import java.usj.LP1;

```
usj.LP1.setProfessor("Jaime Miranda Junior");
usj.LP1.setAlunos(    "Leonardo Ferreira Costa ",
                      "Lucas Gustavo Machado",
                      "Rafael Lapa Valgas");
```

import java.sql.*;

```
String CRUD = “CREATE READ UPDATE DELETE”;
```

/* O que iremos apresentar?

Apresentaremos as classes, interfaces e métodos básicos para a implementação do CRUD em Java, fazem parte do pacote java.sql.

CLASSES E INTERFACES: */

Classe: java.sql.DriverManager;

Interface: java.sql.Connection;

Interface: java.sql.PreparedStatement;

Interface: java.sql.ResultSet; // bônus

/* O que iremos apresentar?

```
MÉTODOS:        */
DriverManager.getConnection(String url, String user, String login);
Connection.preparedStatement(String sql);
Connection.close();
PreparedStatement.setString(String texto);  // variações desse método
ResultSet.next();
```

/* O que é o pacote java.sql?

Fornece os recursos necessários para acessar diferentes tipos de banco de dados.

Trabalha conjuntamente do driver do banco de dados.

Possibilita interface necessária para ler e gravar quaisquer fontes de dados em formato tabular (tabelas).

Atualmente está na versão 4.1. */

/* Funcionamento geral...

Possibilita efetuar uma conexão ao driver de banco de dados através da classe **DriverManager**. E enviar comandos SQL (em inglês SQL statements) através das interfaces **Connection, Statement** e **PreparedStatement** (existem outras)**.**

Por fim, é capaz de receber os resultados da consulta SQL através da interface **ResultSet**. Além de propiciar um conjunto de interfaces e classes para o mapeamento e tratamento dos dados. */

Class DriverManager

/* Fornece os recursos para carregar e conectar com diferentes drivers para diferentes bancos de dados. Atualmente não é necessário o sistema carregar os drivers, pois a classe consegue carregar automaticamente. Para isso é necessário que o driver esteja nas bibliotecas e referenciado corretamente (ver na documentação). */

Interface Connection

/* Uma sessão de conexão com uma base de dados. Essa interface é responsável por gerar e executar comandos SQL além de retornar e manipular os dados de uma consulta SQL. Possui diferentes métodos para consulta. Também é responsável pelo commit (alterações definitivas na base de dados) e pelo fechamento da conexão. */

Interface PreparedStatement

/* É um objeto que representa um pre comando SQL em uma String, esse comando pode ter valores atribuídos com métodos setters específicos dessa interface.

Funciona basicamente assim: */

```
String sql = “SELECT * FROM pessoa WHERE ID = ?”;
//É possível substituir o ‘?’ por qualquer valor com métodos setters específicos dessa classe.
```

Interface ResultSet

/* O objeto ResultSet contém o resultado da consulta SQL, uma tabela. E possui todas as funcionalidades para navegar na tabela, alterar e coletar dados.

observação:

Se por um lado a interface PreparedStatement funciona como setters de atributos para gerar comandos SQL, a interface ResultSet funciona como getters de atributos do banco de dados para gerar objetos nos sistemas implementados em java. */

/* Utilização geral...

É necessário efetuar a abertura de uma conexão com o driver através de um objeto **Connection** instanciado pela classe **DriverManager** e o método **.getConnection().**

A conexão/connection é utilizada para criar comandos SQL através de uma String junto da interface **PreparedStatement** que é capaz de executar a query através do método **.execute().**

Se necessário armazenar os resultados da query, é necessário instanciar um objeto da interface **ResultSet** através do método e .**executeQuery().**

O objeto **ResultSet** possui métodos de navegação nas tabelas - apresentaremos apenas **.next()** e **.previous()**.

Por fim é necessário fechar a conexão através do método .close() da interface **Connection**. */

//Métodos DriverManager e Connection

- DriverManager.getConnection(String url, user, password);

//Gera um objeto Connection e uma conexão com o driver do banco de dados.

- Connection.PrepareStatement(String sql);

//Gera um objeto PreparedStatement para gerenciar comandos SQL.

//Métodos PreparedStatement

- PreparedStatement.setString(Int posição, String valor);
- PreparedStatement.setInt(Int posição, Int valor);

//Substitui valores da String (?) por atributos específicos para a consulta. Existe um método set para cada tipo: Boolean, Char, Float, Double...

- PreparedStatement.execute();

//Executa a query e retorna um boolean indicando se a query foi executada com sucesso.

- PreparedStatement.executeQuery();

//Executa query e retorna o valor da consulta para gerar um objeto ResultSet. É utilizada quando precisa de uma tabela como retorno.

//Métodos ResultSet

- ResultSet.next();

//Caminha o cursos na tabela para o próximo registro (próxima linha).

- ResultSet.previous();

//Caminha o cursos na tabela para trás (próxima linha).

- ResultSet.getString(String nomeColuna);

//Acessa o valor de uma coluna com o nome especificado. Existe variações em cada tipo.

- ResultSet.getString(Int indexColuna);

//Acessa o valor da coluna na ordem especificada. Existe variações em cada tipo.

