

```
import java.sql.*;
```

```
String CRUD = "CREATE READ UPDATE DELETE";
```

/* O que iremos apresentar?

Apresentaremos algumas classes, interfaces e métodos do pacote `java.sql` e que são responsáveis em efetuar o básico do **CRUD** em Java.

CLASSES E INTERFACES: */

Classe: `java.sql.DriverManager;`

Interface: `java.sql.Connection;`

Interface: `java.sql.PreparedStatement;`

Interface: `java.sql.ResultSet; //bonus`

/* O que iremos apresentar?

MÉTODOS: */

DriverManager.getConnection(String url, String user, String login);

Connection.prepareStatement(String sql);

Connection.close();

PreparedStatement.setString(String texto); //variações de tipos

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 através das interfaces **Connection**, **Statement** e **PreparedStatement** (existem outras).

observação: (em inglês SQL statements)

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 = ?";
```

/* Com métodos setters específicos é possível substituir o '?' por qualquer valor. */

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. Se por um lado a interface PreparedStatement
funciona como setters de atributos a interface ResultSet funciona como
getters de atributos.      */
```

/* 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 o resultado da query, é necessário instanciar um objeto da interface **ResultSet** através do método **.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, Floar, 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 frente.

- `ResultSet.previous();`

//Caminha o cursos na tabela para trás.

/* Navegação em tabelas pela 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.

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

Output de navegação:

> A, B, C, 1, 2, 3

A	B	C
1	2	3

```
1 import java.sql.Connection;
2 import java.sql.DriverManager;
3 import java.sql.ResultSet;
4 import java.sql.SQLException;
5 import java.sql.Statement;
6
7 public class main {
8     Run | Debug
9     public static void main(String[] args) {
10         //Cria conexão
11         Connection connection = DBCon.getConnection();
12
13         //gera um objeto pessoaDAO
14         PessoaDAO pessoaDAO = new PessoaDAO();
15
16         //gera um objpessoa1 completo
17         Pessoa objPessoa1 = new Pessoa(00000000000, "Jaime", "Silva", "jaime@silva.br", 4890000000000);
18
19         //cadastra o objpessoa1 na base de dados com as informações passadas no construtor
20         pessoaDAO.cadastrarPessoa(connection, objPessoa1);
21
22         //cria um objPessoa2 pessoa apenas com CPF
23         Pessoa objPessoa2 = new Pessoa();
24         objPessoa2.setCpf(00000000000);
25
26         //consulta base de dados para pessoa com cpf e retorna os valores no objPessoa2
27         objPessoa2 = PessoaDAO.consultarPessoa(connection, objPessoa2);
28
29         //finaliza conexão com o driver do banco de dados
30         connection.close();
31     }
32 }
```

```
7 public class DBCon {
8
9     //Atributos para o método .getConnection();
10    //Uma boa prática seria isolar em uma classe e utilizar métodos getters;
11    private final static String url = "jdbc:postgresql://localhost/dvdrental";
12    private final static String user = "postgres";
13    private final static String password = "entrar";
14
15    //Exemplo de uma rotina básica de conexão ao driver do banco de dados;
16
17    /* No exemplo apresentado é gerado um objeto null Connection nomeado connection;
18     * Tenta gerar uma conexão através da classe e método DriverManager.getConnection();
19     * Se não houver erros irá gerar um objeto, em caso de erros avisará através do try{}catch{};
20     * ao gerar o objeto a função irá retornar esse objeto de conexão ao solicitante*/
21
22    public static Connection getConnection() {
23        //Gera um objeto null do tipo Connection;
24        Connection connection = null;
25        //em caso de erro deve verificar endereço, login ou senha;
26        try {
27            //tenta gerar o objeto Connection através do método .getConnection()
28            connection = DriverManager.getConnection(url, user, password);
29            System.out.println("Conectado com sucesso!");
30            /
31        } catch (SQLException e) {
32            System.out.println("Error - Conexão: "+e.getMessage());
33        }
34        //retorna um objeto connection
35        return connection;
36    }
```



```
7 public class PessoaDAO {
8
9     //Exemplo de uma rotina básica de cadastro no banco de dados;
10
11     /* No exemplo apresentado o método cadastrarPessoa() precisa de um objeto de connection e pessoa;
12     * É gerada uma String com o comando SQL básico a ser executado pelo método;
13     * É gerado um objeto preparedStatement através do objeto connection, utilizando a String anterior;
14     * Os valores de ?, ?, ?, ? são tratados através de métodos especiais e de consulta no objeto pessoa;
15     * é utilizada a função .execute() que executa a query */
16
17     public void cadastrarPessoa(Connection connection, Pessoa pessoa){
18         String sql = "INSERT INTO pessoas (cpf, primeiro_nome, segundo_nome, email, telefone) VALUES (?, ?, ?, ?, ?)";
19         PreparedStatement preparedStatement = connection.prepareStatement(sql);
20         preparedStatement.setInt(pessoa.getCpf());
21         preparedStatement.setString(pessoa.getPrimeiroNome());
22         preparedStatement.setString(pessoa.getSegundoNome());
23         preparedStatement.setString(pessoa.getEmail());
24         preparedStatement.setInt(pessoa.getTelefone());
25         preparedStatement.execute();
26     }
```

```
28 //Exemplo de uma rotina básica de consulta pessoa no banco de dados através do CPF;
29
30 /* No exemplo apresentado o método consultarPessoa() precisa de um objeto connection e pessoa;
31 * observação: o objeto pessoa não necessariamente está completo, mas no mínimo deve vir com o dado CPF;
32 * É gerado uma String com o comando SQL básico a ser executado pelo método;
33 * É gerado um objeto preparedStatement através do objeto connection;
34 * Os valores de ? (o CPF) é tratado através do método especial de preparedStatement e da consulta pessoa;
35 * É gerado um objeto ResultSet através do método .executeQuery() do objeto preparedStatement;
36 * O objeto ResultSet é o retorno da query, referente a consulta;
37 * Existe maneiras de caminhar pelos dados das tabelas, o comando mais comum é o método .next(),
38 * Através da rotina While(resultSet.next()) conseguimos verificar todos os registros retornados
39 * esses dados são incluídos no objeto pessoa inicialmente passado para a consulta
40 * e por fim retorna o objeto pessoa recheado de dados;
41 */
```

```
42
43 public Pessoa consultarPessoa(Connection connection, Pessoa pessoa){
44     String sql = "SELECT * FROM pessoas WHERE cpf = ?";
45     PreparedStatement preparedStatement = connection.prepareStatement(sql);
46     preparedStatement.setInt(1, pessoa.getCpf());
47     ResultSet resultSet = preparedStatement.executeQuery();
48     while(resultSet.next()){
49         pessoa.setPrimeiroNome(resultSet.getString("primeiro_nome"));
50         pessoa.setSegundoNome(resultSet.getString("segundo_nome"));
51         pessoa.setEmail(resultSet.getString("email"));
52         pessoa.setTelefone(resultSet.getString("telefone"));
53         //...
54     }
55     return pessoa;
56 }
57 }
```

```
import java.usj.LP1;
```

```
usj.LP1.setProfessor("Jaime");
```

```
usj.LP1.setAlunos("Léo", "Lucas", "Rafael");
```

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

