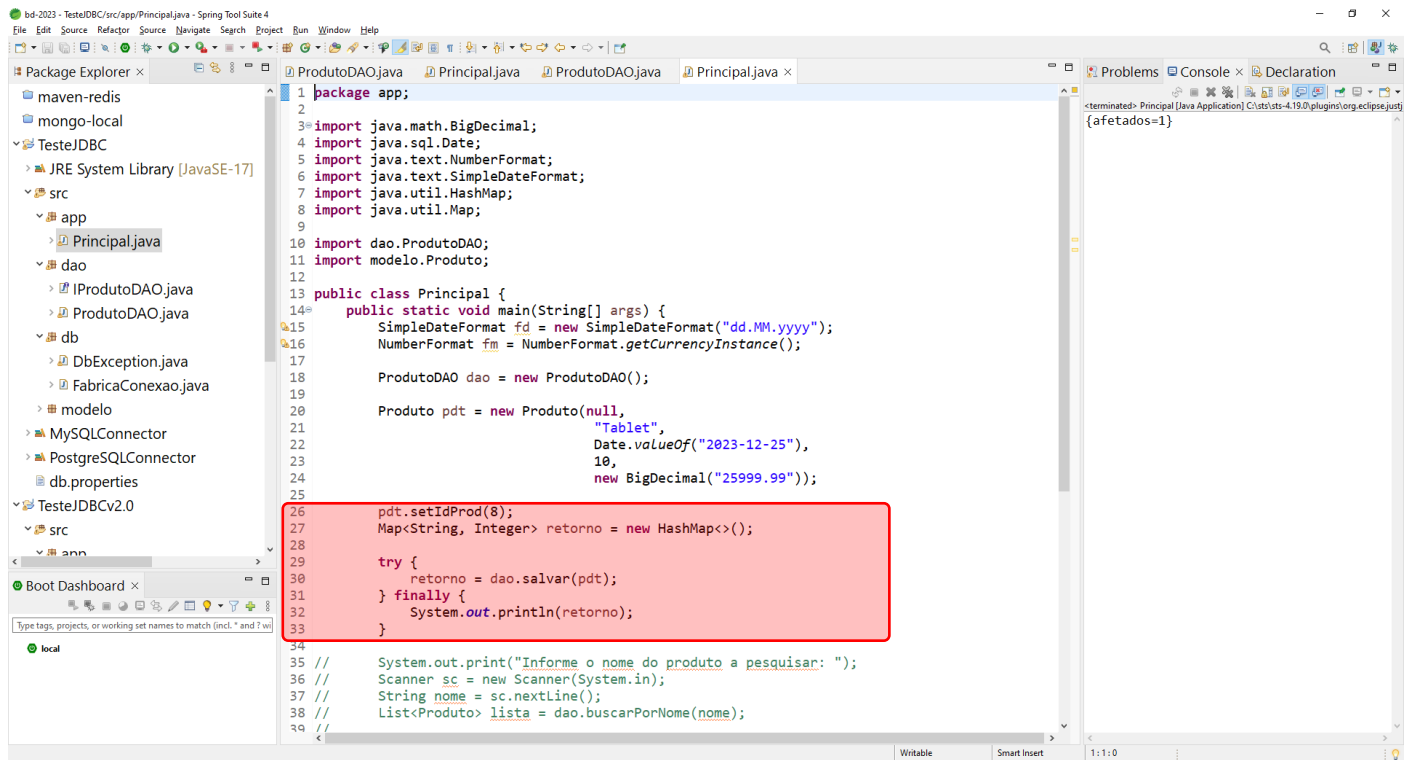
### 5.9.2 Modificação no Método `salvar()`

Código:

```
19 @Override
20 public Map<String, Integer> salvar(Produto p) {
21     conexao = FabricaConexao.getConexao();
22
23     String sql;
24     Map<String, Integer> retorno = new HashMap<>() {{
25         put("afetados", 0);
26     }};
27
28     if (p.getIdProd() == null) {
29         sql = "INSERT INTO produto (nome_prod, data_cadastro, quantidade, preco) "
30             + "VALUES (?, ?, ?, ?)";
31     } else {
32         sql = "UPDATE produto SET nome_prod=?, data_cadastro=?, quantidade=?, preco=? "
33             + "WHERE id_prod=?";
34     }
35
36     try (PreparedStatement stmt = this.conexao.prepareStatement(sql);) {
37         stmt.setString(1, p.getNomeProd());
38         stmt.setDate(2, p.getDataCadastro());
39         stmt.setInt(3, p.getQuantidade());
40         stmt.setBigDecimal(4, p.getPreco());
41
42         if (p.getIdProd() != null) {
43             stmt.setInt(5, p.getIdProd());
44         }
45
46         // Quantos registros foram afetados
47         int registrosAfetados = stmt.executeUpdate();
48         retorno.put("afetados", registrosAfetados);
49         return retorno;
50     } catch (SQLException e) {
51         throw new RuntimeException(e.getMessage());
52     } finally {
53         FabricaConexao.fechaConexao();
54     }
55 }
```

