



UNIVERSIDADE FEDERAL DE ITAJUBÁ

Banco de Dados II

COM 231

Conectividade com Banco de Dados
JDBC
Aula 8

Vanessa Cristina Oliveira de Souza



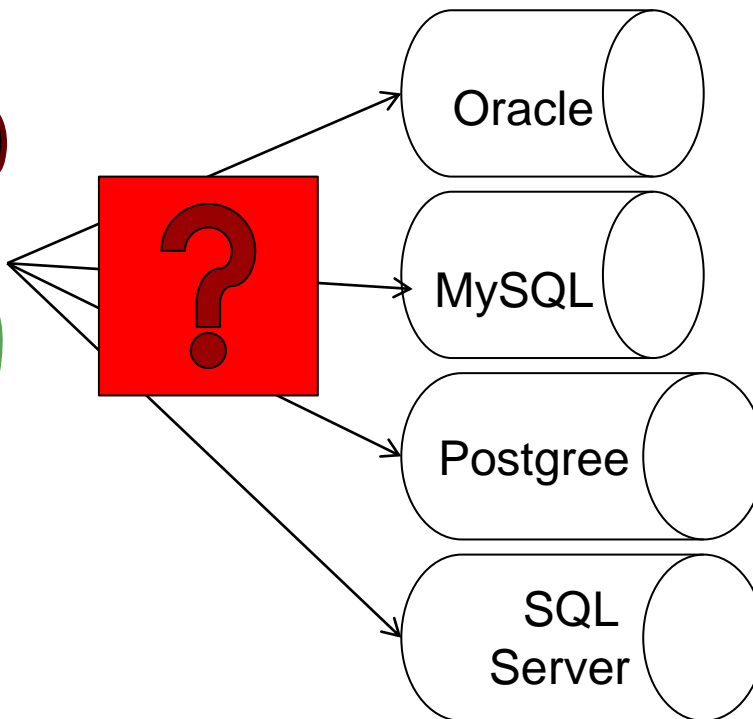
Cenário

Aplicação Cliente

- C
- C++
- PHP
- JAVA
- ...



Dados Servidor



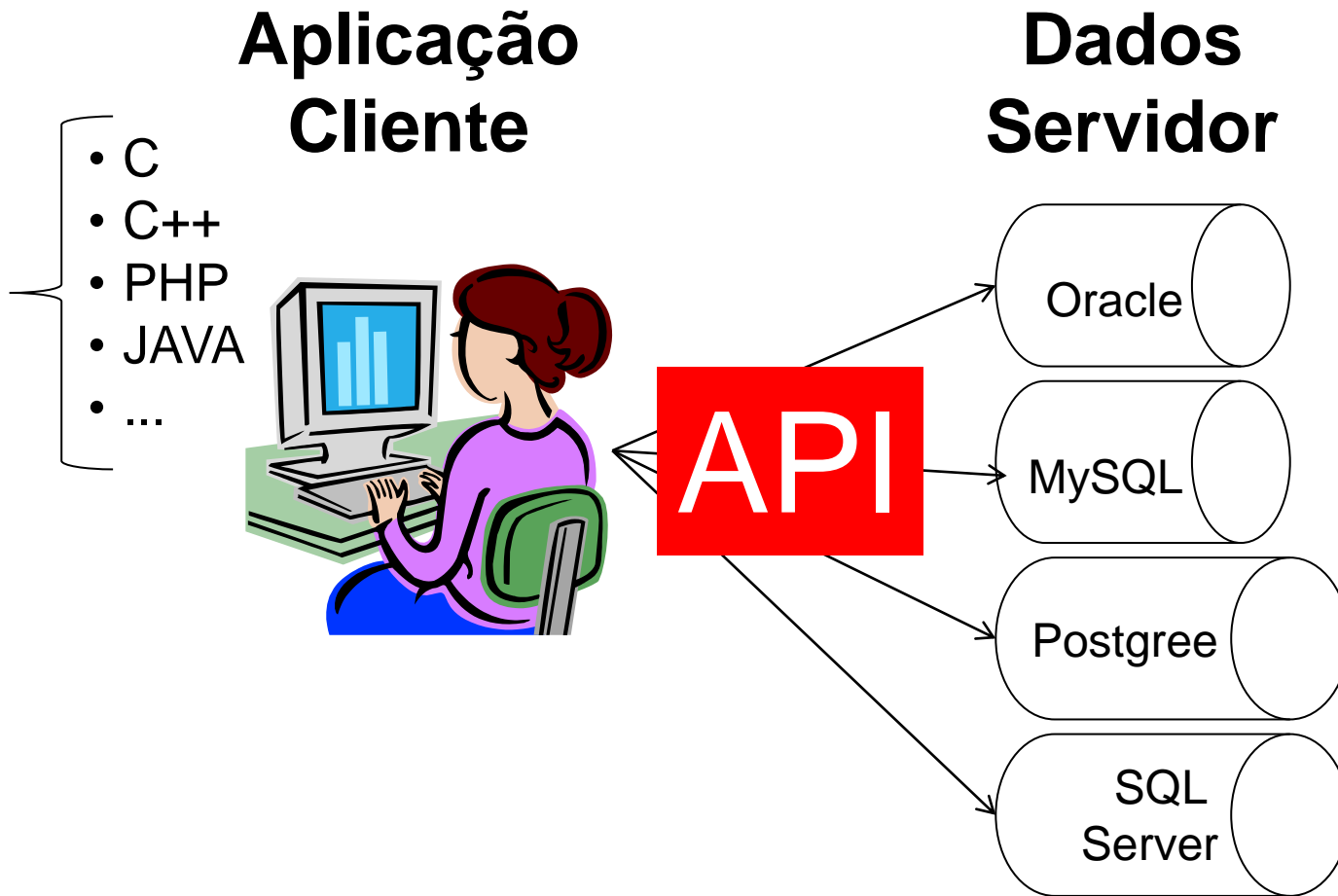


Introdução

- Os programas da interface com o usuário e os programas de aplicação podem ser executados no lado do cliente.
- Quando é necessário acessar o SGBD a partir de uma interface com o usuário, o programa estabelece uma conexão com o SGBD.
- Nesse caso, uma interface de programação (API) é necessária porque existem diversos tipos de bancos de dados que podem ser usados e, preferencialmente, utiliza-se uma única linguagem para comunicar com todos eles.



Cenário



As chamadas aos BDs são padronizadas.



ODBC

- Um padrão denominado ODBC (*Open Database Connectivity*) oferece uma interface de programação de aplicações (API) amplamente aceita para acessar banco de dados.
- Ele é baseado nas especificações da ***Call-Level Interface (CLI)***. Implementado em C e usa SQL como linguagem de acesso aos SGBDs.

