



CRUD COM JDBD E DAO

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI

herculano.debiasi@unoesc.edu.br



TÓPICOS

- CRUD
- Preparação do projeto
- Criação do projeto
- Criação da conexão
- Classe de domínio
- DAO
- Criando uma exceção personalizada
- Externalizando os dados da conexão

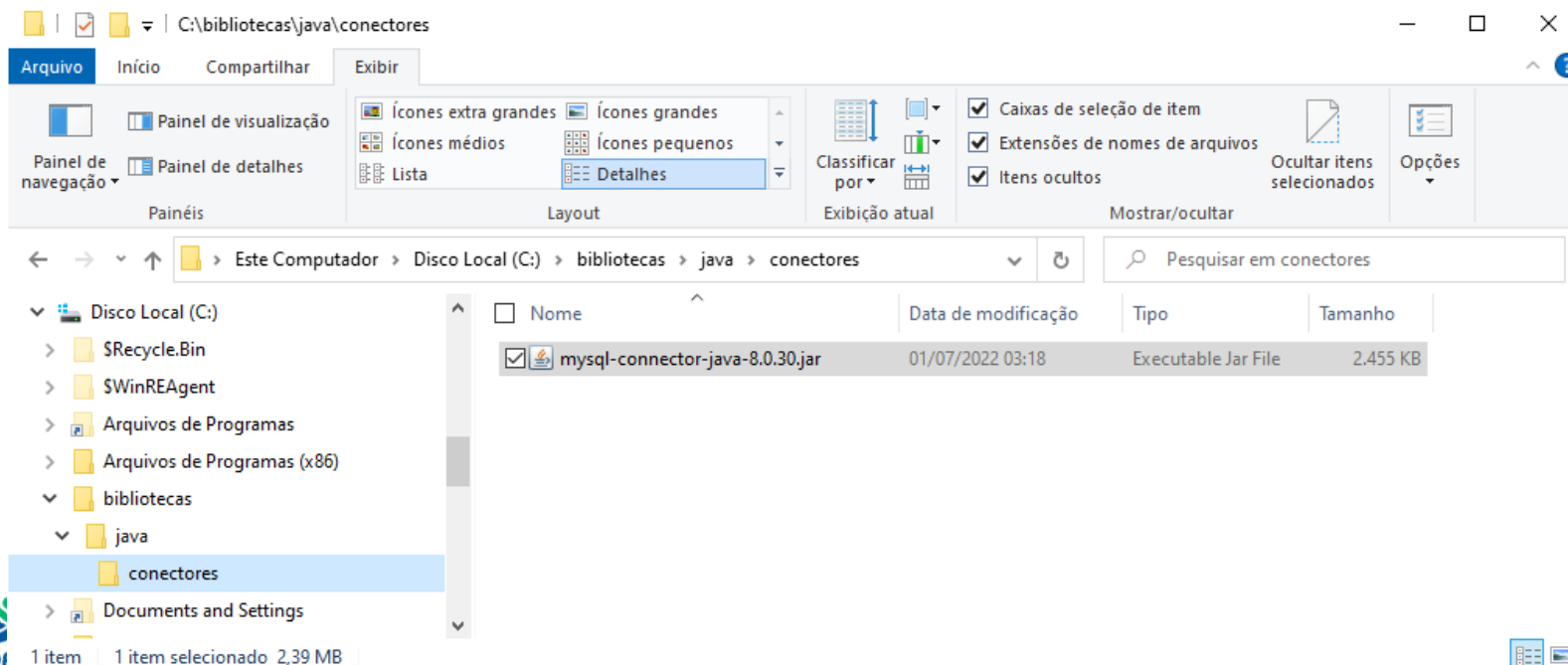
CRUD

- Aplicação CRUD (**C**reate, **R**ead, **U**ppdate e **D**eleite - Criar, Ler, Atualizar e Excluir)