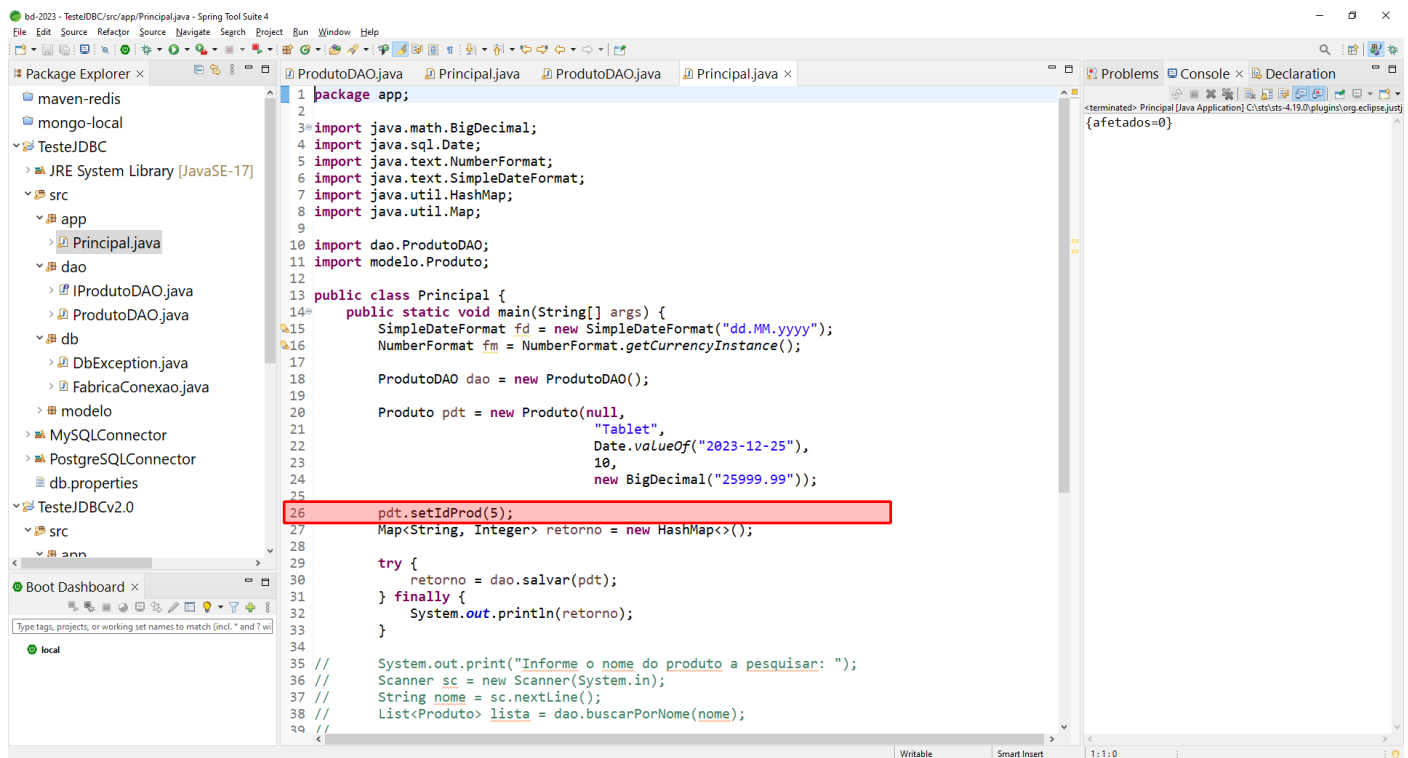
### 5.9.3 Teste no Método `main()`

Código com produto existente:



```
1 package app;
2
3 import java.math.BigDecimal;
4 import java.sql.Date;
5 import java.text.NumberFormat;
6 import java.text.SimpleDateFormat;
7 import java.util.HashMap;
8 import java.util.Map;
9
10 import dao.ProdutoDAO;
11 import modelo.Produto;
12
13 public class Principal {
14     public static void main(String[] args) {
15         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
16         NumberFormat fm = NumberFormat.getCurrencyInstance();
17
18         ProdutoDAO dao = new ProdutoDAO();
19
20         Produto pdt = new Produto(null,
21                                   "Tablet",
22                                   Date.valueOf("2023-12-25"),
23                                   10,
24                                   new BigDecimal("25999.99"));
25
26         pdt.setIdProd(8);
27         Map<String, Integer> retorno = new HashMap<>();
28
29         try {
30             retorno = dao.salvar(pdt);
31         } finally {
32             System.out.println(retorno);
33         }
34
35         // System.out.print("Informe o nome do produto a pesquisar: ");
36         // Scanner sc = new Scanner(System.in);
37         // String nome = sc.nextLine();
38         // List<Produto> lista = dao.buscarPorNome(nome);
39         //
```

