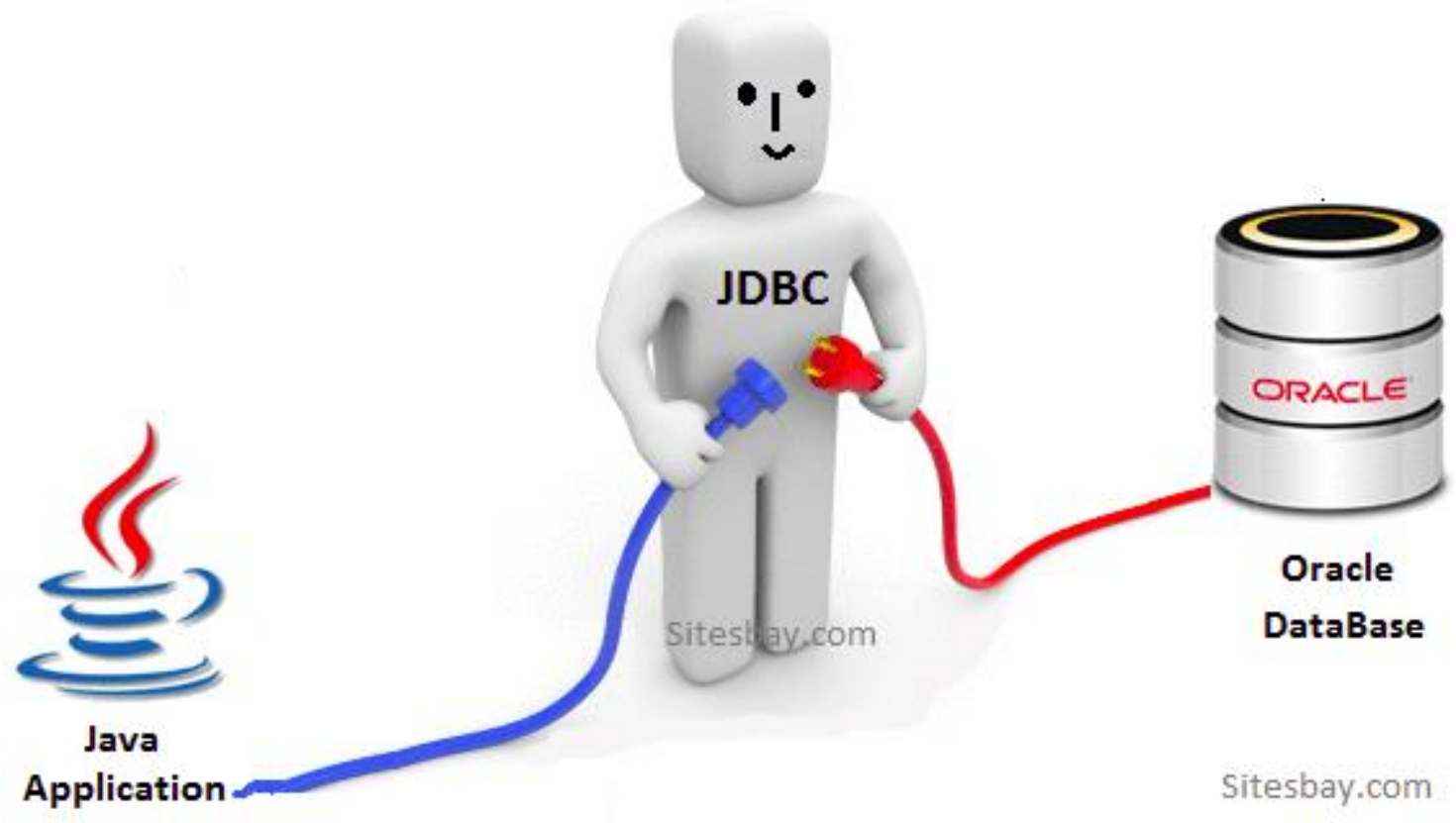



# [Projeto de Banco de Dados]
# Aula 03: Manipulando Bancos de Dados com JDBC

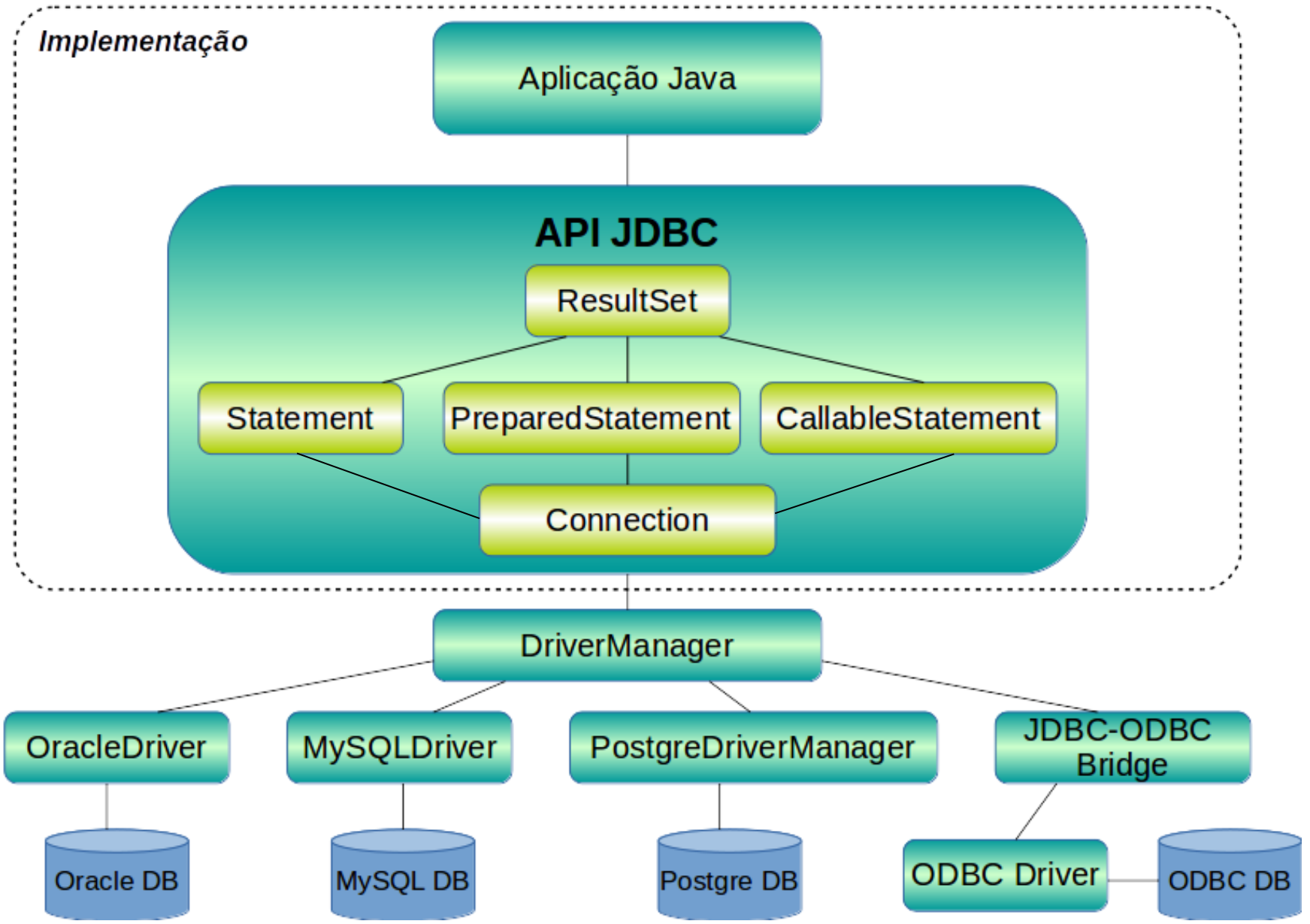
# JDBC

- A linguagem Java não suporta acesso direto a bancos de dados. Ela utiliza uma API (conjunto de classes e interfaces, a nível de código) para fazer a esta comunicação.

- A API JDBC (*Java DataBase Conectivity*) “provê uma acesso universal [a bancos de dados]. Usando JDBC você pode acessar virtualmente qualquer fonte de dados relacionais (…)” (JDBC API)
  - API JDBC: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

- O JDBC é capaz de, entre outras tarefas:
  - gerenciar a conexão e o protocolo de comunicação com BD;
  - executar consultas SQL no banco de dados.

