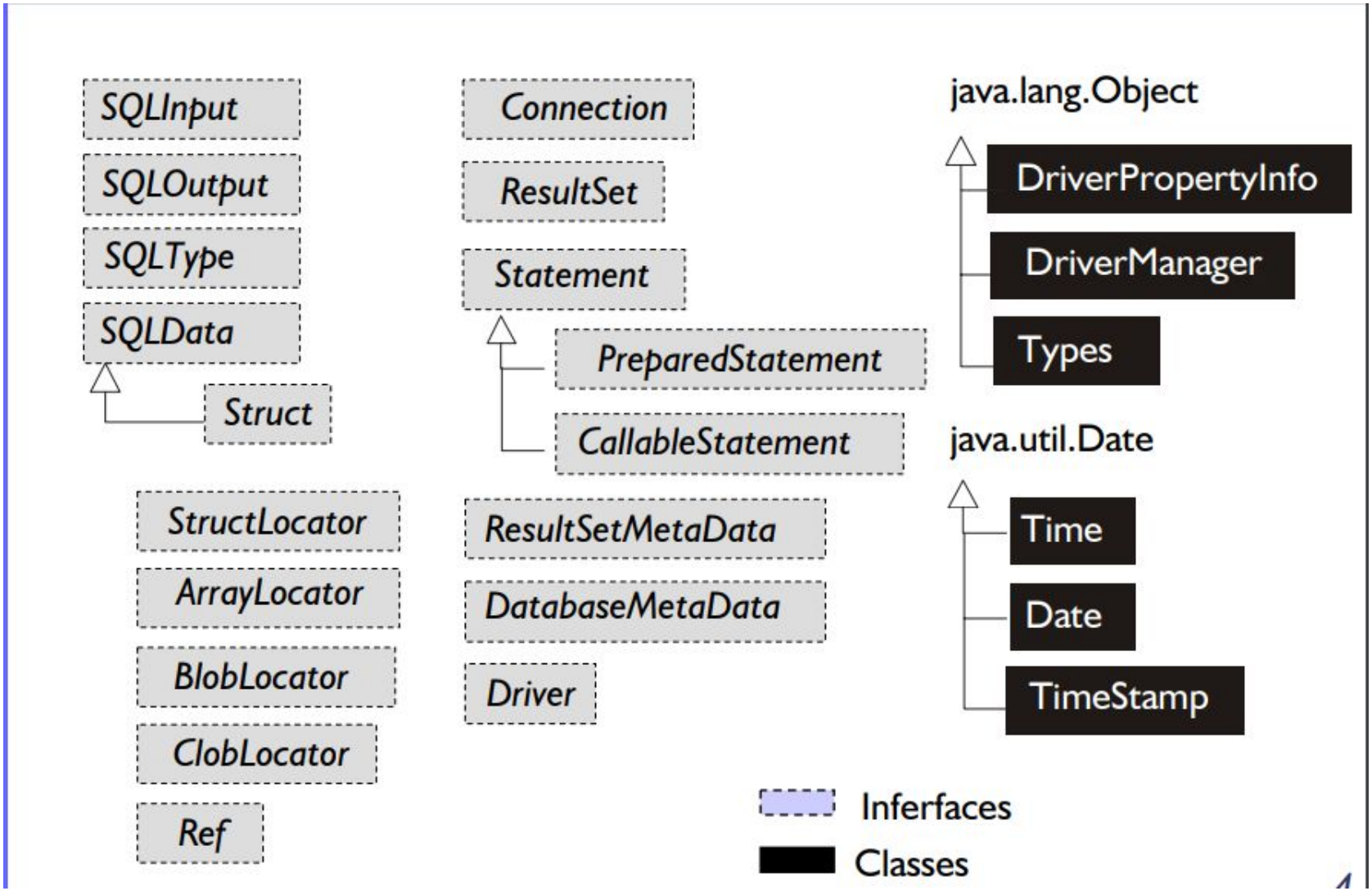
FUNDAMENTOS DE JDBC

JDBC

- JDBC é uma interface baseada em Java para acesso a bancos de dados através de comandos em SQL.
  - Pacote Java padrão: java.sql
  - Baseada em ODBC