Código com produto inexistente:



```
1 package app;
2
3 import java.math.BigDecimal;
4 import java.sql.Date;
5 import java.text.NumberFormat;
6 import java.text.SimpleDateFormat;
7 import java.util.HashMap;
8 import java.util.Map;
9
10 import dao.ProdutoDAO;
11 import modelo.Produto;
12
13 public class Principal {
14     public static void main(String[] args) {
15         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
16         NumberFormat fm = NumberFormat.getCurrencyInstance();
17
18         ProdutoDAO dao = new ProdutoDAO();
19
20         Produto pdt = new Produto(null,
21                                   "Tablet",
22                                   Date.valueOf("2023-12-25"),
23                                   10,
24                                   new BigDecimal("25999.99"));
25
26         pdt.setIdProd(5);
27         Map<String, Integer> retorno = new HashMap<>();
28
29         try {
30             retorno = dao.salvar(pdt);
31         } finally {
32             System.out.println(retorno);
33         }
34
35         // System.out.print("Informe o nome do produto a pesquisar: ");
36         // Scanner sc = new Scanner(System.in);
37         // String nome = sc.nextLine();
38         // List<Produto> lista = dao.buscarPorNome(nome);
39         //
```

Caso aconteça alguma exceção no método `salvar()`, o resultado será o mesmo mostrado na tela acima.

## 5.10 Retorno da Chave do Registro no Caso de Inserção na Tabela

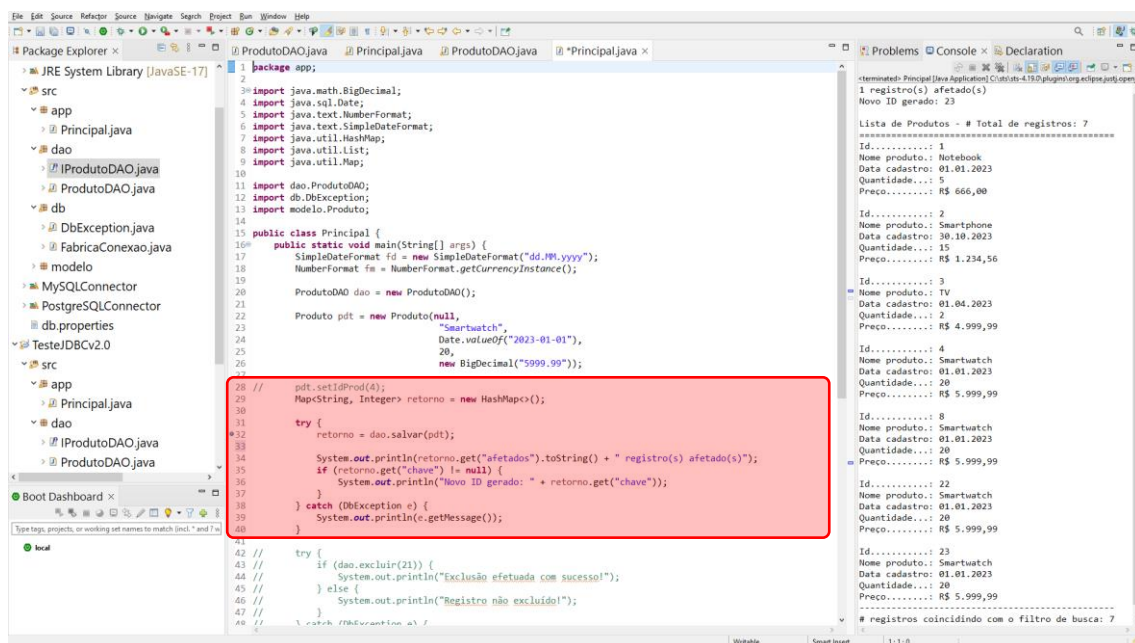
### 5.10.1 Modificação no Método salvar()