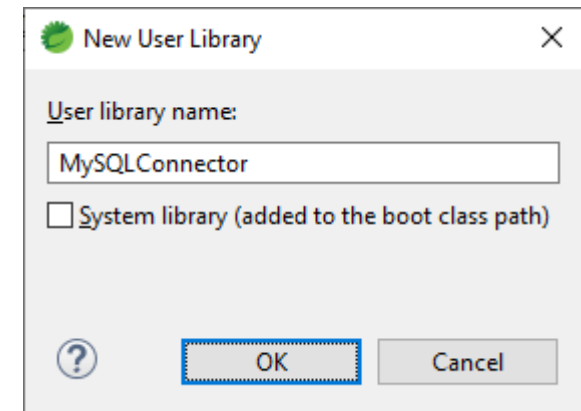
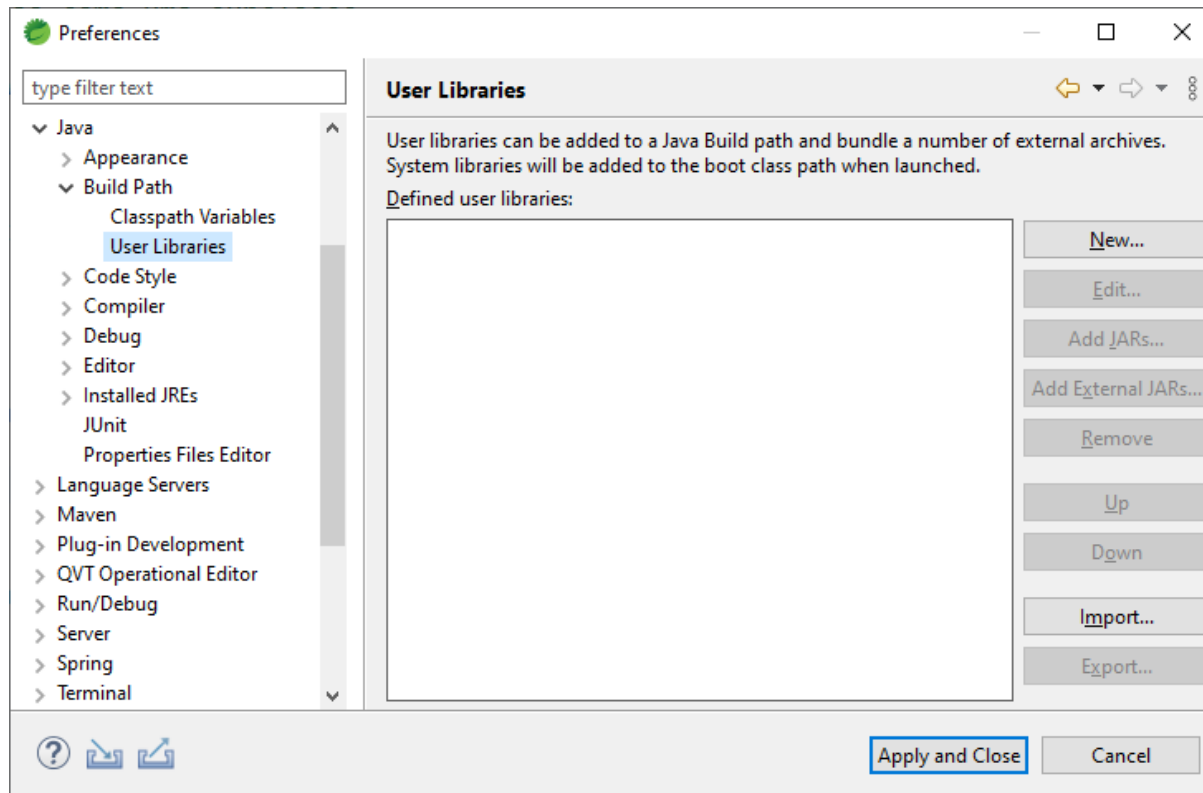
PREPARAÇÃO DO PROJETO

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 chamada *java*; dentro da pasta *java* crie uma pasta chamada *conectores*
3. Abri o arquivo .zip baixado e arraste o arquivo *mysql-connector-java-8.0.30.jar* para dentro da pasta *conectores*



