Padrão MVC com Interface Gráfica (swing)

Kalil de Oliveira
Faculdade Senac Criciúma

Parte 1

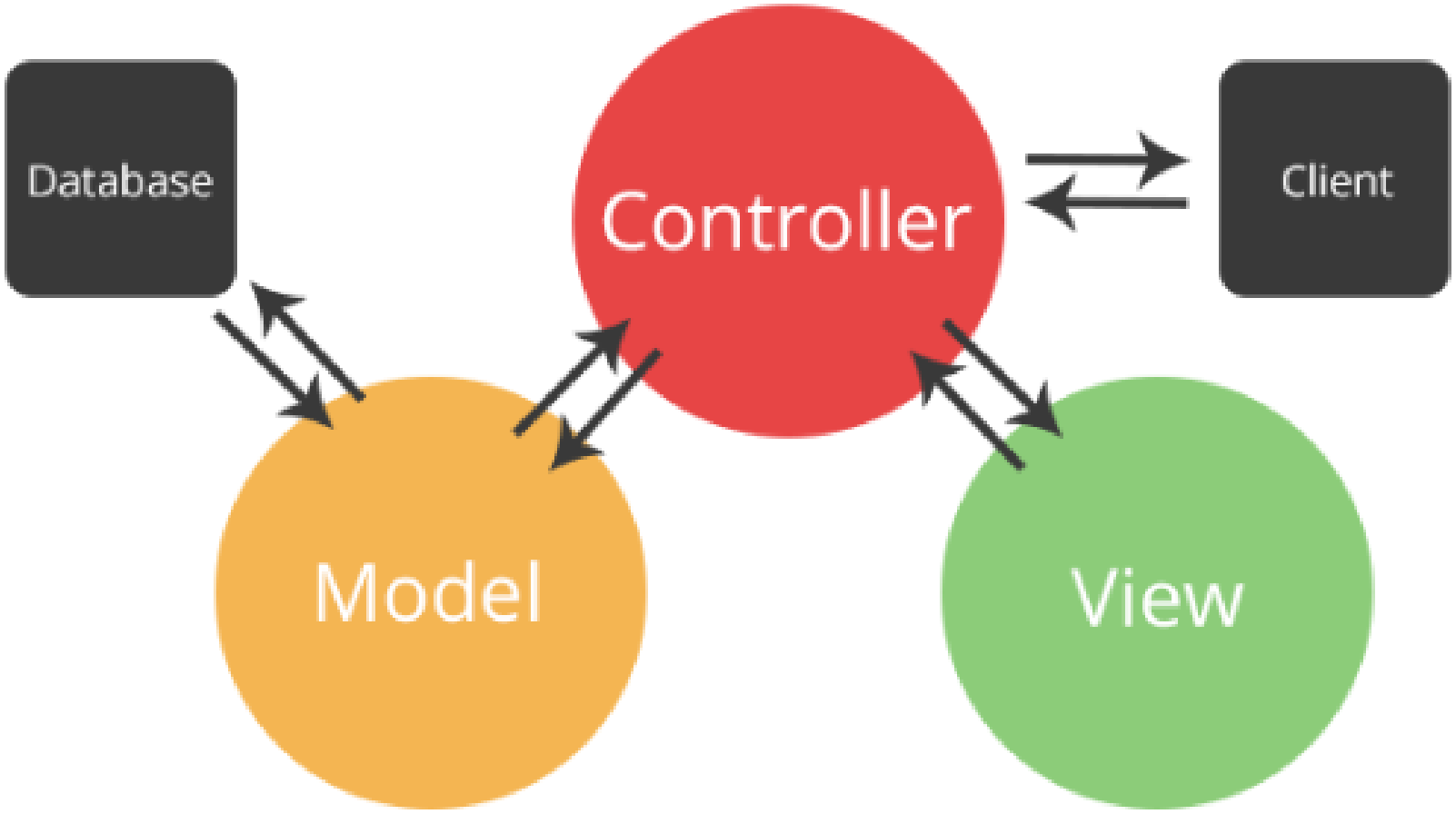
Sobre o MVC

❖ Introdução

- Como manter dados e interface com usuário desacoplados?
- Como dividir o projeto para desenvolvimento em equipe?
- O MVC (Model View Controller) fornece uma estratégia que desacoplam completamente o sistema subjacente da interface com o usuário

❖ Conceitos

- Modelo/Model = sistema
- Visão/View = exibe o conteúdo
- Controle/Controller = processa as entradas do usuário
- Cada parte tem seu conjunto próprio de responsabilidades


