Aplicações de Linguagem de Programação Orientada a Objetos
Prof. Sheila Cáceres

Padrão MVC (Model View Controller)

Originalmente o Model-View-Controller foi criado como um padrão de projeto arquitetural desenvolvido para o ambiente de desenvolvimento Smalltalk, mas ele pode ser utilizado para qualquer aplicação interativa e em diferentes tipos de ambientes.
Model-View-Controller (MVC) é um padrão de arquitetura de software que tem como objetivo separar dados ou lógica de negócios (Model) da interface do usuário (View) e do fluxo da aplicação (Control). A ideia é permitir que uma mesma lógica de negócios possa ser acessada e visualizada através de várias interfaces.
Na arquitetura MVC, a lógica de negócios (chamada aqui de Modelo) não sabe de quantas nem quais interfaces com o usuário estão exibindo seu estado. Alterações feitas na interface não afetarão a manipulação dos dados, e estes poderão ser reorganizados sem alterar a interface do usuário.
O model-view-controller resolve este problema por meio da separação das tarefas de acesso aos dados e lógica do negócio da apresentação e interação com o usuário, introduzindo um componente entre os dois: o Controller.

Podemos descrever este padrão da seguinte forma:

Model
É o objeto da aplicação. Especifica a informação que a aplicação opera.

View
É a representação do Model na tela. "Renderiza" o model em uma forma especifica para a interação, geralmente uma interface de usuário.

Controller
Define o modo em que a interface reage à entrada do usuário
Processa e responde a eventos, geralmente ações do usuário, e pode invocar altero modo ações no Model quando estas se fizerem necessárias.

Modelo – CalculatorModel.java

```
// O modelo faz o calculos necessarios.
// Não sabe nada em relação às vistas

public class CalculatorModel {

	private int calculationValue;

	// O modelo realiza as operações
	public void addTwoNumbers(int firstNumber, int secondNumber){
		calculationValue = firstNumber + secondNumber;
	}

	// O modelo mantem o resultado da soma dos números ingressados na vista
	public int getCalculationValue(){
		return calculationValue;
	}
}
```

Vista – CalculatorView.java

```
// A vista mostra a interface do usuário
// Deve mostrar o estado do Modelo
// Não realiza calculos
import java.awt.event.ActionListener;
import javax.swing.*;

public class CalculatorView extends JFrame{

	private JTextField txtFirstNumber  = new JTextField(10);
	private JLabel lblAddition = new JLabel("+");
	private JTextField txtSecondNumber = new JTextField(10);
	private JButton calculateButton = new JButton("Calculate");
	private JLabel lblResult = new JLabel("Valor do Modelo");
	private JTextField txtCalcSolution = new JTextField(10);

	public CalculatorView(){
		JPanel calcPanel = new JPanel();

		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setSize(600, 200);

		calcPanel.add(txtFirstNumber);
		calcPanel.add(lblAddition);
		calcPanel.add(txtSecondNumber);
		calcPanel.add(calculateButton);
		calcPanel.add(lblResult);
		calcPanel.add(txtCalcSolution);

		this.add(calcPanel);
	}

	public int getFirstNumber(){
		return Integer.parseInt(txtFirstNumber.getText());
	}

	public int getSecondNumber(){
		return Integer.parseInt(txtSecondNumber.getText());
	}

	public int getCalcSolution(){
		return Integer.parseInt(txtCalcSolution.getText());
	}

	public void updateView(int solution){
		txtCalcSolution.setText(Integer.toString(solution));
	}

	// Se o botão é clicado, o controlador trata esse evento
	void addCalculateListener(ActionListener listenForCalcButton){
		calculateButton.addActionListener(listenForCalcButton);
	}

	// Abre um popup que contem a mensagem de erro
	void displayErrorMessage(String errorMessage){
		JOptionPane.showMessageDialog(this, errorMessage);
	}
}
```

Controlador – CalculatorController.java

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

// Coordena a interação entre a vista e o modelo

public class CalculatorController {

	private CalculatorView theView;
	private CalculatorModel theModel;

	public CalculatorController(CalculatorView theView, CalculatorModel theModel)
{
		this.theView = theView;
		this.theModel = theModel;

		// Aloca a vista um listener que será tratado no metodo
actionPerformed no Controlador
		this.theView.addCalculateListener(new CalculateListener());
	}

	class CalculateListener implements ActionListener{

		public void actionPerformed(ActionEvent e) {
			int firstNumber, secondNumber = 0;

			try{
				firstNumber = theView.getFirstNumber();
				secondNumber = theView.getSecondNumber();
				theModel.addTwoNumbers(firstNumber, secondNumber);//
modifica o modelo
				theView.updateView(theModel.getCalculationValue());//
atualiza as vistas
			}

			catch(NumberFormatException ex){
				System.out.println(ex);
				theView.displayErrorMessage("You Need to Enter 2
Integers");
			}
		}
	}
}
```

Main

```
public class MVCCalculator {
    public static void main(String[] args) {
      CalculatorView theView = new CalculatorView();
      CalculatorModel theModel = new CalculatorModel();
      CalculatorController theController=new CalculatorController(theView,theModel);
      theView.setVisible(true);
      theView.setSize(650,80);
   }
}
```

Exercícios para Entrega

1. Implementar outro modelo para o exemplo anterior que salve os dois números e o resultado em uma tabela no banco de dados

2. Adicione outra vista para o exercício anterior que tenha um layout diferente. Pode criar uma interface e fazer que as classes vista implementem dela.
   Lembre que deveram se atualizar todas as vistas.

3. Criar um projeto MVC que siga a especificação a seguir:
   Controller: Adminisra os eventos. Mantera a comunicação entre a vista e o Modelo.
   Modelo: Somente existirá uma classe Aluno com atributos nome e ra.
   Vista: A vista deverá ser um Jframe como mostrado a seguir: