UAST
Unidade Acadêmica de Serra Talhada - PE
UFRPE
Desde 2006
BSi
Bacharelado em Sistemas de Informação

MPOO

Site: https://sigs.ufrpe.br/sigaa/ava/index.jsf

Disciplina: Modelagem e Programação Orientada a Objetos (MPOO)

Profº: Richarlyson D'Emery

LISTA DE EXERCÍCIOS IX

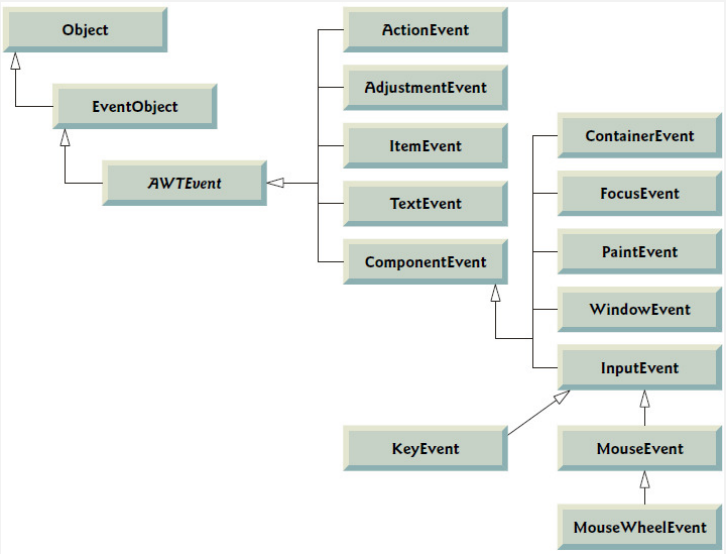
Leia atentamente as instruções gerais:

- No Eclipse crie um novo **projeto chamado br.edu.mpoo.listaIX.SeuNomeSobrenome**, o qual deverá ter **pastas de pacotes** para cada questão: questao1, questao2, e assim sucessivamente, contendo todas as respostas da lista.
- A lista envolve questões práticas.

Fique atento!

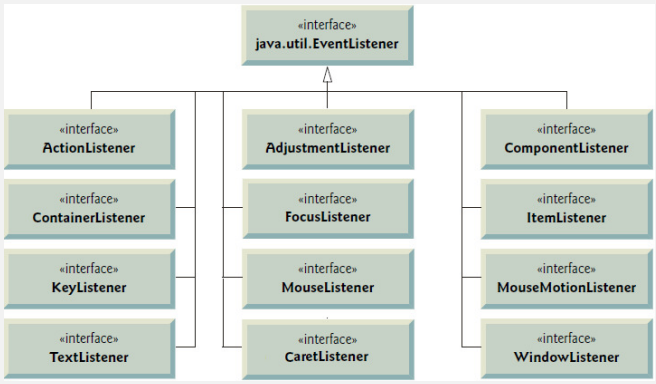
Prezado aluno, esta é a lista de exercícios é relativa ao assunto de "Componentes Gráficos", "Tratamento de Eventos" e sua organização por *architectural pattern* "Model-View-Controller".

Relembrando!



Na aula de MPOO introduzimos o assunto de componentes gráficos em que aprendemos a criar interfaces gráficas, ou simplesmente GUI's (do inglês, *Graphical User Interface*), para os nossos sistemas. Também vimos diversas possibilidades de codificação para a interação com essas telas através de Tratamento de Eventos. As informações de evento são armazenadas em um objeto de uma classe que estende de AWTEvent do pacote java.awt, como, por exemplo, ActionEvent para JButton e ItemEvent para JRadioButton. Tipos adicionais de evento que são específicos dos componentes Swing GUI são declarados no pacote javax.swing.event, como, por exemplo, ChangeEvent para JSliders e CaretEvent para JTextField.

Para cada tipo de objeto de evento, há em geral uma interface listener de eventos correspondentes. Um ouvinte de evento para um evento GUI é um objeto de uma classe que implementa uma ou mais das interfaces ouvintes de evento dos pacotes java.awt.event e javax.swing.event. Muitos dos tipos de ouvinte de evento são comuns aos componentes Swing e AWT. Esses tipos são declarados no pacote java.awt.event Os tipos de ouvinte de evento adicionais que são específicos dos componentes Swing são declarados no pacote javax.swing.event.



