



Modelagem e Desenvolvimento Orientada a Objeto

Prof. Anderson Barros

anderson.barros@uni9.pro.br

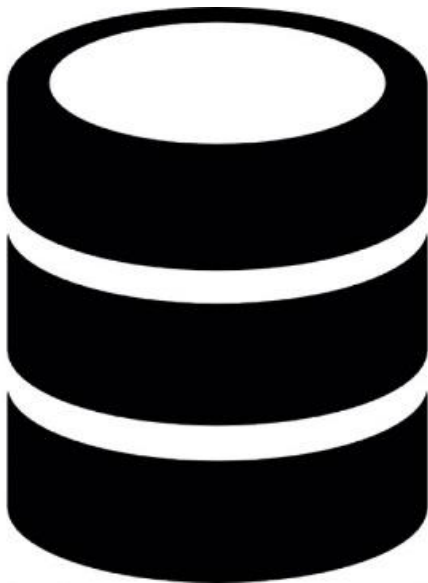
Tópicos:

Introdução de Banco de Dados MySQL

O que é SQL?

SQL é sigla inglesa de “***Structured Query Language***” que significa, em Português, Linguagem de Consulta Estruturada, uma linguagem padrão de gerenciamento de dados que interage com os principais bancos de dados baseados no **modelo relacional**.

Geralmente o banco de dados vem simbolizado por uma sequência de discos empilhados como apresentado na figura abaixo



Alguns dos principais sistemas que utilizam a linguagem SQL, conhecido como **SGBD** (Sistema de Gerenciamento de Banco de Dados) ou **DBMS** (Data Base Management System), são: MySQL, Oracle, Firebird, Microsoft Access, SQL Server, PostgreSQL, DB2, MariaDB e etc.



Dentro de um SGBD, pode conter diversos Bancos de Dados. Em cada Banco de Dados, conterà uma ou mais tabelas e em cada tabela poderá conter 1 ou mais registros. Estas tabelas poderão estar relacionadas umas as outras (modelo relacional).



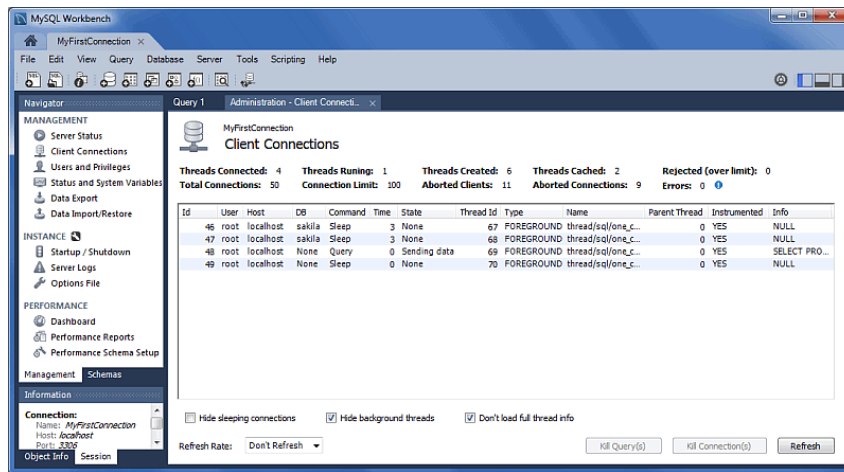
Alguns dos principais comandos SQL para manipulação de dados são: **INSERT** (inserção), **SELECT** (consulta), **UPDATE** (atualização), **DELETE** (exclusão). SQL possibilita ainda a criação de relações entre tabelas e o controle do acesso aos dados.

Através destes comandos que será possível desenvolver sistemas que façam o conhecido **CRUD**.

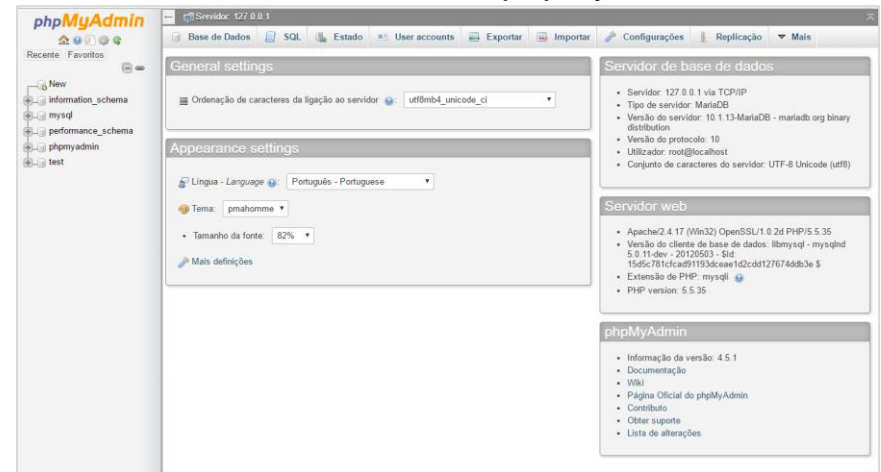
CRUD (acrônimo de **C**reate, **R**ead, **U**ppdate e **D**eleete na língua Inglesa), para as quatro operações básicas utilizadas em bases de dados relacionais (SGBD em Português / RDBMS em Inglês), ou em interface para utilizadores para criação, consulta, atualização e destruição de dados.

Interface Gráfica para Administração do SGBD MySQL

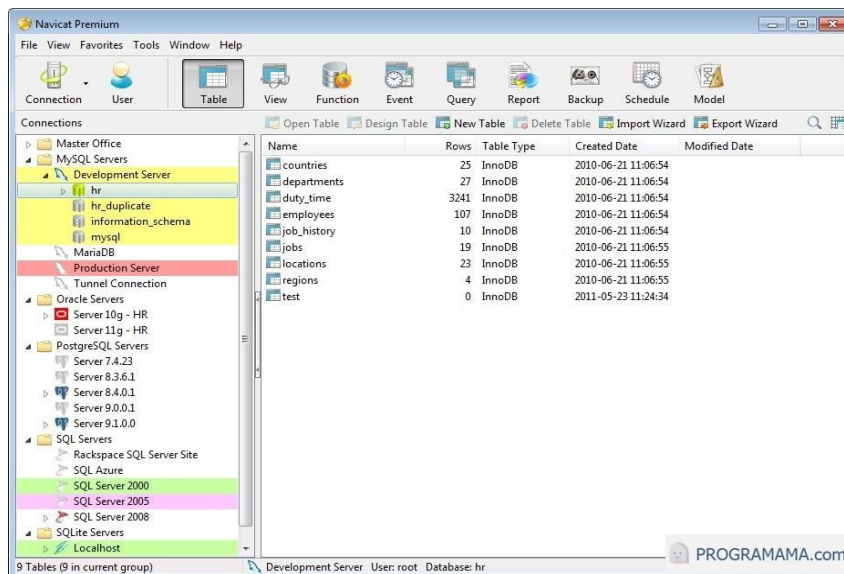
Interface Gráfica MySQL Workbench



Interface Gráfica phpMyAdmin



Interface Gráfica Navicat



ANALISANDO O FRONTEND DE UM SISTEMA

Sistema de Notas dos Alunos

Digite o nome completo do aluno










1º Bimestre

2º Bimestre

3º Bimestre

4º Bimestre

Cadastrar Aluno

Nome	1º Bim	2º Bim	3º Bim	4º Bim	Média	Status	Ação
Humberto Costa Braga	6,5	8	10	7,5	8,0	Aprovado	 
Fernanda Brito	3,5	7,5	2,5	6	4,9	Reprovado	 
Robson Siqueira Prado	2,5	6,5	9,5	6,5	6,3	Recuperação	 
Sandra Costa	10	10	10	10	10,0	Aprovado	 
Jandira Queiroz	3,5	2,5	10	4,5	5,1	Recuperação	 

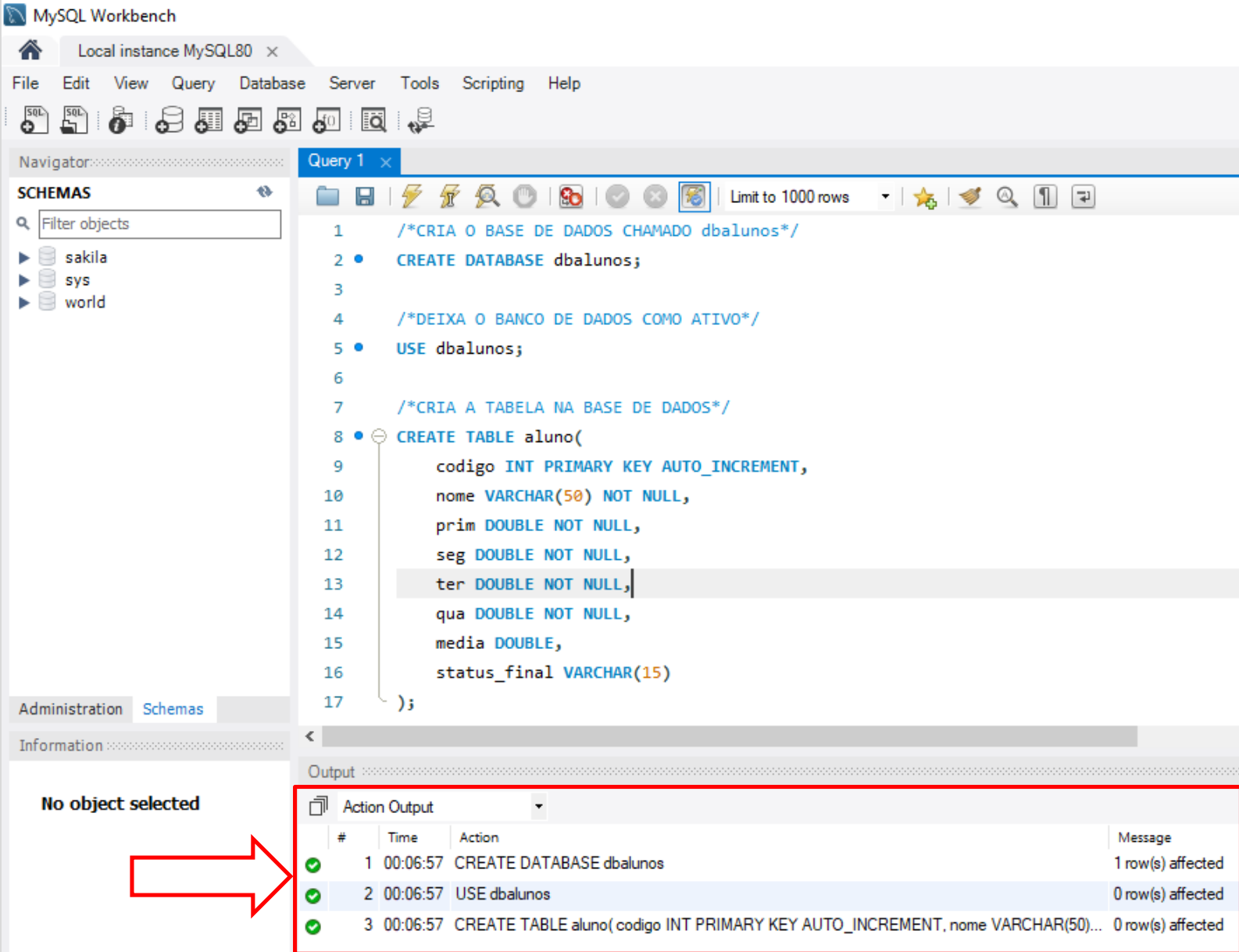
Cadastrar

Mostrar

Alterar

Excluir

CRIANDO A BASE DE DADOS E A TABELA



MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- ▶ sakila
- ▶ sys
- ▶ world

