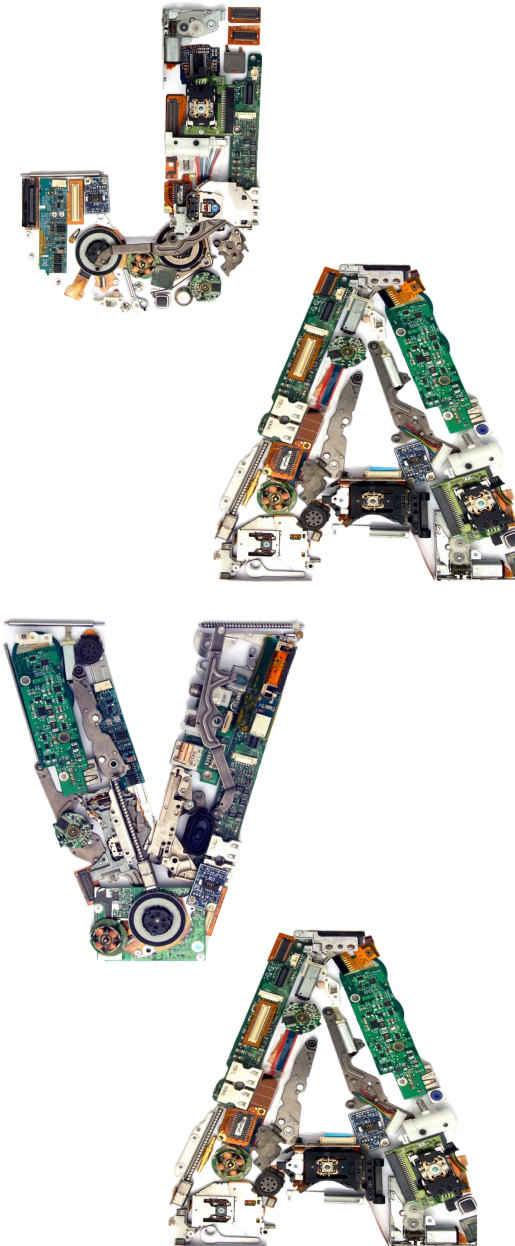


Programação Orientada a Objetos com Java e WEB

Banco de Dados



Introdução

Muitos sistemas precisam manter as informações com as quais trabalham para permitir operações futuras (consultas, alterações, geração de relatórios, dentre outras).

Para que as informações sejam mantidas de forma permanente devemos utilizar arquivos ou banco de dados.

A maioria dos banco de dados utilizados no mercado são chamados de **relacionais**, onde os dados são armazenados em tabelas.

O processo de armazenamento de dados é também chamado de ***persistência***.

Introdução

A biblioteca para manipulação de banco de dados relacionais é chamada de JDBC (*Java Database Connectivity*).

Existem diversas ferramentas do tipo **ORM** (*Object Relational Mapping*) que facilitam bastante o uso do JDBC.

Para que o Java faça a conexão com um banco de dados é utilizado um conjunto de interfaces que devem ser implementadas.

O conjunto de interfaces é chamada de JDBC e está definida no pacote **java.sql**.

Introdução

Ao trabalhar com um banco de dados é necessário que tenhamos classes concretas que implementam a interface.



JDBC

JDBC (*Java Database Connectivity*) é um conjunto de classes e interfaces (API) escritas em Java que faz o envio de instruções SQL para qualquer banco de dados relacional.

É uma API de baixo nível e base para APIs de alto nível. Possibilita o uso de banco de dados já instalados. Para cada banco de dados há um driver específico.

Um programa Java utiliza uma API JDBC única que independe do banco de dados ou driver que estiver sendo utilizado.

Os drivers para conexão e acesso aos principais banco de dados existentes são fornecidos pelos seus fabricantes.