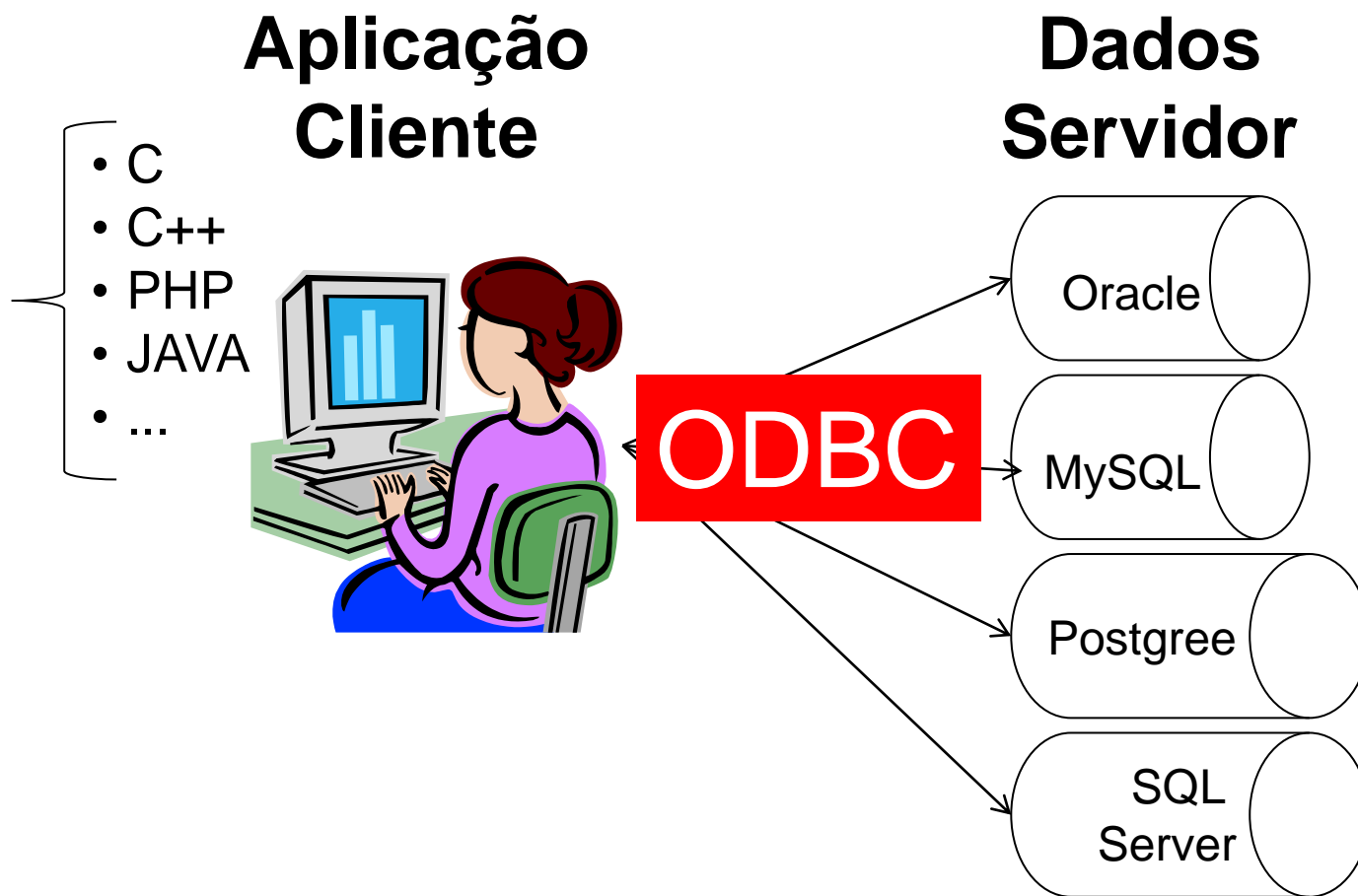


ODBC

- O ODBC permite que os programas do cliente chamem o SGBD, desde que as máquinas cliente e servidor tenham o *software* necessário instalado.
- Um programa cliente pode se conectar a vários SGBDs e enviar solicitações de consulta e transação usando a API da ODBC, que são processadas nos servidores.



Cenário ODBC



Para cada SGBD há um driver ODBC compatível.



ODBC

■ Desvantagens

- ☐ A aplicação fica presa à uma única plataforma
- ☐ Difícil de aprender
- ☐ Requer instalação em cada cliente



JDBC

- *Java Database Connectivity*
- Conjunto de classes e interfaces (API), desenvolvido pela Sun, escritas em Java para acessar qualquer banco de dados relacional
- API de baixo nível e base para APIs de alto nível
- JDBC é o ODBC traduzido numa linguagem orientada objeto de alto nível.



Cenário JDBC

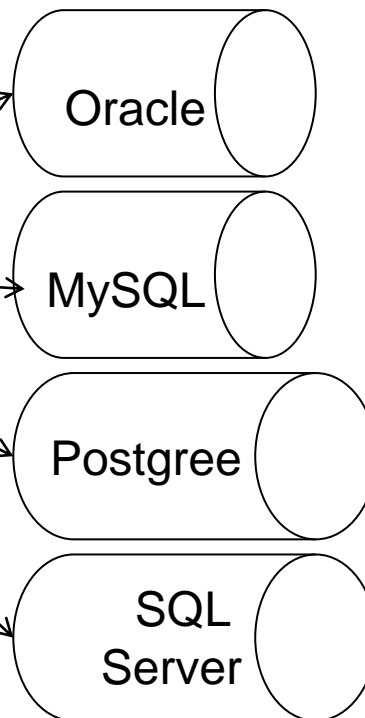
Aplicação Cliente

- JAVA



JDBC

Dados Servidor



Para cada SGBD há um driver JDBC compatível.



ODBC x JDBC

- Pode-se utilizar ODBC em aplicações Java, mas não é aconselhável, pois:
 - ODBC é escrito em C
 - Chamadas de aplicações JAVA para códigos C nativos acarretam num grande número de inconvenientes na segurança, implementação, robustez, e portabilidade das aplicações.
 - Traduções literais do ODBC dentro de um código JAVA não são possíveis, já que as linguagens são muito diferentes (ex. C usa ponteiro)



ODBC x JDBC

- JDBC é mais amigável do que o ODBC
- JDBC puro é portátil
- A versão 2.0 do JDBC já traz compatibilidade com o novo padrão SQL-1999 (Objeto-relacional), não suportado pelo ODBC.

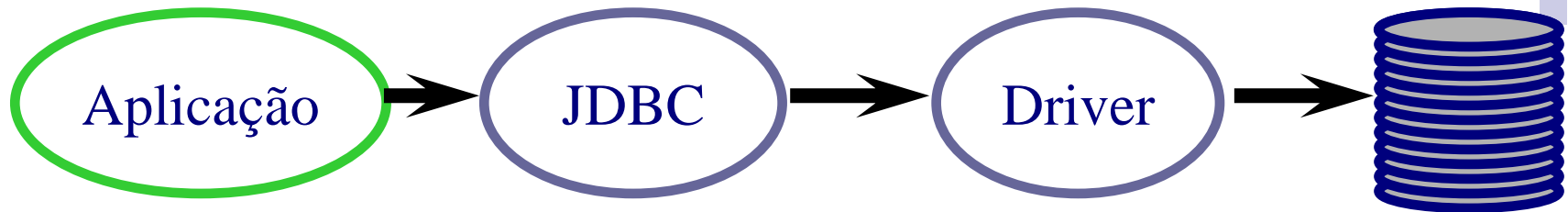


JDBC

- A biblioteca da JDBC provê um conjunto de interfaces de acesso ao BD.
- Uma implementação em particular dessas interfaces é chamada de ***driver***.
- Os próprios fabricantes dos bancos de dados (ou terceiros) são quem implementam os *drivers* JDBC para cada BD, pois são eles que conhecem detalhes dos BDs.
- Cada BD possui um *driver* JDBC específico (que é usado de forma padrão - JDBC).



Arquitetura JDBC



- Código Java chama biblioteca JDBC
- JDBC carrega um *driver*
- Driver conversa com um banco de dados
- Na mesma aplicação pode haver mais de um driver
 - mais do que um banco de dados
- Ideal: mudar o banco de dados sem mudar o código da aplicação



UNIVERSIDADE FEDERAL DE ITAJUBÁ

TIPOS DE DRIVERS

JDBC



Driver Ponte JDBC-ODBC

- É o tipo mais simples.
- Restrito à plataforma Windows.
- Utiliza ODBC para conectar-se com o banco de dados, convertendo métodos JDBC em chamadas às funções do ODBC.
- Esta ponte é normalmente usada quando não há um driver puro-Java para determinado banco de dados, pois seu uso é desencorajado devido à dependência de plataforma.



Driver Ponte JDBC-ODBC

- O JDK é distribuído com um driver desse tipo;



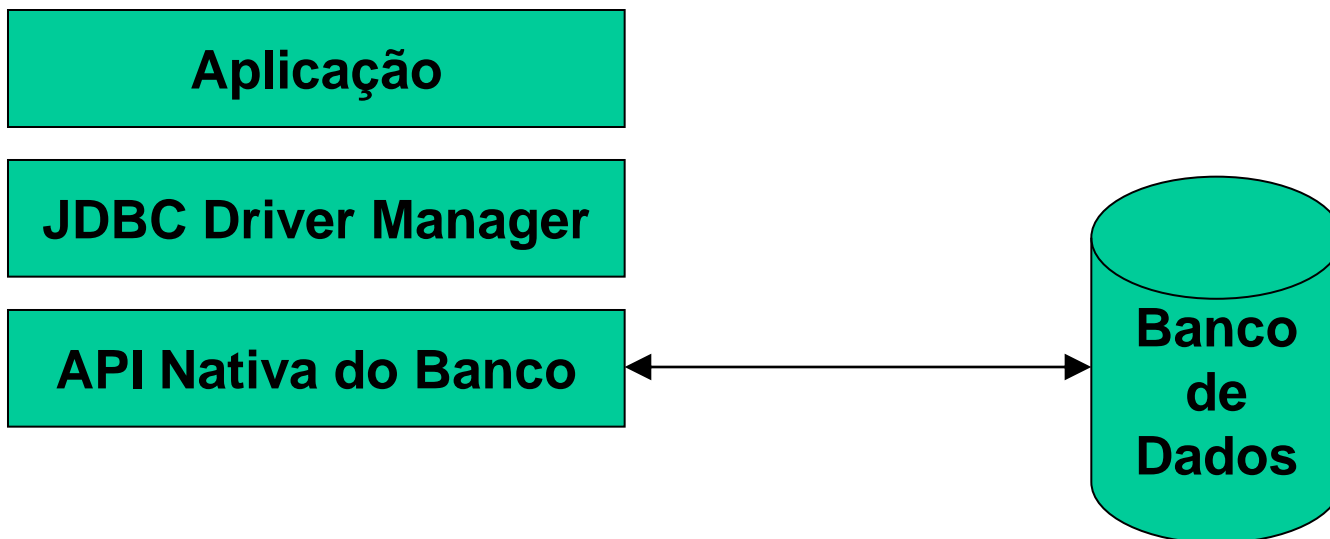


Driver API-Nativa Parcialmente Java

- Traduz as chamadas JDBC para as chamadas da API cliente do banco de dados usado.
- Como a Ponte JDBC-ODBC, pode precisar de *software* extra instalado na máquina cliente.



