

EducaCiência FastCode

Fala Galera,

Neste artigo, abordaremos um tema muito interessante.

- Artigo: 09/2020 Data: Fevereiro/2020
- Público Alvo: Desenvolvedores – Iniciantes ao Avançado
- Tecnologia: Arquitetura Java
- Tema: Padrão MVC em uma aplicação Desktop
- Link: <https://github.com/perucello/DevFP>

Desta vez, escolhi um tema interessante, vamos sair um pouco de códigos e entraremos em Arquitetura!

O tema que escolhi, é justamente o que eu mais atuo, que é o Padrão MVC.

Antes de mais nada, vamos dar uma ênfase no que realmente é esta Arquitetura !
Vamos dar uma olhada rapidinha no que o Wikipedia nos diz:

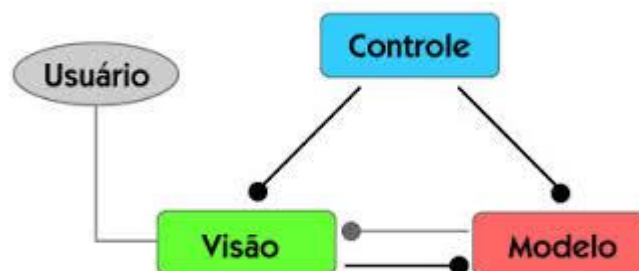
*“ **MVC** é o acrônimo de Model-View-Controller (em português: Arquitetura Modelo-Visão-Controle - MVC) é um padrão de projeto de software ou padrão de arquitetura de software formulado na década de 1970, focado no reuso de código e a separação de conceitos em três camadas interconectadas, onde a apresentação dos dados e interação dos usuários (front-end) são separados dos métodos que interagem com o banco de dados (back-end).*

Normalmente usado para o desenvolvimento de interfaces de usuário que divide uma aplicação partes (camadas/componentes) interconectadas. Isto é feito para separar representações de informação internas dos modos como a informação é apresentada para e aceita pelo usuário, levando ao desenvolvimento paralelo de maneira eficiente. “

“ Tradicionalmente usado para interfaces gráficas de usuário (GUIs), esta arquitetura tornou-se popular para projetar aplicações web e até mesmo para aplicações móveis, para desktop e para outros clientes.

Linguagens de programação populares como Java, C#, Ruby, PHP e outras possuem frameworks MVC populares que são atualmente usados no desenvolvimentos de aplicações web. “

Fonte: <https://pt.wikipedia.org/wiki/MVC>



Dito isso, vamos ao que interessa !

Esta arquitetura é basicamente dividida em três camadas sendo Model, View e Controller, ou seja, Modelo, Visão e Controladora.

Resumindo:

Camada Model (Modelo) – é a camada que “conversa” com a camada View e com a camada Controller, é nela que encontramos a regra de negócio, a lógica e as funções da nossa aplicação!

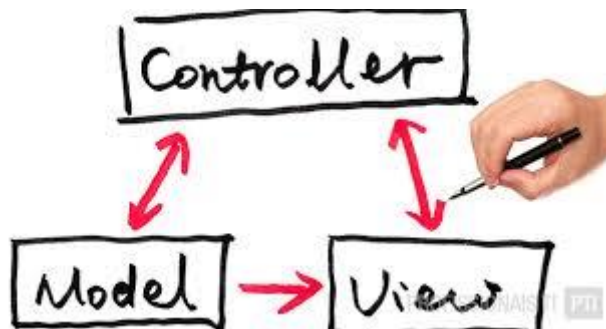
Camada View (Visão) – é nesta camada que mostramos a saída dos dados, ou seja, a interface que o operador ou quem for usufruir do Sistema “verá”, é nessa camada que podemos dizer que ocorre a interação entre o Operador e o “Sistema” (Controller).

Camada Controller (Controladora) – é vamos dizer o “pilar” da aplicação, faz a mediação entre o que o usuário irá “ver” (View) e o que o usuário irá “manipular” (Model), enfim, o Controller envia as ações para o Model e apresenta pela View ao usuário.

Interação dos componentes

Além de dividir a aplicação em três tipos de componentes, o desenho MVC define as interações entre eles.

- O **Controlador (controller)** envia comandos para o modelo para atualizar o seu estado (por exemplo, editando um documento). O controlador também pode enviar comandos para a visão associada para alterar a apresentação da visão do modelo (por exemplo, percorrendo um documento).
- Um **modelo (model)** armazena dados e notifica suas visões e controladores associados quando há uma mudança em seu estado. Estas notificações permitem que as visões produzam saídas atualizadas e que os controladores alterem o conjunto de comandos disponíveis. Uma implementação *passiva* do MVC monta estas notificações, devido a aplicação não necessitar delas ou a plataforma de software não suportá-las.
- A **visão (view)** gera uma representação (Visão) dos dados presentes no modelo solicitado, fazendo a exibição dos dados, sendo ela por meio de um html ou xml.