Administration Schemas

Information

No object selected

Query 1 x

Limit to 1000 rows

```
1  /*CRIA O BASE DE DADOS CHAMADO dbalunos*/
2  • CREATE DATABASE dbalunos;
3
4  /*DEIXA O BANCO DE DADOS COMO ATIVO*/
5  • USE dbalunos;
6
7  /*CRIA A TABELA NA BASE DE DADOS*/
8  • CREATE TABLE aluno(
9      codigo INT PRIMARY KEY AUTO_INCREMENT,
10     nome VARCHAR(50) NOT NULL,
11     prim DOUBLE NOT NULL,
12     seg DOUBLE NOT NULL,
13     ter DOUBLE NOT NULL,
14     qua DOUBLE NOT NULL,
15     media DOUBLE,
16     status_final VARCHAR(15)
17 );
```

Output

Action Output

#	Time	Action	Message
✓ 1	00:06:57	CREATE DATABASE dbalunos	1 row(s) affected
✓ 2	00:06:57	USE dbalunos	0 row(s) affected
✓ 3	00:06:57	CREATE TABLE aluno(codigo INT PRIMARY KEY AUTO_INCREMENT, nome VARCHAR(50)...	0 row(s) affected

INSERINDO DADOS DENTRO DA TABELA

- 1 • `INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Carlos Siqueira',7.5,8.5,9.0,10);`
- 2 • `INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Pedro Alcantara',5.0,4.5,7.0,4.5);`
- 3 • `INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Flavia Macedo',3.0,6.5,4.0,3.5);`
- 4 • `INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Marcela Nara',3.0,1.5,10.0,5.0);`
- 5 • `INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Humberto Braga',10,10,10,10);`
- 6 • `INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Nayara Yago',3.5,7.5,6.0,4);`

✓	4	00:18:53	INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Carlos Siqueira',7.5,8.5,9.0,10)	1 row(s) affected
✓	5	00:18:53	INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Pedro Alcantara',5.0,4.5,7.0,4.5)	1 row(s) affected
✓	6	00:18:53	INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Flavia Macedo',3.0,6.5,4.0,3.5)	1 row(s) affected
✓	7	00:18:53	INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Marcela Nara',3.0,1.5,10.0,5.0)	1 row(s) affected
✓	8	00:18:54	INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Humberto Braga',10,10,10,10)	1 row(s) affected
✓	9	00:18:54	INSERT INTO aluno(nome,prim,seg,ter,qua) VALUES('Nayara Yago',3.5,7.5,6.0,4)	1 row(s) affected

MOSTRANDO DADOS DA TABELA

```
1      /* MOSTRAR TODAS AS COLUNAS DE TODOS OS ALUNOS */
2 •    SELECT * FROM aluno;
3
4      /* MOSTRAR SOMENTE O ALUNO COM O CODIGO 2 */
5 •    SELECT * FROM aluno WHERE codigo=2;
6
7      /* MOSTRAR SOMENTE O NOME E A MÉDIA DO ALUNO COM O CODIGO 5 */
8 •    SELECT nome, media FROM aluno WHERE codigo=5;
```

EXCLUINDO DADOS DA TABELA

```
1      /* EXCLUI O ALUNO COM CÓDIGO */
2 •    DELETE FROM aluno WHERE codigo = 3;
3
4      /* EXCLUI TODAS AS ALUNAS QUE COMEÇA COM MARCELA */
5 •    DELETE FROM aluno WHERE nome LIKE 'Marcela%';
```

ALTERANDO DADOS DA TABELA

```
1      /* ALTERA O PARA MAYARA YAGO O ALUNO COM O CÓDIGO 6 */
2  ●   UPDATE aluno SET nome='Mayara Yago' WHERE codigo=6;
3
4      /* ALTERA AS NOTAS DO ALUNO COM CÓDIGO 4 */
5  ●   UPDATE aluno set prim=3.5, seg=9.5, ter=7.5, qua=9.0 WHERE codigo=4;
```

Programação Orientado a Objeto (pré-requisito)

Analizando o método main()

Foi estudado anteriormente sobre os seguintes tópicos:

public - modificador de acesso

static – referência a atributo ou método da classe

void – tipo de retorno vazio

String[] – vetor de caracteres

Todos estes elementos aparecem em um método especial que tivemos contato ao começar a trabalhar com java. O método main. Este método é o principal método do java, ou seja, quando iniciamos um programa este é procurado e chamado como ponto inicial da execução do programa.

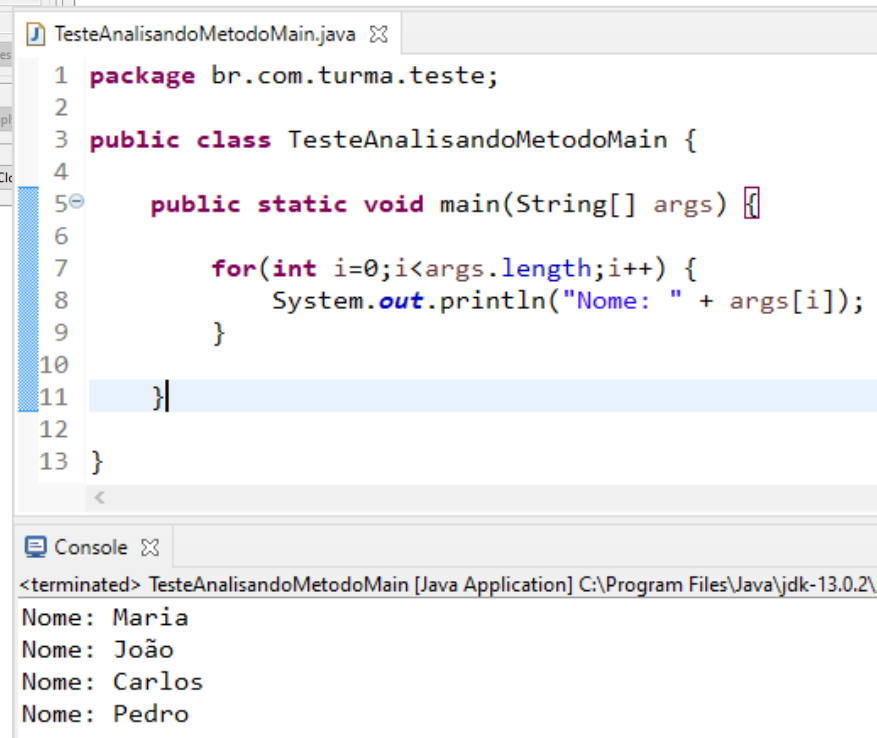
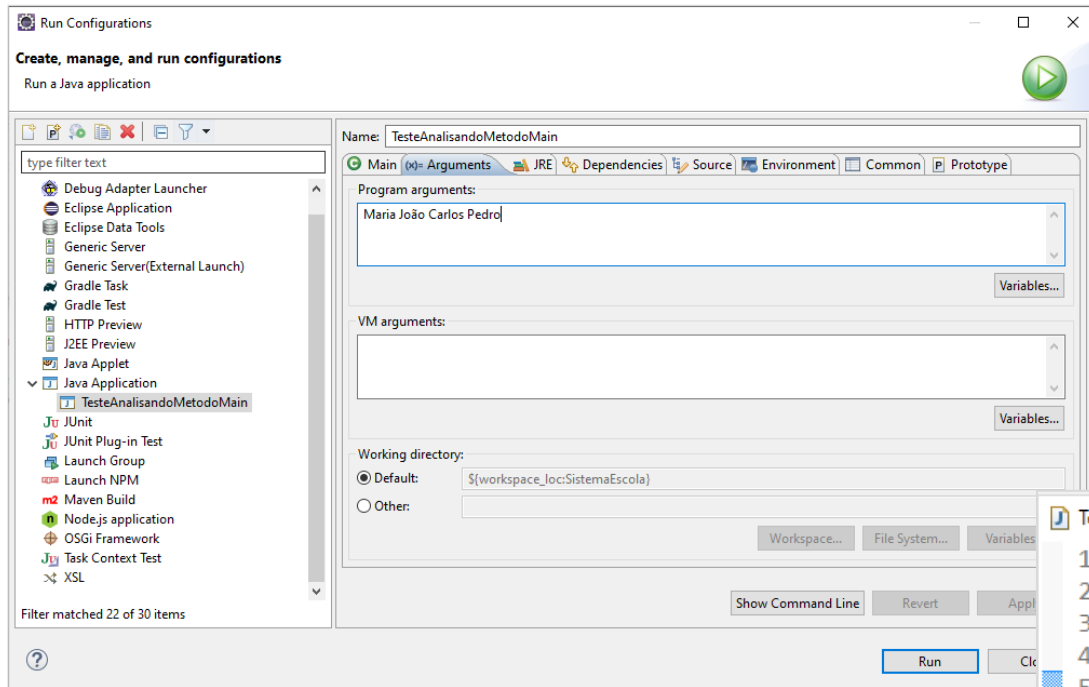
Analizando o método main()

```
TesteAnalizandoMetodoMain.java
1 package br.com.turma.teste;
2
3 public class TesteAnalizandoMetodoMain {
4
5     public static void main(String[] args) {
6
7
8     }
9
10 }
```

Método que é o ponto inicial na execução de um projeto java.

O argumento ou parâmetro significa que é possível executar o método já enviando parâmetros a ele. Pense na execução de um programa java, utilizando por exemplo o prompt de command (command prompt).

Analizando o método main()



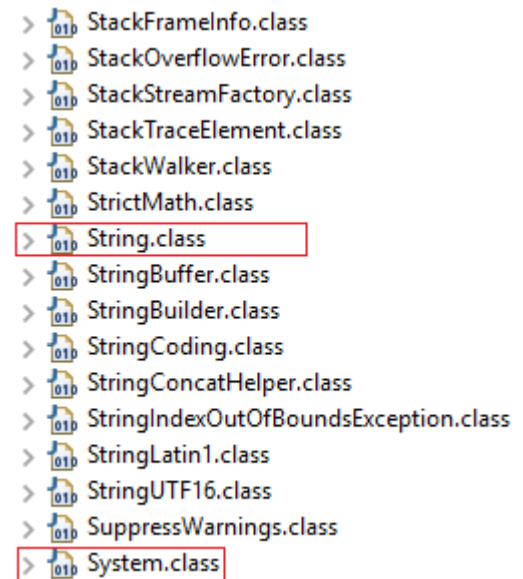
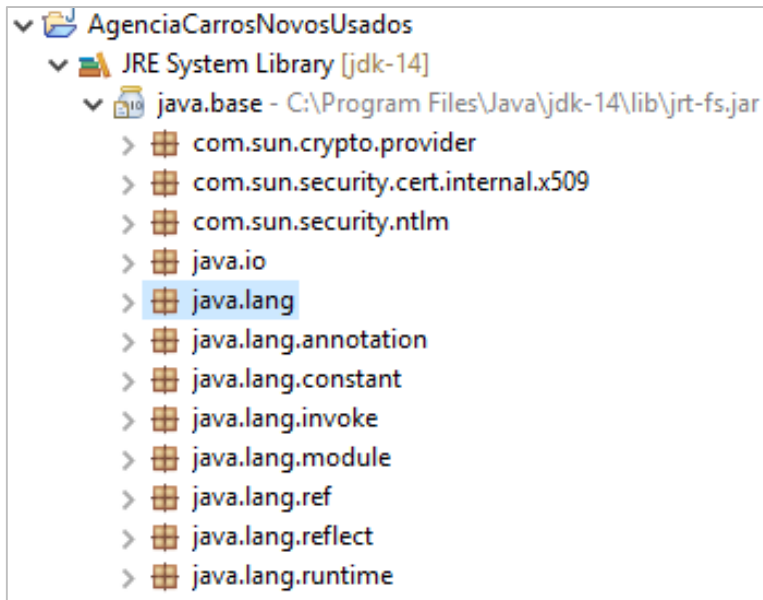
Entendendo a Estrutura de Pacotes (Packages)

... é um namespace usado para organizar um conjunto de interfaces e classes relacionadas.