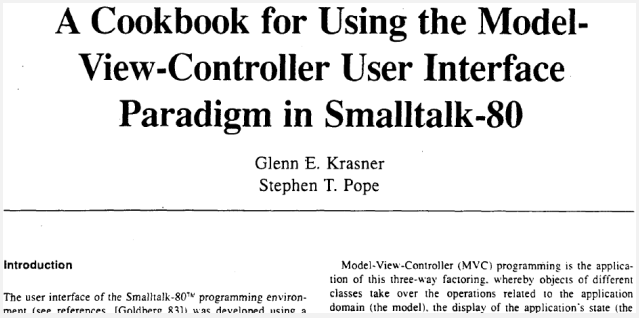
Conheça outras especializações de EventListener em: https://docs.oracle.com/javase/10/docs/api/java/util/EventListener.html

Saiba mais sobre Tratamento de Eventos em: https://docs.oracle.com/javase/tutorial/uiswing/events/intro.html

Você Sabia?

Os padrões de design permitem que os desenvolvedores projetem partes específicas dos sistemas, como abstrair instanciações de objetos ou agregar classes a estruturas maiores. Os padrões de design também promovem o acoplamento fraco entre objetos. Padrões arquitetônicos promovem o acoplamento fraco entre subsistemas. Esses padrões especificam como os subsistemas interagem um com o outro. Um dos padrões arquitetônicos mais populares é o Model-View-Controller (MVC), o qual separa a codificação em camadas.

MVC foi desenvolvido para construir interfaces gráficas em Smalltalk por Trygve Reenskaug enquanto trabalhava na Xerox PARC e descrito inicialmente nos trabalhos de Krasner e Pope (1988) e Burberck (1992).



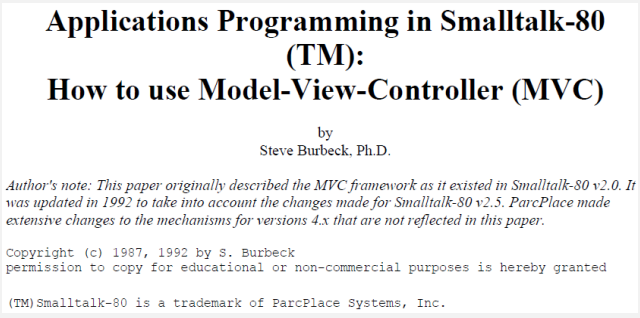
A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80

Glenn E. Krasner
Stephen T. Pope

Introduction

The user interface of the Smalltalk-80™ programming environment (see references, [Goldberg 83]) was developed using a

Model-View-Controller (MVC) programming is the application of this three-way factoring, whereby objects of different classes take over the operations related to the application domain (the model), the display of the application's state (the



Applications Programming in Smalltalk-80 (TM):
How to use Model-View-Controller (MVC)

by
Steve Burbeck, Ph.D.

Author's note: This paper originally described the MVC framework as it existed in Smalltalk-80 v2.0. It was updated in 1992 to take into account the changes made for Smalltalk-80 v2.5. ParcPlace made extensive changes to the mechanisms for versions 4.x that are not reflected in this paper.

Copyright (c) 1987, 1992 by S. Burbeck
permission to copy for educational or non-commercial purposes is hereby granted

(TM) Smalltalk-80 is a trademark of ParcPlace Systems, Inc.

MVC é composto por três tipos de objetos: (i) o "*modelo*" é o objeto de aplicação, a "*vista*" é a apresentação na tela e o "*controlador*" define a maneira como a interface do usuário reage às entradas do mesmo. Antes do MVC, os projetos de interface para o usuário tendiam em agrupar esses objetos em um único módulo, consequentemente, havia muitas linhas de código dificultando o entendimento, a manutenção e a reutilização desse código. MVC para aumentar a flexibilidade e a reutilização (GAMMA et al. 2000, p. 20[1]). O MVC tem como principal objetivo: separar dados ou lógicos de negócios (Model) da interface do usuário (View) e o fluxo da aplicação (Controller), a ideia é permitir que uma mensagem da lógica de negócios pudesse ser acessada e visualizada através de várias interfaces. Na arquitetura MVC, á lógica de negócios, ou seja, nosso Model não sabe quantas nem quais as interfaces com o usuário esta exibindo seu estado, a view não se importa de onde esta recebendo os dados, mas ela tem que garantir que sua aparência reflita o estado do modelo, ou seja, sempre que os estados do modelo mudam, o modelo notifica as view para que as mesmas atualizem-se (*sendo esta última uma perspectiva atualmente adotada pelo padrão Observer – cenas dos próximos capítulos!*).

