ACESSO DADOS COM DAO

Esse tutorial pretende mostrar como operações como busca, deleção, atualização e inserção de dados no banco podem ser feitas com diversas tecnologias. Assim, podemos ter um repositório de tutorias sobre as mais diversas API's, tecnologias e frameworks que estão no mundo Java com um exemplo simples e de fácil entendimento.

Embora aqui no fórum tenhamos muitos tutoriais com acesso a banco de dados, esse pretende ser um pouco diferente, pois iremos utilizar as classes demonstradas hoje para construir outras partes em outras tecnologias.

Primeiramente parte iremos abordar somente o banco de dados e a manipulação dos dados em modo texto.

Futuramente, pretendemos usar essas mesmas classes com Java Swing entre outras tecnologias.

Por ser um primeiro exemplo simples, essa primeira parte desconsiderou algumas boas práticas e uso de alguns comandos um pouco mais complexos, e quebramos alguns padrões, como exibição de mensagem ser da responsabilidade da view. Assim, na segunda parte iremos mostrar como deixar o acesso mais limpo e flexível, eliminando muitos pontos fracos do código!

CADASTRO PESSOA

Nossa pessoa precisa se identificar, RG! Mas você não chama as pessoas pelo RG, então colocaremos o nome da pessoa também e mais estado, cidade, idade e só.

Toda tabela precisa de um identificador único que chamamos de chave primária, ou seja, algo que diferencie uma tabela das outras. No caso da pessoa, a chave é o RG.

Agora vamos criar isso no banco de dados, estamos usando MySql.

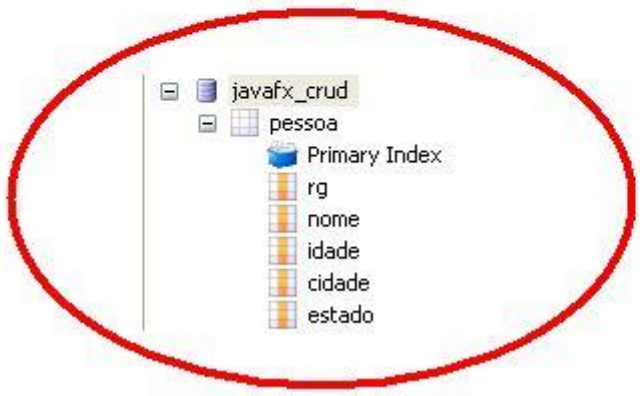
Criamos um database com o nome "javafx_crud"(você pode criar com o nome que quiser), o script para a criação da tabela ficou assim:

```
1.  use javafx_crud;
2.  create table pessoa(
3.     rg varchar(20) not null,
4.     nome varchar(20) not null,
5.     idade int(2) not null,
6.     cidade varchar(20) not null,
7.     estado varchar(2) not null,
8.     primary key(rg)
9.  );
```

Rodando o Script você terá sua tabela no banco de dados fisicamente.

Sugestão: Usem MySql e MySql - Front

```
1 use javafx_crud;
2 create table pessoa(
3    rg varchar(20) not null,
4    nome varchar(20) not null,
5    idade int(2) not null,
6    cidade varchar(20) not null,
7    estado varchar(2) not null,
8    primary key(rg)
9 );
```



Para salvar os dados você tem que logar no banco e ir inserindo usando o comando INSERT, ou pela interface que os SGBDs(programa que gerencia o banco) oferecem. Mas isso não é usual, o interessante é ter um aplicativo que faz isso, acessa o banco e permite as funcionalidades, sem o usuário conhecer o banco de dados.
Em Java, como em qualquer outra linguagem(exceto algumas específicas, como ABAP, mas isso já são outros 500), você tem que se conectar ao banco de dados para realizar ações com os dados. Dependendo do banco que você utiliza, existe um .jar que permite

o acesso.

O JAR para o MySQL está anexo!

Quanto a manipulação, você pode usar um framework(ex: Hibernate, neste caso até a conexão) de persistência ou a solução nativa de java JDBC. JDBC é menos flexível que um framework, mas é mais simples para nosso projetinho, vamos usar JDBC!