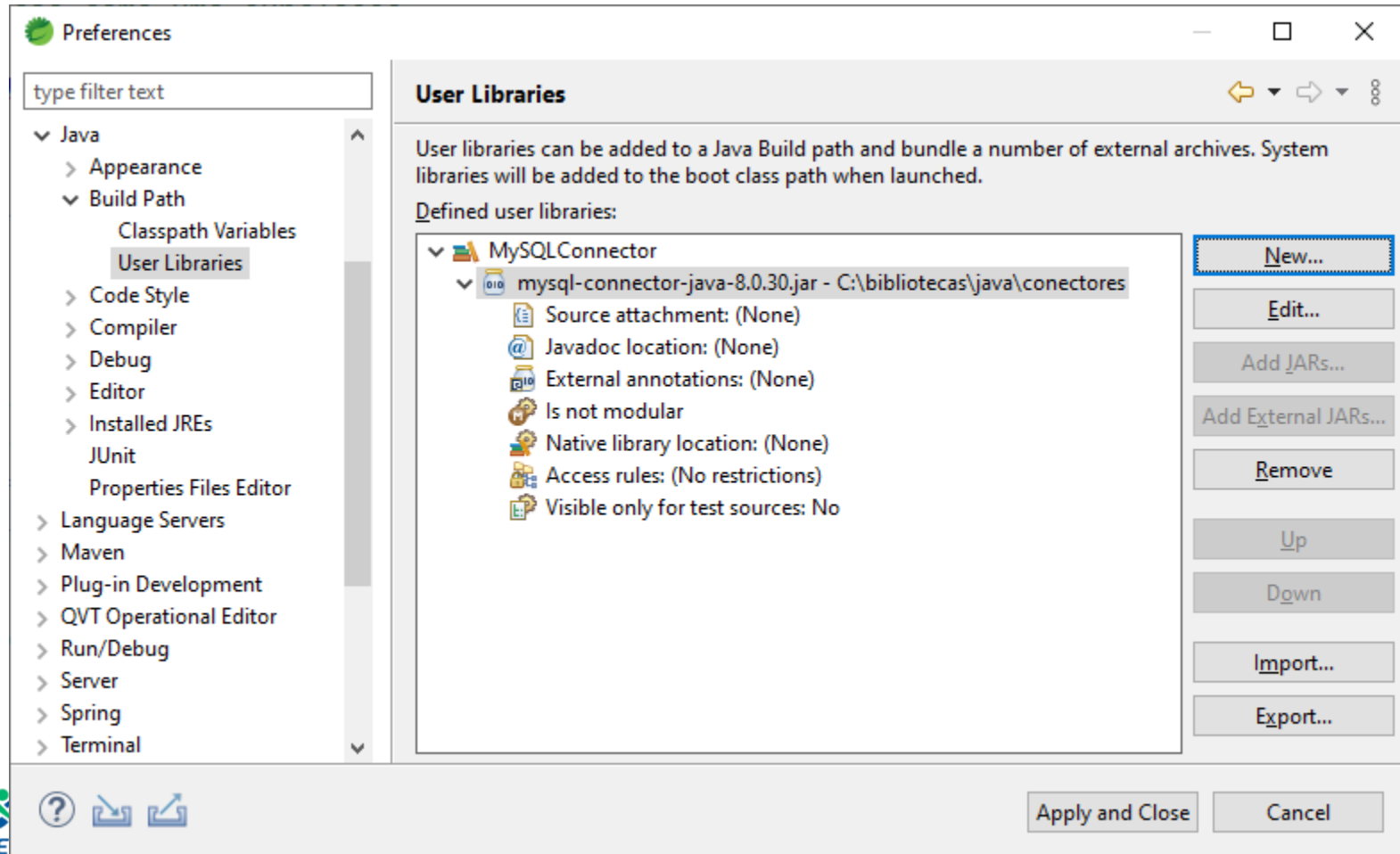
PREPARAÇÃO DO PROJETO

4. No STS vá em *Window* → *Preferences* e clique no botão *New ...*



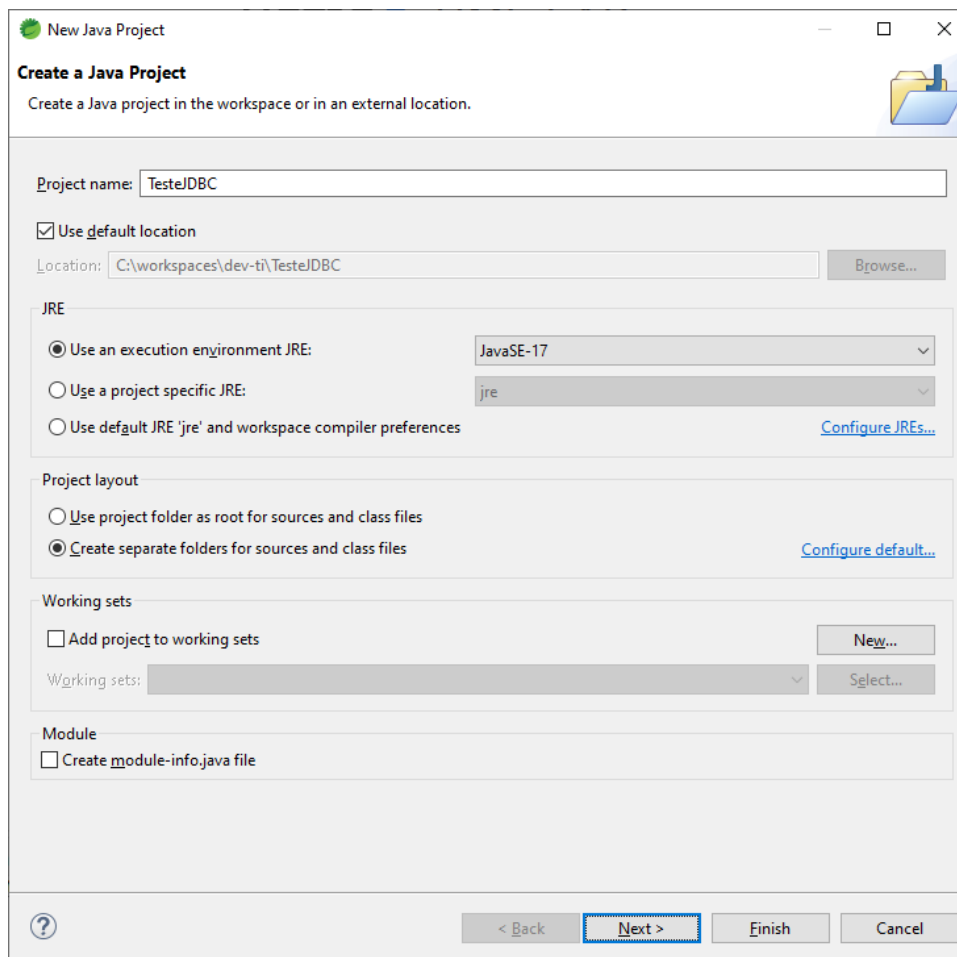
PREPARAÇÃO DO PROJETO

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



CRIAÇÃO DO PROJETO

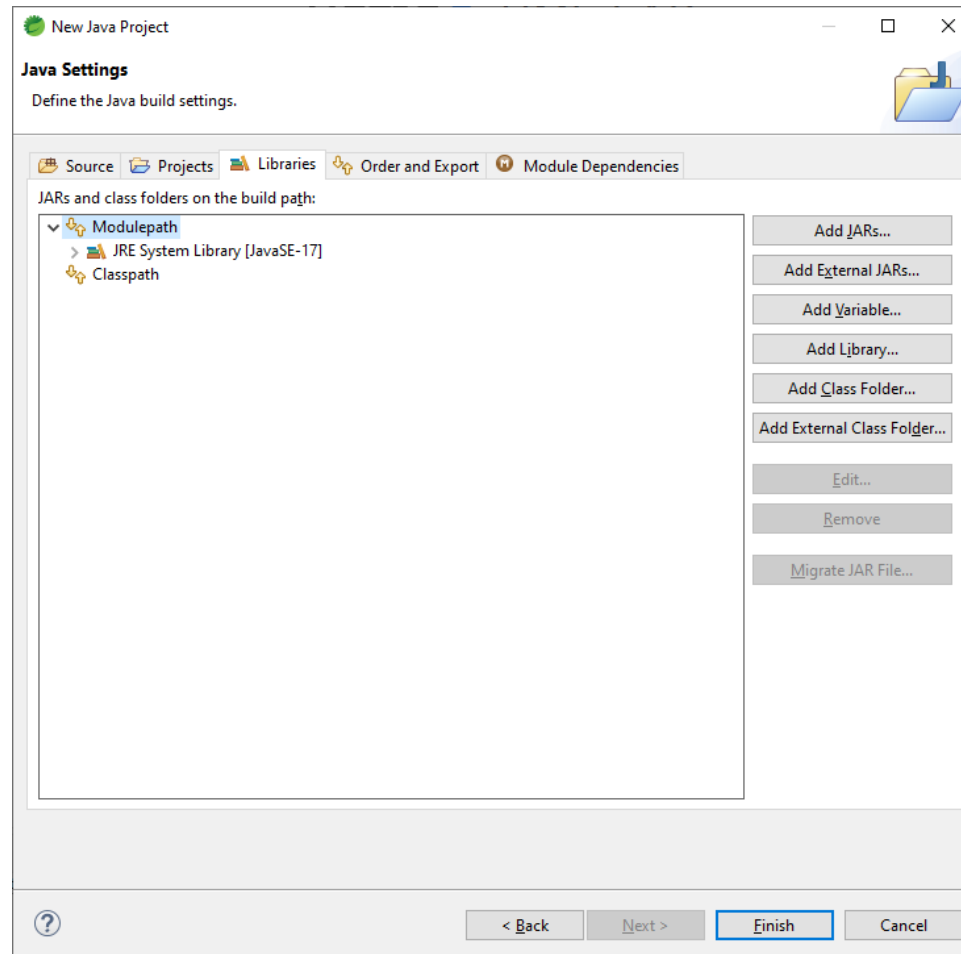
1. Crie um novo projeto Java (*File* → *New* → *Java Project*) – dê o nome de *TesteJDBC* e clique em *Next*



The screenshot shows the 'New Java Project' dialog box in the Eclipse IDE. The dialog is titled 'New Java Project' and has a subtitle 'Create a Java Project'. Below the subtitle, it says 'Create a Java project in the workspace or in an external location.' The 'Project name' field is filled with 'TesteJDBC'. The 'Use default location' checkbox is checked, and the 'Location' field shows 'C:\workspaces\dev-ti\TesteJDBC'. Under the 'JRE' section, the 'Use an execution environment JRE' radio button is selected, and the dropdown menu shows 'JavaSE-17'. There are also options for 'Use a project specific JRE' and 'Use default JRE 'jre' and workspace compiler preferences'. The 'Project layout' section has two options: 'Use project folder as root for sources and class files' and 'Create separate folders for sources and class files', with the latter being selected. The 'Working sets' section has an 'Add project to working sets' checkbox and a 'New...' button. The 'Module' section has a 'Create module-info.java file' checkbox. At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