Não é recomendado quando o sistema não tenha uma interdependência de dados, ou seja, quando se trata de um sistema embarcado, único ou muito pequeno, devido ao seu custo de desenvolvimento (aumento da separabilidade de código em classes distintas).

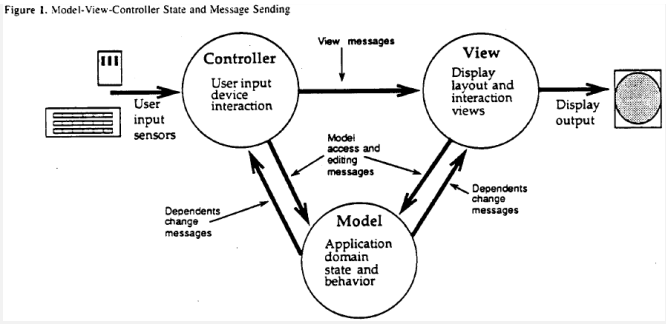
O MVC inicialmente foi desenvolvido no intuito de mapear o método tradicional de entrada, processamento e saída que os diversos programas baseados em GUI utilizavam. No padrão MVC, teríamos então o mapeamento de cada uma dessas três partes para o padrão MVC:

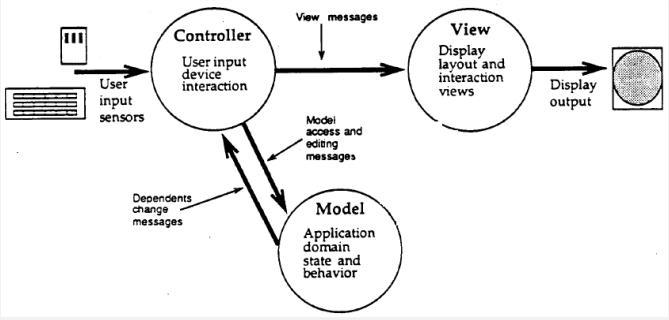
Input ➤ Processing ➤ Output
Controller ➤ Model ➤ View

Em suma: (i) model contém os dados da aplicação com a lógica da aplicação e as regras de negócios; (ii) view's são as interfaces gráficas em que o usuário interage; e (iii) controller intermedia o fluxo de informação entre view e model. Cada um com suas funcionalidades específicas.

Originalmente, Krasner e Pope (1988) descrevem o modelo (*esquerda*) em que todos os imputs são iniciados pelo Controller gerenciando a comunicação para as telas (views) e o domínio da aplicação (model). Entretanto, hoje em dia observa-se que na maioria das soluções que utilizam o MVC a remoção da comunicação entre View e Model, deixando essa comunicação por intermédio do Controlador (*direita*).

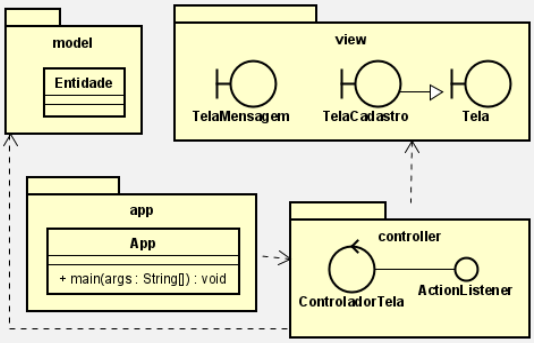
[1] GAMMA, E. et al. ***Padrões de Projeto:*** *Soluções reutilizáveis de software Orientado a Objetos.* Porto Alegre: Bookman, 2000.