- Este módulo apresenta uma introdução superficial do JDBC mas suficiente para integrar aplicações Java com bancos de dados relacionais que possuam drivers JDBC

Pacote java.sql



JDBC

- JDBC é uma interface de nível de código
  - Código SQL é usado explicitamente dentro do código Java
  - O pacote java.sql consiste de um conjunto de classes e interfaces que permitem embutir código SQL em métodos.
- Com JDBC é possível construir uma aplicação Java para acesso a qualquer banco de dados SQL.
  - O banco deve ter pelo menos um driver ODBC, se não tiver driver JDBC
- Para usar JDBC é preciso ter um driver JDBC
  - O J2SE distribui um driver ODBC que permite o acesso a bancos que não suportam JDBC mas suportam ODBC

Tipos de Drivers JDBC

- Tipo 1: ponte ODBC-JDBC

  - Usam uma ponte para ter acesso a um banco de dados. Este tipo de solução requer a instalação de software do lado do cliente.

  - Esta ponte é normalmente usada quando não há um driver puro-Java (tipo 4) para determinado banco de dados , pois seu uso é desencorajado devido à dependência de plataforma e provê um acesso mais lento.

Tipos de Drivers JDBC

- Tipo 2: solução com código nativo

  - Usam uma API nativa. Esses drivers contém métodos Java implementados em C ou C++. Requer software no cliente.

  - Se houver alteração do banco de dados, será necessário alterar a API nativa (que é específica do primeiro banco de dados)

  - Este driver é mais rápido que os do Tipo 1, pois elimina a sobrecarga gerada pelo ODBC

Tipos de Drivers JDBC

- Tipo 3: solução 100% Java no cliente

  - Oferecem uma API de rede via middleware que traduz requisições para API do driver desejado. Não requer software no cliente. Um único driver no cliente pode fornecer acesso a múltiplos bancos de dados. É o modelo mais flexível.

  - O servidor de aplicação funciona como um proxy JDBC, fazendo as chamadas ao banco de dados pelo cliente.

Tipos de Drivers JDBC

- Tipo 4: solução 100% Java

  - Drivers que se comunicam diretamente com o banco de dados usando soquetes de rede. É uma solução puro Java. Não requer código adicional do lado do cliente.