Um pacote (package) é um namespace usado para organizar um conjunto de classes relacionadas. Podemos, por analogia, pensar nos pacotes como pastas que contém classes que trabalham em conjunto.

Programas em Java podem se tornar imensos, contendo centenas ou até mesmo milhares de classes, e por isso a organização dessas classes em pacotes é muito importante.

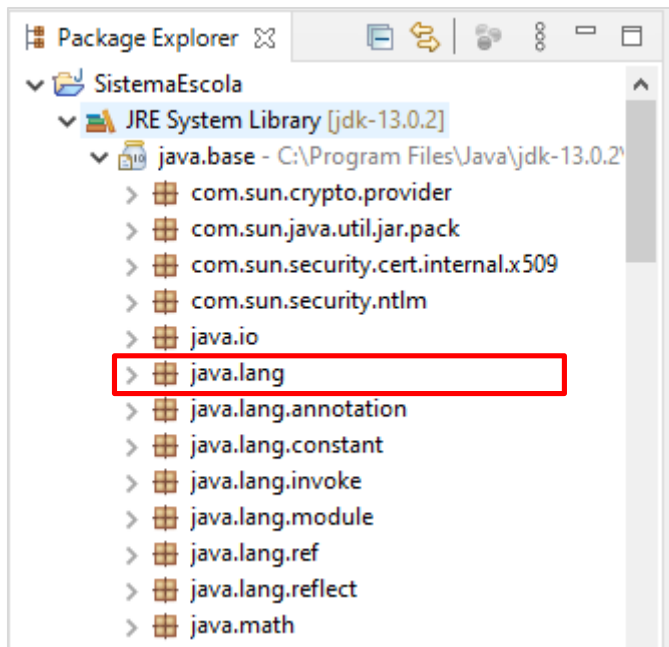
Os pacotes são usados para agrupar classes relacionadas. Desta forma, conseguimos evitar conflitos de nomes entre classes, e também escrever código mais fácil de manter e atualizar. Além disso, os pacotes facilitam a busca ou pesquisa de classes, interfaces e enumerações.



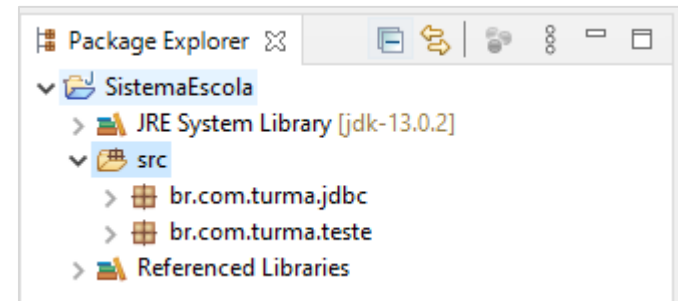
Existem duas categorias de pacotes em Java:

- Pacotes internos (built-in), da API do Java
- Pacotes definidos pelo usuário (criados por você mesmo)

build-in

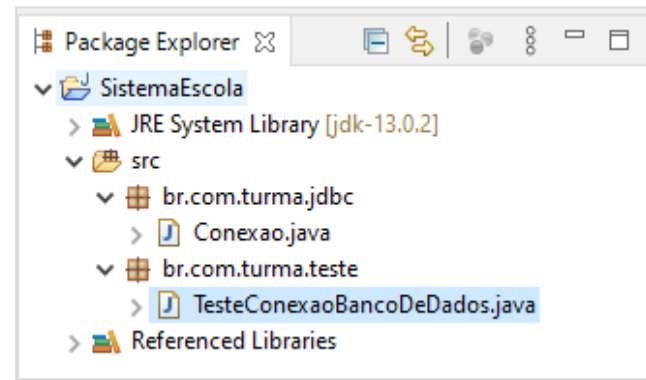


definido pelo usuário



Obs.: As classes do pacote **java.lang** são por padrão importados a classe, portanto não há necessidade de deixá-lo explícito.

Pacotes diferentes:



Sempre que uma classe é invocada, a procura primária é no pacote da classe que esta chamando, caso não esteja, será necessário informar a localização do pacote onde esta a classe invocada através do comando **import**.

Em nosso exemplo, como as classes de teste estão em um pacote a parte, será necessário chamar o comando import caso queira utilizar as classes do pacote br.com.turmas.jdbc.

Essa informação ficará abaixo do nome do pacote e antes do nome da classe.

```
1 package br.com.turma.teste;
2
3 import br.com.turma.jdbc.Conexao;
4
5 public class TesteConexaoBancoDeDados {
6
7     public static void main(String[] args) {
8
9         String msg = Conexao.obterConexao();
10        System.out.println(msg);
11
12    }
13 }
```

**Importando classes em
outros pacotes**

Diferença entre public (+), private (-) e protected (#)

Já entendemos a diferença entre private e public, vamos repassar

private – Os membros da classe não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos.

public - Uma declaração com o modificador public pode ser acessada de qualquer lugar e por qualquer entidade que possa visualizar a classe a que ela pertence.

protected - torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

Entendimento Base sobre Tratamento de Exceções

Aqui um problema maior por falta de atenção:

```
String[] nomes = new String[3];  
nomes[1] = "Maria";  
nomes[2] = "João";  
nomes[3] = "Francisca";  
  
for(int i=0;i<=nomes.length;i++) {  
    System.out.println("Nome: " + nomes[i]);  
}
```

Exception in thread "main" [java.lang.ArrayIndexOutOfBoundsException](#): Index 3 out of bounds for length 3
at br.com.turma.teste.TesteAnalizandoMetodoMain.main([TesteAnalizandoMetodoMain.java:10](#))

Entendimento Base sobre Tratamento de Exceções

Aparentemente um código pode não apresentar problema nenhum, como no exemplo abaixo:

```
int valor1 = 10;
int valor2 = 5;
int resultado = valor1/valor2;

System.out.println("Resultado: " + resultado);
```

A lógica acima funciona perfeitamente, mas uma mudança pode ocasionar problema em tempo de execução do sistema.

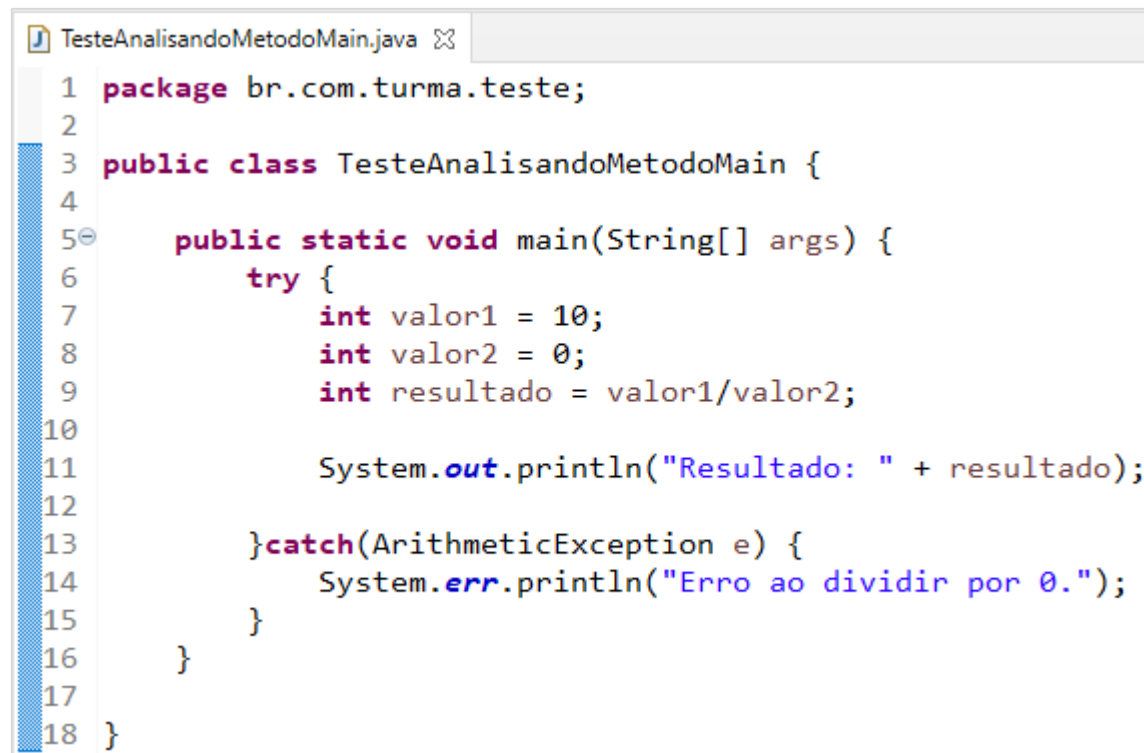
```
int valor1 = 10;
int valor2 = 0;
int resultado = valor1/valor2;

System.out.println("Resultado: " + resultado);
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at br.com.turma.teste.TesteAnalisandoMetodoMain.main(TesteAnalisandoMetodoMain.java:9)
```


Entendimento Base sobre Tratamento de Exceções

Repare que foi utilizado o comando try/catch que trata o problema de exceção (Exception). Por trás das “câmeras”, uma exception é um objeto que “cai” no código provocando uma parada repentina no sistema e por isso deve ser tratado, como apresentado abaixo.

A screenshot of a Java IDE window titled 'TesteAnalizandoMetodoMain.java'. The code is as follows:

```
1 package br.com.turma.teste;
2
3 public class TesteAnalizandoMetodoMain {
4
5     public static void main(String[] args) {
6         try {
7             int valor1 = 10;
8             int valor2 = 0;
9             int resultado = valor1/valor2;
10
11             System.out.println("Resultado: " + resultado);
12
13         } catch (ArithmeticException e) {
14             System.err.println("Erro ao dividir por 0.");
15         }
16     }
17
18 }
```

A vantagem de utilizar listas

Uma das impossibilidades que temos ao trabalhar com vetor (exemplo visto anteriormente), é que em sua declaração colocamos o tamanho do vetor. Isto nos limita, porque se precisarmos de um tamanho maior ou menor teremos dificuldades em redimensiona-lo.

```
String[] nomes = new String[3];  
nomes[0] = "Maria";  
nomes[1] = "João";  
nomes[2] = "Francisca";
```

Exemplo de um vetor com 3 posições. Esta forma dificulta o decremento ou incremento de posições.

Existe uma forma melhor de trabalharmos, este é feito com listas. Faz parte API Java Collection que é uma coleção de classes e interfaces, no entanto veremos apenas como funciona o chamado ArrayList do pacote java.util.