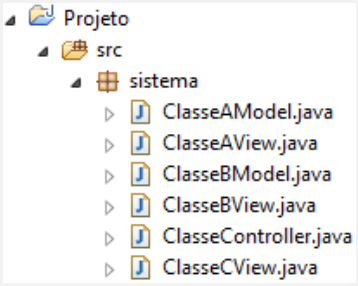
A representação MVC utiliza a mesma do padrão boundary-control-entity (BCE), em que: as entidades pertencem ao model, control ao controller e tela à boundary. Vale destacar que as diferenças entre os padrões arquiteturais se são principalmente quanto às regras de negócios do controle de BCE e controlador de MVC, uma vez que o primeiro encapsula também a lógica de negócios do caso de uso, enquanto o controlador MVC processa a entrada do usuário que seria de responsabilidade da fronteira (boundary) de BCE. O controle de BCE aumenta a separação de interesses na arquitetura ao encapsular a lógica de negócios que não está diretamente relacionada a uma entidade.

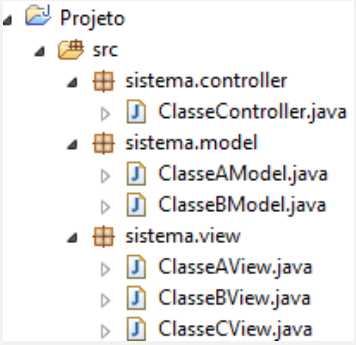
Vejamos uma exemplificação da modelagem MVC fazendo uso dos elementos BCE:



Fique atento!

Não existe uma regra quanto à organização dos arquivos no projeto Java. Alguns desenvolvedores optam por separar as camadas apenas nomeando os arquivos ou os organizando em pacotes. Também se podem variar as nomenclaturas para as camadas por sinônimos que as representam, como, por exemplo, entidade para model, tela ou gui para view, e controle ou controlador(a) para controller.





Na disciplina MPOO, vamos adotar a padronização por separação de camadas em pacotes!

Mão na Massa!

Você, aluno de MPOO, está experienciando situações-problemas do universo de desenvolvimento de software e começará a ser desafiado a solucionar novos problemas a partir de conhecimentos de POO, Componentes Gráficos, Tratamento de Eventos e de *Model-View-Controller.*

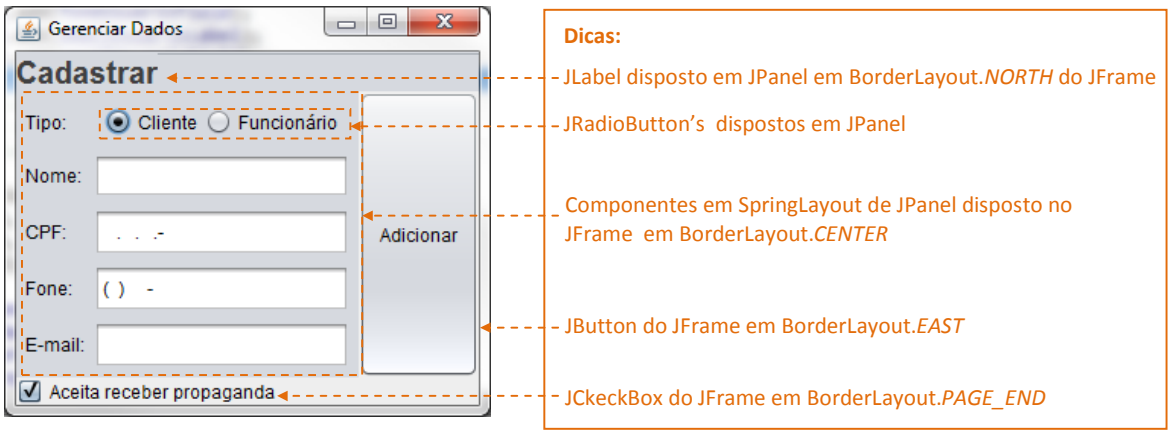
1) Na **Lista de Exercícios VIII**, foi realizada a criação de GUI's para o login de um sistema e o cadastro de usuários. Agora organize a codificação fazendo uso do padrão arquitetural MVC.

Desafio!

2) Na **Lista de Exercícios VIII**, foi solicitado a criação da GUI para o cadastro de usuários de uma empresa. Entretanto, o gerenciador de layout adotado foi o FlowLayout, tornando a interface não tão atraente! Agora veja a seção **Saiba Mais!** a seguir e melhore a GUI.

Saiba Mais!