Driver API-Nativa Parcialmente Java



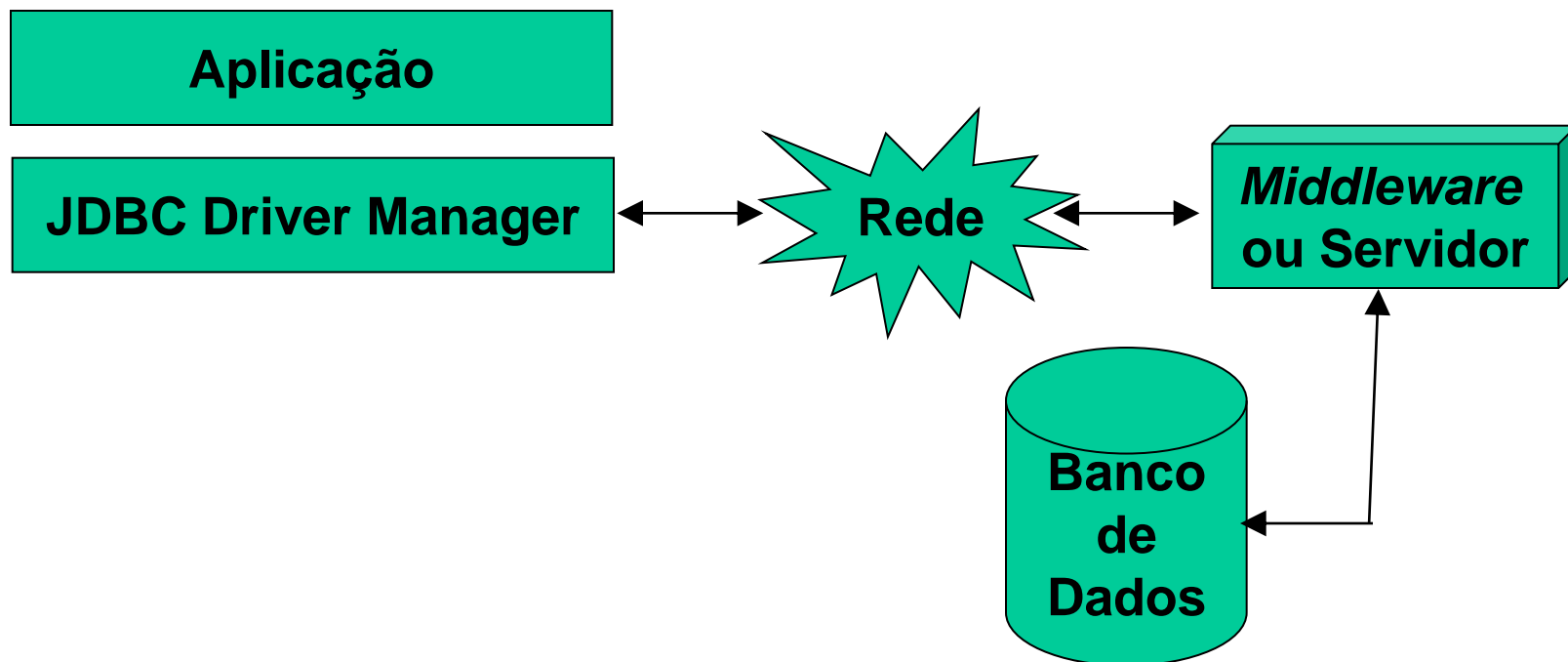


Driver Java c/ Net-Protocol

- Utiliza um *middleware* para a conexão com o banco de dados.
- Não há necessidade de configuração da máquina cliente.
- O *middleware* converte a chamada de alto nível da API na chamada ao SGBD;
- O *middleware* é capaz de conectar aplicações Java com diversos SGBDs.



Driver Java c/ Net-Protocol



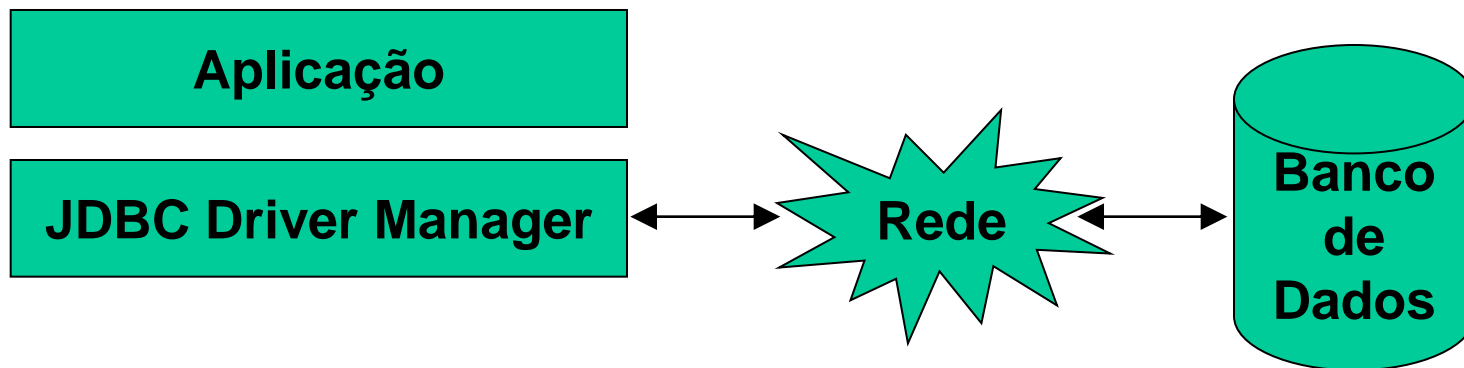


Driver Java Puro

- Acesso direto ao servidor utilizando-se o protocolo do próprio SGBD.
- Conhece todo o protocolo de comunicação com o BD e pode acessar o BD sem software extra.
- É o tipo de *driver* mais indicado.