A vantagem de utilizar listas

Usando o mesmo exemplo anterior, precisamos de uma lista de String's. Cada posição terá um nome, podemos percorrer os nomes utilizando o comando for.

```
ArrayList<String> nomes = new ArrayList<String>();
nomes.add("Maria");      //A posição 0 esta o nome Maria
nomes.add("João");       //A posição 1 esta o nome João
nomes.add("Francisca");  //A posição 2 esta o nome Francisca

for(int i=0;i<nomes.size();i++) {
    System.out.println("Nomes: " + nomes.get(i));
}
System.out.println();
//-----ADICIONANDO NOVAS POSIÇÕES NA LISTA -----
nomes.add("Katia");      //A posição 3 agora é preenchida com nome Katia
nomes.add("Lauro");      //A posição 4 agora é preenchida com nome Lauro

for(int i=0;i<nomes.size();i++) {
    System.out.println("Nomes: " + nomes.get(i));
}
System.out.println();
//----- REMOVENDO POSIÇÕES NA LISTA -----
nomes.remove(1);         //Remove a posição 1 da lista (a lista é reorganizada)
for(int i=0;i<nomes.size();i++) {
    System.out.println("Nomes: " + nomes.get(i));
}
```

Resultado:

```
Nomes: Maria
Nomes: João
Nomes: Francisca
```

```
Nomes: Maria
Nomes: João
Nomes: Francisca
Nomes: Katia
Nomes: Lauro
```

```
Nomes: Maria
Nomes: Francisca
Nomes: Katia
Nomes: Lauro
```

A vantagem de utilizar listas

Para terminar veja a diferença entre ambos:

```
String[] nomes = new String[3];
nomes[0] = "Maria";      //A posição 0 esta o nome Maria
nomes[1] = "João";       //A posição 1 esta o nome João
nomes[2] = "Francisca";  //A posição 2 esta o nome Francisca

//PRIMEIRO MODELO PARA FOR
for(int i=0;i<nomes.length;i++) {
    System.out.println("Nome: " + nomes[i]);
}

//SEGUNDO MODELO PARA FOR (CHAMADO DE FOREACH)
for(String nome:nomes) {
    System.out.println("Nome: " + nome);
}
```

Utilizando o vetor que **NÃO** possibilita o incremento e decremento de posições.

```
ArrayList<String> nomes = new ArrayList<String>();
nomes.add("Maria");      //A posição 0 esta o nome Maria
nomes.add("João");       //A posição 1 esta o nome João
nomes.add("Francisca");  //A posição 2 esta o nome Francisca

//PRIMEIRO MODELO PARA FOR
for(int i=0;i<nomes.size();i++) {
    System.out.println("Nome: " + nomes.get(i));
}

//SEGUNDO MODELO PARA FOR (CHAMADO DE FOREACH)
for(String nome:nomes) {
    System.out.println("Nome: " + nome);
}
```

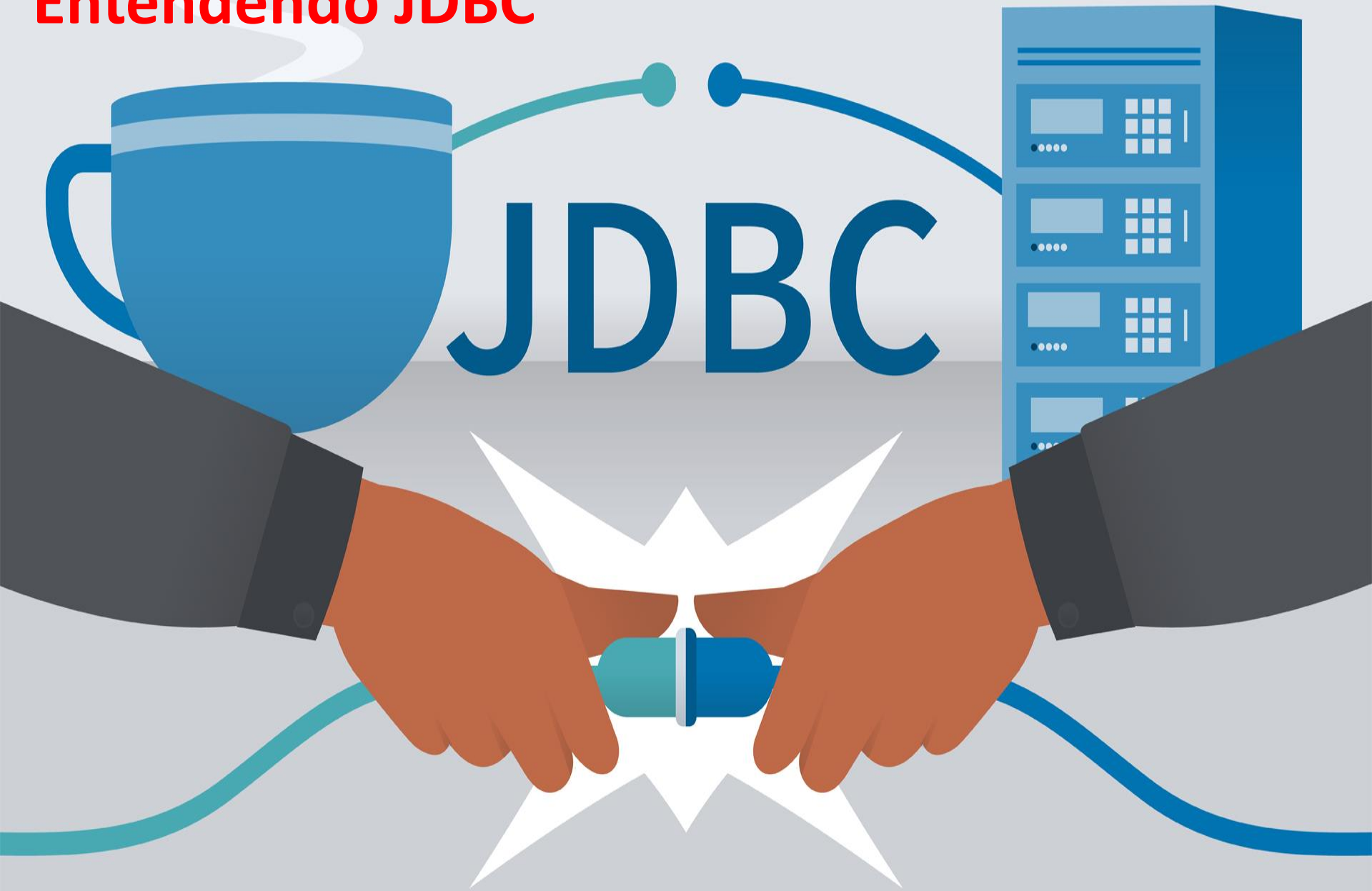
Utilizando o vetor que possibilita o incremento e decremento de posições.

JDBC

Java Database Connectivity

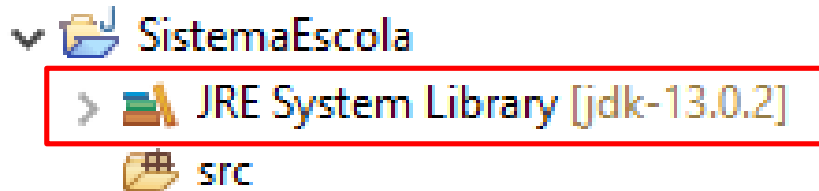
Java Database Connectivity ou **JDBC** é um conjunto de classes e interfaces escritas em Java que fazem o envio de instruções SQL para qualquer banco de dados relacional;

Entendendo JDBC
























Entendendo JDBC

Ao criar um projeto JAVA, as API's que fazem parte da biblioteca do JAVA são inseridas no projeto em virtude da instalação anterior do JDK (Java Development Kit). Na organização destas biblioteca podemos encontrar API's responsáveis por: leitura e gravação em arquivos, conexão com banco de dados, manipulação de arquivos XML e etc.



Entendendo JDBC

A API de conexão com o banco de dados encontra-se em java.sql

- ▼ SistemaEscola
 - ▼ JRE System Library [jdk-13.0.2]
 - >  java.base - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.compiler - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.datatransfer - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.desktop - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.instrument - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.logging - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.management - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.management.rmi - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.naming - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.net.http - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.prefs - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.rmi - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.scripting - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.se - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.security.jgss - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.security.sasl - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  **java.sql - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar**
 - >  java.sql.rowset - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.transaction.xa - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.xml - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar
 - >  java.xml.crypto - C:\Program Files\Java\jdk-13.0.2\lib\jrt-fs.jar

Entendendo JDBC

Projeto Java: SistemaEscola

Driver
JDBC
Oracle

Driver
JDBC
MySQL

Driver
JDBC
Postgre

Biblioteca JAVA

JDBC
(Java Database
Connectivity)



Baixando e adicionando o driver JDBC

driver jdbc mysql

Todas Videos Imagens Notícias Shopping

Aproximadamente 3.950.000 resultados (0,42 segundos)

MySQL Connector/J - MySQL :: Developer Zone

dev.mysql.com > downloads > connector > 5.1.html Traduzir

MySQL Connector/J is the official JDBC driver for MySQL. MySQL Connector/J is compatible with all MySQL versions starting with MySQL 5.6. Additional

Neste caso foi baixado um arquivo ZIP, descompacte este arquivo retirando o arquivo **mysql-connector-java-5.1.49.jar** Coloque em uma pasta qualquer em seu sistema operacional.

dev.mysql.com/downloads/connector/j/5.1.html

MySQL Community Downloads

Connector/J

General Availability (GA) Releases

Archives

Connector/J 5.1.49

Select Operating System:

Platform Independent

Looking for the latest GA version?

Recommended Windows Download:

MySQL Installer
for Windows

All MySQL Products. For All Windows Platforms.
In One Package.



Starting with MySQL 5.6 the MySQL Installer package replaces the standalone MSI packages.

Windows (x86, 32 & 64-bit), MySQL Installer MSI

Go to Download Page >

Platform Independent (Architecture Independent),
Compressed TAR Archive

(mysql-connector-java-5.1.49.tar.gz)

5.1.49

3.2M

Download

MD5: e7bc11a55398bad0ea8548163deabaa8 | Signature

Platform Independent (Architecture Independent), ZIP
Archive

(mysql-connector-java-5.1.49.zip)