São diversos os gerenciadores de layout que servem para determinar o tamanho e a posição dos componentes em uma GUI. Entretanto, são diversas as possibilidades, inclusive da utilização de soluções disponibilizadas em classes. Para desafiar "O Furão", seu Scrum Master o apresentou as possibilidades de configuração para JTextComponent, ao SpringLayout através de um exemplo (Fig. 4) e de um tutorial de JavaSE da Oracle disponível em https://docs.oracle.com/javase/tutorial/uiswing/layout/spring.html. Mostre que você também é capaz de aplicar soluções de layouts, em especial, a tela de cadastro de usuário (Fig. 1). Para isso, utilize a personalização pas os tipos de JTextComponent e a combinação de JPanels e gerenciadores de layout como BorderLayout e SpringLayout.



Melhorando a Interação!

Em todas as GUI's anteriores, podemos observar que as confirmações (JButton's) de interação do sistema só deveriam ser realizadas após o preenchimento de campos! Então por que deixá-las liberadas? Faria mais sentido só poderem ser acionadas quando todas as opções de preenchimento fossem realizadas, ou seja, os JButton's deveriam estar desabilitados até que as condições fossem atendidas. Mas antes de prosseguir, aprenda mais nas próximas seções sobre Programação Orientada a Objetos e a linguagem Java.

Saiba Mais!

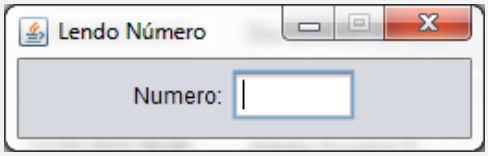
Muitas das interfaces de java.util.EventListener são compostas de diversos métodos responsáveis por tratar uma ação específica. Entretanto, muitas vezes, não precisamos utilizar todos os métodos dessas interfaces e para isso podemos simplificá-los através de classes adaptadoras.

Quando uma classe implementar uma interface, a classe terá um relacionamento é um com essa interface. Todas as subclasses diretas e indiretas dessa classe herdam essa interface. Portanto, um objeto de uma classe que estende uma classe adaptadora de evento é um objeto do tipo ouvinte de evento correspondente (por exemplo, um objeto de uma subclasse de KeyAdapter é um KeyListener).

Em Java estão disponíveis algumas classes adaptadoras, vejamos:

Classe adaptadora para java.awt.event	Interface correspondente
ComponentAdapter	ComponentListener
ContainerAdapter	ContainerListener
FocusAdapter	FocusListener
KeyAdapter	KeyListener
MouseAdapter	MouseListener
MouseMotionAdapter	MouseMotionListener
WindowAdapter	WindowListener

Exemplificando o tratamento de evento de teclado com KeyAdapter em uma arquitetura MVC:




```
//Tela.java
package sistema.view;

import java.awt.BorderLayout;
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class Tela extends JFrame {
	JLabel numeroLabel;
	JTextField numeroField;

	public Tela(){
		super("Lendo Número");
		setSize(250, 70);
		setResizable(false);
		setLayout(new FlowLayout());
		setLocationRelativeTo(null);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		numeroLabel = new JLabel("Numero:");
		numeroLabel.setAlignmentX(LEFT_ALIGNMENT);
		numeroField = new JTextField(5);
		add(numeroLabel);
		add(numeroField);
		setVisible(true);
	}

	public JTextField getNumeroField() { return numeroField; }
}


//Controller.java
package sistema.controller;

import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import sistema.view.Tela;

public class Controller{
	KeyHandler keyHandler;
	Tela tela;

	public Controller(){
		keyHandler = new KeyHandler();
		tela = new Tela();
		control();
	}

	private void control(){
	      tela.getNumeroField().addKeyListener(keyHandler);
	}

	private class KeyHandler extends KeyAdapter{
		@Override
		public void keyTyped(KeyEvent event) {
			if (!Character.isDigit(event.getKeyChar()))
				event.consume();
		/* Observe que apenas keyTyped está sendo utilizado. Se Controller implements KeyListener
		* então teríamos os 3 métodos da interface: keyPressed, keyReleased e keyTyped
		 */
	      }
	}
}
```

(continua na próxima página...)

```java
//App.java
package sistema.app;

import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import sistema.controller.Controller;

public class App {
	public static void main(String[] args) {
		try {
			UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
		} catch (ClassNotFoundException | InstantiationException
				| IllegalAccessException | UnsupportedLookAndFeelException e) {
			e.printStackTrace();
		}
		Controller controller = new Controller();
	}
}
```

Saiba Mais!