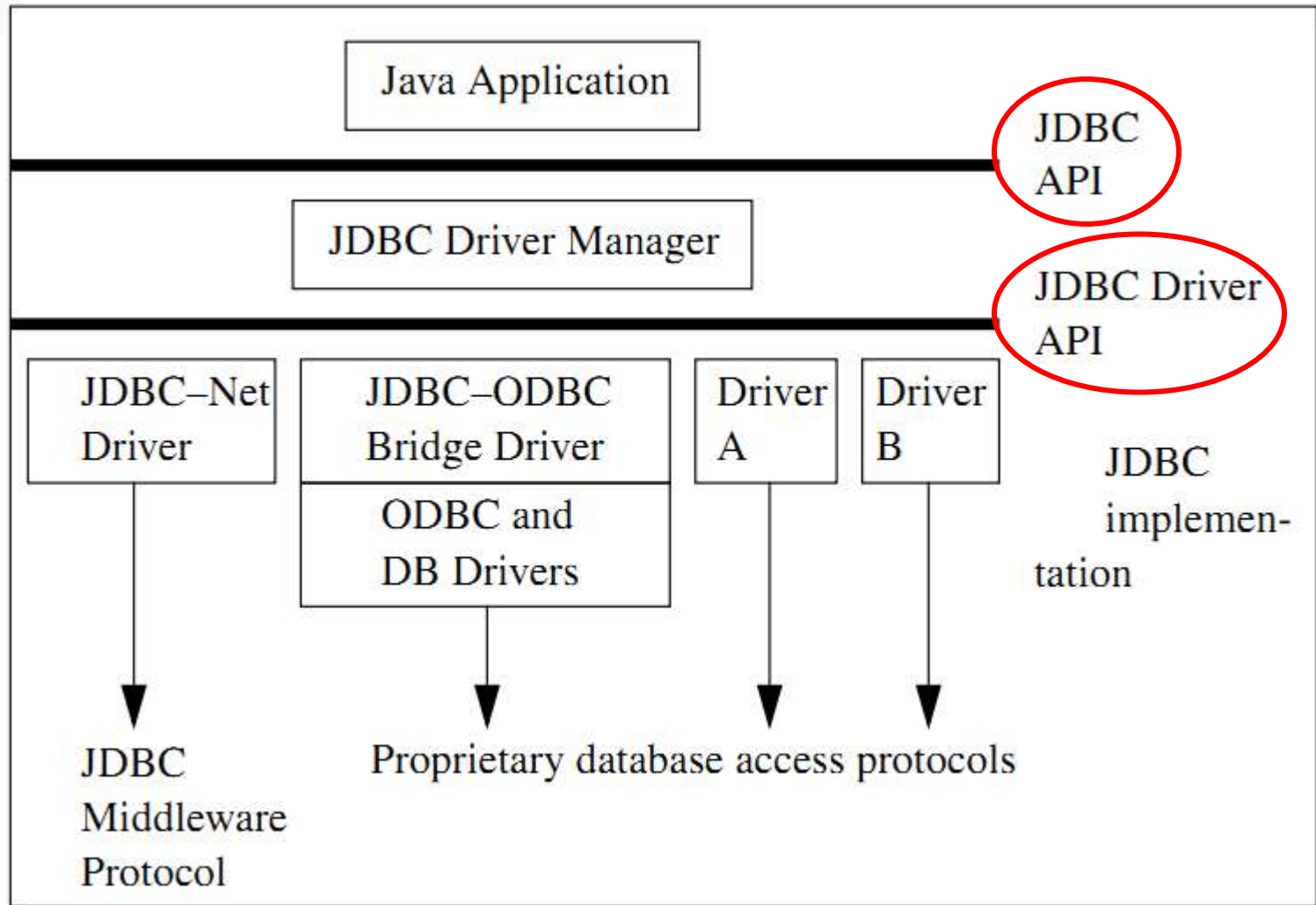


Driver Java Puro





Drivers JDBC





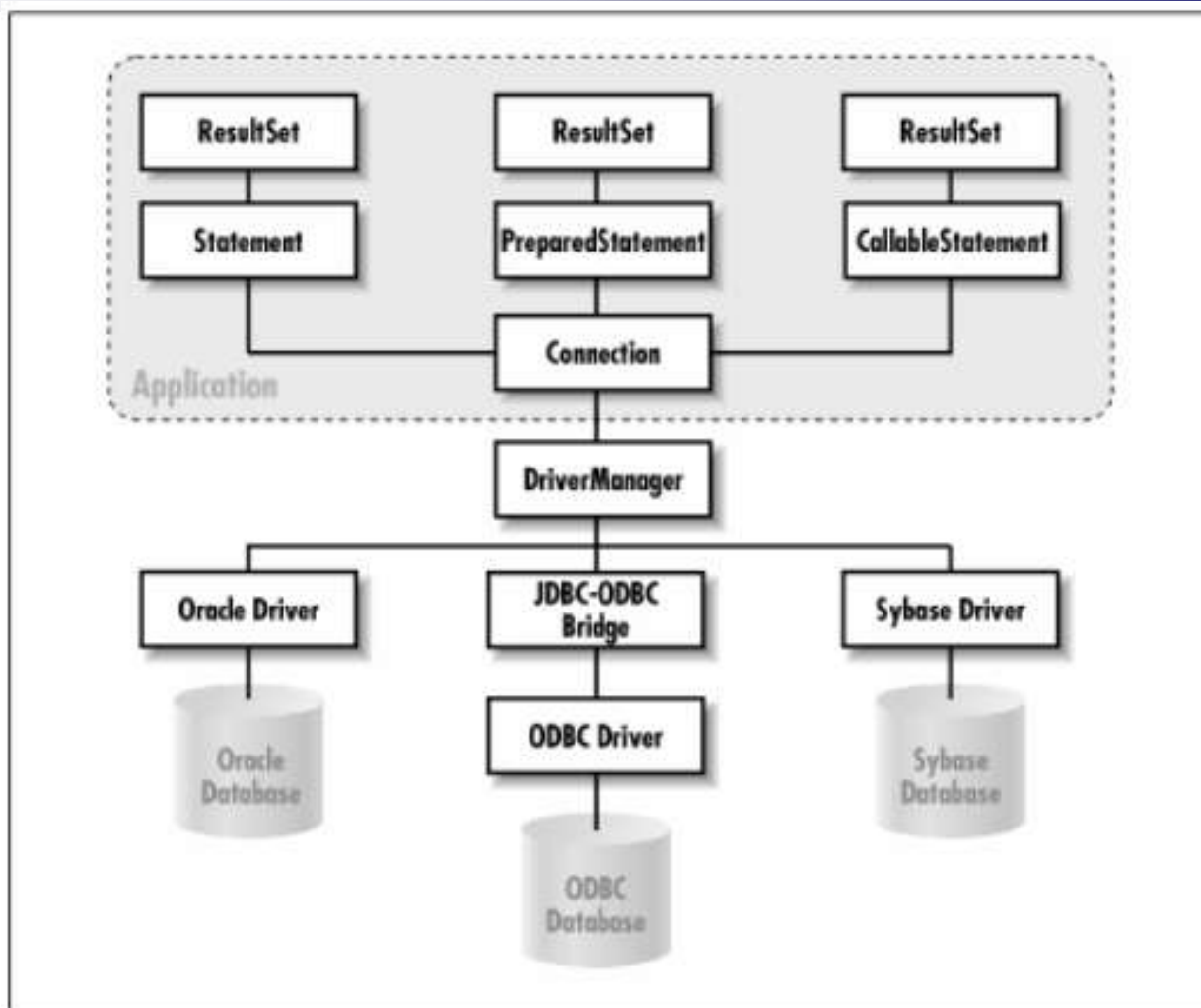
UNIVERSIDADE FEDERAL DE ITAJUBÁ

Arquitetura da JDBC



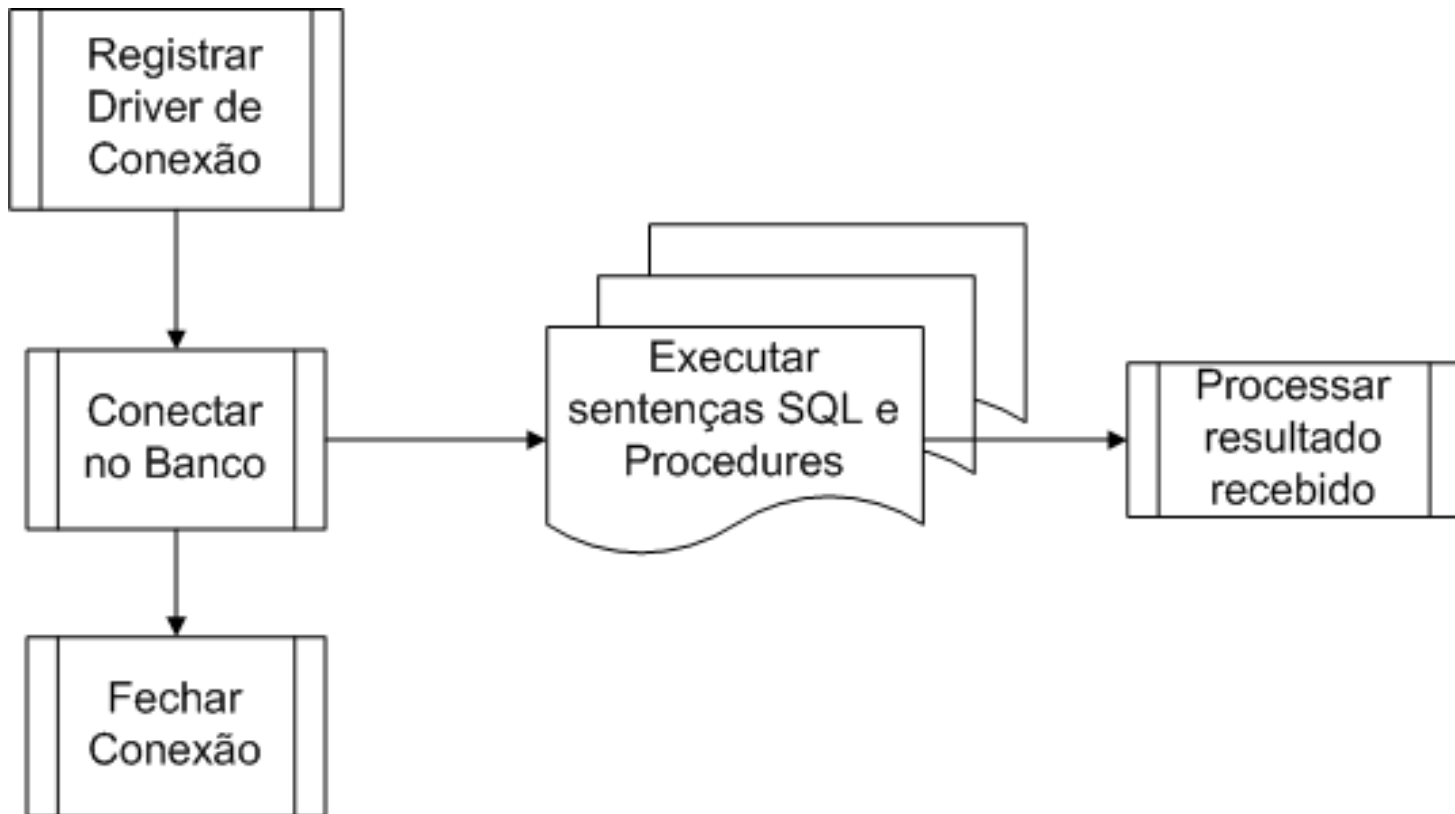
Arquitetura da JDBC

Visão Geral





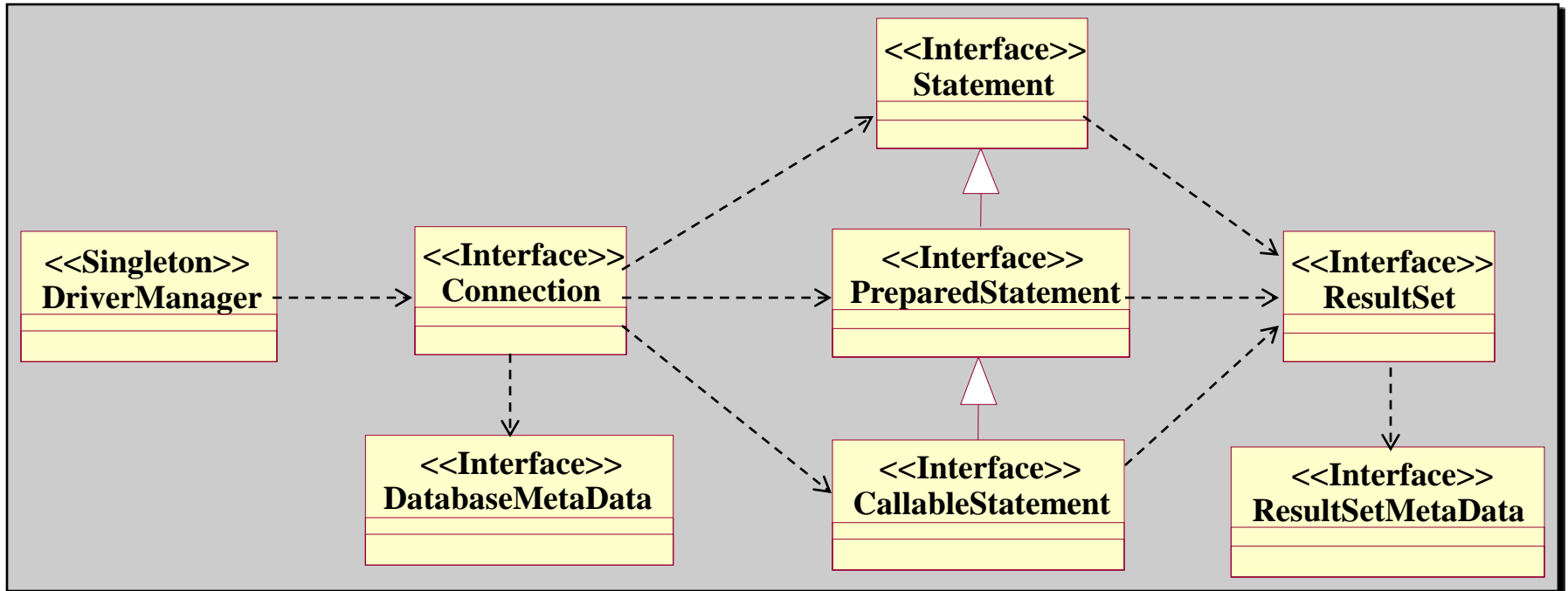
Passos JDBC





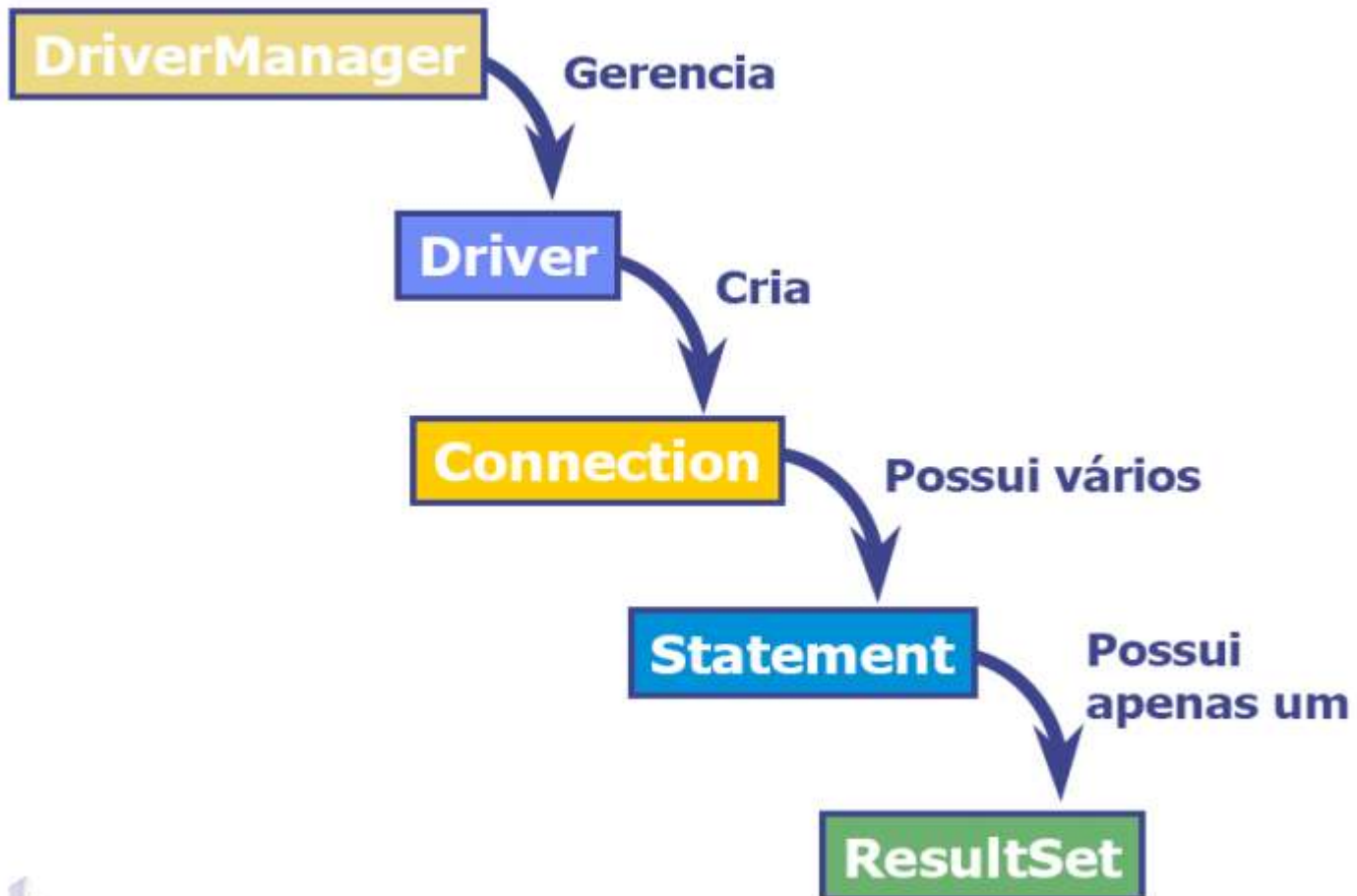
Classes JDBC

- `java.sql.*` provê um conjunto de interfaces para serem usadas pelas aplicações