Database
Controller
Client
Model
View

❖ Model

- Fornece acesso a funcionalidade básica do sistema
  - Contém as informações de estado do sistema
- É a camada que gerencia o comportamento básico e o estado do sistema. Responde as consultas sobre seu estado a partir do modo de visualização e do controlador e aos pedidos de mudança de estado do controlador

❖ View

- Apresenta os dados e receber entrada do usuário
- Pode ser no formato texto, com Scanner, ou numa interface gráfica com a biblioteca swing, ou ainda com um formulário desenvolvido com html

❖ Controller

- Recebe eventos da View, interpreta-os e aciona as operações correspondentes no Modelo. Também atualiza a View com base nas mudanças no Modelo.
- O Controller facilita a separação de preocupações, torna o código mais modular e promove a reutilização.

Parte 2

Nosso exemplo

❖ Requisitos

- O sistema em modo gráfico será composto por um formulário com duas entradas e um botão de OK.
- O usuário informará seu nome na primeira entrada e sobrenome na segunda e o sistema retornará uma saudação exibindo a concatenação.

❖ A classe NomeModel

```
class NomeModel {

   private String nome, snome;

   public String getNome() { return nome; }

    public void setNome(String nome) { this.nome = nome; }

    public String getSnome() { return snome;   }

    public void setSnome(String snome) { this.snome = snome; }

    public String getCompleto() { return nome + " " + snome;  }}
```

❖ Class NomeView : imports

```
import javax.swing.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;
```

❖ NomeView : atributos

```
class NomeView {

private JTextField txtNome, txtSnome;

...
```

❖ NomeView : instanciar a janela

```
public NomeCompletoView() {

JFrame janela = new JFrame(“Bem-vindo");

janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

...
```

❖ NomeView : instanciar um container

```
BOX caixa = Box.createVerticalBox();

...
```

❖ NomeView : rótulos e caixas de texto

```
JLabel lblNome = new JLabel("Nome: ");

txtNome = new JTextField(20);
```

```
JLabel lblSnome = new JLabel("Sobrenome: ");

txtSnome = new JTextField(20);

...
```

❖ NomeView : botão OK

JButton btnOK = new JButton(«OK»);

...

❖ NomeView : add elementos ao container

```
caixa.add(lblNome);

caixa.add(txtNome);

caixa.add(lblSnome);

caixa.add(txtSnome);

caixa.add(btnOK);

...
```

❖ NomeView : finalizando a janela

```
janela.add(caixa);

janela.setSize(300, 200);

janela.setVisible(true);

...
```

❖ NomeView : métodos get

```
public String getNome() {
 return txtNome.getText();  }
public String getSnome() {
return txtSnome.getText();   }
...
```

❖ NomeView : evento para o btnOK

```
public void setBtnOKEscuta(ActionListener escuta) {
        txtNome.addActionListener(escuta);
        txtSnome.addActionListener(escuta);      }
...
```

❖ Classe NomeController: atributos

```
class NomeController {

private NomeModel model;

private NomeView view;

...
```

❖ NomeController: construtor

```
public NomeController(NomeModel model, NomeView view)
{    this.model = model;   this.view = view;
        view. setBtnOKEscuta(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
                atualizaView();
                showNome ();            }      });    }
```

❖ NomeController: métodos privados

```
private void atualizaView() {

model.setNome(view.getNome());

model.setSnome(view.getSnome());   }

private void showNome() {

JOptionPane.showMessageDialog(null, “Bem-vindo: " +

model.getCompleto());   }
```

Parte 3

Lista de Desafios

❖ Desafio 01

Crie, a patir do que aprendeu sobre MVC hoje, um pequeno sistema de agenda telefônica, que cadastre nome, sobrenome e telefone.

❖ Desafio 02

Aproveite o código desta aula para escrever um programa que leia o código de itens pedidos por um cliente de uma lanchonete (você pode criar a lista: cachorro quente, hambúrguer, refrigerante...) e retorne um relatório com o total geral do pedido.

❖ Desafio 03

Amplie o código desta aula para criar um pequeno Quiz com 5 perguntas. Se o jogador acertar todas, o sistema retorna "Parabéns, nome-do-usuário, você passou de fase". Caso contrário, informa "Tente novamente" e reinicia.

❖ Desafio 04

Faça, com MVC, um programa que simule uma bomba de combustível. Os atributos: tipoCombustivel, valorLitro, qtdCombutivel. Os métodos: abastecerPorValor(), abastecerPorLitro().