Você sabe como mudar a posição do cursor entre componentes de texto em uma GUI? A resposta é CaretListener! Com esta interface podemos, por exemplo, saber ou modificar os campos de JTextArea ou JTextField sem que o usuário tenha digitado.

E se a pergunta agora fosse: Você sabe como mudar o foco de uma GUI? Se sua resposta é não, vamos analisar algumas situações:

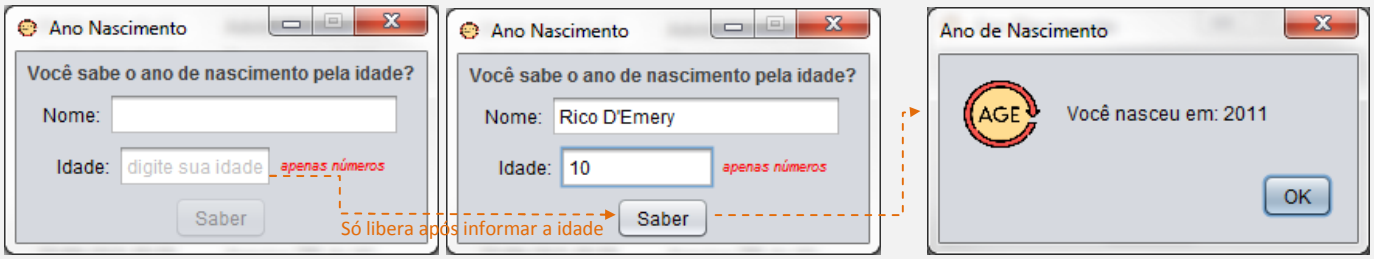
Focus em componentes por método de Component

É possível direcionar o foco para um componente gráfico, como, por exemplo, para um JLabel tem-se:

```java
label.setFocusable(true);
```

Focus por Interface

Uma situação corriqueira de GUI's: Um JFrame contendo JLabel e JTextField o foco ficará no JTextField, ou seja, o campo de entrada de dados ficará ativo neste componente! Para que se possa mudar o foco podemos utilizar a interface FocusListener. Vejamos um exemplo:




```java
// Icone.java
package sistema.model;

import javax.swing.ImageIcon;

public class Icone {
	ImageIcon icone;
	public static final String URL_ICONE_AGE ="img/age.png";

	public Icone(String urlFile){
		icone = new ImageIcon(urlFile);
	}

	public ImageIcon getIcone() { return icone; }
}
```

(continua na próxima página...)

```java
//Mensagem.Java
package sistema.view;

import javax.swing.JOptionPane;
import sistema.model.Icone;

public class Mensagem extends JOptionPane {
	public static void exibirMensagemAniversario(String mensagem, String urlIcone){
			showMessageDialog(null, mensagem, "Ano de Nascimento", JOptionPane.OK_OPTION, new
Icone(urlIcone).getIcone());
	}

	public static void exibirMensagemFalha(String mensagem){
		showMessageDialog(null, mensagem, "Erro sistema", JOptionPane.ERROR_MESSAGE);
	}
}

//Tela.java
package sistema.view;

import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class Tela extends JFrame {
	JLabel tituloLabel, nomeLabel, idadeLabel, infoLabel;
	JTextField nomeField, idadeField;
	JButton saberButton;

	public Tela(ImageIcon icone){
		super("Ano Nascimento");
		setSize(270, 150);
		setResizable(false);
		setLayout(new FlowLayout());
		setLocationRelativeTo(null);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		setIconImage(icone.getImage());
		tituloLabel = new JLabel("Você sabe o ano de nascimento pela idade?",JLabel.TRAILING);
		tituloLabel.setFont(new Font("", Font.BOLD, 12));
		tituloLabel.setForeground(Color.DARK_GRAY);
		tituloLabel.setFocusable(true);//foco no título
		nomeLabel = new JLabel("Nome:");
		nomeField = new JTextField(16);
		idadeLabel = new JLabel("Idade:");
		idadeField = new JTextField("digite sua idade", 8);
		idadeField.setForeground(Color.LIGHT_GRAY);
		infoLabel = new JLabel("apenas números");
		infoLabel.setFont(new Font("", Font.ITALIC, 9));
		infoLabel.setForeground(Color.RED);
		saberButton = new JButton("Saber");
		saberButton.setEnabled(false); //botão inativo
		add(tituloLabel);
		add(nomeLabel);
		add(nomeField);
		add(idadeLabel);
		add(idadeField);
		add(infoLabel);
		add(saberButton);
		setVisible(true);
	}

	public JTextField getIdadeField() { return idadeField; }

	public JButton getSaberButton() { return saberButton; }
}
```