JDBC – Principais Componentes

Componente	Descrição
DriverManager	Responsável por encontrar o driver e estabelecer a conexão com o SGBDR
Connection	Representa a conexão com o SGBDR por onde serão passados os comandos SQL
Statement	Execução de comando SQL
PreparedStatement	
CallableStatement	
ResultSet	Representa os registros retornados de um Statement, PreparedStatement ou CallableStatement

Drivers e Strings de Conexão

Banco	Driver JDBC	String de Conexão
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://nomeDoHost:porta/serviço
Oracle	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@nomeDoHost:númeroDaPorta:serviço
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:nomeDoHost:númeroDaPorta/nomeDoBanco
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:nomeDoHost:númeroDaPorta/nomeDoBanco
SQLServer	com.microsoft.sqlserver.jdbc.SQLServerDriver	jdbc:sqlserver://nomeDoHost;user=?;password=?;

Conexão com Banco de Dados

As classes java para manipulação de banco de dados estão no pacote *java.sql*.

Para estabelecer uma conexão com um banco de dados é preciso realizar duas tarefas distintas:

- ☐ registrar o driver que será utilizado.
- ☐ solicitar do gerenciador de drivers a abertura da conexão.

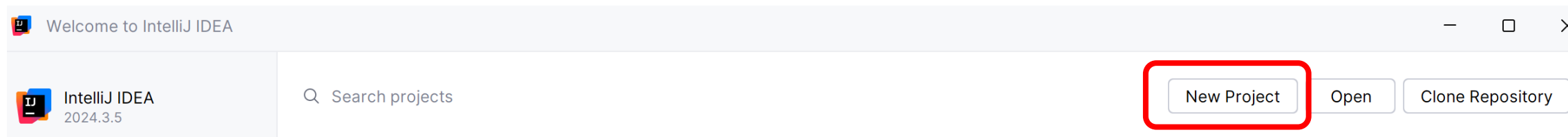
Apache Maven

Maven é uma ferramenta de gerenciamento de dependências de software para projetos Java. Ele ajuda a automatizar o processo de construção, testes e distribuição de projetos Java. Principais características:

1. **Gerenciamento de dependências:** permite que você gerencie as dependências do seu projeto Java. Ele faz o download automaticamente dos pacotes necessários e os integra ao seu projeto.
2. **Padrão de diretório:** tem um padrão de diretório bem definido que é seguido por todos os projetos Maven. Isso ajuda a garantir que o seu projeto Java seja organizado e fácil de entender.



Criação de um projeto Maven no IntelliJ



Clicar em New Project

Criação de um projeto Maven no IntelliJ

Escolha o tipo de projeto que será criado

1º

The screenshot shows the 'New Project' dialog in IntelliJ IDEA. On the left, under the 'Generators' section, 'Maven Archetype' is selected and highlighted with a red box. The right pane shows the configuration for a general Maven project. The 'Name' field is set to 'untitled' and is highlighted with a red box. The 'Location' field shows the path '~\OneDrive\Área de Trabalho\solei'. The 'JDK' is set to '21 Oracle OpenJDK 21.0.2'. The 'Catalog' is set to 'Internal' and is highlighted with a red box. The 'Archetype' dropdown is set to 'org.apache.maven.archetypes:maven-archetype-quickstart' and is highlighted with a red box. The 'Version' is set to '1.1'. At the bottom right, the 'Create' button is highlighted with a red box.