Crie um projeto Java no eclipse , chame como quiser :), depois comece a codificação!

Hora de código

O comum em Java é criarmos classes Pojo, classes que são semelhantes a tabelas do banco, assim podemos manipular de forma igual para igual ao que está no banco. Nossa classe pessoa:

```
1.  package model;
2.
3.  public class Pessoa {
4.     private String rg;
5.     private String nome;
6.     private int idade;
7.     private String estado;
8.     private String cidade;
9.
10.    public void setRg(String rg) {
11.       this.rg = rg;
12.    }
13.
14.    public void setNome(String nome) {
15.       this.nome = nome;
16.    }
17.
18.    public void setIdade(int idade) {
19.       this.idade = idade;
20.    }
21.
22.    public void setEstado(String estado) {
23.       this.estado = estado;
24.    }
25.
26.    public void setCidade(String cidade) {
27.       this.cidade = cidade;
28.    }
29.
30.    public String getRg() {
31.       return this.rg;
32.    }
33.
34.    public String getNome() {
35.       return this.nome;
36.    }
37.
38.    public int getIdade() {
39.       return this.idade;
40.    }
41.
42.    public String getEstado() {
43.       return this.estado;
44.    }
45.
```

```
46.    public String getCidade() {
47.       return this.cidade;
48.    }
49. }
```

Claro, todos os atributos estão encapsulados, nunca use atributo público, use sempre métodos de acesso a atributos declarados como privados.

Essa classe é parte de um projeto MVC, ela é o modelo, ou model, por isso estará no pacote(usamos para dividir nosso projeto melhor) model, então crie um pacote e coloque esse código em um arquivo .java lá.

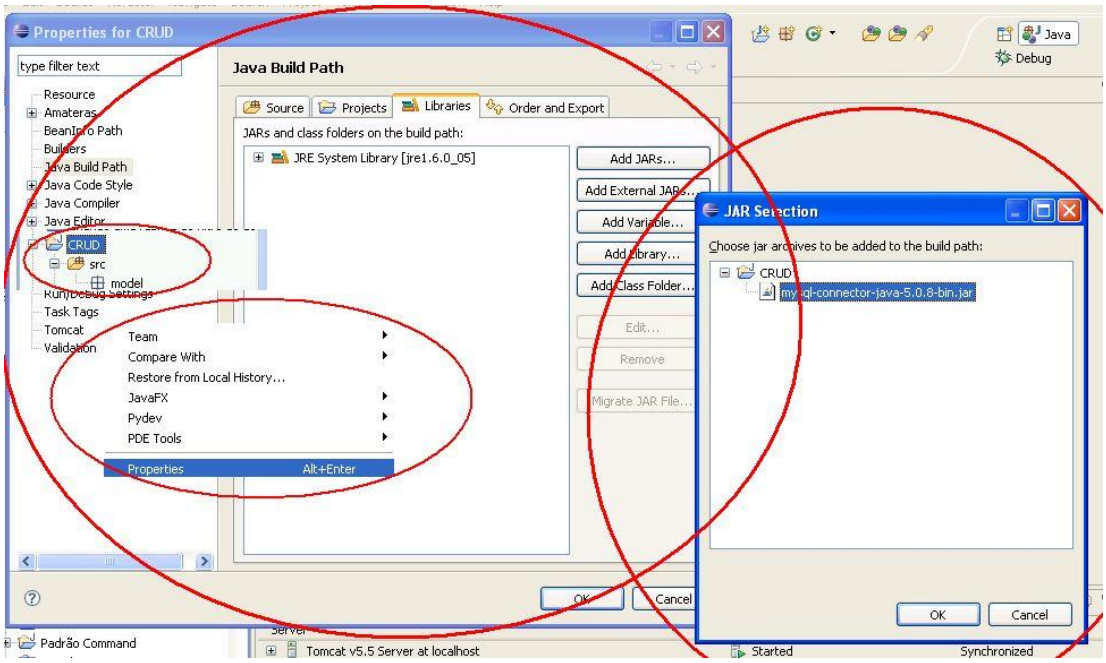
É importante a divisão de papéis: quem é Model, Control e View, assim podemos usar o que for definido hoje nos projetos futuros.
Agora vamos implementar a parte de banco de dados, meio chato mais temos que fazer.