(continua na próxima página...)

```java
//Controller.java
package sistema.controller;

import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.FocusEvent;
import java.awt.event.FocusListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.time.LocalDateTime;
import javax.swing.event.CaretEvent;
import javax.swing.event.CaretListener;
import sistema.model.Icone;
import sistema.view.Mensagem;
import sistema.view.Tela;

public class Controller{
	KeyHandler keyHandler;
	FieldHandler fieldHandler;
	ButtonHandler buttonHandler;
	Tela tela;
	Icone icone;
	public Controller(){
		keyHandler = new KeyHandler();
		fieldHandler = new FieldHandler();
		buttonHandler = new ButtonHandler();
		icone = new Icone(Icone.URL_ICONE_AGE);
		tela = new Tela(icone.getIcone());

		control();
	}

	private void control(){
		tela.getIdadeField().addKeyListener(keyHandler);
		tela.getIdadeField().addCaretListener(fieldHandler);
		tela.getIdadeField().addFocusListener(fieldHandler);
		tela.getSaberButton().addActionListener(buttonHandler);
	}

	private class KeyHandler extends KeyAdapter{
		@Override
		public void keyTyped(KeyEvent event) {
			if (!Character.isDigit(event.getKeyChar()))
				event.consume();
		}
	}
	private class FieldHandler implements CaretListener, FocusListener{
		@Override
		public void caretUpdate(CaretEvent e) {
			if (tela.getIdadeField().getText().length()!=0 &&
			    !tela.getIdadeField().getText().contains("digite sua idade"))
				tela.getSaberButton().setEnabled(true);
			else
				tela.getSaberButton().setEnabled(false);
		}

		//Continua na próxima página...
```

Em caretUpadate é verificado se o usuário digitou uma idade válida. Se sim, libera o botão Saber, caso contrário o botão permanece desabilitado.

Verifica se o campo de texto para idade está sendo utilizado.

- Em focusGained remove o texto dica e configura para uma entrada padrão.
- Se outro campo estiver em uso e uma idade não tiver sido informada, então em focusLost é retomada a configuração inicial do campo de texto para idade.

```
		@Override
		public void focusGained(FocusEvent e) {
			if(e.getSource()==tela.getIdadeField()){
				if(tela.getIdadeField().getText().equals("digite sua idade")) {
					tela.getIdadeField().setText("");
					tela.getIdadeField().setForeground(Color.BLACK);
				}
			}
		}

		@Override
		public void focusLost(FocusEvent e) {
			if(e.getSource()==tela.getIdadeField()){
				if(tela.getIdadeField().getText().equals("")) {
					tela.getIdadeField().setText("digite sua idade");
					tela.getIdadeField().setForeground(Color.LIGHT_GRAY);
				}
			}
		}
	}

	private class ButtonHandler implements ActionListener{
		@Override
		public void actionPerformed(ActionEvent e) {
			int idade, anoAtual, anoNascimento;

			if (e.getSource()==tela.getSaberButton()){
				idade= Integer.parseInt(tela.getIdadeField().getText());
				anoAtual = LocalDateTime.now().getYear();
				anoNascimento = anoAtual-idade;

				Mensagem.exibirMensagemAniversario("Você nasceu em: " + anoNascimento,
Icone.URL_ICONE_AGE);
			}
		}
	}
}

//App.java
package sistema.app;

import javax.swing.UIManager;
import javax.swing.UnsupportedLookAndFeelException;
import sistema.controller.Controller;

public class App {
	public static void main(String[] args) {
		try {
			UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
		} catch (ClassNotFoundException | InstantiationException
				| IllegalAccessException | UnsupportedLookAndFeelException e) {
			e.printStackTrace();
		}
		Controller controller = new Controller();
	}
}
```

Tratamento de evento para o botão Saber: Exibe o ano de nascimento do usuário a partir da idade informada. É considerado o ano corrente do sistema.