# Arquitetura JDBC



# JDBC

- Por ser uma interface a nível de código, o código SQL é executado explicitamente dentro do código Java. As classes/interfaces estão agrupadas no pacote **java.sql**
  - http://docs.oracle.com/javase/8/docs/api/java/sql/package-summary.html

- Interfaces e Classes mais utilizadas:
  - ***java.sql.Connection***
    - Representa a conexão com o Banco de Dados.
    - Permitem criar objetos que criam e executam instruções SQL..
    - Os objetos são criados como resultado da classe *DriverManager*.
    - Exemplo:

```
import java.sql.Connection;
(...)
{
   Connection conn = DriverManager.getConnection
            (url, userBD, passwordBD);
}
```

    - Se o DriverManager não se conectar (ou outro erro de conexão), lança uma *java.sql.SQLException*.

# JDBC

- Interfaces e Classes mais utilizadas:
  - ***java.sql.Statement***
    - Controla e envia as declarações SQL para o BD.
    - Os objetos *statements* são criados e retornados pela conexão.
    - Exemplo:

```
import java.sql.Connection;
import java.sql.Statement;
(...)
{
   Connection conn = DriverManager.getConnection
            (url, userBD, passwordBD);
   Statement stmt = conn.createStatement();
}
```

    - Se houverem erros, lança uma *java.sql.SQLException*.

# JDBC

- Interfaces e Classes mais utilizadas:
  - *java.sql.PreparedStatement*
    - Versão estentida da Statement.
      - Utilizada para execução de SQLs mais elaboradas.
      - Mais fácil de ser entendida, mais confiável e mais rápidas
      - Código mais limpo e compreensível.
      - Um objeto representa uma declaração SQL pré-compilada.
    - Os objetos *PreparedStatements* são criados e retornados pela conexão.
    - Exemplo:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
(...)
{
 Connection conn = DriverManager.getConnection
            (url, userBD, passwordBD);
 PreparedStatement pStmt = conn.prepareStatement(sql);
}
```

    - Se houverem erros, lança uma *java.sql.SQLException*.



Veremos mais detalhes posteriormente.

# JDBC

- Interfaces e Classes mais utilizadas:
  - ***java.sql.ResultSet***
    - Conjunto de dados resultante de uma consulta SQL...
    - Inicialmente, o cursor do ResultSet é posicionado antes da primeira linha.
    - Se, quando invocado, o ResultSet for capaz de posicionar-se na próxima linha, retorna true. Caso contrário, false.
      - Os objetos ResulsetSet padrões não são atualizáveis, e o curso move-se somente para frente.
    - Exemplo:

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
(...)
{
   Connection conn = DriverManager.getConnection
                (url, userBD, passwordBD);
  Statement stmt = conn.createStatement();
  ResultSet rs = stmt.executeQuery(sql);
}
```

    - Um erro comum é tentar acessar o conteúdo do ResultSet antes de posicionar o cursor na primeira linha.
      - Se houverem erros, lança uma *java.sql.SQLException*.

# Tutorial JDBC: exemplo

1. Adicionar o conector JDBC ao Projeto
   - Depende da IDE.

2. Criando a classe de Conexão:

   2.1. Objetos:
   - **`java.sql.Connection`**
   - **`java.sql.Statement`** e/ou **`PreparedStatement`**
     - depende da implementação
   - **`java.sql.ResultSet`**

   2.2 Métodos:
   - **`public Construtor();`**
   - **`(private) ou (public) void setConnection();`**
   - **`public ResultSet executeQuery(String sql);`**
   - **`(??) executeUpdate(String query);`**
   - **`void desconect();`**

# Tutorial JDBC: exemplo

2. Criando a classe de Conexão:

2.3. Construtor

- No construtor serão inicializados as instâncias globais e os primeiros métodos.