5.1.49

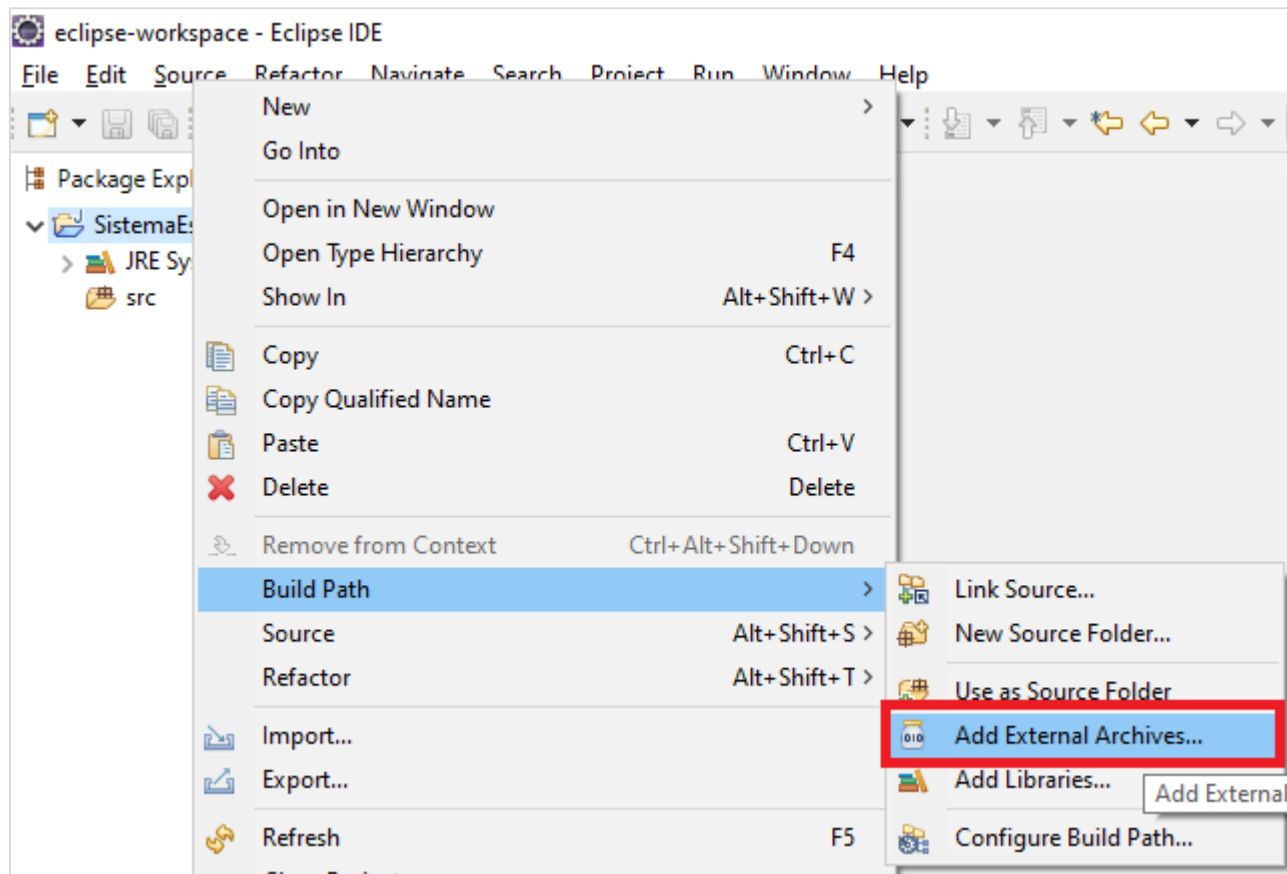
3.5M

Download

MD5: 5ecd588e13f14de07faa5c67f5ca43f1 | Signature

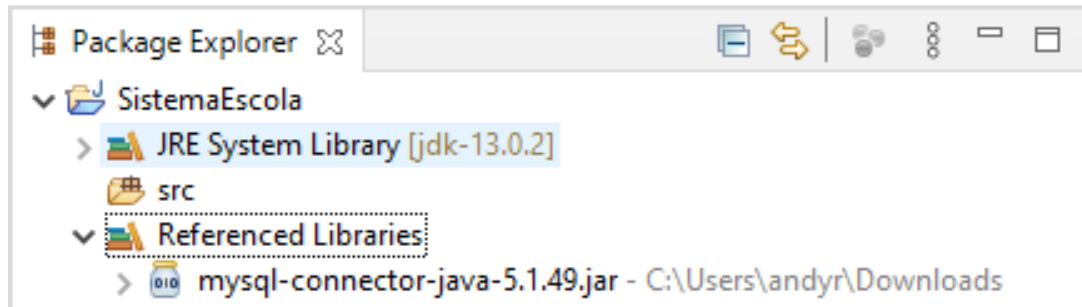
Baixando e adicionando o driver JDBC

Clique com o botão direito do mouse sobre seu projeto, vá até Add External Archives e selecione o arquivo descompactado: **mysql-connector-java-5.1.49.jar**



Baixando e adicionando o driver JDBC

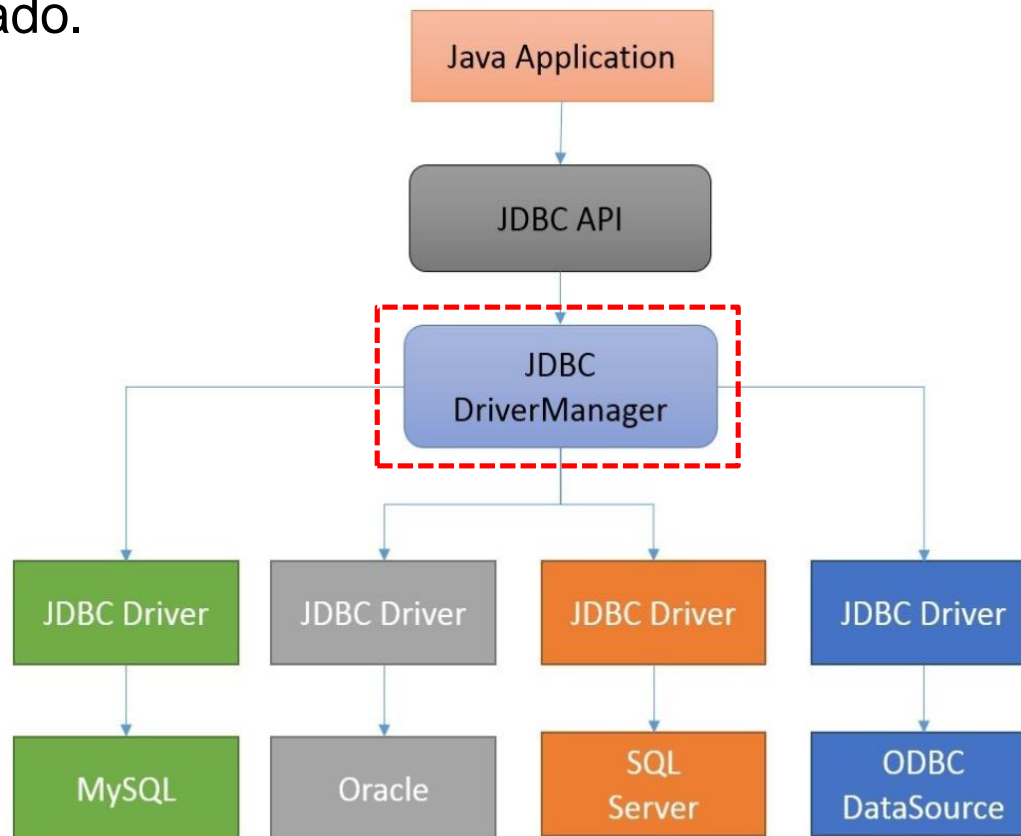
Após adicioná-lo, verifique que o projeto será apresentado da seguinte forma no Eclipse:



Agora podemos entender como este driver é gerenciado através de um componente chamado **DriverManager**.

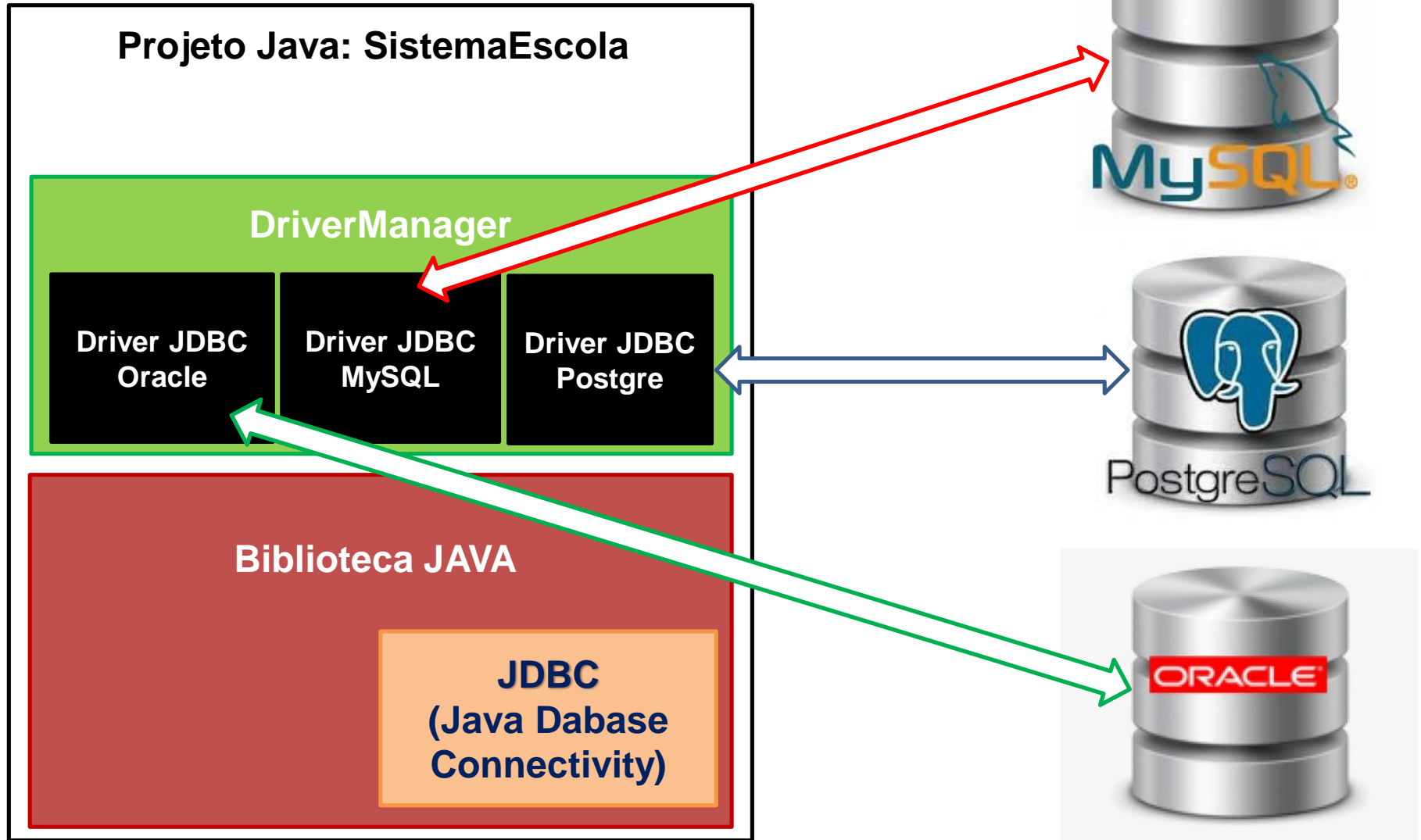
Conhecendo o DriverManager e Testando a Conexão ao BD

DriverManager gerencia uma lista de drivers de banco de dados. Combina a requisição de conexão da aplicação JAVA, com o driver de banco de dados apropriado.