/* Navegação em tabelas pelo **ResultSet**

A interface ResultSet possui diferentes métodos para navegação nas tabelas. A navegação padrão mais utilizada é através dos métodos .next() e .previous(), basicamente caminhando para frente e para trás nos registros de cada tabela. */

```
While(ResultSet.next()){
• //comandos}
```

Output de navegação:

> A, A1, A2

A	B	C	D
A1	B1	C1	D1
A2	B2	C2	D2

main.java | DBCon.java | PessoaDAO.java

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class main {
    Run | Debug
    public static void main(String[] args) {

        //  Cria conexão.
        Connection connection = DBCon.getConnection();

        //  Gera um objeto pessoaDAO para as operações.  ---> Melhor Static?!
        PessoaDAO pessoaDAO = new PessoaDAO();

        //  Gera um objeto pessoa através de um construtor e cadastra no banco de dados.
        Pessoa objPessoa1 = new Pessoa(00000000000, "Jaime", "Miranda", "jaime@miranda.junior.br", 489000000000);
        pessoaDAO.cadastrarPessoa(connection, objPessoa1);

        //  Cria um objeto pessoa apenas com CPF e identifica os dados através de uma consulta na base de dados.
        Pessoa objPessoa2 = new Pessoa();
        objPessoa2.setCpf(00000000000);
        objPessoa2 = PessoaDAO.consultarPessoa(connection, objPessoa2);

        //  Finaliza conexão com o driver do banco de dados.
        connection.close();
        }
}
```

```java
public class DBCon {

    //  Atributos para o método .getConnection().
    //  Uma boa prática seria isolar em uma classe e utilizar métodos getters.
    private final static String url = "jdbc:postgresql://localhost/dvdrental";
    private final static String user = "postgres";
    private final static String password = "entrar";

    //Exemplo de uma rotina básica de conexão ao driver do banco de dados;

    /*  No exemplo apresentado é gerado um objeto null Connection nomeado connection;
    *   Tenta gerar uma conexão através da classe e método DriverManager.getConnection();
    *   Se não houver erros irá gerar um objeto.
    *   Em caso de erros avisará através do primeiro try{}cacth{};
    *   ao gerar o objeto a função irá retornar esse objeto de conexão ao solicitante.  */

    public static Connection getConnection() {
        //  Gera um objeto null do tipo Connection.
        Connection connection = null;
        //  Em caso de erro é preciso verificar endereço, login ou senha.
        try {
            //  Gera o objeto Connection através do método .getConnection().
            connection = DriverManager.getConnection(url, user, password);
            System.out.println("Conectado com sucesso!");
            /
        } catch (SQLException e) {
            System.out.println("Error - Conexão: "+e.getMessage());
        }
        //  Retorna o objeto connection.
        return connection;
    }
```

```java
public class PessoaDAO {

    //  Exemplo de uma rotina básica de cadastro no banco de dados;

    /*  No exemplo apresentado o método cadastrarPessoa() precisa de um objeto tipo Connection e Pessoa.
    *   É gerada uma String com o comando SQL básico a ser executado pelo método.
    *   É gerado um objeto preparedStatement através do objeto connection, utilizando a String.
    *   Os valores de ?, ?, ?, ? são tratados através de métodos especiais e de consulta no objeto pessoa.
    *   é utilizada a função .execute() que executa a query.     */

    public void cadastrarPessoa(Connection connection, Pessoa pessoa){
        String sql = "INSERT INTO pessoas (cpf, primeiro_nome, segundo_nome, email, telefone) VALUES (?, ?, ?, ?, ?)";
        PreparedStatement prepareStatement = connection.preparedStatement(sql);
        prepareStatement.setInt(1, pessoa.getCpf());
        prepareStatement.setString(2, pessoa.getPrimeiroNome());
        prepareStatement.setString(3, pessoa.getSegundoNome());
        prepareStatement.setString(4, pessoa.getEMail());
        prepareStatement.setInt(5, pessoa.getTelefone());
        prepareStatement.execute();
    }
```

```java
    //  Exemplo de uma rotina básica de consulta pessoa no banco de dados através do CPF.

/*  No exemplo apresentado o método consultarPessoa() precisa de um objeto connection e pessoa.
*   observação: o objeto pessoa não necessáriamente está completo, mas no mínimo deve vir com o dado CPF.
*   É gerado uma String com o comando SQL básico a ser executado pelo método.
*   É gerado um objeto preparedStatement através do objeto connection.
*   Os valores de ? (o CPF) é tratado através do método especial do preparedStatement e do .getCpf() da pessoa.
*   É gerado um objeto ResultSet através do método .executeQuery() do objeto preparedStatement.
*       O objeto ResultSet é exatamente a tabela retornada na query, referente a consulta.
*   Através da rotina While(resultSet.next()) conseguimos verificar todos os dados retornados na consulta.
*   esses dados são incluidos no objeto pessoa inicialmente passado para a consulta.
*   e por fim retorna o objeto pessoa recheado de dados.        */

public Pessoa consultarPessoa(Connection connection, Pessoa pessoa){
    String sql = "SELECT * FROM pessoas WHERE cpf = ?";
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setInt(1, pessoa.getCpf());
    ResultSet resultSet = preparedStatement.executeQuery();
    while(resultSet.next()){
        pessoa.setPrimeiroNome(resultSet.getString("primeiro_nome"));
        pessoa.setSegundoNome(resultSet.getString("segundo_nome"));
        pessoa.setEMail(resultSet.getString("email"));
        pessoa.setTelefone(resultSet.getString("telefone"));
        //...
    }
    return pessoa;
}
}
```

```
usj.LP1.setNota(10);
usj.LP1. connection.close();
```

Obrigado! ☺