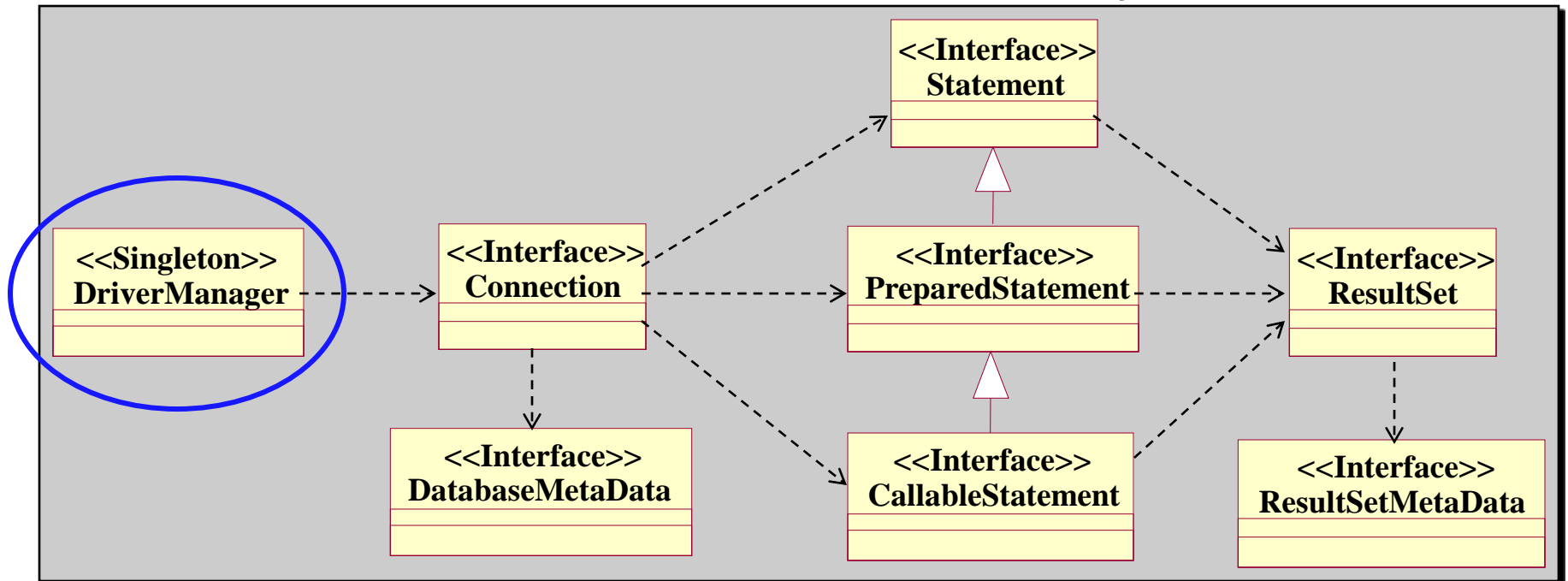
Hierarquia JDBC





Classes JDBC

- `java.sql.*` provê um conjunto de interfaces para serem usadas pelas aplicações





Registro do Driver

- Usar a classe `driveManager` para realizar uma conexão com o banco envolve dois passos:
 - ☐ Carregar o driver
 - ☐ Fazer a conexão

- O *driver* é um jar (se for um driver puro java), e você deve tê-lo em seu *classpath*.