Código:

```
20 public Map<String, Integer> salvar(Produto p) {
21     conexao = FabricaConexao.getConexao();
22
23     String sql;
24     Map<String, Integer> retorno = new HashMap<>() {{
25         put("afetados", 0);
26         put("chave", null);
27     }};
28
29     if (p.getIdProd() == null) {
30         sql = "INSERT INTO produto (nome_prod, data_cadastro, quantidade, preco) "
31             + "VALUES (?, ?, ?, ?);";
32     } else {
33         sql = "UPDATE produto SET nome_prod=?, data_cadastro=?, quantidade=?, preco=? "
34             + "WHERE id_prod=?";
35     }
36
37     try {
38         PreparedStatement stmt = this.conexao.prepareStatement(sql,
39             Statement.RETURN_GENERATED_KEYS);
40     } {
41         stmt.setString(1, p.getNomeProd());
42         stmt.setDate(2, p.getDataCadastro());
43         stmt.setInt(3, p.getQuantidade());
44         stmt.setBigDecimal(4, p.getPreco());
45
46         if (p.getIdProd() != null) {
47             stmt.setInt(5, p.getIdProd());
48         }
49
50         // Quantos registros foram afetados
51         int registrosAfetados = stmt.executeUpdate();
52         retorno.put("afetados", registrosAfetados);
53
54         // No caso de inserção, retorna ID gerado
55         try (ResultSet rs = stmt.getGeneratedKeys()) {
56             if (rs.next()) {
57                 retorno.put("chave", rs.getInt(1));
58             }
59         }
60         return retorno;
61     } catch (SQLException e) {
62         throw new RuntimeException(e.getMessage());
63     } finally {
64         FabricaConexao.fechaConexao();
65     }
66 }
```

### 5.10.2 Teste no Método main()

Modificação no programa principal para teste de inserção:





## 5.11 Retorno *Booleano* do Método `excluir()` e Posterior Verificação de Exclusão

### 5.11.1 Modificação na *Interface*

Código:

```
1 package dao;
2
3 import java.util.List;
4 import java.util.Map;
5
6 import modelo.Produto;
7
8 public interface IProdutoDAO {
9     Map<String, Integer> salvar(Produto p);
10    boolean excluir(Integer id);
11    List<Produto> listarTodos();
12    Produto buscarPorId(Integer id);
13    List<Produto> buscarPorNome(String nome);
14    int getNumeroRegistros();
15 }
```

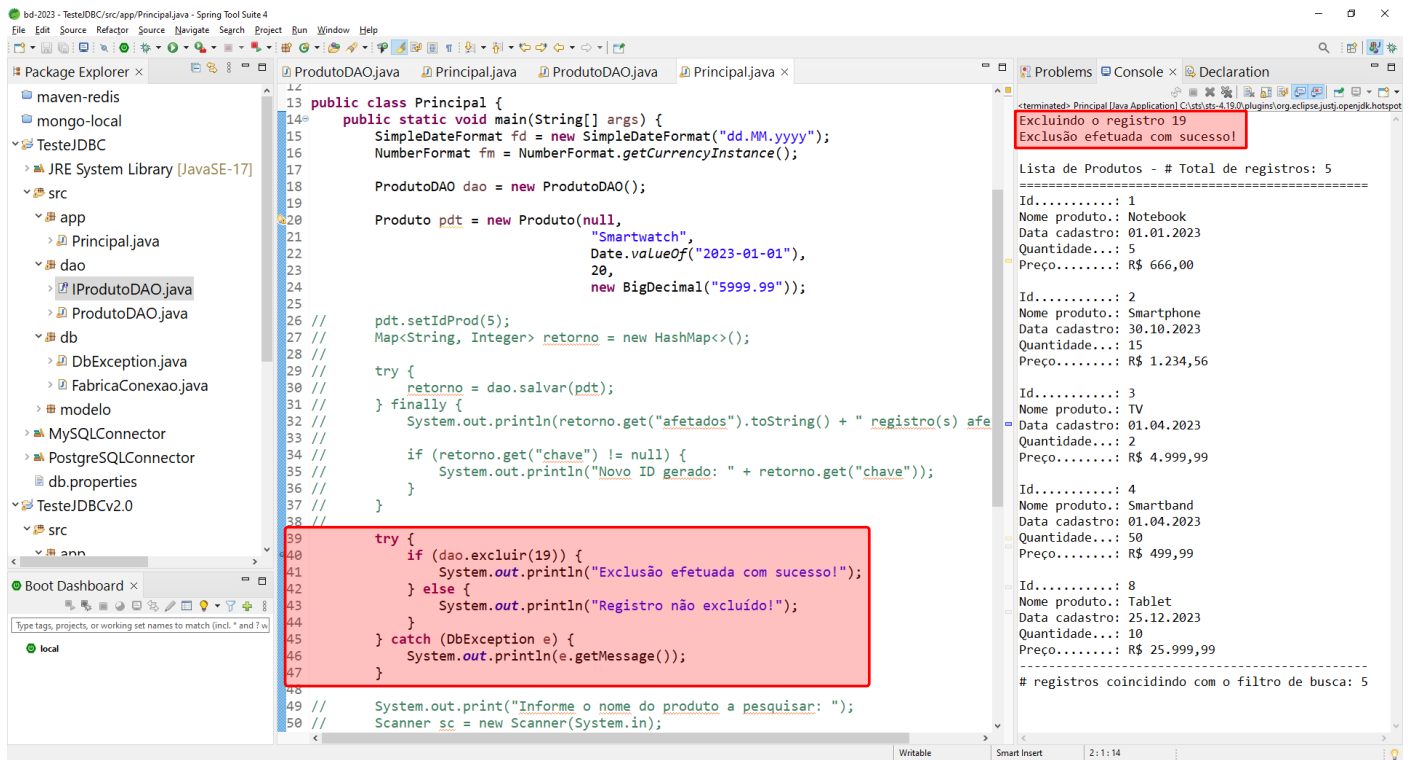
### 5.11.2 Modificação no Método `excluir()`