JDBC Architecture

Conhecendo o DriverManager e Testando a Conexão ao BD



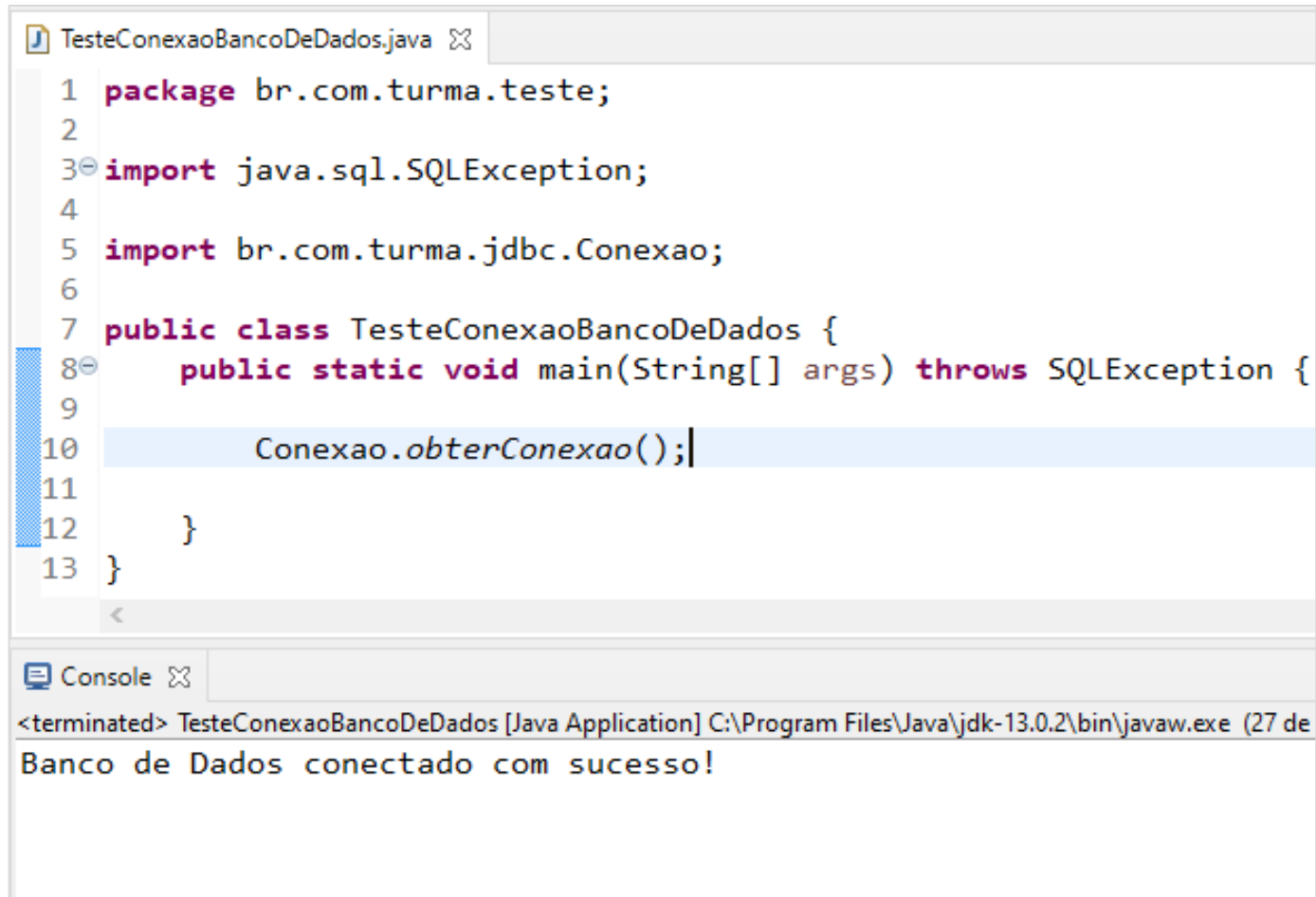
Conhecendo o DriverManager e Testando a Conexão ao BD

Para colocar a teoria em prática, criaremos uma classe chamada `Conexao` e dentro dele um método chamado `obterConexao()`. A finalidade deste método é receber uma conexão permitindo a comunicação entre a aplicação e o banco de dados. Esta é a razão do tipo de dado para retorno ser `Connection`. Se o retorno for nulo, significa que não foi possível estabelecer uma conexão com o banco de dados.

```
Conexao.java x
1 package br.com.turma.jdbc;
2
3 import java.sql.Connection;
4 import java.sql.DriverManager;
5 import java.sql.SQLException;
6
7 public class Conexao {
8
9     public static Connection obterConexao(){
10
11         Connection con = null;
12         try {
13             con = DriverManager.getConnection("jdbc:mysql://localhost/dbalunos?useSSL=false","root","root");
14             System.out.println("Banco de Dados conectado com sucesso!");
15         } catch (SQLException e) {
16             System.err.println("Não foi possível conectar ao banco de dados");
17             e.printStackTrace();
18         }
19         return con;
20     }
21 }
22
23 }
```

Conhecendo o DriverManager e Testando a Conexão ao BD

Vamos criar uma classe para testar a conexão com o banco de dados.



```
TesteConexaoBancoDeDados.java
1 package br.com.turma.teste;
2
3 import java.sql.SQLException;
4
5 import br.com.turma.jdbc.Conexao;
6
7 public class TesteConexaoBancoDeDados {
8     public static void main(String[] args) throws SQLException {
9
10         Conexao.obterConexao();
11
12     }
13 }
```

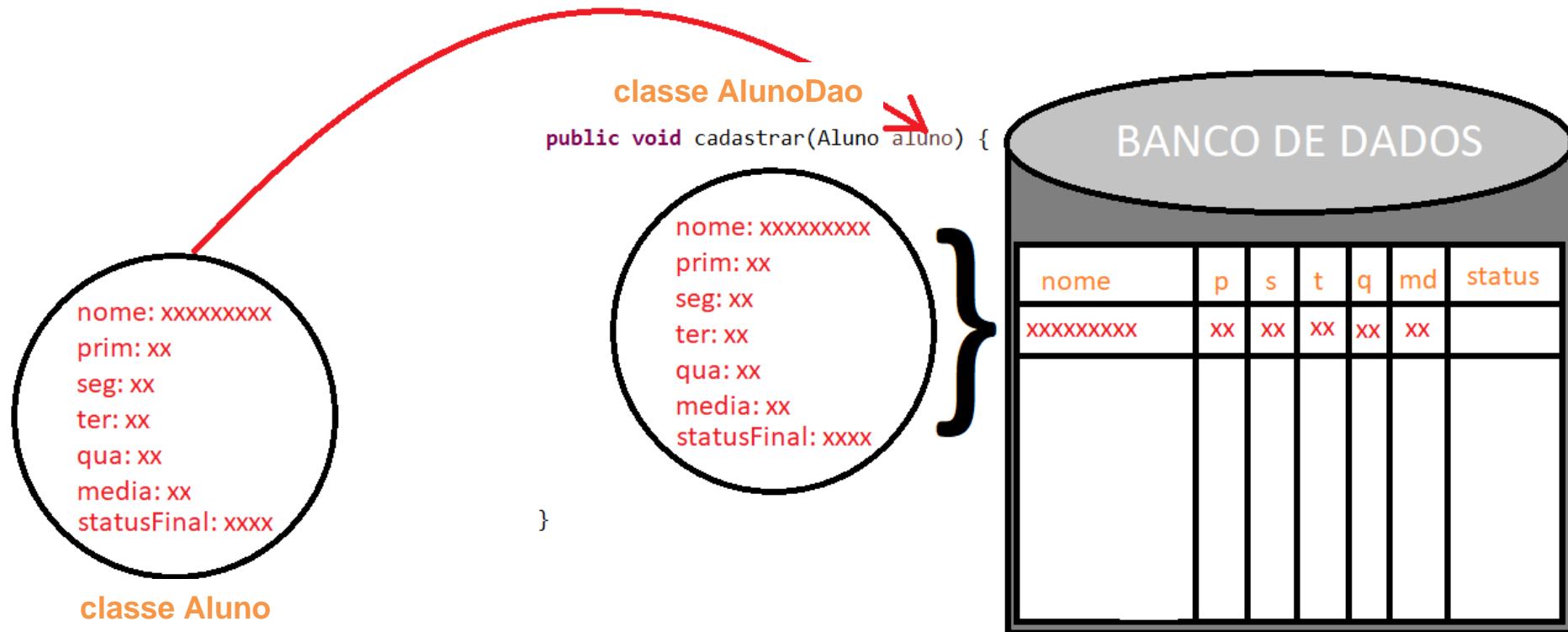
Console

<terminated> TesteConexaoBancoDeDados [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (27 de Banco de Dados conectado com sucesso!

**Inserindo Dados no Banco de
Dados a partir da aplicação**

Inserindo Dados no Banco de Dados a partir da aplicação

A inserção dos dados dentro de um banco de dados consiste em receber um objeto com todos os dados necessários completos. Enviá-lo para um método que terá a capacidade de resgatar todos os dados nos atributos do objeto e gravá-los na tabela dentro do banco de dados.



Inserindo Dados no Banco de Dados a partir da aplicação

Criando a classe Aluno

Aluno.java

```
1 package br.com.turma.entidade;
2
3 public class Aluno {
4
5     //Construtor
6     public Aluno(String nome, double prim, double seg, double ter, double qua) {
7         this.nome = nome;
8         this.prim = prim;
9         this.seg = seg;
10        this.ter = ter;
11        this.qua = qua;
12        this.media = this.calcularMedia();
13        this.statusFinal = this.verificarStatus();
14    }
15
16    //Propriedades ou Atributos
17    private int codigo;
18    private String nome;
19    private double prim;
20    private double seg;
21    private double ter;
22    private double qua;
23    private double media;
24    private String statusFinal;
25
26    //Método para calcular a média
27    public double calcularMedia() {
28        return (this.prim + this.seg + this.ter + this.qua)/2;
29    }
30
31    //Método para verificar o status do aluno
32    public String verificarStatus() {
33
34        if(this.media >= 7 && this.media <= 10) {
35            return "APROVADO";
36        }else if(this.media >= 5 && this.media < 7) {
37            return "RECUPERAÇÃO";
38        }else if(this.media >= 0 && this.media < 5) {
39            return "REPROVADO";
40        }else {
41            return "NOTA INVÁLIDA";
42        }
43    }
44 }
```

```
46    //Métodos Getters
47    public int getCodigo() {
48        return codigo;
49    }
50    public String getNome() {
51        return nome;
52    }
53    public double getPrim() {
54        return prim;
55    }
56    public double getSeg() {
57        return seg;
58    }
59    public double getTer() {
60        return ter;
61    }
62    public double getQua() {
63        return qua;
64    }
65    public double getMedia() {
66        return media;
67    }
68    public String getStatusFinal() {
69        return statusFinal;
70    }
71
72
73 }
```

Inserindo Dados no Banco de Dados a partir da aplicação

- 1) Obtém uma conexão com o banco de dados
- 2) Cria a String que será utilizada para ser executada no banco de dados.
- 3) Cria um objeto com o objetivo de preparar a linha de instrução (statement) antes de ser executado no banco de dados.
- 4) Esta preparação consiste em alterar os pontos de interrogação por valores que estão dentro do objeto.
- 5) Após a preparação executar o comando no banco de dados e fechar a conexão com o banco de dados.

Inserindo Dados no Banco de Dados a partir da aplicação

Criando a classe AlunoDao

```
AlunoDao.java ✕
1 package br.com.turma.dao;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.SQLException;
6
7 import br.com.turma.entidade.Aluno;
8 import br.com.turma.jdbc.Conexao;
9
10 public class AlunoDao {
11
12     public void cadastrar(Aluno aluno) {
13
14         Connection con = Conexao.obterConexao();
15
16         String sql = "INSERT INTO aluno(nome,prim,seg,ter,qua,media,status_final) VALUES(?,?,?,?,?,?,?)";
17
18         try {
19             PreparedStatement preparador = con.prepareStatement(sql);
20             preparador.setString(1, aluno.getNome());
21             preparador.setDouble(2, aluno.getPrim());
22             preparador.setDouble(3, aluno.getSeg());
23             preparador.setDouble(4, aluno.getTer());
24             preparador.setDouble(5, aluno.getQua());
25             preparador.setDouble(6, aluno.getMedia());
26             preparador.setString(7, aluno.getStatusFinal());
27
28             preparador.execute();
29
30             preparador.close();
31
32             System.out.println("O aluno foi cadastrado com sucesso!");
33
34         } catch (SQLException e) {
35             System.err.println("Erro ao cadastrar o aluno!");
36             e.printStackTrace();
37         }
38     }
39 }
40
41 }
```