2º Forneça o nome do projeto

3º

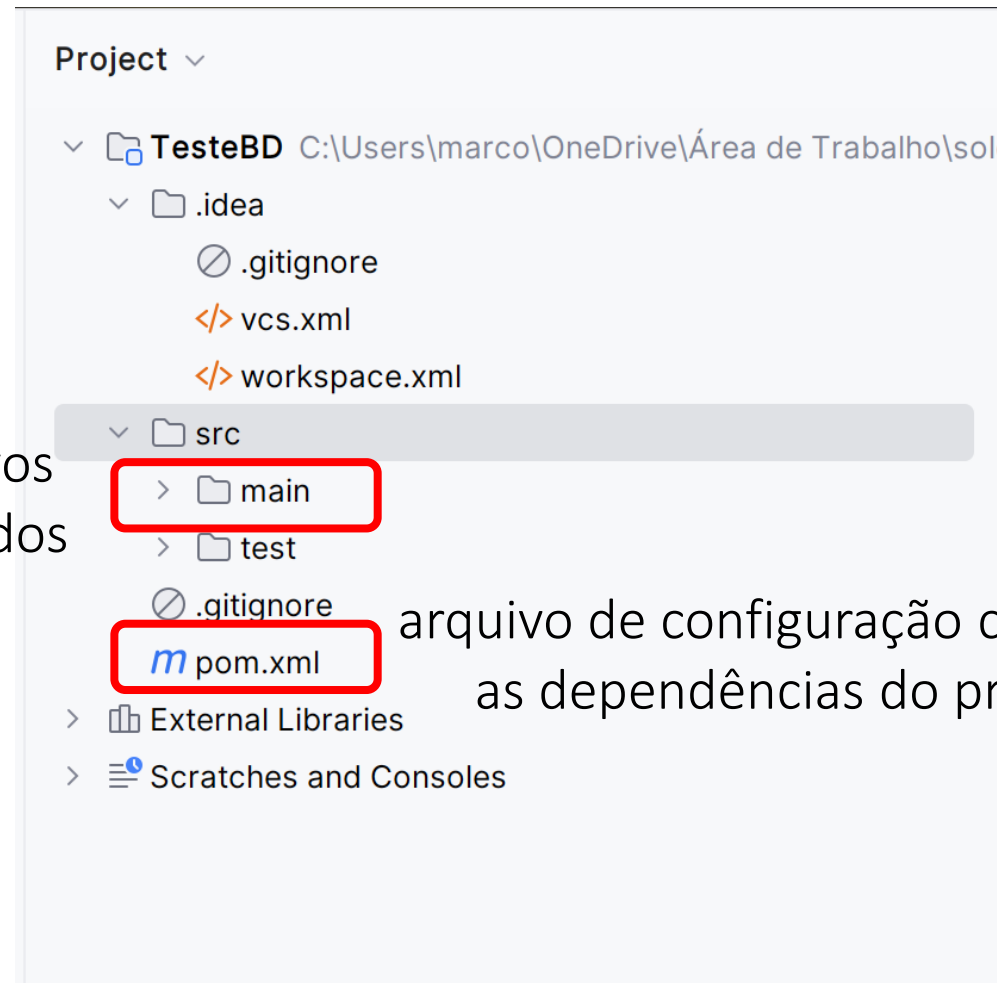
Escolha a opção quickstart

4º

Clique em **Create** para criar o projeto

Criação de um projeto Maven no IntelliJ

pasta onde os arquivos
Java serão armazenados



arquivo de configuração contendo
as dependências do projeto

Conexão com Banco de Dados – carregar driver

Para carregar o driver de conexão:

```
try {  
    Class.forName("driver_jdbc");  
}  
catch(ClassNotFoundException ex)  
{  
    System.out.println("Erro ao carregar o driver");  
}
```

O método estático *forName()* é utilizado para carregar o driver apropriado do banco de dados.

Conexão com Banco de Dados – drivers

Exemplos de drivers utilizados nos principais banco de dados:

```
Class.forName("com.mysql.jdbc.Driver");  
Class.forName("net.sourceforge.jtds.jdbc.Driver");  
Class.forName("oracle.jdbc.driver.OracleDriver");  
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
Class.forName("org.hsqldb.jdbcDriver");
```

Os drivers devem ser baixados diretamente com os fabricantes do SGBDR, no caso do MySQL, o driver é o *Connector/J* (<http://dev.mysql.com/downloads/connector/j>)

Conexão com Banco de Dados – abertura de conexão

Para solicitar a abertura da conexão:

```
try {  
    Connection c;  
    c = DriverManager.getConnection(string_de_conexão);  
}  
catch (SQLException s) {  
    System.out.println("Erro");  
}
```



Conexão mostrada no
slide 7.