Código:

```
1 package dao;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 import java.sql.SQLException;
7 import java.sql.Statement;
8 import java.util.ArrayList;
9 import java.util.HashMap;
10 import java.util.List;
11 import java.util.Map;
12
13 import db.DbException;
14 import db.FabricaConexao;
15 import modelo.Produto;
16
17 public class ProdutoDAO implements IProdutoDAO {
18     private Connection conexao;
19
20     public Map<String, Integer> salvar(Produto p) {}
21
22     @Override
23     public boolean excluir(Integer id) {
24         System.out.println("Excluindo o registro " + id);
25         conexao = FabricaConexao.getConexao();
26         String sql = "DELETE FROM produto WHERE id_prod=?";
27
28         try (
29             PreparedStatement stmt = this.conexao.prepareStatement(sql);
30         ) {
31             stmt.setInt(1, id);
32             int registrosAfetados = stmt.executeUpdate();
33
34             if (1 == 1) {
35                 throw new SQLException();
36             }
37
38             return (registrosAfetados == 1);
39         } catch (SQLException e) {
40             throw new DbException(e.getMessage());
41         } finally {
42             FabricaConexao.fechaConexao();
43         }
44     }
45 }
```

### 5.11.3 Teste no Método `main()`

Modificação no programa principal para teste de exclusão. Na linha 40 utilize o `id` de um registro existente.



```
13 public class Principal {
14     public static void main(String[] args) {
15         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
16         NumberFormat fm = NumberFormat.getCurrencyInstance();
17
18         ProdutoDAO dao = new ProdutoDAO();
19
20         Produto pdt = new Produto(null,
21             "Smartwatch",
22             Date.valueOf("2023-01-01"),
23             20,
24             new BigDecimal("5999.99"));
25
26         pdt.setIdProd(5);
27         Map<String, Integer> retorno = new HashMap<>();
28
29         try {
30             retorno = dao.salvar(pdt);
31         } finally {
32             System.out.println(retorno.get("afetados") + " registro(s) afetados");
33
34             if (retorno.get("chave") != null) {
35                 System.out.println("Novo ID gerado: " + retorno.get("chave"));
36             }
37         }
38
39         try {
40             if (dao.excluir(19)) {
41                 System.out.println("Exclusão efetuada com sucesso!");
42             } else {
43                 System.out.println("Registro não excluído!");
44             }
45         } catch (DbException e) {
46             System.out.println(e.getMessage());
47         }
48
49         System.out.print("Informe o nome do produto a pesquisar: ");
50         Scanner sc = new Scanner(System.in);
```