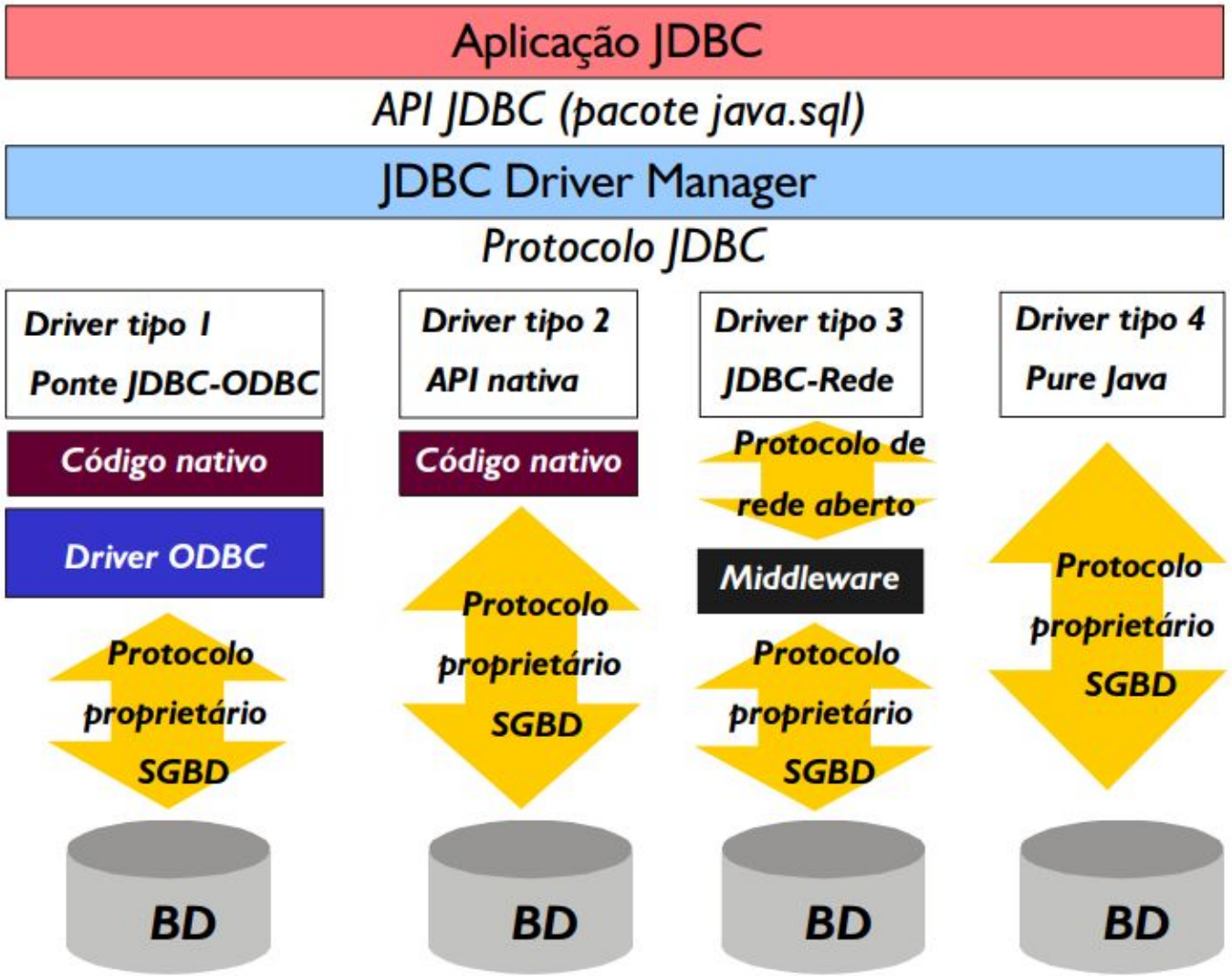
  - Implementado em Java, normalmente é independente de plataforma e escrito pelos próprios desenvolvedores. É o tipo mais recomendado para ser usado.

  - Este é o driver que fornece o maior desempenho dos tipos existentes. Um exemplo do Tipo 4 é o driver fornecido pela MySQL denominado Connector/J.

O que usar?

- Se você está utilizando um banco de dados de grande porte como Oracle, Sybase, IBM, Sql Server o driver recomendado é o Tipo 4;
- Se a aplicação Java vai acessar tipos de bancos de dados simultaneamente o driver recomendado é o Tipo 3;
- Driver do Tipo 2 são utilizados em situações onde os tipos 3 e 4 não estão disponíveis;
- Os drivers do Tipo 1 não são considerados como drivers para aplicações em produção, mas podem ser utilizados em casos onde não há outra opção.

Arquitetura JDBC



DriverManager e Driver

- A interface Driver é utilizada apenas pelas implementações de drivers JDBC
  - É preciso carregar a classe do driver na aplicação que irá utilizá-lo. Isto pode ser feito com Class.forName():

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

  - A partir do Java 6, esse comando não é mais necessário, pois o mesmo utiliza a JDVC 4.0(JSR 221).