Conexão com Banco de Dados – fechamento de conexão

Depois de utilizar uma conexão, a mesma deverá ser fechada

```
try {  
    c.close();  
}  
catch( SQLException s ) {  
    System.out.println("Problemas ao fechar a conexão");  
}
```


Conexão com Banco de Dados – resumo

Em resumo, temos o seguinte código para conexão com um banco de dados:

```
import java.sql.*;
try {
    Class.forName("driver_conexão");
    Connection c = DriverManager.getConnection(string, login, senha);
}
catch (ClassNotFoundException ex) {
    System.out.println("Erro ao carregar o driver");
}
catch (SQLException s) {
    System.out.println("Erro ao estabelecer conexão");
}
```

Interface PreparedStatement

Para realizar operações em um banco de dados é utilizada a interface *PreparedStatement* (*pacote java.sql*).

Armazena um comando SQL predefinido. Ideal para comandos configuráveis e usados repetidas vezes.

Um objeto *PreparedStatement* é obtido com o método *prepareStatement()* da classe *Connection*.

A string SQL do *PreparedStatement* pode ser parametrizada com sinais *?*. Os métodos *setXXX()* são utilizados para atribuir valores a esses parâmetros.

Interface PreparedStatement

Para realizar operações em um banco de dados é utilizada a interface *PreparedStatement* (*pacote java.sql*).

O método *executeUpdate()* de *PreparedStatement* é utilizado para executar as cláusulas: *INSERT*, *UPDATE* e *DELETE*.

O método *executeQuery()* de *PreparedStatement* é utilizado para executar a cláusula *SELECT*. O método retorna um objeto *ResultSet*.

Interface PreparedStatement

Exemplo para inserir dados em uma tabela chamada **Aluno**:

```
...
String sql = "INSERT INTO Aluno VALUES (?, ?)";
Connection c = DriverManager.getConnection(conexão);
PreparedStatement p = c.prepareStatement(sql);
//configura o valor do primeiro ?
p.setInt(1, 50231);
//configura o valor do segundo ?
p.setString(2, "Pedro");
int i = p.executeUpdate();
...
```

Interface PreparedStatement

Principais métodos de PreparedStatement:

- ❑ **public ResultSet executeQuery() throws SQLException**
 - Executa uma query e retorna um conjunto de resultados (ResultSet).
- ❑ **public int executeUpdate() throws SQLException**
 - Executa um comando SQL INSERT, UPDATE, DELETE ou qualquer outro que não tenha retorno. Este método retorna apenas o número de registros que foram alterados.

Interface PreparedStatement

Métodos setXXX():

- São utilizados para substituir a *i-ésima* ocorrência de ? pelo valor passado no segundo parâmetro recebido pelo método.
- O tipo passado como parâmetro deve coincidir com o tipo de dado SQL que está sendo afetado na tabela no banco de dados.
 - `public void setDate(int i, Date d);`
 - `public void setInt(int i, int d);`
 - `public void setFloat(int i, float d);`
 - `public void setDouble(int i, double d);`
 - `public void setString(int i, String d);`
 - `public void setObject(int i, Object d);`

Interface ResultSet

Ao executar uma consulta (***SELECT***) em um banco de dados, o método ***executeQuery()*** de ***PreparedStatement*** retorna um objeto ***ResultSet***.

ResultSet é uma interface do pacote ***java.sql*** utilizada para manipular os dados resultantes de uma consulta com a cláusula ***SELECT***.

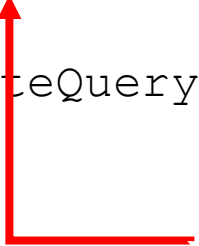
Um objeto ***ResultSet*** representa uma “tabela” contendo todos os dados da consulta. Mantém um cursor posicionado no registro corrente.