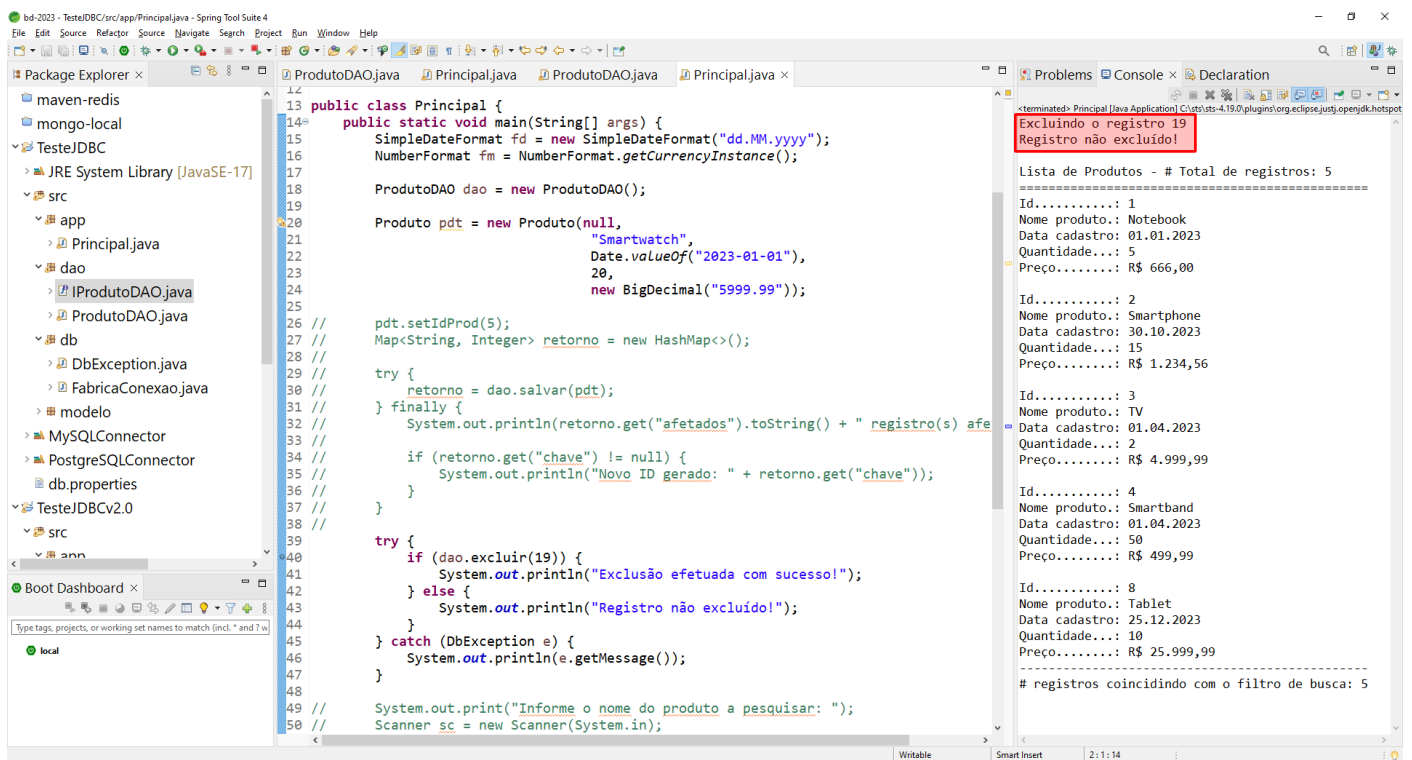
Excluindo o registro 19  
Exclusão efetuada com sucesso!

Lista de Produtos - # Total de registros: 5

Id	Nome produto	Data cadastro	Quantidade	Preço
1	Notebook	01.01.2023	5	R\$ 666,00
2	Smartphone	30.10.2023	15	R\$ 1.234,56
3	TV	01.04.2023	2	R\$ 4.999,99
4	Smartband	01.04.2023	50	R\$ 499,99
8	Tablet	25.12.2023	10	R\$ 25.999,99

# registros coincidindo com o filtro de busca: 5

Ao tentar excluir o mesmo registro novamente, a mensagem abaixo deverá aparecer:



```
13 public class Principal {
14     public static void main(String[] args) {
15         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
16         NumberFormat fm = NumberFormat.getCurrencyInstance();
17
18         ProdutoDAO dao = new ProdutoDAO();
19
20         Produto pdt = new Produto(null,
21             "Smartwatch",
22             Date.valueOf("2023-01-01"),
23             20,
24             new BigDecimal("5999.99"));
25
26         pdt.setIdProd(5);
27         Map<String, Integer> retorno = new HashMap<>();
28
29         try {
30             retorno = dao.salvar(pdt);
31         } finally {
32             System.out.println(retorno.get("afetados") + " registro(s) afetados");
33
34             if (retorno.get("chave") != null) {
35                 System.out.println("Novo ID gerado: " + retorno.get("chave"));
36             }
37         }
38
39         try {
40             if (dao.excluir(19)) {
41                 System.out.println("Exclusão efetuada com sucesso!");
42             } else {
43                 System.out.println("Registro não excluído!");
44             }
45         } catch (DbException e) {
46             System.out.println(e.getMessage());
47         }
48
49         System.out.print("Informe o nome do produto a pesquisar: ");
50         Scanner sc = new Scanner(System.in);
```

Excluindo o registro 19  
Registro não excluído!

Lista de Produtos - # Total de registros: 5

Id	Nome produto	Data cadastro	Quantidade	Preço
1	Notebook	01.01.2023	5	R\$ 666,00
2	Smartphone	30.10.2023	15	R\$ 1.234,56
3	TV	01.04.2023	2	R\$ 4.999,99
4	Smartband	01.04.2023	50	R\$ 499,99
8	Tablet	25.12.2023	10	R\$ 25.999,99