Registro do Driver

- Existem algumas formas de carregar o *driver* na JVM:
 - Via driver direto
 - `com.mysql.jdbc.Driver Driver = new com.mysql.jdbc.Driver();`
 - Via class `ForName`
 - Mais indicado
 - Mais utilizado
 - **`Class.forName ("com.mysql.jdbc.Driver");`**



Registro do Driver

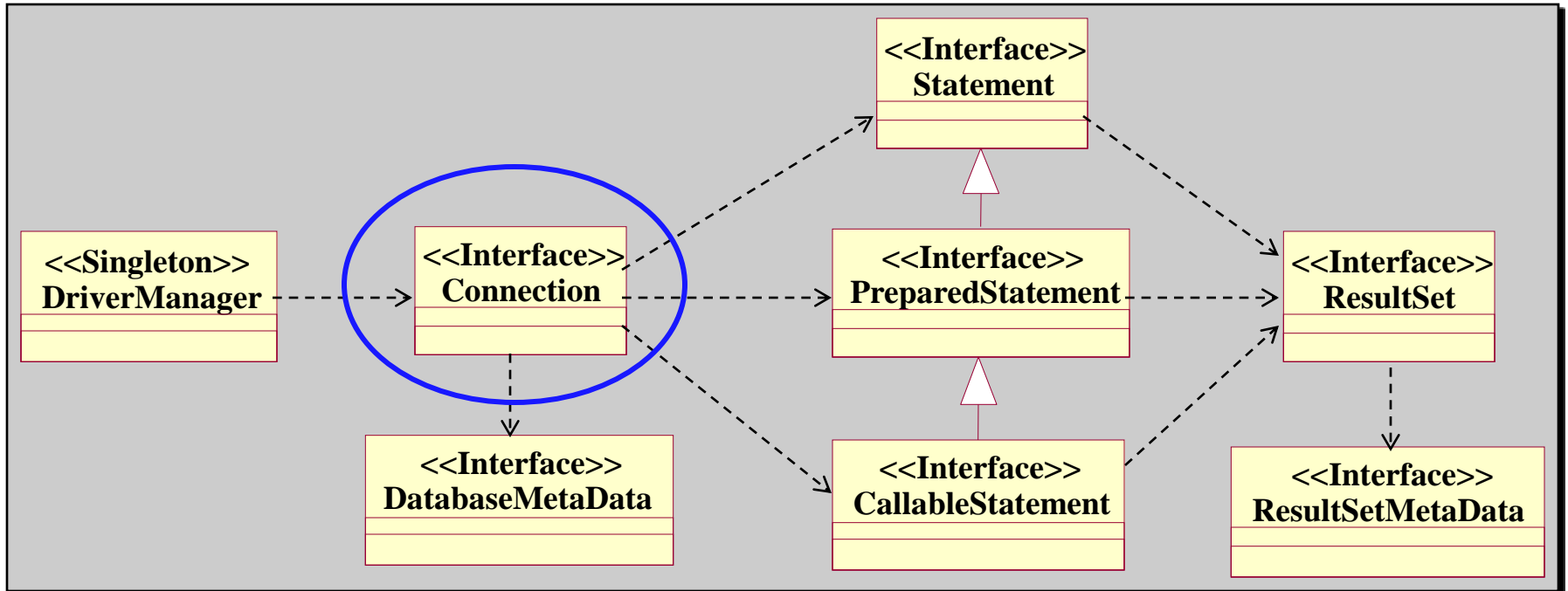
■ Class.forName

- ☐ Quando o método estático `Class.forName()` é utilizado, o Class Loader tenta inicializar a classe (driver jdbc), de tal maneira que o Java saiba que ele existe.
- ☐ `Class.forName` internamente chama o método `DriverManager.registerDriver`.



Classes JDBC

- `java.sql.*` provê um conjunto de interfaces para serem usadas pelas aplicações





Realizar a Conexão

- Após o registro do *driver* é preciso fornecer informações ao DriverManager para a conexão;

```
Connection conexao = DriverManager.getConnection ("DATABASE_URL", "user", "senha");
```

- Sintaxe geral de URLs:

- ☐ "jdbc:<subprotocol>://<server>:<port>/<database>"
- ☐ "jdbc:mysql://localhost/universidade"
- ☐ "jdbc:postgresql://localhost/test"



Realizar a Conexão

- Após o registro do *driver* é preciso fornecer informações ao DriverManager para a conexão;

```
Connection conexao = DriverManager.getConnection ("DATABASE_URL", "user", "senha");
```

- Sintaxe geral de URLs:

- ☐ "jdbc:<subprotocol>://<server>:<port>/<database>"
- ☐ "jdbc:mysql://localhost/universidade"

STRING DE CONEXÃO

```
package jdbc,  
import java.sql.*;
```

```
/**
```

```
 *  
 * @author vanessa  
 */
```

```
public class JDBC {
```

```
    static final String JDBC_DRIVER="org.postgresql.Driver";
```

```
    static final String DATABASE_URL = "jdbc:postgresql://localhost:5432/northwind";
```

```
    public static void main(String[] args) throws Exception {
```

```
        Class.forName(JDBC_DRIVER);
```

```
        Connection conexao = DriverManager.getConnection(DATABASE_URL, "postgres", "postgres");  
        System.out.print("Conexão efetuada com sucesso \n");
```

```
    }
```

```
}
```



Exception

A captura de exceções em Java é obrigatória para usarmos JDBC.

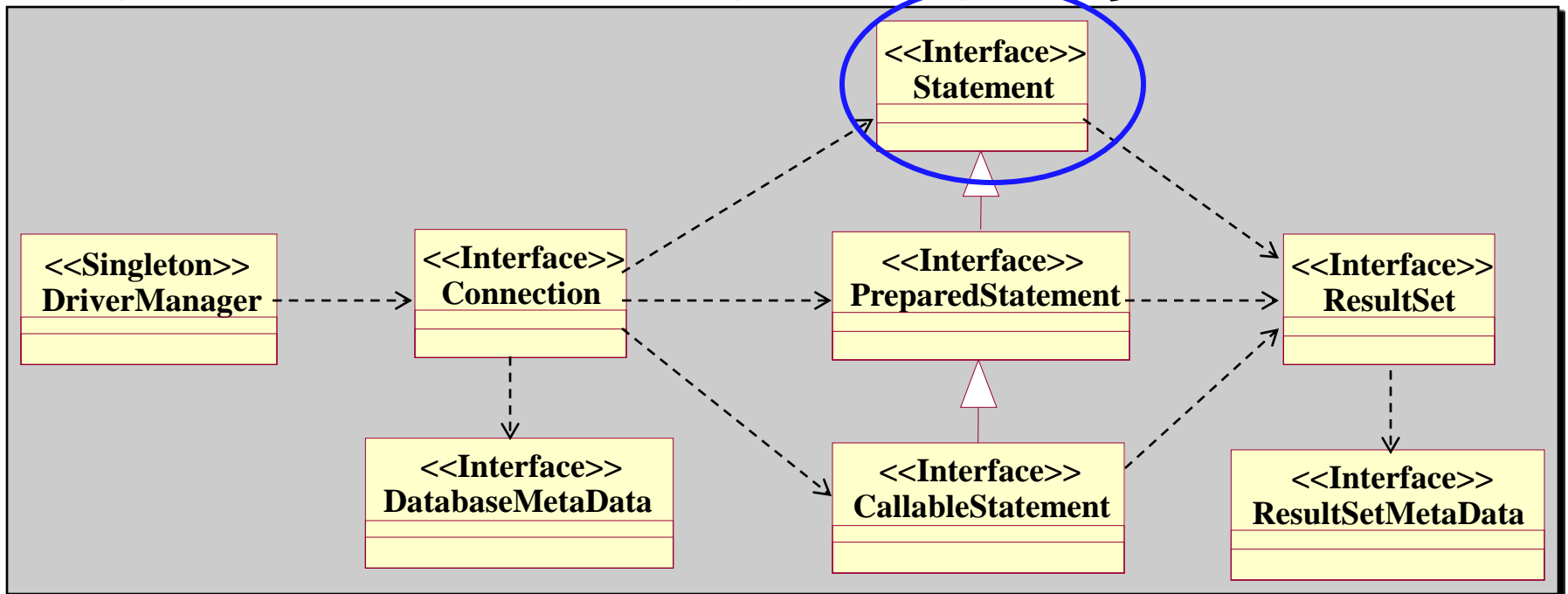
arquivo de configuração: **pg_hba.conf**.

```
# IPv6 local connections:
host      northwind      postgres      127.0.0.1/32      md5
host      all             all           ::1/128           md5
```



Classes JDBC

- `java.sql.*` provê um conjunto de interfaces para serem usadas pelas aplicações





Objetos do tipo statement

- Statement createStatement()
 - Retorna um novo objeto Statement, que é usado para executar um comando SQL no BD.
 - **executeUpdate()**
 - **executeQuery()**



Objetos do tipo statement

- Statement createStatement()
 - O método **executeUpdate()** de Statement recebe o SQL que será executado.
 - Este método deve ser usado para DDLs e comandos SQL de INSERT, UPDATE ou DELETE.