Agora que definimos de maneira direta o que vem a ser o Padrão de Arquitetura MVC , vamos para os códigos onde certamente você entenderá melhor !

No Nosso projeto, vamos fazer uma tela em que traremos um cadastro já realizado previamente. Não iremos abordar como Inserimos, como Pesquisamos, como filtramos os dados, daremos uma ênfase neste ponto em um outro artigo, dessa vez vamos colocar em prática o que realmente o Padrão MVC de arquitetura faz.

Criando Banco de Dados e inserindo dados manualmente

```

1 • create database db_Artigo9;
2 • use db_Artigo9;
3
4 • create table tb_usuario
5   (
6     id int not null auto_increment,
7     Nome varchar(50),
8     Sobrenome varchar(15),
9     primary key (id)
10  ) engine = InnoDB;
11
12 • insert into tb_usuario values
13   ('1', 'Clemente', 'Fulano');
14
15 • select * from tb_usuario;
16

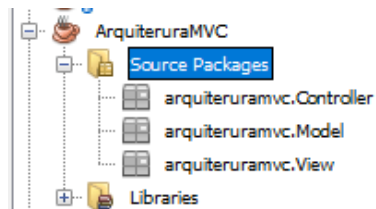
```

id	Nome	Sobrenome
1	Clemente	Fulano

Pronto, nosso Banco de Dados criado, vamos ao Netbeans11 e prepararemos nossa Camada de Arquitetura MVC !

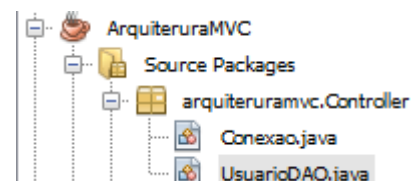
Vamos criar nossos “Pacotes”:

- ✓ **Model**
- ✓ **View**
- ✓ **Controller**

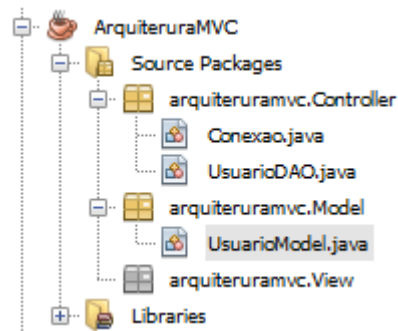


Como nós vimos, o Controller é nosso pilar do Sistema, fará a mediação entre o usuário e o Sistema, para isso, precisamos fazer com que nosso Sistema se comunique com o Banco de Dados, e é justamente aqui, no Controller que criaremos a “classe” **Conexao** responsável por isso!

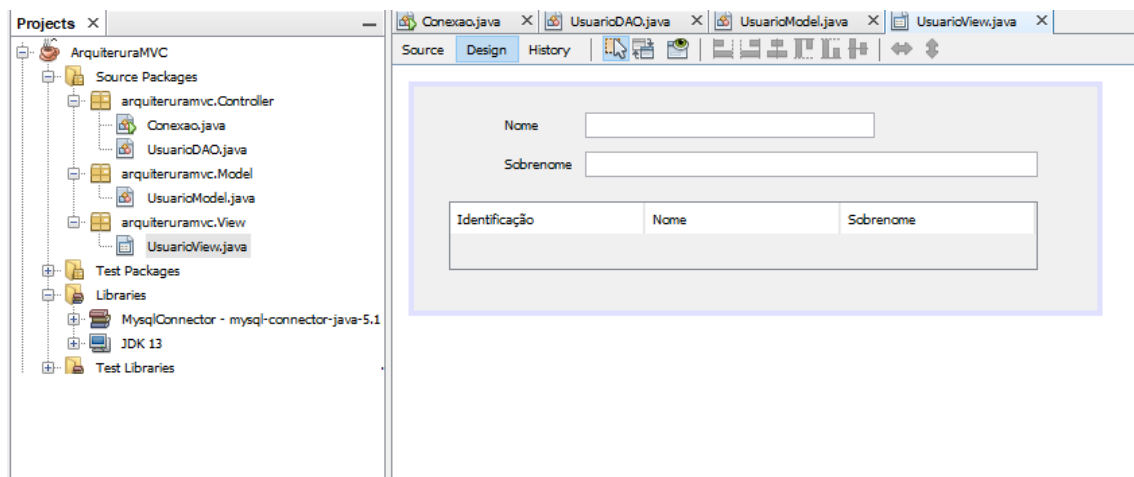
É no Controller também que haverá a requisição da interatividade , portanto, é no Controller que também criaremos nossa “classe” **UsuarioDAO**.