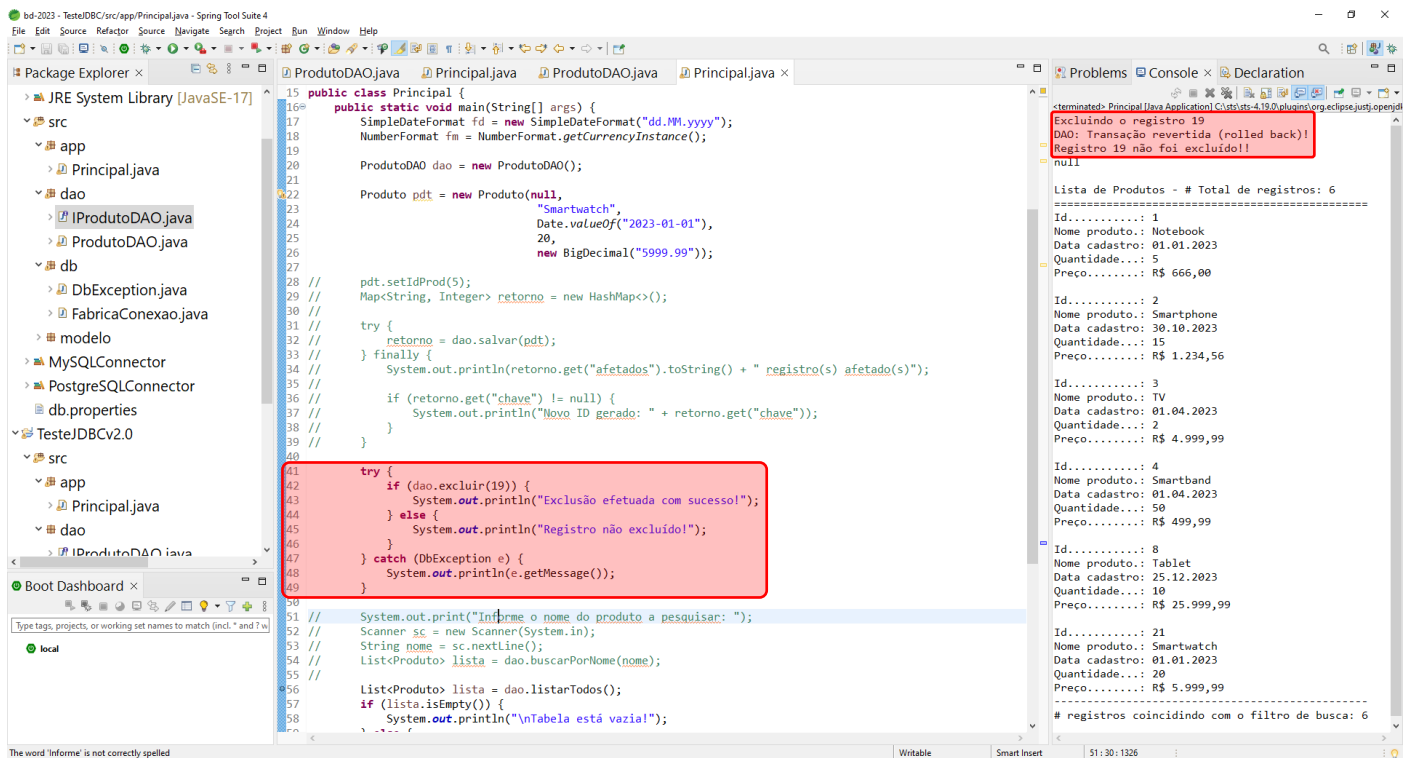
# registros coincidindo com o filtro de busca: 5

## 5.12 Controle Transacional com `setAutoCommit(false)`, `commit()` e `rollback()`

Código da função `excluir()`. Repare o código das linhas 88 a 90, que foram descomentados de forma a produzir deliberadamente uma exceção de forma a não executar o `commit` dos dados e, logo após, realizar o `rollback`.

```
74 @Override
75 public boolean excluir(Integer id) {
76     System.out.println("Excluindo o registro " + id);
77     conexao = FabricaConexao.getConexao();
78     String sql = "DELETE FROM produto WHERE id_prod=?";
79
80     try (
81         PreparedStatement stmt = this.conexao.prepareStatement(sql);
82     ) {
83         stmt.setInt(1, id);
84
85         conexao.setAutoCommit(false);
86         int registrosAfetados = stmt.executeUpdate();
87
88         if (1 == 1) {
89             throw new SQLException();
90         }
91
92         conexao.commit();
93
94         return (registrosAfetados == 1);
95     } catch (SQLException e) {
96         try {
97             conexao.rollback();
98             throw new DbException("Transação revertida (rolled back)!\n"
99                 + "Registro " + id + " não foi excluído!!\n"
100                 + e.getMessage());
101         } catch (SQLException e1) {
102             throw new DbException("ihhh, deu ruim! " + e1.getMessage());
103         }
104     } finally {
105         FabricaConexao.fechaConexao();
106     }
107 }
```