Para a conexão temos uma classe que faz a função de fábrica de conexões. Isso mesmo, o padrão Factory que usamos em um projeto muito antigo, mas serve para esse da mesma forma. Obeserve que para outros bancos de dados você deve alterar a fábrica, essa só tem a parte do MySql.
Essa classe foi alocada no pacote "banco", então não perca tempo e crie o pacote "banco"(que estará em outro pacote futuramente, o dao).

```
1.  package dao.banco;
2.
3.  import java.sql.Connection;
4.  import java.sql.DriverManager;
5.  import java.sql.SQLException;
6.
7.  public class ConFactory {
8.
9.     public static final int MYSQL = 0;
10.    private static final String MySQLDriver = "com.mysql.jdbc.Driver";
11.
12.    public static Connection conexao(String url, String nome, String senha,
13.          int banco) throws ClassNotFoundException, SQLException {
14.       switch (banco) {
15.       case MYSQL:
16.          Class.forName(MySQLDriver);
17.          break;
18.       }
19.       return DriverManager.getConnection(url, nome, senha);
20.    }
21. }
```

Perceba que a classe lança um exceção quando algo da errado na conexão com o banco, e muita coisa pode dar errado!

Lembre-se de incluir o JAR no seu projeto para o driver do MySQL!!



Clique com o botão direito em cima do seu projeto, escolha propriedade. Na janela propriedades, escolha JavaBuild Path, clique em Add JARs ou Add External JARs:

Para acessar os dados usamos o padrão DAO, assim encapsulamos todo o trabalho com o banco, e nossas classes que querer usar e manipular os dados simplesmente devem conhecer nossa classe DAO, sem se preocupar com abrir conexão, fechar e inserir comandos.
Olhem nossa classe de acesso, percebam que ela permite todas operações com a tabela Pessoa(Está no pacote dao):

```
1.  package dao;
2.
3.  import java.sql.Connection;
4.  import java.sql.ResultSet;
5.  import java.sql.SQLException;
6.  import java.sql.Statement;
7.  import java.util.Vector;
8.
9.  import javax.swing.JOptionPane;
10.
11. import model.Pessoa;
12. import dao.banco.ConFactory;
13.
14. public class DaoPessoa {
15.    // Configura essas variáveis de acordo com o seu banco
16.    private final String URL = "jdbc:mysql://localhost/javafx_crud",
17.          NOME = "root", SENHA = "senha";
18.
19.    private Connection con;
20.    private Statement comando;
21.
22.    public void apagar(String rg) {
23.       conectar();
24.       try {
25.          comando
26.                .executeUpdate("DELETE FROM pessoa WHERE rg = '" + rg
27.                      + "';");
```

```
28.       } catch (SQLException e) {
29.          imprimeErro("Erro ao apagar pessoas", e.getMessage());
30.       } finally {
31.          fechar();
32.       }
33.    }
34.
35.    public Vector<Pessoa> buscarTodos() {
36.       conectar();
37.       Vector<Pessoa> resultados = new Vector<Pessoa>();
38.       ResultSet rs;
39.       try {
40.          rs = comando.executeQuery("SELECT * FROM pessoa");
41.          while (rs.next()) {
42.             Pessoa temp = new Pessoa();
43.             // pega todos os atributos da pessoa
44.             temp.setRg(rs.getString("rg"));
45.             temp.setNome(rs.getString("nome"));
46.             temp.setIdade(rs.getInt("idade"));
47.             temp.setCidade(rs.getString("cidade"));
48.             temp.setEstado(rs.getString("estado"));
49.             resultados.add(temp);
50.          }
51.          return resultados;
52.       } catch (SQLException e) {
53.          imprimeErro("Erro ao buscar pessoas", e.getMessage());
54.          return null;
55.       }
56.    }
57.
58.    public void atualizar(Pessoa pessoa) {
59.       conectar();
60.       String com = "UPDATE pessoa SET nome = '" + pessoa.getNome()
61.             + "', idade =" + pessoa.getIdade() + ", cidade = '"
62.             + pessoa.getCidade() + "', estado ='" + pessoa.getEstado()
63.             + "' WHERE  rg = '" + pessoa.getRg() + "';";
64.       System.out.println("Atualizada!");
65.       try {
66.          comando.executeUpdate(com);
67.       } catch (SQLException e) {
68.          e.printStackTrace();
69.       } finally {
70.          fechar();
71.       }
72.    }
73.
74.    public Vector<Pessoa> buscar(String rg) {
75.       conectar();
76.       Vector<Pessoa> resultados = new Vector<Pessoa>();
77.       ResultSet rs;
78.       try {
79.          rs = comando.executeQuery("SELECT * FROM pessoa WHERE rg LIKE '"
80.                + rg + "%';");
81.          while (rs.next()) {
82.             Pessoa temp = new Pessoa();
83.             // pega todos os atributos da pessoa
84.             temp.setRg(rs.getString("rg"));
85.             temp.setNome(rs.getString("nome"));
86.             temp.setIdade(rs.getInt("idade"));
87.             temp.setCidade(rs.getString("cidade"));
88.             temp.setEstado(rs.getString("estado"));
89.             resultados.add(temp);
90.          }
91.          return resultados;
92.       } catch (SQLException e) {
93.          imprimeErro("Erro ao buscar pessoa", e.getMessage());
```

```
94.          return null;
95.       }
96.
97.    }
98.
99.    public void insere(Pessoa pessoa){
100.             conectar();
101.             try {
102.                comando.executeUpdate("INSERT INTO Pessoa VALUES('"
103.                      + pessoa.getRg() + "', '" + pessoa.getNome() + "',"
104.                      + pessoa.getIdade() + ",'" + pessoa.getCidade() + "','"
105.                      + pessoa.getEstado() + "')");
106.                System.out.println("Inserida!");
107.             } catch (SQLException e) {
108.                imprimeErro("Erro ao inserir Pessoa", e.getMessage());
109.             } finally {
110.                fechar();
111.             }
112.          }
113.
114.          private void conectar() {
115.             try {
116.                con = ConFactory.conexao(URL, NOME, SENHA, ConFactory.MYSQL);
117.                comando = con.createStatement();
118.                System.out.println("Conectado!");
119.             } catch (ClassNotFoundException e) {
120.                imprimeErro("Erro ao carregar o driver", e.getMessage());
121.             } catch (SQLException e) {
122.                imprimeErro("Erro ao conectar", e.getMessage());
123.             }
124.          }
125.
126.          private void fechar() {
127.             try {
128.                comando.close();
129.                con.close();
130.                System.out.println("Conexão Fechada");
131.             } catch (SQLException e) {
132.                imprimeErro("Erro ao fechar conexão", e.getMessage());
133.             }
134.          }
135.
136.          private void imprimeErro(String msg, String msgErro) {
137.             JOptionPane.showMessageDialog(null, msg, "Erro crítico", 0);
138.             System.err.println(msg);
139.             System.out.println(msgErro);
140.             System.exit(0);
141.          }
142.       }
```

Eu, particularmente, costumo criar classes que herdam de um pai(ou implementam uma interface) DAO, para evitar repetir código, mas como temos somente uma tabela(entidade para alguns) iremos usar esse DAO específico, o DaoPessoa.

Quanto ao código não temos segredo. Primeiro conectamos, com nossa conexão criamos um comando. Depois do comando enviamos uma String que corresponde ao comando de banco de dados(como o DELETE, INSERT, SELECT e UPDATE). Perceba que para cada comando tem um tratamento de exceção, assim identificamos o erro "na lata"!