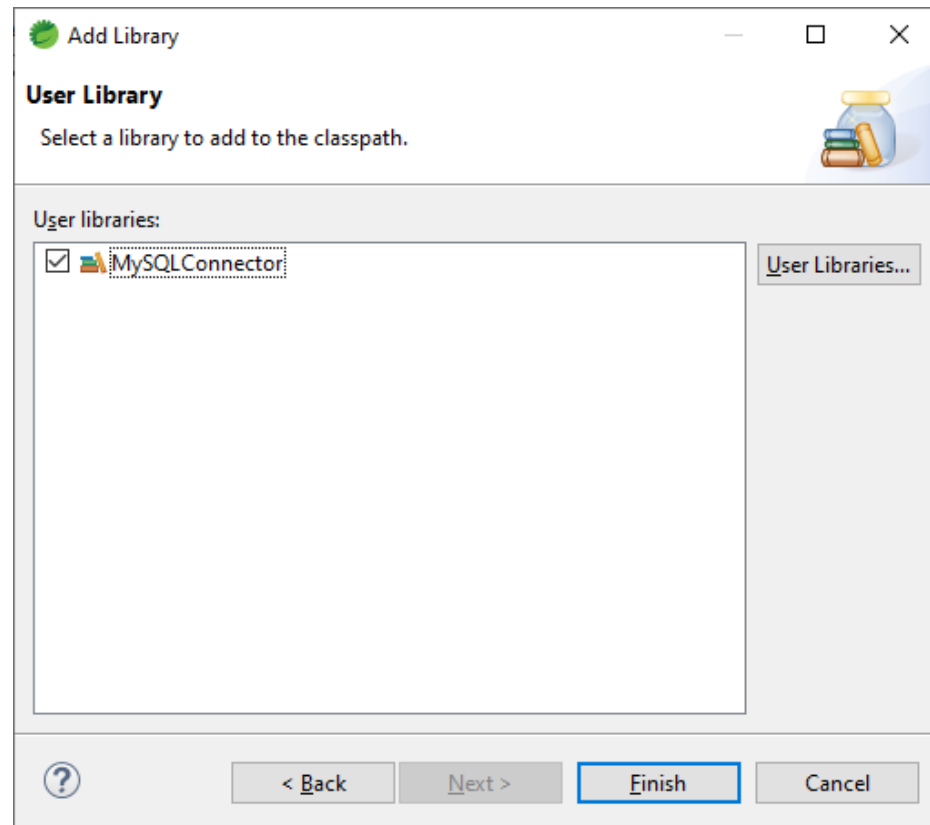
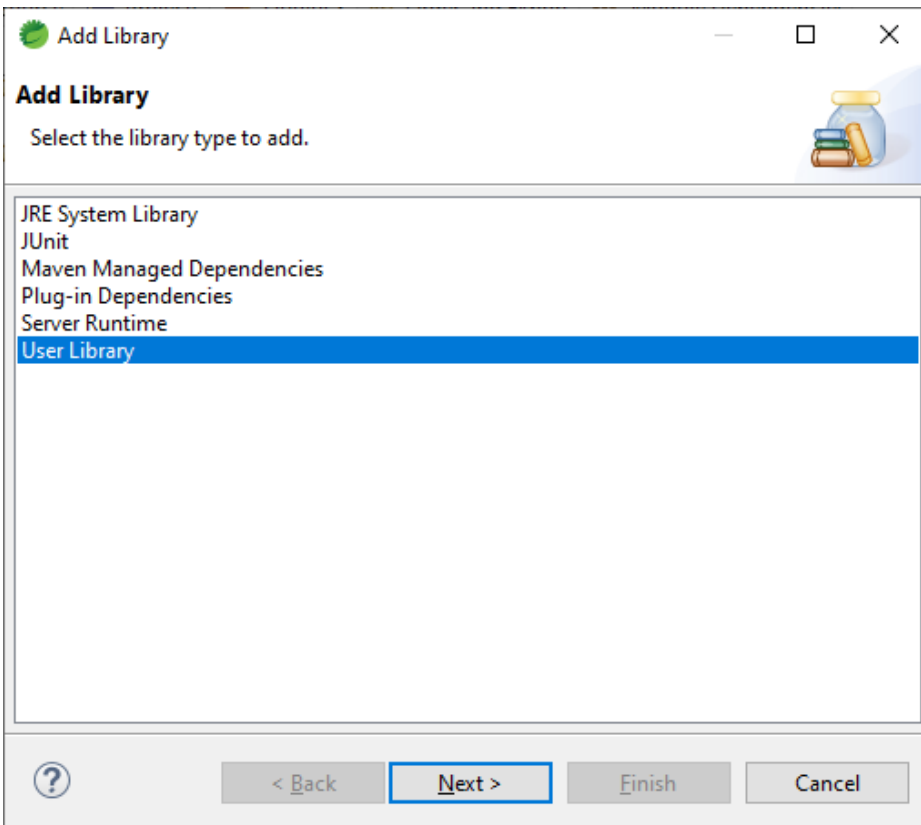
CRIAÇÃO DO PROJETO