DriverManager e Driver

- A classe DriverManager manipula objetos do tipo Driver.
  - Possui métodos para registrar drivers, removê-los ou listá-los.
  - É usado para retornar Connection, que representa uma conexão a um banco de dados, a partir de uma URL JDBC recebida como parâmetro

```
Connection con = DriverManager.getConnection
  ("jdbc:odbc:dados", "nome","senha");
```

Connection, Statement e ResultSet

- Interfaces que contém métodos implementados em todos os drivers JDBC.
- **Connection**
  - Representa uma conexão ao banco de dados, que é retornada pelo DriverManager na forma de um objeto.
- Statement
  - Oferece meios de passar instruções SQL para o sistema de bancos de dados.
- ResultSet
  - É um cursor para os dados recebidos.

MySQL Workbench - Criando o Banco de Dados

- Baixar no Drive “PROGRAMAR-COM-VC” na pasta “Scripts_Banco_Dados” o arquivo “Tarefa1.sql”;
- Executar o mesmo no Workbench;
- Pronto! Agora temos um banco de dados com uma tabela e alguns registros;

Instalar o Driver JDBC do MySQL

- Baixar no Drive na pasta “INSTALADORES” o driver do MySQL, arquivo “mysql-connector-java-5.1.47.zip”
- Descompactar esse arquivo dentro do diretório de seus projetos em JAVA.
- Criar um novo projeto no Eclipse chamado “Integracao Banco de Dados”

Instalar o Driver JDBC do MySQL

- Com o botão direito no nome do projeto, clique em “Properties” ou digite “ALT + ENTER”;
- Clicar em “Java Build Path”,acessar a aba “Libraries” e clicar no botão “Add External JARs...”
- Vá até a pasta onde você descompactou o driver de conexão e acesse o arquivo “mysql-connector-java-5.1.47.jar”.
- Clique no botão “Apply and Close”;
- Driver MySQl Connector instalado no seu projeto;

Classe Conexão

- Na pasta “src” crie um novo pacote “br.com.cursopcv.jdbc_conn”
- Criar uma classe “Conexao” com o método main para testar o acesso ao Banco de Dados (Vide comandos no projetor);
- Execute o programa para testar a conexão;

Connection, Statement e ResultSet

- Interfaces que contém métodos implementados em todos os drivers JDBC.
- Connection
  - Representa uma conexão ao banco de dados, que é retornada pelo DriverManager na forma de um objeto.
- **Statement**
  - Oferece meios de passar instruções SQL para o sistema de bancos de dados.
- ResultSet
  - É um cursor para os dados recebidos.

Statement

- Obtendo-se um objeto Connection, chama-se sobre ele o método createStatement() para obter um objeto do tipo Statement:

```
Statement stmt = con.createStatement()
```

que poderá usar métodos como execute(), executeQuery(), executeBatch() e executeUpdate() para enviar instruções SQL ao BD.

- Subinterfaces:
  - PreparedStatement e CallableStatement

```
PreparedStatement pstmt = con.prepareStatement(...)
CallableStatement cstmt = con.prepareCall(...);
```

Enviando instruções

- Exempos de uso de Statement

```
stmt.execute("CREATE TABLE dinossauros " + "(codigo INT
PRIMARY KEY, " + "genero CHAR(20), " + "especie
CHAR(20));");
```

```
int linhasModificadas = stmt.executeUpdate("INSERT INTO
dinossauros " + "(codigo, genero, especie) VALUES " +
"(499,'Fernandosaurus','brasiliensis')");
```

Exemplos: Criando uma tabela no Banco


```
public void criarTabela() throws SQLException {

    Statement stat = (Statement) connection.createStatement();
    stat.executeUpdate("drop table if exists cafes;");
    stat.executeUpdate("create table cafes (codigo int not null auto_increment , custo numeric(10,2), descricao varchar(200),"
                        + "primary key (codigo));");

    // Verificar se existe a tabela cafes no banco de dados
    DatabaseMetaData dbm = (DatabaseMetaData) connection.getMetaData();
    ResultSet tabela = dbm.getTables(null, null, "cafes", null);

    if (tabela.next()) {
        System.out.println("Tabela Criada Com Sucesso!!");
    } else {
       System.out.println("Tabela Não Existe");
   }
    //connection.close();
}
```