Os nossos métodos de busca retornam um Vector de pessoas, poderíamos utilizar outra estutura de armazenamento múltiplo(Collection), como o ArrayList...

Para mostrar o erro usamos um método, assim encapsulamos como o erro será exibido(pois podemos não querer sair do programa, ou exibir o erro no console ou uma janela de diálogo...), sem problemas!

Agora chegou a hora mais interessante, vamos testar o que fizemos até agora:

```
1.  import java.util.Iterator;
2.  import java.util.Vector;
3.
4.  import model.Pessoa;
5.  import dao.DaoPessoa;
6.
7.  public class Teste {
8.
9.     public static void main(String[] args) {
10.       Pessoa pessoa = new Pessoa();
11.       DaoPessoa daoPessoa = new DaoPessoa();
12.       pessoa.setNome("José da Silva");
13.       pessoa.setRg("12345678X");
14.       pessoa.setIdade(20);
15.       pessoa.setEstado("SP");
16.       pessoa.setCidade("São Paulo");
17.
18.       daoPessoa.insere(pessoa);
19.
20.       Vector<Pessoa> resultado = daoPessoa.buscar("12345678X");
21.
22.       for (Iterator<Pessoa> iterator = resultado.iterator(); iterator
23.             .hasNext();) {
24.          Pessoa p = iterator.next();
25.          System.out.println("Pessoa Encontrada: " + p.getNome());
26.       }
27.       pessoa.setNome("José da Silva Sauro");
28.
29.       daoPessoa.atualizar(pessoa);
30.
31.       resultado = daoPessoa.buscar("12345678X");
32.
33.       for (Iterator<Pessoa> iterator = resultado.iterator(); iterator
34.             .hasNext();) {
35.          Pessoa p = iterator.next();
36.          System.out.println("Pessoa Encontrada: " + p.getNome());
37.       }
38.
39.       daoPessoa.apagar("12345678X");
40.
41.       resultado = daoPessoa.buscar("12345678X");
42.
43.       if (resultado.size() == 0) {
44.          System.out.println("Pessoa foi apagada com sucesso");
45.       } else {
46.          System.out.println("A pessoa permanece no banco de dados");
47.       }
48.
49.    }
50. }
```

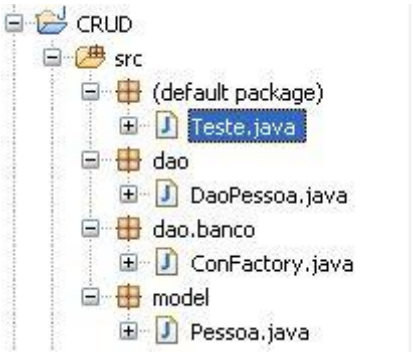
Nada demais neste teste, estamos inserindo uma pessoa, depois buscando para ver se ela foi inserida mesmo! Em seguida atualizamos o nome e buscamos para ver se aconteceu a atualização, por fim apagamos a pessoa e tentamos realizar a busca para ter certeza que nosso dado foi apagado mesmo!
Ao rodar essa classe você deve obter o seguinte resultado no console:

Quote:

Conectado!
Inserida!
Conexão Fechada
Conectado!
Pessoa Encontrada: José da Silva
Conectado!
Atualizada!
Conexão Fechada
Conectado!
Pessoa Encontrada: José da Silva Sauro
Conectado!
Conexão Fechada
Conectado!
Pessoa foi apagada com sucesso

Se você não conseguiu, volte e tente fazer tudo desde o começo!

Chegamos ao fim desse primeiro tutorial, nossas classes ficaram como mostrado abaixo:



Quem quiser terminar o que foi implementado até agora e mostrar para o pessoal, fique a vontade! Mostre suas habilidades em alguma das tecnologias abaixo, usando esse DAO:

-Java Swing
-JSP
-JavaFX
-WebServices(Oferecer o acesso como um serviço)
-Uso com algum FrameWork(Struts, JSF...)
-Outros...