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



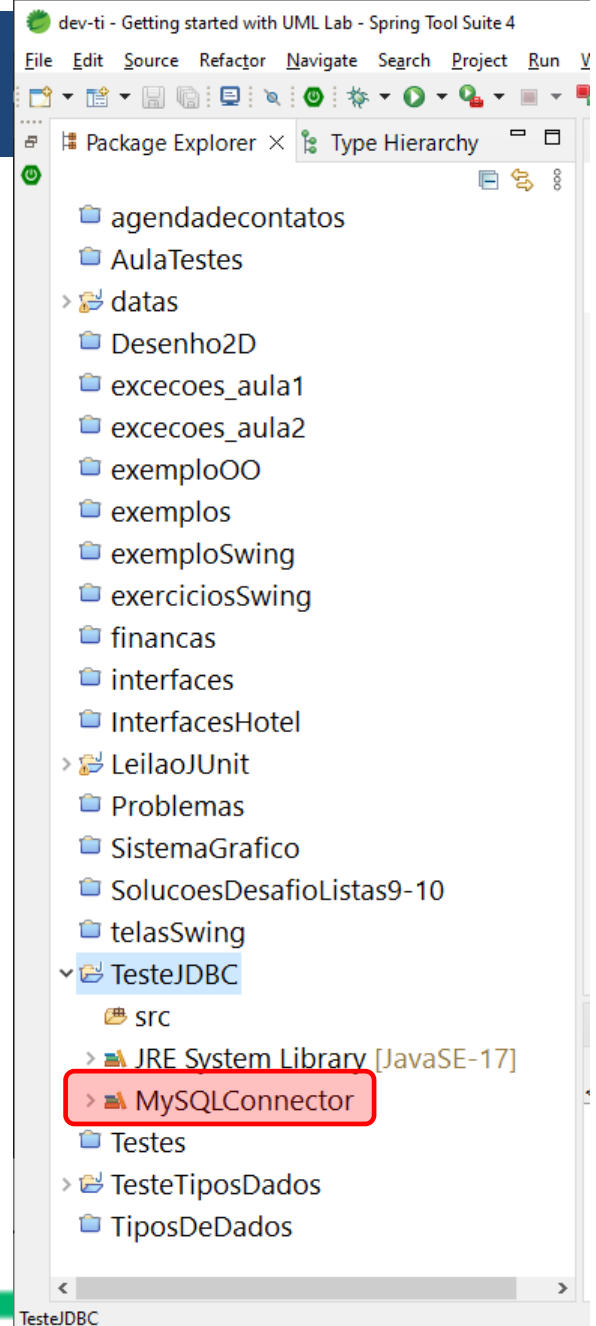
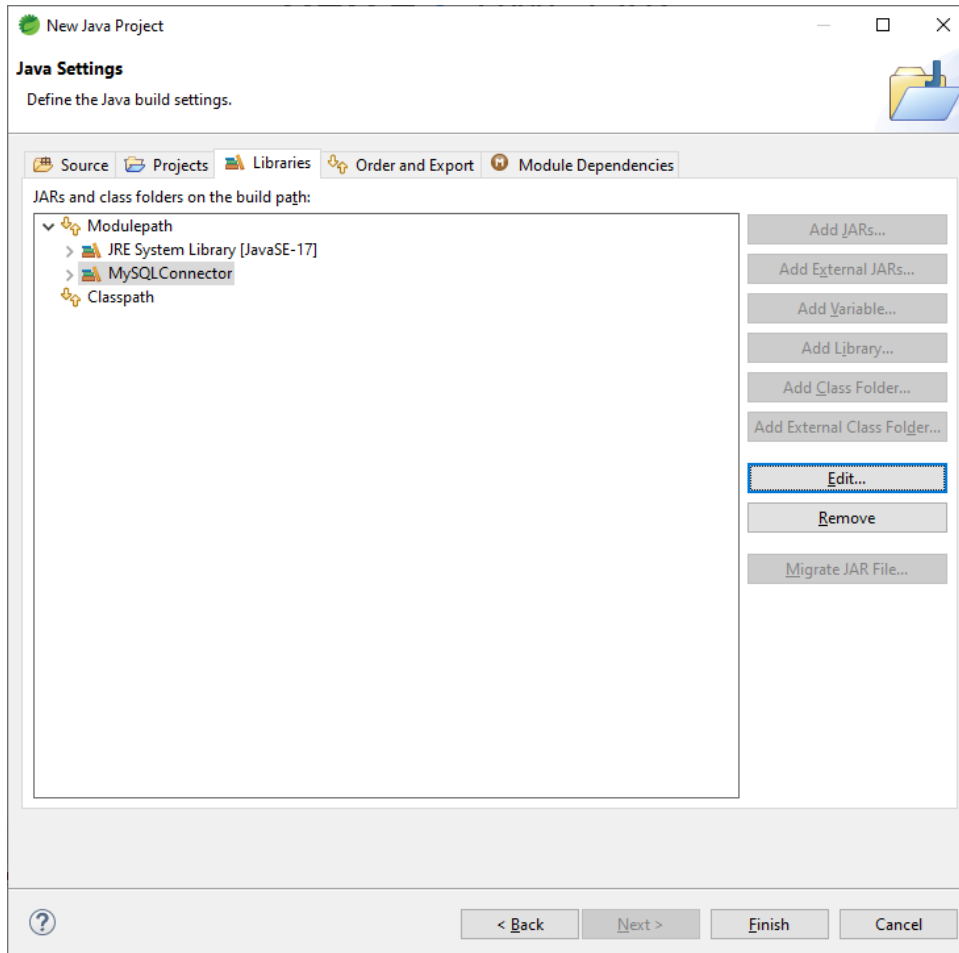
CRIAÇÃO DO PROJETO

3. Clique na aba *Libraries* e então no botão *Add Library ...*



CRIAÇÃO DO PROJETO

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



CRIAÇÃO DA CONEXÃO

- Para criar uma conexão são utilizados 2 padrões de projeto ([*design patterns*](#))

- [*Singleton*](#)