Desafio!

3) Atualize todas as GUI's do sistema de maneira que os botões de confirmação de ações só sejam liberados quando os campos obrigatórios forem devidamente preenchidos! Mas atenção, cuidado para o usuário não burlar o sistema, por exemplo: usuário preenche os campos e o botão é liberado, então apagada o dado de um campo e o botão continua liberado! Nesse caso o botão deve voltar a ficar desabilitado. É exemplo de GUI com botão desabilitado:

Melhorando o Projeto!

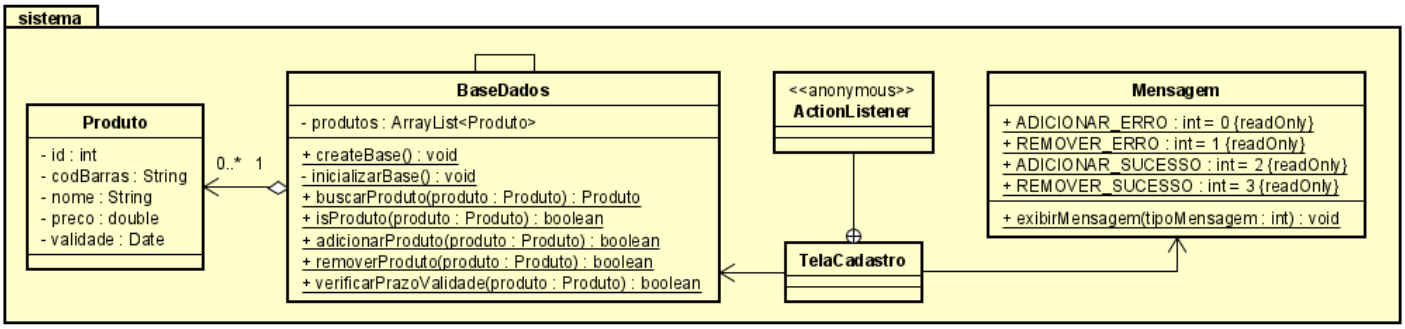
Se você chegou até aqui, então faça a seguinte análise: "*E o meu projeto UML, está organizado?*", "*O diagrama de classes do meu sistema segue MVC?*". Mostre que você também sabe diagramar a solução do seu projeto:

4) Crie o diagrama de classes do sistema anterior; e

Solucione o próximo problema!

Mão na Massa!

5) A empresa MPOOSoftware LTDA solicitou a um de seus programadores (*de codinome mustela putórius furo – "O Furão"*) que resolvesse uma demanda de funcionalidades de seu sistema. Esta demanda envolve a manutenção de produtos. Entretanto, *O Furão* não havia estudado o padrão arquitetura *Model-View-Controller*, consequentemente, condensando diversas funcionalidades. A Tela Cadastro possui menus para cada comportamento disponível na base de dados, os quais permitem a atualização de telas conforme a funcionalidade pretendida. Projete as telas que *O Furão* possa ter pensado. Abaixo está o diagrama pensado por *O Furão* com as classes utilizadas e seus relacionamentos. Sendo assim, codifique em Java a solução pensada!



6) Aproveite a oportunidade e mostre que você tem a capacidade de ocupar a vaga de "O Furão" em MPOOSoftware LTDA. Observe que parte da codificação da questão anterior é condensada em um único arquivo (tela, tratamento de eventos e regras de negócios), isso porque *O Furão* não havia estudado o padrão arquitetural MVC. Após a leitura das seções "Você Sabia?" e "Fique Atento!" sobre MVC, responda as próximas perguntas para garantir sua vaga na empresa:

6.1) Quais classes pensadas por *O Furão* condensam diversas funcionalidades que poderiam ser separadas? Explique os porquês?

6.2) Apresente uma solução (diagrama de classes e codificação Java) adequada para o problema. *Dica: Lembre-se que você poderá aproveitar a opção de copiar e colar os códigos-fonte em uma nova pasta de pacotes (pasta de pacotes questao6_2)*