Inicialmente o cursor está posicionado antes da primeira linha.

Interface ResultSet

O cursor é deslocado com o método *next()* de *ResultSet*, que retorna o valor *false* quando não encontrar mais registros no *ResultSet*. Exemplo:

```
....
String sql = "select * from aluno";
PreparedStatement stmt = c.prepareStatement(sql);
// executa um select
ResultSet rs = stmt.executeQuery();
// itera no ResultSet
while (rs.next()) {
}
....
```



conexão devidamente estabelecida

Interface ResultSet

Para iterar sobre os objetos de um *ResultSet*, a interface oferece uma série de métodos sobrecarregados *getXXX()*.

Os principais métodos são:

- ❑ `getString(int index)` ou `getString(String nomeDaColuna)`;
- ❑ `getInt(int index)` ou `getInt(String nomeDaColuna)`;
- ❑ `getDouble(int index)` ou `getDouble(String nomeDaColuna)`;
- ❑ `getFloat(int index)` ou `getFloat(String nomeDaColuna)`;
- ❑ `getLong(int index)` ou `getLong(String nomeDaColuna)`;
- ❑ `getBigDecimal(int index)` ou `getBigDecimal (String nomeDaColuna)`: retorna um objeto `BigDecimal` (pacote `java.lang.Math`) com toda precisão;
- ❑ `getObject(int index)` ou `getObject(String nomeDaColuna)`;

Interface ResultSet – exemplo

O trecho de código abaixo pesquisa todos os registros de uma tabela chamada **Aluno**. A tabela apresenta apenas dois campos: **rm** e **nome**. Os dados pesquisados são impressos no vídeo.

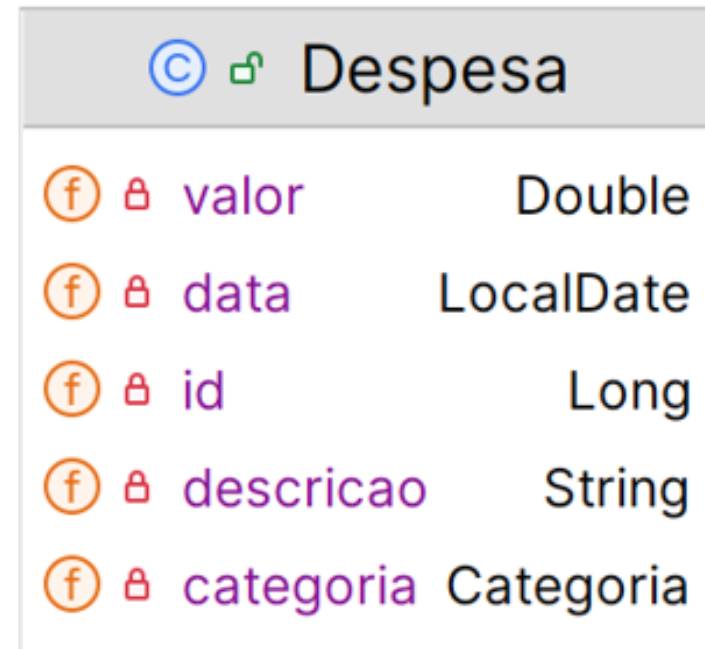
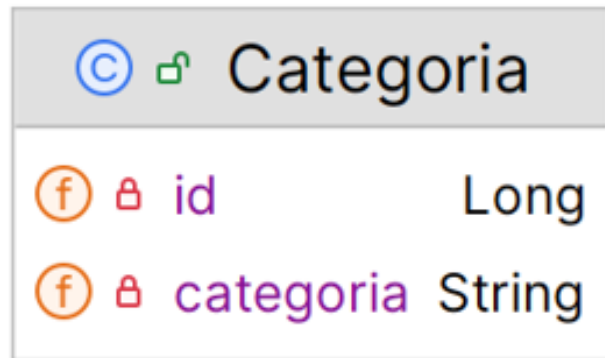
```
....
String sql = "select * from aluno";
PreparedStatement p = c.prepareStatement(sql);

// executa um select
ResultSet rs = stmt.executeQuery();

// itera no ResultSet
int rm;
String nome;
while (rs.next()) {
    a = rs.getInt("rm");
    nome = rs.getString("nome");
    System.out.println(rm+"\t"+nome+"\n");
}
```