- É um dos 23 padrões de projeto originais GoF ([*Gang of Four*](#))
- Permite criar objetos únicos para os quais há apenas uma instância
- Este padrão garante que uma classe tenha apenas uma instância de si mesma, a própria classe sempre vai oferecer a própria instância dela e caso não tenha ainda uma instância, então ela mesma cria e retorna essa nova instância criada
 - A classe gerencia sua própria instância além de evitar que qualquer outra classe crie uma instância dela
 - Isso é conseguido declarando-se o construtor padrão como privado
 - Para criar a instância é necessário que passar pela classe obrigatoriamente, ou seja, nenhuma outra classe pode instanciar ela
- O padrão deve fornecer um ponto global de acesso à sua única instância, o que é conseguido através da implementação de um método público estático
- É considerado por muitos desenvolvedores como um [*antipadrão de projeto*](#)

| Singleton |
|-----------------------------|
| - singleton : Singleton |
| - Singleton() |
| + getInstance() : Singleton |

```
1 public class Singleton {
2     private static final Singleton INSTANCE = new Singleton();
3
4     private Singleton() {}
5
6     public static Singleton getInstance() {
7         return INSTANCE;
8     }
9 }
```

CRIAÇÃO DA CONEXÃO

- Para criar uma conexão são utilizados 2 padrões de projeto

- Fábrica

- Objeto para criação de outros objetos – formalmente uma fábrica é um método que retorna os objetos de uma classe
- É uma abstração de um construtor de uma classe
- Um *singleton* implementado pelo padrão *singleton* é uma fábrica formal pois retorna um objeto, mas não cria novas instâncias além da original

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 }
```

CRIAÇÃO DA CONEXÃO

- 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 }
```

CLASSE DE DOMÍNIO

- A entidade de domínio representa um produto, crie-a dentro do pacote `modelo`

```
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 + "];"
43    }
44 }
```

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 }
```


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 para reduzir o tamanho do código

- Fica com exercício realizar esta limpeza 😊

```
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 p = new Produto();
28
29                 p.setIdProd(rs.getInt("id_prod"));
30                 p.setNomeProd(rs.getString("nome_prod"));
31                 p.setDataCadastro(rs.getDate("data_cadastro"));
32                 p.setQuantidade(rs.getInt("quantidade"));
33                 p.setPreco(rs.getBigDecimal("preco"));
34
35                 lista.add(p);
36             }
37         } catch (SQLException e) {
38             throw new RuntimeException(e.getMessage());
39         } finally {
40             FabricaConexao.fechaConexao();
41         }
42
43         return lista;
44     }
```

DAO

- 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 }
```

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 }
```

DAO

- 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 }
```

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 }
```

DAO

- 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("2021-08-26"),
21                                     51,
22                                     new BigDecimal("123.45"));
23         prod.setIdProd(3);
24         dao.alterar(prod);
25         dao.adicionar(prod);
26
27         List<Produto> lista = dao.listarTodos();
28
29         System.out.println("Lista de Contatos");
30         System.out.println("-----");
31
32         for (Produto produto : lista) {
33             System.out.println("Id.....: " + produto.getIdProd());
34             System.out.println("Nome produto: " + produto.getNomeProd());
35             System.out.println("Data cadastro: " + fd.format(produto.getDataCadastro()));
36             System.out.println("Quantidade....: " + produto.getQuantidade());
37             System.out.println("Preço.....: " + fm.format(produto.getPreco()));
38             System.out.println();
39         }
40     }
41 }
```

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
```

DAO

- 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("2021-08-26"),
21                                     51,
22                                     new BigDecimal("123.45"));
23         prod.setIdProd(3);
24         // dao.alterar(prod);
25         // dao.adicionar(prod);
26         dao.excluir(3);
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 }
```


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 }
```

DAO

- 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("2021-08-26"),
21                                     51,
22                                     new BigDecimal("123.45"));
23         prod.setIdProd(3);
24         // dao.alterar(prod);
25         // dao.adicionar(prod);
26         // dao.excluir(3);
27
28         prod = dao.buscarPorId(2);
29         if (prod != null) {
30             System.out.println("Id.....: " + prod.getIdProd());
31             System.out.println("Nome produto.: " + prod.getNomeProd());
32             System.out.println("Data cadastro: " + fd.format(prod.getDataCadastro()));
33             System.out.println("Quantidade...: " + prod.getQuantidade());
34             System.out.println("Preço.....: " + fm.format(prod.getPreco()));
35             System.out.println();
36         }
```