Agora, vamos ao nosso “Model”, é aqui a camada responsável por fazer nossa aplicação “conversar” com o Banco de Dados e com o Usuario, portanto, vamos criar nossa “classe” **UsuarioModel**.



Agora, restou apenas criar nossa “View” que é a tela que o usuário verá com os dados ao serem apresentados, sendo assim, criaremos nossa FORM chamado **UsuarioView**. Vamos criar uma tela em que trará o Nome e o Sobrenome que foram inseridos no Banco de Dados manualmente mais acima.



Agora nos resta programar, vamos lá !

Mas antes , não se esqueça de importar a Biblioteca do MySql no projeto, nós já escrevemos uma artigo anteriormente sobre este tema.

```

1  package arquiteturaMVC.Controller;
2
3  import java.sql.Connection;
4  import java.sql.DriverManager;
5  import java.sql.SQLException;
6
7  public class Conexao {
8      private static final String DATABASE="db_Artigo9";
9      private static final String HOST="jdbc:mysql://localhost:3306/db_Artigo9";
10     private static final String DRIVER="com.mysql.jdbc.Driver";
11     private static final String URL="jdbc:mysql://localhost:3306/db_Artigo9";
12     private static final String USER="root";
13     private static final String PWD="";
14
15     public static Connection Conectar(){
16         //Metodo conectar
17         try{
18             Class.forName(DRIVER);
19             return DriverManager.getConnection(URL, USER, PWD);
20         }
21         catch (ClassNotFoundException | SQLException e) {
22             System.out.println("ERRO: " + e.getMessage());
23             return null;
24         }
25     }
26
27     //Método Desconectar
28     public static void Desconectar (Connection con){
29         try{
30             if (con != null){
31                 con.close();
32             }
33         }
34         catch (SQLException e){
35             System.out.println("ERRO: " + e.getMessage());
36         }
37     }
38
39     //Metodo Main
40     public static void main(String[] args){
41         if (Conectar() != null){
42             System.out.println("Conexão realizada com sucesso ao Banco de Dados!");
43         }
44     }
45 }

```

Output - ArquiteturaMVC (run)

```

run:
Conexão realizada com sucesso ao Banco de Dados!
BUILD SUCCESSFUL (total time: 1 second)

```

Feito isso, vamos aos códigos do nosso **UsuarioModel** do pacote Model, é note que teremos nossos métodos Getters e Setters.

Uma maneira rápida de criar os Métodos Getters and Setters é utilizarmos do atalho “Alt+Insert” – para o NetBeans 11

```
package arquiteturaamvc.Model;

public class UsuarioModel {
    public int id;
    public String Nome;
    public String Sobrenome;

    public UsuarioModel() {
        this.id = id;
        this.Nome = Nome;
        this.Sobrenome = Sobrenome;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return Nome;
    }

    public void setName(String Nome) {
        this.Nome = Nome;
    }

    public String getSobrenome() {
        return Sobrenome;
    }

    public void setSobrenome(String Sobrenome) {
        this.Sobrenome = Sobrenome;
    }

    @Override
    public String toString() {
        return super.toString(); //To change body of generated methods, choose Tools | Templates.
    }
}
```

Feito isso, vamos aos códigos do nosso **UsuarioDAO** do pacote Controller, aqui como vemos, receberemos a instrução do Banco de Dados, neste caso criamos apenas o método Listar cujo qual trará pelo Select nossa informação.

```
package arquiteruramvc.Controller;

import arquiteruramvc.Model.UsuarioModel;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class UsuarioDAO {
    public static List<UsuarioModel> list() {
        throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
    }
    private final Connection con;
    private PreparedStatement cmd;

    public UsuarioDAO() {
        this.con = Conexao.Conectar();
    }

    public List<UsuarioModel> listar() {
        try {
            String sql = "select * from tb_usuario";
            cmd = con.prepareStatement(sql);
            ResultSet rs = cmd.executeQuery();

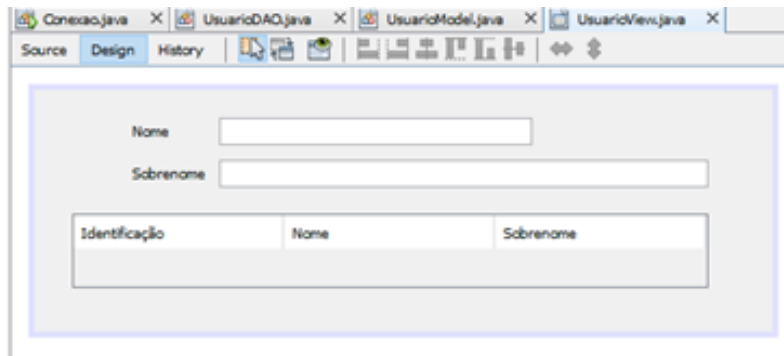
            List<UsuarioModel> lista = new ArrayList<>();
            while(rs.next()) {

                UsuarioModel usr = new UsuarioModel();
                usr.setId(rs.getInt("id"));
                usr.setNome(rs.getString("Nome"));
                usr.setSobrenome(rs.getString("Sobrenome"));

                lista.add(usr);
            }
            return lista;
        } catch (SQLException e) {
            System.out.println("ERRO: " + e.getMessage());
            return null;
        } finally {
            Conexao.Desconectar(con);
        }
    }
}
```