2.4. setConnection()

- Este método é responsável por abrir a conexão com o BD.

```
Connection conn = DriverManager.getConnection
      (url, user, password);
```

- Onde:
  - **url**: caminho do BD
    - jdbc:nome_sgbd://host:porta/nomeBD
    - exemplo: `jdbc:postgresql://localhost:5432/pbd20171`
  - **user**: usuário do BD
  - **password**: senha do BD

# Tutorial JDBC: exemplo

2. Criando a classe de Conexão:

   2.4.1 Registrar Drive de Conexão

   - Até a versão 3 do JDBC (atualmente está na 4.2), antes de chamar o DriverManager.getConnection() era necessário registrar o driver JDBC que iria ser utilizado, e essa se comunicava com o DriverManager.
   - Se necessário, deve ser feito antes de instanciar a conexão.

```
Class.forName("nome_driverSGBD")
```

     - O driverSGBD é diferente para cada SGBD.
       - PostGreSQL: **`org.postgresql.Driver`**
       - MySQL: **`com.mysql.jdbc.Driver`**

   - A partir do JDBC 4, que está presente no Java 6, esse passo não é mais necessário.
   - Isso também pode ser necessário em alguns servidores de aplicação e web, como no Tomcat 7 ou posterior, por proteção para possíveis vazamentos de memória.

# Tutorial JDBC: exemplo

2. Criando a classe de Conexão:
   2.5. executeQuery()
   - Responsável por fazer as consultas no BD.

   - Exemplo 1: com Statements

```
Statement stmt = conn.createStatement();
```

   - As “statements” são liberadas pela conexão, para executarem as rotinas.
   - Os SQLs serão passados para as statements.
   - Devem ser instanciadas nos métodos Update e Query.

   - O resultado da consulta é armazenado num ResultSet.

```
ResultSet rs = stmt.executeQuery(sql);
```

# Tutorial JDBC

2. Criando a classe de Conexão:
   2.5.1 executeQuery(): ResultSet

- ResultSet possui método para recuperar os dados resultantes de uma consulta, além de retornar os metadados da consulta.
- Deve ser instanciada no métodos que fazem consulta.
  - Seja por Statement ou PreparedStatement.

- Importante:
  - `rs.next()`
    - retorna true se existem linhas ou próxima linha.

  - rs.getTipo("nomeColuna"); ou rs.getTipo(numeroColuna) retornam os valores armazenados nas colunas.
    - Exemplo:

```
rs.getInt("id");
rs.getString("nome");
```

# Tutorial JDBC

2. Criando a classe de Conexão:
   2.5. executeQuery()
   - Responsável por fazer as consultas no BD.
   - Exemplo 2: com PreparedStatements

```
PreparedStatement stmt = conn.preparedStatement(sql);
```

   - As “preparedStatements” são liberadas pela conexão, para executarem as rotinas.
   - Devem ser instanciadas nos métodos Update e Query.
   - O resultado da consulta é armazenado num ResultSet.
   - Exemplo:

```
PreparedStatement stmt = conn.prepareStatement
("select * from pessoa where id = ? nnd nome != ?");
stmt.setInt(1, 2);
stmt.setString(2, “paulo");
ResultSet rs = stmt.executeQuery();
```

# Tutorial JDBC: exemplo

2. Criando a classe de Conexão:
   2.6. executeUpdate()
   - Responsável por fazer CRUD no BD.
   - Pode ser feito com Statements ou PreparedStatements.

```
stmt = conn.createStatement();
stmt.executeUpdate(query);
```

Ou

```
pStmt = conn.createStatement();
pStmt.executeUpdate(sql);
```

- O método executeUpdate retorna 1 se houve sucesso.

# Tutorial JDBC: exemplo

2. Criando a classe de Conexão:
   2.7. desconect()
   - Sempre que acabar de utilizar o BD, devem ser fechadas as conexões, para liberar memória.

```
rs.close();
statement.close();
conn.close();
```

# Boas práticas

- Utilizar padrões de projeto
  - MVC
  - Singleton
  - Fábrica de conexões.
  - DAO (essencial).
    - Criar DAOS específicos para cada bean.

  - Obs.:
    - Utilizar padrões de projeto é altamente recomendável
    - Torna a arquitetura do sistema torna-se mais flexível e segura (depende do padrão)
    - É altamente recomendável.
    - Exige alterações nos códigos de exemplo.

# Referências


- ROB, P. Coronel, C. (2011) Sistemas de Banco de Dados. 8ºEd. Editora Cengage Learning.
- ELMASRI, R. Navathe, B. S. (2011) Sistemas de Banco de Dados. 6º Ed. Editora Pearson.
- HEUSER, Carlos A. Projeto de Banco de Dados. (1991)