Exercício de programação

Implemente as classes a seguir.



Exercício de programação

Implemente as classes a seguir.

© DespesaDAO		
f	rs	ResultSet
f	ps	PreparedStatement
m	salvar(Despesa)	void
m	atualizar(Despesa)	void
m	listar()	List<Despesa>
m	excluir(Long)	void

© CategoriaDAO		
f	rs	ResultSet
f	ps	PreparedStatement
m	buscarId(String)	Long
m	salvar(Categoria)	void
m	listar()	List<Categoria>
m	excluir(String)	void
m	atualizar(Categoria)	void

Bibliografia



- ❑ DEITEL, H. M., DEITEL, P. J. **JAVA como programar**. 10ª edição. São Paulo: Prentice-Hall, 2010.
- ❑ SCHILDT, H. **Java para Iniciantes – Crie, Compile e Execute Programas Java Rapidamente**. 6ª Edição, Editora Bookman, Porto Alegre, RS, 2015.



Apêndice – Principais Instruções SQL

❑ Criação de Tabela, CREATE TABLE:

```
CREATE TABLE tb_cliente (  
    ID INTEGER NOT NULL, /*Integer – auto numeração*/  
    nome varchar2(100) NOT NULL, /*String*/  
    cpf varchar2(11) NOT NULL, /*char - 11 posições*/  
    ddd INTEGER, /*Integer*/  
    telefone varchar2(20), /*String*/  
    ativo NUMBER(1) NOT NULL,  
    Valor_Ultima_Compra number(10,2) NOT NULL, /*Double -2 casas*/  
    PRIMARY KEY (ID)  
)
```

❑ Exclusão de Tabela, DROP TABLE:

```
DROP TABLE tb_cliente
```

Apêndice – Principais Instruções SQL

❑ Seleção, SELECT:

```
SELECT * FROM tb_cliente;
```

```
SELECT Ativo, Nome, CPF FROM tb_cliente;
```

❑ Condição, WHERE:

```
SELECT * FROM tb_cliente WHERE ativo = true;
```

```
SELECT CPF, Nome FROM tb_cliente WHERE Nome LIKE '%ia%';
```

```
SELECT * FROM tb_cliente WHERE (DDD is null) AND (CPF is not null) AND ((Nome = 'Maria') OR (NOME = 'Tiago')) AND Valor_Ultima_Compra <> 1000;
```

❑ Ordem, ORDER BY:

```
SELECT * FROM tb_cliente ORDER BY Nome ASC;
```

```
SELECT Ativo, Nome, CPF FROM tb_cliente ORDER BY ATIVO DESC, NOME ASC;
```

Apêndice – Principais Instruções SQL

❑ Inclusão, INSERT:

```
INSERT INTO tb_cliente (Nome,CPF,DDD,Telefone,Ativo,Valor_Ultima_Compra)
VALUES ('Maria', '26875436687', null, null, true, 452.78);
```

```
INSERT INTO tb_cliente(Nome,CPF) values ('Tiago', '30152223678');
```

```
INSERT INTO tb_cliente(Nome,CPF) values ('Felix', '01235478222');
```

❑ Alteração, UPDATE:

```
UPDATE tb_cliente SET DDD = 11 WHERE DDD is null;
```

❑ Exclusão, DELETE:

```
DELETE FROM tb_cliente WHERE Nome = 'Felix';
```