Objetos do tipo statement

```
public class JDBC {  
  
    static final String JDBC_DRIVER="org.postgresql.Driver";  
    static final String DATABASE_URL = "jdbc:postgresql://localhost:5432/northwind";  
  
    public static void main(String[] args) throws Exception {  
        Class.forName(JDBC_DRIVER);  
        Connection conexao = DriverManager.getConnection(DATABASE_URL, "postgres", "postgres");  
        System.out.print("Conexão efetuada com sucesso \n");  
  
        Statement stm = conexao.createStatement();  
        String SQL = "CREATE TABLE teste (a int primary key)";  
        stm.executeUpdate(SQL);  
        stm.close();  
        System.out.print("Tabela criada com sucesso\n");  
    }  
}
```



Exercício

- Criar a tabela 'teste' no seu banco
- Insira valores nessa tabela
- Acrescente o atributo 'nome' a tabela 'teste'



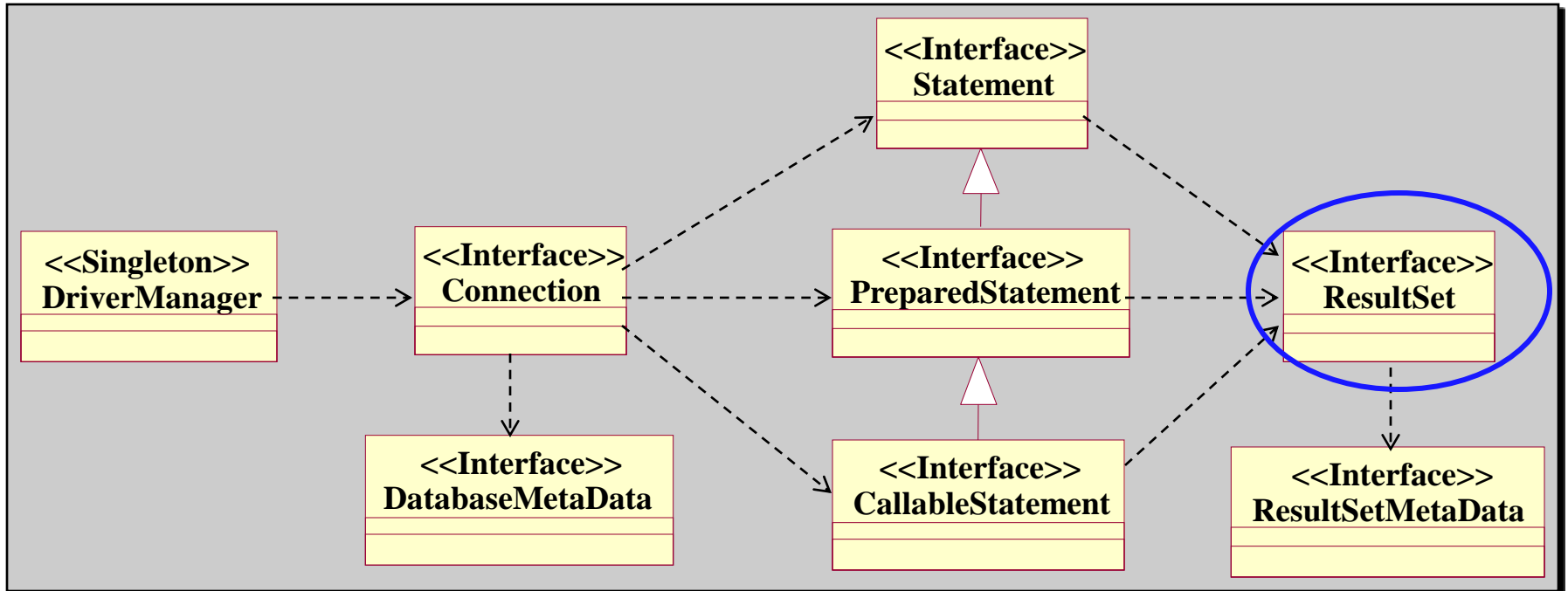
Objetos do tipo statement

- Statement createStatement()
 - O método **executeQuery()** de Statement executa uma consulta SQL e retorna um objeto ResultSet, que contém os dados recuperados.



Classes JDBC

- `java.sql.*` provê um conjunto de interfaces para serem usadas pelas aplicações





Objetos do tipo statement

■ ResultSet()

- ☐ Através do método `next()` de **ResultSet**, o cursor interno é movido para o próximo registro.
- ☐ O método retorna `false` caso não existam mais registros ou `true` caso contrário.
- ☐ Os valores dos registros podem ser recuperados como o método `get`, que recebe o nome do campo ou seu alias.



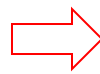
Objetos do tipo statement

Método	Tipos (java.sql.Types)
getByte	TINYINT
getShort	SMALLINT
getInt	INTEGER
getLong	BIGINT
getFloat	REAL
getDouble	FLOAT, DOUBLE
getBigDecimal	DECIMAL, NUMERIC
getBoolean	CHAR
getString	VARCHAR, LONGVARCHAR
getBytes	BINARY, VARBINARY
getDate	DATE
getTime	TIME
getTimestamp	TIMESTAMP
getAsciiStream	LONGVARCHAR
getUnicodeStream	LONGVARCHAR
getBinaryStream	LONGVARBINARY
getObject	Todos os tipos



Objetos do tipo statement

ResultSet rs



campoa	compob	campoc	campod
1	2	3	4
5	6	7	8

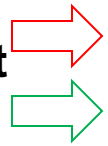
```
getString ("campoa") = 1  
getString ("campod") = 4
```



Objetos do tipo statement

ResultSet rs

rs.next



campoa	compob	campoc	campod
1	2	3	4
5	6	7	8

getString ("campoa") = 5
getString ("campod") = 8



Objetos do tipo statement

```
Statement stm = conexao.createStatement();
SQL = "SELECT * FROM northwind.categories;";
ResultSet res = stm.executeQuery(SQL);
int id;
String nome;
while (res.next())
{
    id = res.getInt(1);
    nome = res.getString(2);
    System.out.print(id + " , " + nome + '\n');
}
stm.close();
```



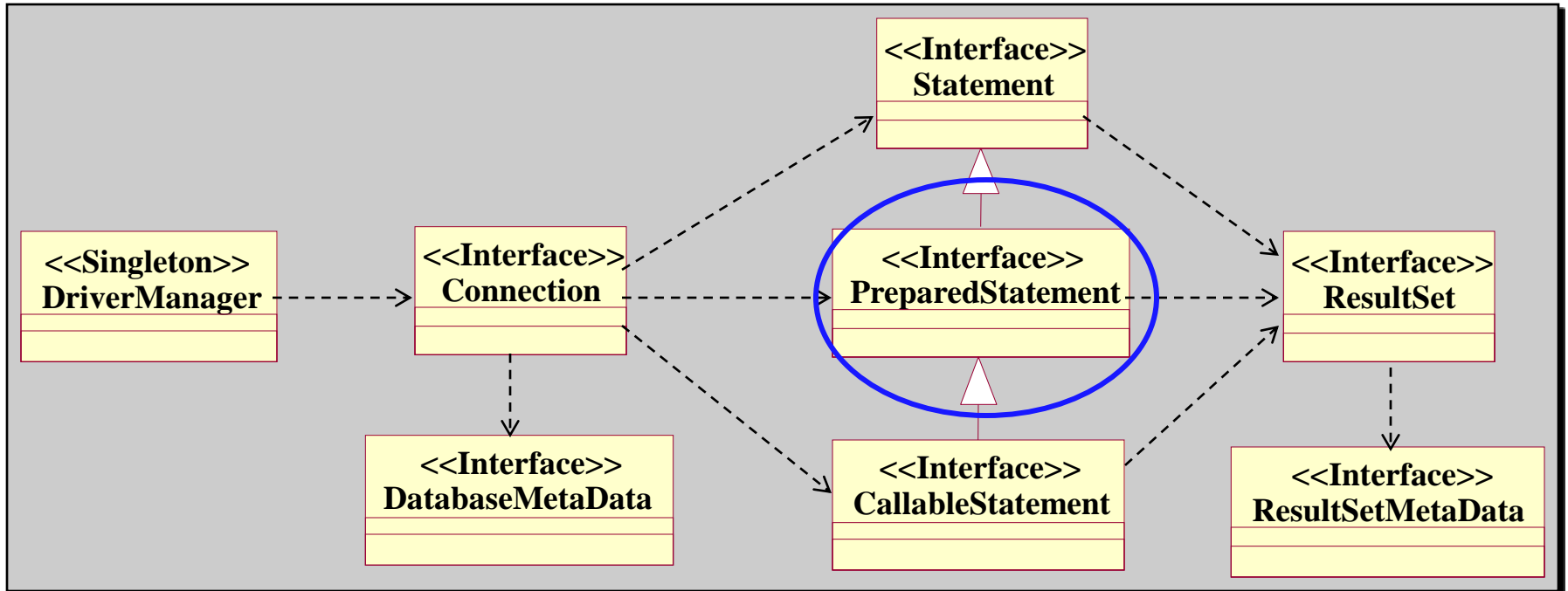
Exercício

- Recupere todos os dados da tabela 'teste'



Classes JDBC

- `java.sql.*` provê um conjunto de interfaces para serem usadas pelas aplicações