Finalmente, vamos apresentar os dados ao usuário, porém também precisamos preparar a interface, vamos nessa !

Nesta fase, ou seja, nosso FORM foi montado e traremos os comandos que farão ou melhor, serão responsáveis pela interatividade do operador com o Sistema.



Veja que no nosso FORM trouxemos uma caixa de texto para o Nome e Sobrenome e nossa Tabela, sendo que na nossa tabela trará nosso Select do Banco de Dados e o campo Nome e Sobrenome também informará os mesmos dados após selecionados.

O Projeto é simples, porém nosso intuito é apenas mostrar que a arquitetura MVC é simples e fácil de se usar e um dos benefícios maiores é a reutilização dos códigos.

Resumindo o código que apresentaremos a seguir, vou tentar detalhar:

Fizemos algumas importações importantes:

```
import arquiteruramvc.Controller.UsuarioDAO; -> nessa importação, acionaremos o Pacote Controller e a Classe UsuarioDAO
import arquiteruramvc.Model.UsuarioModel; -> nessa importação, acionaremos o Pacote Model e a classe UsuarioModel
import java.util.List; -> utilizamos deste utilitário para trabalharmos com nossa Lista
import javax.swing.table.DefaultTableModel; -> utilizaremos desta Swing para criarmos nossa tabela.
```

Criamos a Classe **UsuarioView** que é estendida do Java Swing e deixaremos Público nosso **UsuarioView**, claro trazendo a Inicialização dos componentes e Formulário configurado.

Depois, por segurança, deixamos privado nossa **TabResultado** onde passamos por parâmetro o objeto **tabResultad** trazendo o **SelectRow** e logo a seguir criamos uma cláusula **IF** para os campos **txtNome** e **txtSobrenome** criados no nosso FORM.

Feito isso configuramos nosso método **configurarFormulário** passando com parâmetro **"preencherTabela"** fazendo uma importação de um "novo" **UsuarioDao** (classe **UsuarioDAO** do Pacote Controller que importamos mais acima) e trazendo o método **Listar** de maneira automática para nós.