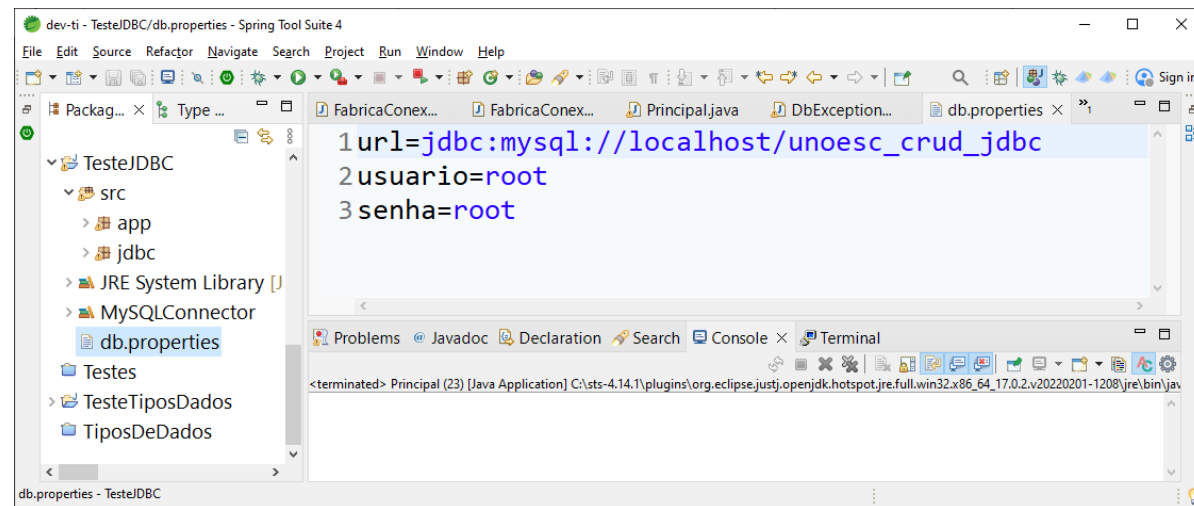
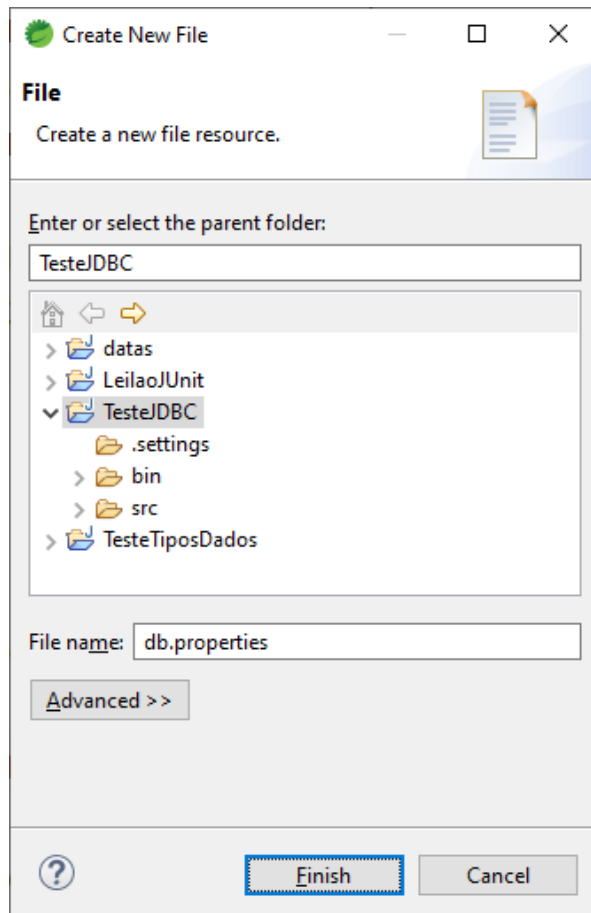
CRIANDO UMA EXCEÇÃO PERSONALIZADA

- Cria a classe `DbException` dentro do pacote `db`

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

EXTERNALIZANDO OS DADOS DA CONEXÃO

- Armazenando os dados da conexão em um arquivo externo
 - Clique com o botão direito sobre o projeto *TesteJDBC* e escolha *New* → *File*
 - Crie o arquivo chamado *db.properties* e insira nele o conteúdo mostrado abaixo



EXTERNALIZANDO OS D

- Modifique a classe de exceção e acrescente o código para ler o arquivo de configuração

```
1 package db;
2 import java.io.FileInputStream;
3 import java.io.FileNotFoundException;
4 import java.io.IOException;
5 import java.sql.Connection;
6 import java.sql.DriverManager;
7 import java.sql.SQLException;
8 import java.util.Properties;
9
10 public class FabricaConexao {
11     private static Connection conexao = null;
12
13     private FabricaConexao() { }
14
15     public static Connection getConexao() {
16         try {
17             if (conexao != null && !conexao.isClosed()) {
18                 return conexao;
19             }
20
21             Properties prop = loadProperties();
22
23             final String url = prop.getProperty("url");
24             final String usuario = prop.getProperty("usuario");
25             final String senha = prop.getProperty("senha");
26
27             System.out.println("Conectando ao banco de dados...");
28             conexao = DriverManager.getConnection(url, usuario, senha);
29             System.out.println("Conexão realizada com sucesso!");
30
31             return conexao;
32         } catch (SQLException | IOException e) {
33             // Converte exceção checkada em uma não checkada
34             throw new DbException(e.getMessage());
35         }
36     }
37
38     public static void fechaConexao() {
39         if (conexao != null) {
40             try {
41                 System.out.println("Fechando conexão...");
42                 conexao.close();
43                 System.out.println("Conexão fechada com sucesso!");
44             } catch (SQLException e) {
45                 throw new DbException(e.getMessage());
46             }
47         }
48     }
49
50     private static Properties loadProperties() throws FileNotFoundException, IOException {
51         try (FileInputStream fs = new FileInputStream("db.properties")) {
52             Properties prop = new Properties();
53             prop.load(fs);
54             return prop;
55         }
56     }
57 }
```