Exemplos: Inserir Registro Tabela


```
public void inserirRegTab() throws SQLException {

    PreparedStatement prep = (PreparedStatement) connection
                                    .prepareStatement("INSERT INTO cafes (nome,custo,descricao) values (?, ?, ?);");

    prep.setString(1, "Café Expresso");
    prep.setDouble(2, 12.50);
    prep.setString(3, "servido apenas com o mais puro café, sem qualquer mistura com leite ou outro ingrediente. \r\n" +
                      "A famosa espuma que se forma sobre a bebida é originária do próprio grão moído, e é produzida no  momento da extração");
    prep.addBatch();

    connection.setAutoCommit(false);
    prep.executeBatch();
    connection.setAutoCommit(true);
    connection.close();
}
```

Exemplos: Alterar Registro Tabela - Parte 1


```
public void alteraRegistro(int codigo, Double custo) throws SQLException {
    String sqlConsultaPorCod = "Select codigo,nome,custo from " + "cafes where codigo = ?";
    String sqlAlteracao = "update cafes set custo = ?" + " where codigo = ?";
    if (custo == null) {
        PreparedStatement prep = (PreparedStatement) connection.prepareStatement(sqlConsultaPorCod);
        prep.setInt(1, codigo);
        ResultSet rs = prep.executeQuery();
        if (rs.next()) {
        System.out.println("=============================================================");
        System.out.println("Dados do Café Para Alteração");
        System.out.println("Código..: " + rs.getString("codigo"));
        System.out.println("Nome..: " + rs.getString("nome"));
        System.out.println("Custo.: " + rs.getString("custo"));
        System.out.println("=============================================================");
```

Exemplos: Alterar Registro Tabela - Parte 2


```
.......
} else {
            System.out.println("Registro não Encontrado!");
            System.exit(0);
        }
        } else {
              StringBuffer srtRet = new StringBuffer();
              try {
             PreparedStatement prep = (PreparedStatement) connection.prepareStatement(sqlAlteracao);
            prep.setDouble(1, custo);
            prep.setInt(2, codigo);
            prep.execute();
            System.out.println("Registro alterado com sucesso!");
            catch (Exception e) {
            System.out.println("Erro: " + e.getMessage());
}
    }
      }
```

Exemplos: Deletar Registro Tabela


```
public void deletarRegistro(int codigo) throws SQLException {
    PreparedStatement prep = (PreparedStatement)
connection.prepareStatement("DELETE FROM cafes where codigo = ? ");
connection.setAutoCommit(true);
prep.setInt(1, codigo);
prep.executeUpdate();
}
```

ResultSet

- O método executeQuery(), da interface Statement, retorna um objeto ResultSet.
  - Cursor para as linhas de uma tabela.
  - Pode-se navegar pelas linhas da tabela recuperar as informações armazenadas nas colunas

```
ResultSet cursor = stmt.executeQuery("SELECT genero, especie " + " FROM dinossauros " + " WHERE codigo = 355");
```

- Os métodos de navegação são:
  - next(), previous(), absolute(), first() e last()
  - Métodos para obtenção de dados: getInt(), getString(), getDate(), getXXX(), ...

Tipos JDBC e métodos getXXX()

Método de ResultSet	*Tipo de dados SQL92*
`getInt()`	`INTEGER`
`getLong()`	`BIG INT`
`getFloat()`	`REAL`
`getDouble()`	`FLOAT`
`getBignum()`	`DECIMAL`
`getBoolean()`	`BIT`
`getString()`	`CHAR, VARCHAR`
`getDate()`	`DATE`
`getTime()`	`TIME`
`getTimestamp()`	`TIME STAMP`
`getObject()`	*Qualquer tipo (Blob)*