Objetos do tipo PreparedStatement

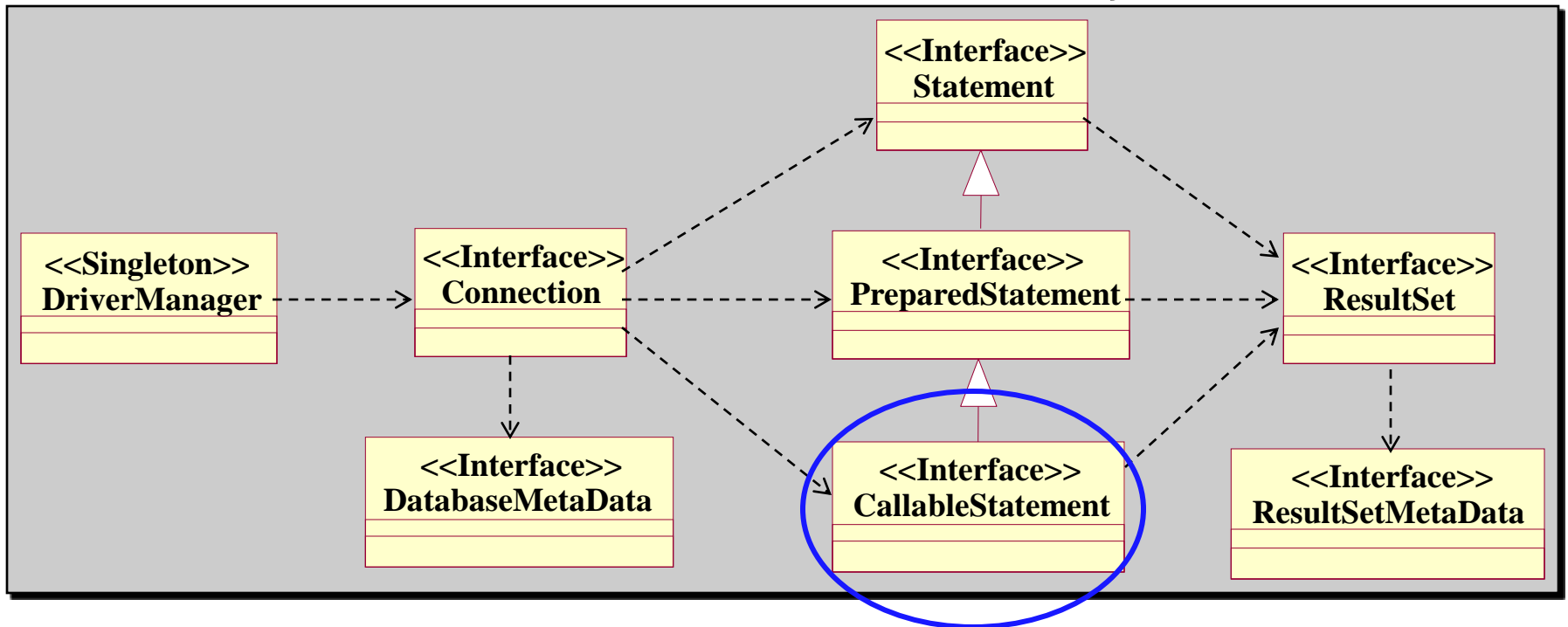
- PreparedStatement prepareStatement(String sql)
 - PreparedStatement tem vantagens sobre Statement, pois ela cria instruções SQL pré-compiladas, economizando recursos do próprio banco de dados, otimizando o programa.
 - Possível inserir dinamismo a partir do coringa '?'

```
public static void main(String[] args) throws Exception {
    Scanner leitura = new Scanner (System.in);
    System.out.print("Digite o nome da categoria: \n");
    String categoria = leitura.next();
    String SQL;
    Class.forName(JDBC_DRIVER);
    Connection conexao = DriverManager.getConnection(DATABASE_URL, "postgres", "postgres");
    System.out.print("Conexão efetuada com sucesso \n");
    SQL = "SELECT * FROM northwind.categories WHERE categoryname = ?";
    PreparedStatement pstmt = conexao.prepareStatement(SQL);
    pstmt.setString(1, categoria);
    ResultSet res = pstmt.executeQuery();
    int id;
    String nome;
    while (res.next())
    {
        id = res.getInt(1);
        nome = res.getString(2);
        System.out.print(id + " , " + nome + '\n');
    }
    pstmt.close();
}
```



Classes JDBC

- `java.sql.*` provê um conjunto de interfaces para serem usadas pelas aplicações





Objetos do tipo CallableStatement

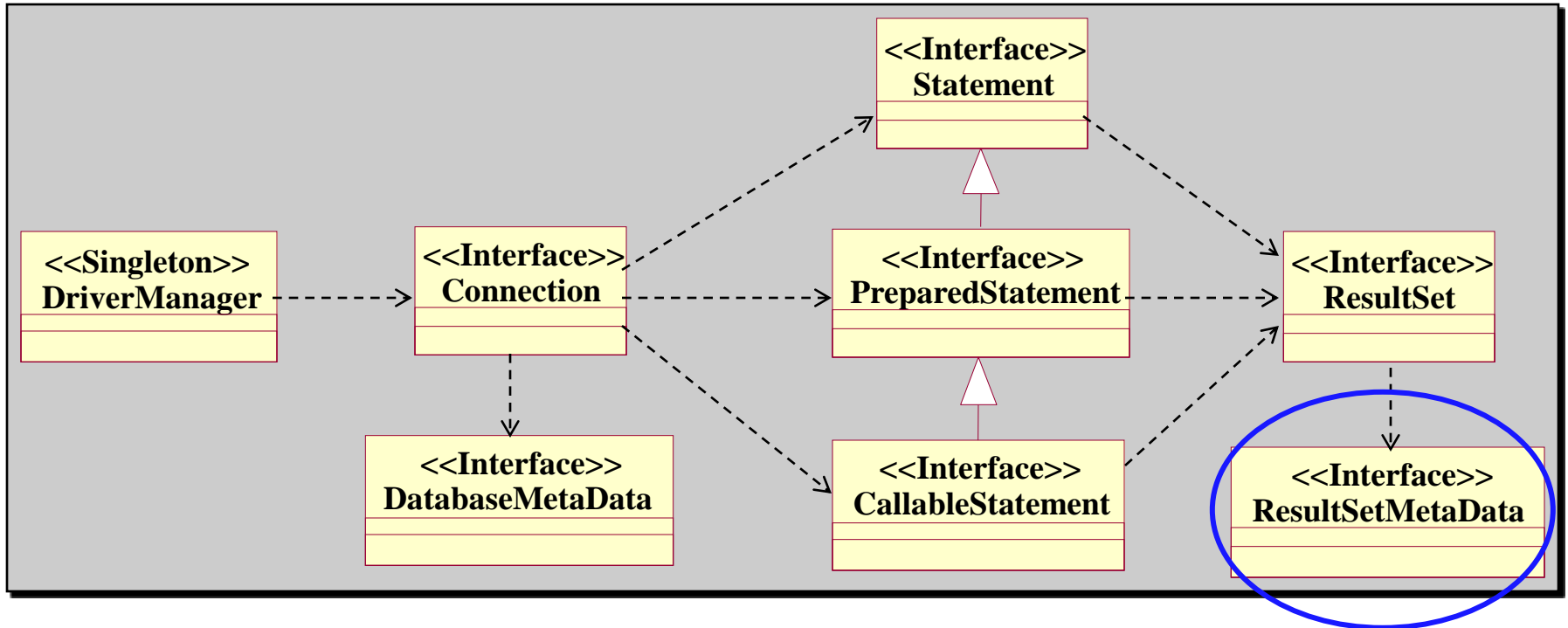
- CallableStatement prepareCall(String sql)
 - Retorna um novo objeto CallableStatement
 - Usado para chamar Stored Procedures.
 - Pode ou não retornar um resultSet

```
try {
    Class.forName("com.mysql.jdbc.Driver");
    Connection conexao = DriverManager.getConnection("jdbc:mysql://localhost/universidade", "root", "root");
    String SQL = "CALL altera_cpf(" + mat.getText() + ",'" + cpf_.getText() + "')";
    System.out.println(SQL);
    CallableStatement proc = conexao.prepareCall(SQL);
    System.out.println("Procedimento executado com sucesso");
    proc.execute();
    proc.close();
    conexao.close();
}
```



Classes JDBC

- `java.sql.*` provê um conjunto de interfaces para serem usadas pelas aplicações





Objetos do tipo ResultSetMetaData

- Tratar o resultSet
- Utilizar o resultSetMetadata
 - ☐ instancia o resultSet
 - ☐ ResultSetMetaData met = RS.getMetadata();
 - ☐ met.getColumnCount();
 - ☐ met.getColumnName();



Exercício

- Crie um novo projeto no netBeans ou Eclipse e crie uma função onde o usuário possa recuperar os registros da tabela 'northwind.employees', recendo o valor de lastname e firstname.
- Mostre os resultados da consulta – utilize o resultSetMetadata



Exercício

Digite o first name: Nancy
Digite o last name: Davolio

employeeid : 1
lastname : Davolio
firstname : Nancy
title : Sales Representative
titleofcourtesy : Ms.
birthdate : 1948-12-08 00:00:00
hiredate : 1992-05-01 00:00:00
address : 507 - 20th Ave. E.
Apt. 2A
city : Seattle
region : WA
postalcode : 98122
country : USA
homephone : (206) 555-9857
extension : 5467
reportsto : 2
notes : Education includes a BA in ,