Logo abaixo temos o **"preencherTabela"** onde preencherá nossa tabela com os objetos recebidos, lembrando que no **PacoteModem** temos todos nossos Getters and Setters, sendo assim, nesse ponto em caso da nossa lista ter algum dado, no nosso caso temos 1, ele trará automaticamente para nós esta informação.


```

1 package arquiteururamvc.View;
2
3 import arquiteururamvc.Controller.UsuarioDAO;
4 import arquiteururamvc.Model.UsuarioModel;
5 import java.util.List;
6 import javax.swing.table.DefaultTableModel;
7
8 public class UsuarioView extends javax.swing.JFrame {
9     public UsuarioView() {
10         initComponents();
11         configurarFormulario();
12     }
13     @SuppressWarnings("unchecked")
14     Generated Code
15
16
17
18
19
20
21
22 private void tabResultadoMouseClicked(java.awt.event.MouseEvent evt) {
23     int linha = tabResultado.getSelectedRow();
24     if (linha >= 0) {
25         txtNome.setText(tabResultado.getValueAt(linha, 1).toString());
26         txtSobrenome.setText(tabResultado.getValueAt(linha, 2).toString());
27     }
28 }
29
30
31
32 private void txtNomeActionPerformed(java.awt.event.ActionEvent evt) {
33     // TODO add your handling code here:
34 }
35
36 public static void main(String args[]) throws ClassNotFoundException, InstantiationException, IllegalAccessException {
37     try {
38         for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
39             if ("".equals(info.getName())) {
40                 javax.swing.UIManager.setLookAndFeel(info.getClassName());
41                 break;
42             }
43         }
44     } catch (javax.swing.UnsupportedLookAndFeelException ex) {
45         java.util.logging.Logger.getLogger(UsuarioView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
46     }
47     java.awt.EventQueue.invokeLater(() -> {
48         new UsuarioView().setVisible(true);
49     });
50 }
51
52 // Variables declaration - do not modify
53 private javax.swing.JLabel lblNome;
54 private javax.swing.JLabel lblSobrenome;
55 private javax.swing.JPanel pnMenu;
56 private javax.swing.JScrollPane spTabResultado;
57 private javax.swing.JTable tabResultado;
58 private javax.swing.JTextField txtNome;
59 private javax.swing.JTextField txtSobrenome;
60 // End of variables declaration
61
62
63 private void configurarFormulario(){
64     this.setResizable(true);
65     this.setLocationRelativeTo(null);
66
67     preencherTabela(new UsuarioDAO().listar());
68 }
69
70 private void configurarTabela(){
71     DefaultTableModel m = new DefaultTableModel() {
72         @Override
73         public boolean isCellEditable(int row, int column){
74             return false;
75         }
76     };
77     m.addColumn("Id");
78     m.addColumn("Nome");
79     m.addColumn("Sobrenome");
80     tabResultado.setModel(m);
81 }
82
83 private void preencherTabela(List<UsuarioModel> lista){
84     if (lista.size() > 0) {
85         configurarTabela();
86         DefaultTableModel m = (DefaultTableModel) tabResultado.getModel();
87         for (UsuarioModel usr : lista) {
88             m.addRow(new Object[] {
89                 usr.getId(), usr.getNome(), usr.getSobrenome()
90             });
91         }
92         tabResultado.setModel(m);
93     }
94 }
95
96 }
97
98 }
99

```

Vamos executar apenas para mostrar que nosso Sistema está OK com o padrão MVC que criamos.

Banco de Dados

```

4 • create table tb_usuario
5 • (
6 •   id int not null auto_increment,
7 •   Nome varchar(50),
8 •   Sobrenome varchar(15),
9 •   primary key (id)
10 • ) engine = InnoDB;
11
12 • insert into tb_usuario values
13 • ('1', 'Clemente', 'Fulano');
14
15 • select * from tb_usuario;
16
17
18 • select Id, Nome, Sobrenome from tb_usuario where id = 1;

```

Result Grid

Id	Nome	Sobrenome
1	Clemente	Fulano

Sistema

```

1 • class UsuarioDAO {
2 •   public static List<UsuarioModel> list() {
3 •     throw new UnsupportedOperationException();
4 •   }
5 •   private final Connection conn;
6 •   private PreparedStatement stmt;
7 •
8 •   public UsuarioDAO(Connection conn) {
9 •     this.conn = conn;
10 •   }
11 •
12 •   public List<UsuarioModel> list() {
13 •     try {
14 •       String sql = "select * from tb_usuario";
15 •       stmt = conn.prepareStatement(sql);
16 •       ResultSet rs = stmt.executeQuery();
17 •
18 •       List<UsuarioModel> lista = new ArrayList<>();
19 •       while(rs.next()) {
20 •         UsuarioModel usr = new UsuarioModel();
21 •         usr.setId(rs.getInt("id"));
22 •         usr.setNome(rs.getString("nome"));
23 •         usr.setSobrenome(rs.getString("sobrenome"));
24 •         lista.add(usr);
25 •       }
26 •       return lista;
27 •     } catch (SQLException e) {
28 •       throw new RuntimeException(e);
29 •     }
30 •   }
31 • }

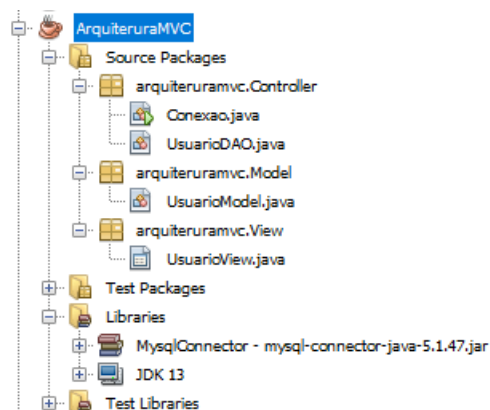
```

Artigo 09 - EducaCiência FastCode

Nome: Clemente

Sobrenome: Fulano

Id	Nome	Sobrenome
1	Clemente	Fulano



Até mais !

Espero ter ajudado !