Inserindo Dados no Banco de Dados a partir da aplicação

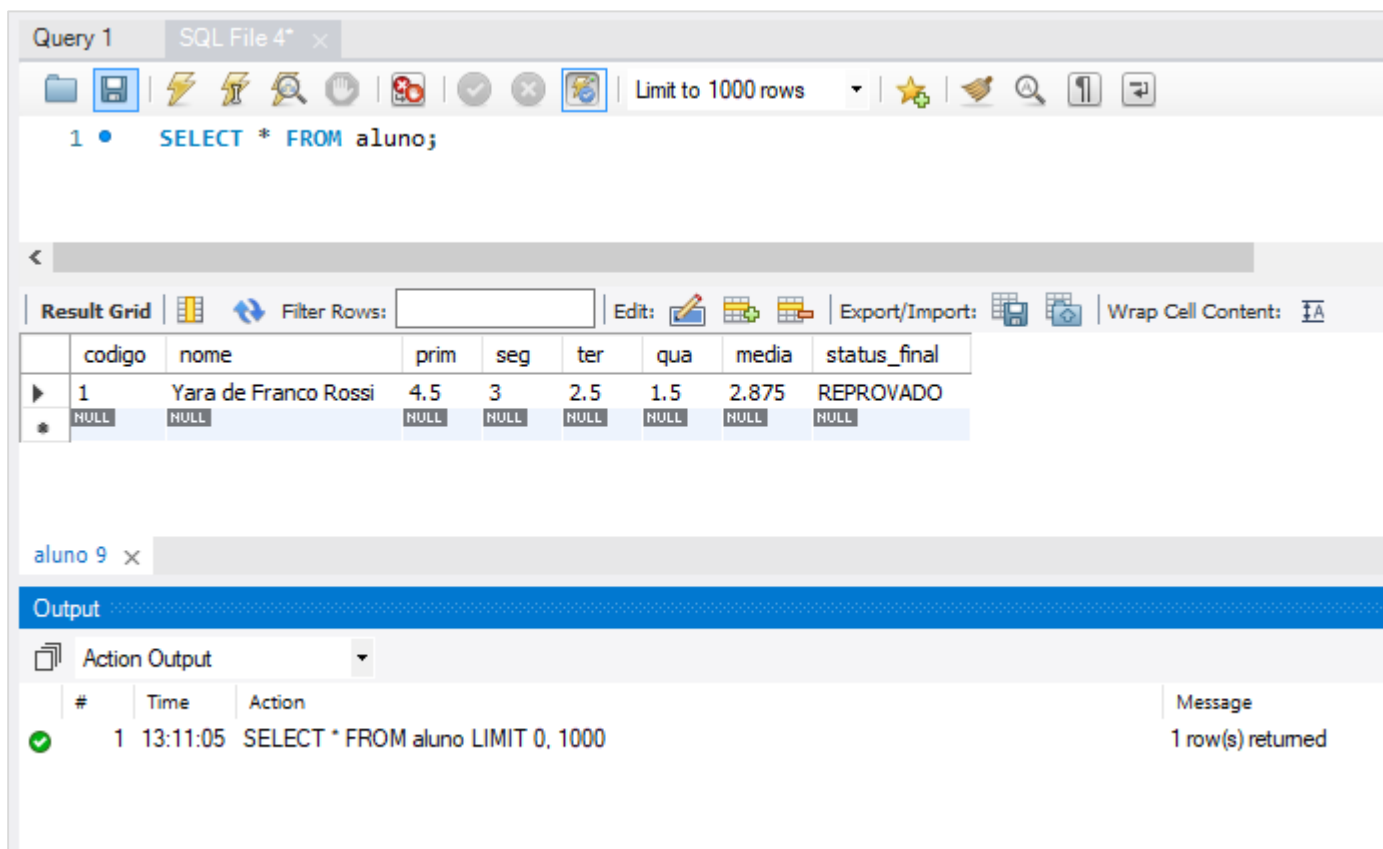
Criando a classe para Teste

```
TesteCadastrarAluno.java
1 package br.com.turma.teste;
2
3 import br.com.turma.dao.AlunoDao;
4 import br.com.turma.entidade.Aluno;
5
6 public class TesteCadastrarAluno {
7
8     public static void main(String[] args) {
9
10         Aluno aluno01 = new Aluno("Yara de Franco Rossi",4.5,3.0,2.5,1.5);
11
12         AlunoDao alunoDao = new AlunoDao();
13         alunoDao.cadastrar(aluno01);
14
15     }
16 }
17
18
```

```
Console
<terminated> TesteCadastrarAluno [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (27 de set de 2020 13:10:08 - 13:10:09)
Banco de Dados conectado com sucesso!
0 aluno foi cadastrado com sucesso!
```

Inserindo Dados no Banco de Dados a partir da aplicação

Confirmando no banco de dados



The screenshot displays a SQL IDE interface. At the top, a tab labeled 'Query 1' shows the SQL query: `SELECT * FROM aluno;`. Below the query editor, a 'Result Grid' shows the query results. The first row contains data for a student named 'Yara de Franco Rossi' with various scores and a status of 'REPROVADO'. The second row is a header row with all cells set to 'NULL'. Below the result grid, a tab labeled 'aluno 9' is visible. At the bottom, an 'Output' panel shows the execution log. The log entry indicates that the query `SELECT * FROM aluno LIMIT 0, 1000` was executed at 13:11:05, returning 1 row(s).

Query 1 SQL File 4* x

Limit to 1000 rows

1 • `SELECT * FROM aluno;`

Result Grid

	codigo	nome	prim	seg	ter	qua	media	status_final
▶	1	Yara de Franco Rossi	4.5	3	2.5	1.5	2.875	REPROVADO
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

aluno 9 x

Output

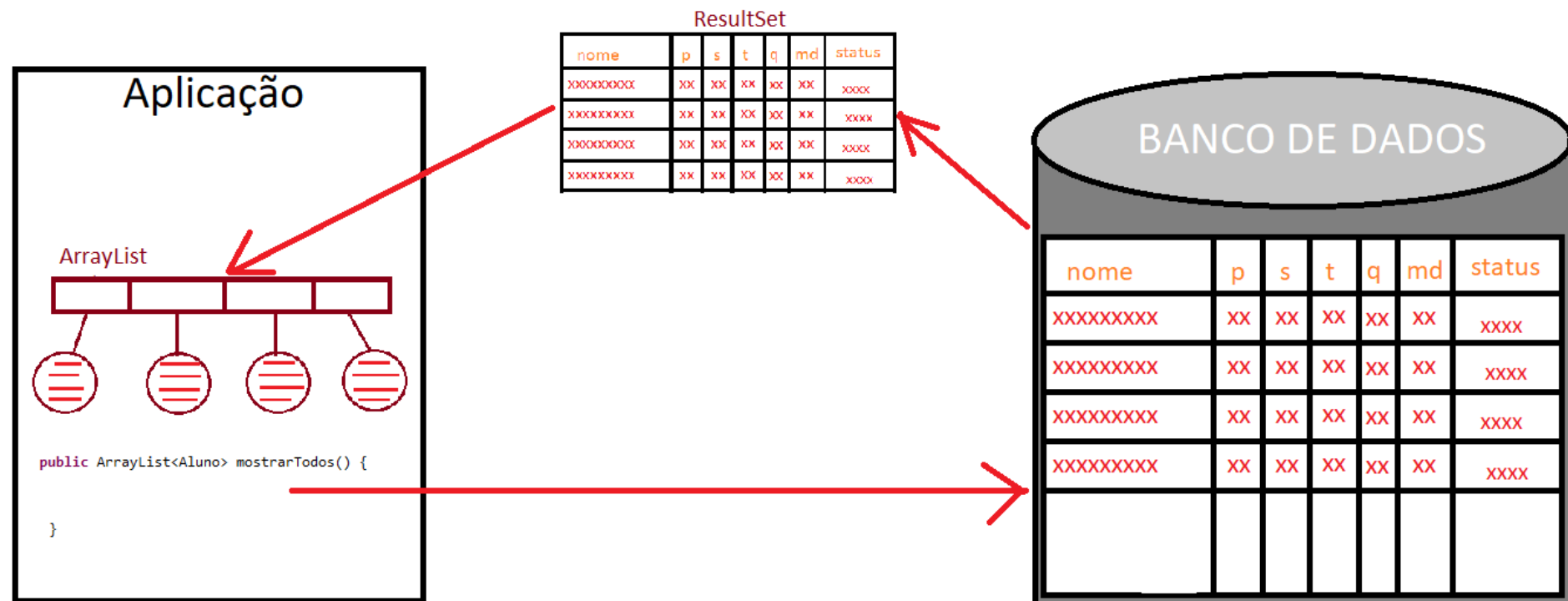
Action Output

#	Time	Action	Message
✓ 1	13:11:05	<code>SELECT * FROM aluno LIMIT 0, 1000</code>	1 row(s) returned

**Mostrando os Dados do Banco
de Dados para a aplicação**

Mostrando os Dados do Banco de Dados para a aplicação

Mostrar os dados da tabela de um banco de dados consiste em selecionar os dados desejados e enviá-lo a uma tabela intermediária chamada de `ResultSet`. Acessando os dados desta tabela intermediária é possível resgatar seus dados e coloca-los dentro de uma lista (`ArrayList`).



Mostrando os Dados do Banco de Dados para a aplicação

- 1) Obtém uma conexão com o banco de dados
- 2) Cria a String que será utilizada para ser executada no banco de dados.
- 3) Cria um objeto com o objetivo de preparar a linha de instrução (statement) antes de ser executado no banco de dados.
- 4) Após a preparação executar o comando no banco de dados. O comando retornará a resposta da pesquisa colocando-o em um objeto do tipo ResultSet.
- 5) Cada linha do ResultSet será colocado em um objeto da classe de sua entidade (no exemplo: Aluno), e cada um destes objetos serão colocados dentro do vetor.
- 6) O retorno será este vetor com os vários objetos.

Mostrando os Dados do Banco de Dados para a aplicação

Coloque este método dentro da classe AlunoDao

```
public ArrayList<Aluno> mostrarTodos() {  
  
    Connection con = Conexao.obterConexao();  
  
    String sql = "SELECT * FROM aluno";  
  
    try {  
        PreparedStatement preparador = con.prepareStatement(sql);  
        ResultSet resultado = preparador.executeQuery();  
  
        while(resultado.next()) {  
            Aluno aluno = new Aluno();  
            aluno.setCodigo(resultado.getInt("codigo"));  
            aluno.setNome(resultado.getString("nome"));  
            aluno.setPrim(resultado.getDouble("prim"));  
            aluno.setSeg(resultado.getDouble("seg"));  
            aluno.setTer(resultado.getDouble("ter"));  
            aluno.setQua(resultado.getDouble("qua"));  
            aluno.setMedia(resultado.getDouble("media"));  
            aluno.setStatusFinal(resultado.getNString("status_final"));  
  
            alunos.add(aluno);  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
  
    return alunos;  
}
```

Mostrando os Dados do Banco de Dados para a aplicação

Crie o método abaixo no pacote teste

```
TesteMostrarTodosAlunos.java
1 package br.com.turma.teste;
2
3 import java.util.ArrayList;
4
5 import br.com.turma.dao.AlunoDao;
6 import br.com.turma.entidade.Aluno;
7
8 public class TesteMostrarTodosAlunos {
9
10     public static void main(String[] args) {
11
12         AlunoDao alunoDao = new AlunoDao();
13         ArrayList<Aluno> alunosDoBanco = alunoDao.mostrarTodos();
14
15         for(int i=0;i<alunosDoBanco.size();i++) {
16
17             Aluno aluno = alunosDoBanco.get(i);
18
19             System.out.println("Código: " + aluno.getCodigo());
20             System.out.println("Nome: " + aluno.getNome());
21             System.out.printf("1º Bim: %.1f - 2º Bim: %.1f - 3º Bim: %.1f - 4º Bim: %.1f\n", aluno.getPrim(),aluno.getSeg(),aluno.getTer(),aluno.getQua());
22             System.out.printf("Média: %.1f ", aluno.getMedia());
23             System.out.println("Status: " + aluno.getStatusFinal());
24             System.out.println();
25
26         }
27     }
28 }
29
30 }
```