ResultSet

- Exemplos de uso de ResultSet

```
ResultSet rs = stmt.executeQuery("SELECT Numero, Texto, " +
   " Data FROM Anuncios");

while (rs.next()) {
   int x = rs.getInt("Numero");
   String s = rs.getString("Texto");
   java.sql.Date d = rs.getDate("Data");
   // faça algo com os valores obtidos...
 }
```

ResultSet


```
public String getDadosCafe(String nomeCafe) {

    String sqlConsulta = "Select codigo,nome,custo,descricao from cafes where nome like ?";

    PreparedStatement prepareStmt = null;
    ResultSet retQuery = null;
    StringBuffer dadodRet = new StringBuffer();

    try {
        prepareStmt = (PreparedStatement) this.getConnection().prepareStatement(sqlConsulta);
        prepareStmt.setString(1, "%" + nomeCafe + "%");
        retQuery = prepareStmt.executeQuery();
```

ResultSet


```
....
        if (retQuery.next()) {
            dadodRet.append("Código: ");
            dadodRet.append(retQuery.getInt("codigo"));
            dadodRet.append("\nNome: ");
            dadodRet.append(retQuery.getString("nome"));
            dadodRet.append("\n\tDescrição: ");
            dadodRet.append(retQuery.getString("descricao"));
            dadodRet.append("\nValor: R$ ");
            dadodRet.append(retQuery.getFloat("custo"));
        }
    } catch (Exception e) {
        dadodRet.append("Erro ao executar a consulta: "+e.getMessage());
    }
    return dadodRet.toString();
 }
```

Exercício Fixação

Criar um programa para pesquisar os dados das tabelas “aluno” e “professor” do banco “escola” e apresentar no console o resultado da pesquisa.

Obs.:

- Classes exemplos, baixar no Drive pasta “Códigos Fontes > JDBC > JDBC-MYSQL”;
- Classes dos exemplos desta aula, na pasta “Códigos Fontes > JDBC > Aula JDBC”;

Programa Com JDBC

1. Criar no MySQL Workbench um database chamado "banconovo"

....

create database banconovo;

use banconovo;

....

Programa Com JDBC

2. No Eclipse criar:

a) Um novo projeto chamado "Trabalhando com Banco de Dados"

b) Criar um novo pacote chamado "bcdadosnew"

3. Criar uma classe para desenvolver os metodos que serão utilizados no projeto, chamado "AcessoBancoNovo“

4. Na classe "AcessoBancoNovo" criar um construtor para fazer a conexão com o Banco criado.

Programa Com JDBC

5. Criar uma nova classe chamada “ChamadasBancoNovo”, esta classe será a principal com o método “main” e criar o objeto de conexão “abn” para testar a conexão. Passar dessa fase somente se a conexão retornar com sucesso;

6. Criar um método “criarTabela" para criar a nossa tabela chamada "dinossauros“ com os campos “código(autoincremento), Gênero e Espécie (ambos varchar(50), chamar a mesma na classe principal e fazer um teste para verficiar se a tabela foi criada com sucesso, enviar mensagem;

Programa Com JDBC

7. Criar um novo método “inserirRegTab()” para inserir novos registros na tabela criada;

8. Neste mesmo método, após inserir os registros faça comandos para capturar os dados registrados e mostrar na tela os valores dos registros inseridos;

Programa Com JDBC

9. Fazer um método onde o usuário irá digitar um determinado código e permitir que se faça uma

alteração no campo "espécie“ e “gênero”, método “alteraRegistro” e imprimir os dados do registro para confirmar a alteração;

10. Fazer a chamada a esse método na classe principal;

Programa Com JDBC

11. Fazer um método onde o usuário irá informar o código do dinossauro e a partir desse codigo o registro deverá ser apagado da tabela;

12. Fazer a chamada na classe principal;

Exercício de Fixação

Verificar no programa anterior, o que está repetido, fazer de uma forma que economize menos códigos, criar uma rotina para toda vez que precisarmos verificar o conteúdo dos dados de uma chamada.