Código de teste na função `main()`:



```
16 public class Principal {
17     public static void main(String[] args) {
18         SimpleDateFormat fd = new SimpleDateFormat("dd.MM.yyyy");
19         NumberFormat fm = NumberFormat.getCurrencyInstance();
20
21         ProdutoDAO dao = new ProdutoDAO();
22
23         Produto pdt = new Produto(null,
24             "Smartwatch",
25             Date.valueOf("2023-01-01"),
26             20,
27             new BigDecimal("5999.99"));
28
29         pdt.setIdProd(5);
30         Map<String, Integer> retorno = new HashMap<>();
31
32         try {
33             retorno = dao.salvar(pdt);
34         } finally {
35             System.out.println(retorno.get("afetados").toString() + " registro(s) afetado(s)");
36
37             if (retorno.get("chave") != null) {
38                 System.out.println("Novo ID gerado: " + retorno.get("chave"));
39             }
40         }
41
42         try {
43             if (dao.excluir(19)) {
44                 System.out.println("Exclusão efetuada com sucesso!");
45             } else {
46                 System.out.println("Registro não excluído!");
47             }
48         } catch (DbException e) {
49             System.out.println(e.getMessage());
50         }
51
52         System.out.print("Informe o nome do produto a pesquisar: ");
53         Scanner sc = new Scanner(System.in);
54         String nome = sc.nextLine();
55         List<Produto> lista = dao.buscarPorNome(nome);
56
57         List<Produto> lista2 = dao.listarTodos();
58         if (lista.isEmpty()) {
59             System.out.println("A tabela está vazia!");
60         }
61     }
62 }
```

Console Output:

```
Excluindo o registro 19
DAO: Transação revertida (rolled back)!
Registro 19 não foi excluído!!
null
Lista de Produtos - # Total de registros: 6
=====
Id.....: 1
Nome produto.: Notebook
Data cadastro: 01.01.2023
Quantidade....: 5
Preço.....: R$ 666,00
Id.....: 2
Nome produto.: Smartphone
Data cadastro: 30.10.2023
Quantidade....: 15
Preço.....: R$ 1.234,56
Id.....: 3
Nome produto.: TV
Data cadastro: 01.04.2023
Quantidade....: 2
Preço.....: R$ 4.999,99
Id.....: 4
Nome produto.: Smartband
Data cadastro: 01.04.2023
Quantidade....: 50
Preço.....: R$ 499,99
Id.....: 10
Nome produto.: Tablet
Data cadastro: 25.12.2023
Quantidade....: 10
Preço.....: R$ 25.999,99
Id.....: 21
Nome produto.: Smartwatch
Data cadastro: 01.01.2023
Quantidade....: 20
Preço.....: R$ 5.999,99
=====
# registros coincidindo com o filtro de busca: 6
Exclusão efetuada com sucesso!
```

Comente novamente as linhas 88 – 90 do método `excluir()` e execute de novo o programa e a exclusão funcionará:

```
Excluindo o registro 21
Exclusão efetuada com sucesso!
```



Código da função salvar():

```
21 public Map<String, Integer> salvar(Produto p) {
22     conexao = FabricaConexao.getConexao();
23
24     Map<String, Integer> retorno = new HashMap<>() {{
25         put("afetados", 0);
26         put("chave", null);
27     }};
28
29     String sql;
30     if (p.getIdProd() == null) {
31         sql = "INSERT INTO produto (nome_prod, data_cadastro, quantidade, preco) "
32             + "VALUES (?, ?, ?, ?);";
33     } else {
34         sql = "UPDATE produto SET nome_prod=?, data_cadastro=?, quantidade=?, preco=? "
35             + "WHERE id_prod=?";
36     }
37
38     try (
39         PreparedStatement stmt = this.conexao.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
40     ) {
41         stmt.setString(1, p.getNomeProd());
42         stmt.setDate(2, p.getDataCadastro());
43         stmt.setInt(3, p.getQuantidade());
44         stmt.setBigDecimal(4, p.getPreco());
45
46         if (p.getIdProd() != null) {
47             stmt.setInt(5, p.getIdProd());
48         }
49
50         conexao.setAutoCommit(false); // Desativa o commit automático
51         int registrosAfetados = stmt.executeUpdate(); // Executa o SQL
52         retorno.put("afetados", registrosAfetados); // Quantos registros foram afetados
53
54         // No caso de inserção, retorna ID gerado
55         try (ResultSet rs = stmt.getGeneratedKeys()) {
56             if (rs.next()) {
57                 retorno.put("chave", rs.getInt(1));
58             }
59         }
60
61         conexao.commit(); // Comita os dados no banco
62         return retorno;
63     } catch (SQLException e) {
64         try {
65             conexao.rollback();
66             throw new DbException("Transação revertida (rolled back)! " + e.getMessage());
67         } catch (SQLException e1) {
68             throw new DbException("ihhh, deu ruim! " + e1.getMessage());
69         }
70     } finally {
71         FabricaConexao.fechaConexao();
72     }
73 }
```