Mostrando os Dados do Banco de Dados para a aplicação

Coloque este método dentro da classe AlunoDao (busca pelo código)

```
public Aluno mostrarPorCodigo(int codigo) {  
  
    Connection con = Conexao.obterConexao();  
    Aluno aluno = null;  
  
    String sql = "SELECT * FROM aluno WHERE codigo=?";  
  
    try {  
        PreparedStatement preparador = con.prepareStatement(sql);  
        preparador.setInt(1, codigo);  
        ResultSet resultado = preparador.executeQuery();  
  
        if(resultado.next()) {  
            aluno = new Aluno();  
            aluno.setCodigo(resultado.getInt("codigo"));  
            aluno.setNome(resultado.getString("nome"));  
            aluno.setPrim(resultado.getDouble("prim"));  
            aluno.setSeg(resultado.getDouble("seg"));  
            aluno.setTer(resultado.getDouble("ter"));  
            aluno.setQua(resultado.getDouble("qua"));  
            aluno.setMedia(resultado.getDouble("media"));  
            aluno.setStatusFinal(resultado.getString("status_final"));  
        }  
  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
  
    return aluno;  
}
```

Este método retornará somente
um objeto do tipo Aluno

Mostrando os Dados do Banco de Dados para a aplicação

Crie este outro método no pacote teste

Este método retornará somente um objeto do tipo Aluno

```
1 package br.com.turma.teste;
2
3 import br.com.turma.dao.AlunoDao;
4 import br.com.turma.entidade.Aluno;
5
6 public class TesteMostrarAlunoPorCodigo {
7
8     public static void main(String[] args) {
9
10         AlunoDao alunoDao = new AlunoDao();
11         Aluno aluno = alunoDao.mostrarPorCodigo(5);
12
13         System.out.println("Código: " + aluno.getCodigo());
14         System.out.println("Nome: " + aluno.getNome());
15         System.out.printf("1º Bim: %.1f - 2º Bim: %.1f - 3º Bim: %.1f - 4º Bim: %.1f\n", aluno.getPrim(), aluno.getSeg(), aluno.getTer(), aluno.getQua());
16         System.out.printf("Média: %.1f ", aluno.getMedia());
17         System.out.println("Status: " + aluno.getStatusFinal());
18         System.out.println();
19     }
20 }
```

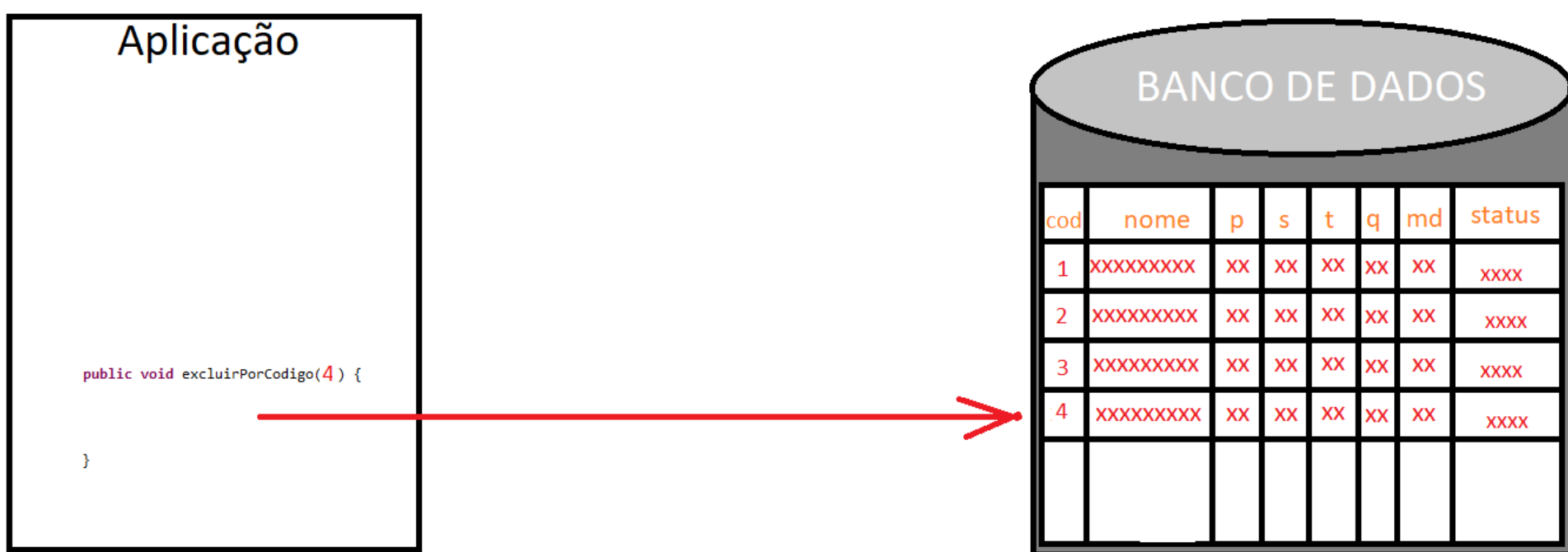
<terminated> TesteMostrarAlunoPorCodigo [Java Application] C:\Program Files\Java\jdk-13.0.2\bin\javaw.exe (27 de set de 2020 17:16:28 – 17:16:29)

Banco de Dados conectado com sucesso!
Código: 5
Nome: Robson Siqueira
1º Bim: 3,0 - 2º Bim: 3,0 - 3º Bim: 8,0 - 4º Bim: 8,5
Média: 5,6 Status: RECUPERAÇÃO

Excluindo os Dados do Banco de Dados

Excluindo os Dados do Banco de Dados

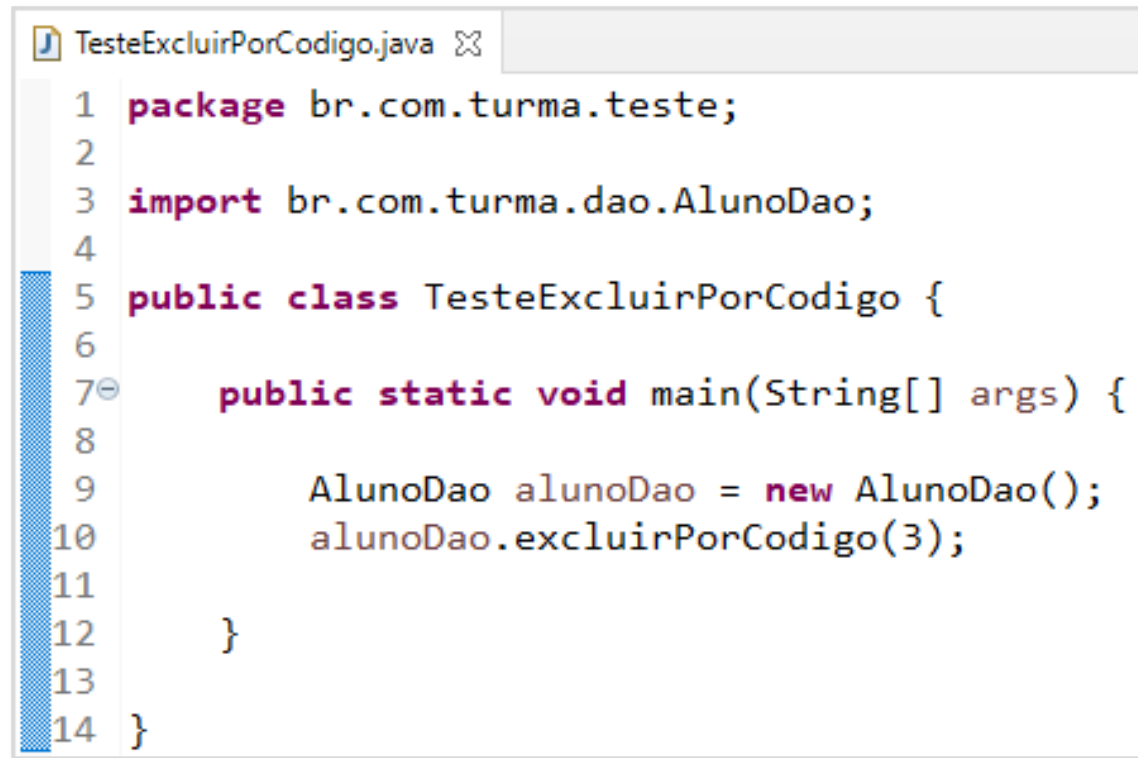
O método a ser criado terá a responsabilidade de receber código do aluno a ser excluído e coloca-lo no comando SQL responsável pela exclusão.



Excluindo os Dados do Banco de Dados

```
public void excluirPorCodigo(int codigo) {  
  
    Connection con = Conexao.obterConexao();  
  
    String sql = "DELETE FROM aluno WHERE codigo=?";  
  
    try {  
        PreparedStatement preparador = con.prepareStatement(sql);  
        preparador.setInt(1, codigo);  
  
        preparador.execute();  
        preparador.close();  
  
        System.out.println("Aluno excluído com sucesso!!!");  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
  
}
```

Excluindo os Dados do Banco de Dados



```
TesteExcluirPorCodigo.java ✕
1 package br.com.turma.teste;
2
3 import br.com.turma.dao.AlunoDao;
4
5 public class TesteExcluirPorCodigo {
6
7     public static void main(String[] args) {
8
9         AlunoDao alunoDao = new AlunoDao();
10        alunoDao.excluirPorCodigo(3);
11
12    }
13
14 }
```

Verifique no banco de dados se o aluno com o código descrito foi realmente excluído

Alterando Dados do Banco de Dados

Alterando Dados do Banco de Dados

Este método receberá um objeto do tipo Aluno que conterá todos os dados que serão alterados no banco de dados. O objeto deverá conter o código que será alterado no banco de dados

```
public void alterarPorCodigo(Aluno aluno) {  
  
    Connection con = Conexao.obterConexao();  
  
    String sql = "UPDATE aluno SET nome=?,prim=?,seg=?,ter=?,qua=?,media=?,status_final=? WHERE codigo=?";  
  
    try {  
        PreparedStatement preparador = con.prepareStatement(sql);  
        preparador.setString(1, aluno.getNome());  
        preparador.setDouble(2, aluno.getPrim());  
        preparador.setDouble(3, aluno.getSeg());  
        preparador.setDouble(4, aluno.getTer());  
        preparador.setDouble(5, aluno.getQua());  
        preparador.setDouble(6, aluno.getMedia());  
        preparador.setString(7, aluno.getStatusFinal());  
        preparador.setInt(8, aluno.getCodigo());  
  
        preparador.execute();  
        preparador.close();  
  
        System.out.println("O aluno foi alterado com sucesso!");  
    } catch (SQLException e) {  
        System.err.println("Erro ao alterar o aluno!");  
        e.printStackTrace();  
    }  
}
```

Alterando Dados do Banco de Dados

```
TesteAlterarPorCodigo.java
1 package br.com.turma.teste;
2
3 import br.com.turma.dao.AlunoDao;
4 import br.com.turma.entidade.Aluno;
5
6 public class TesteAlterarPorCodigo {
7
8     public static void main(String[] args) {
9
10
11         Aluno aluno01 = new Aluno();
12         aluno01.setCodigo(5);
13         aluno01.setNome("Nelson Meirelles");
14         aluno01.setPrim(6.5);
15         aluno01.setSeg(9.5);
16         aluno01.setTer(8.5);
17         aluno01.setQua(7.5);
18         aluno01.setMedia(aluno01.calcularMedia());
19         aluno01.setStatusFinal(aluno01.verificarStatus());
20
21         AlunoDao alunoDao = new AlunoDao();
22         alunoDao.alterarPorCodigo(aluno01);
23
24     }
25
26 }
```

DÚVIDAS

