



Universidade do Minho
Departamento de Informática

Aplicações Multi-camada

Construção de interfaces em Java/Swing

Aula 1

Contribuições de:

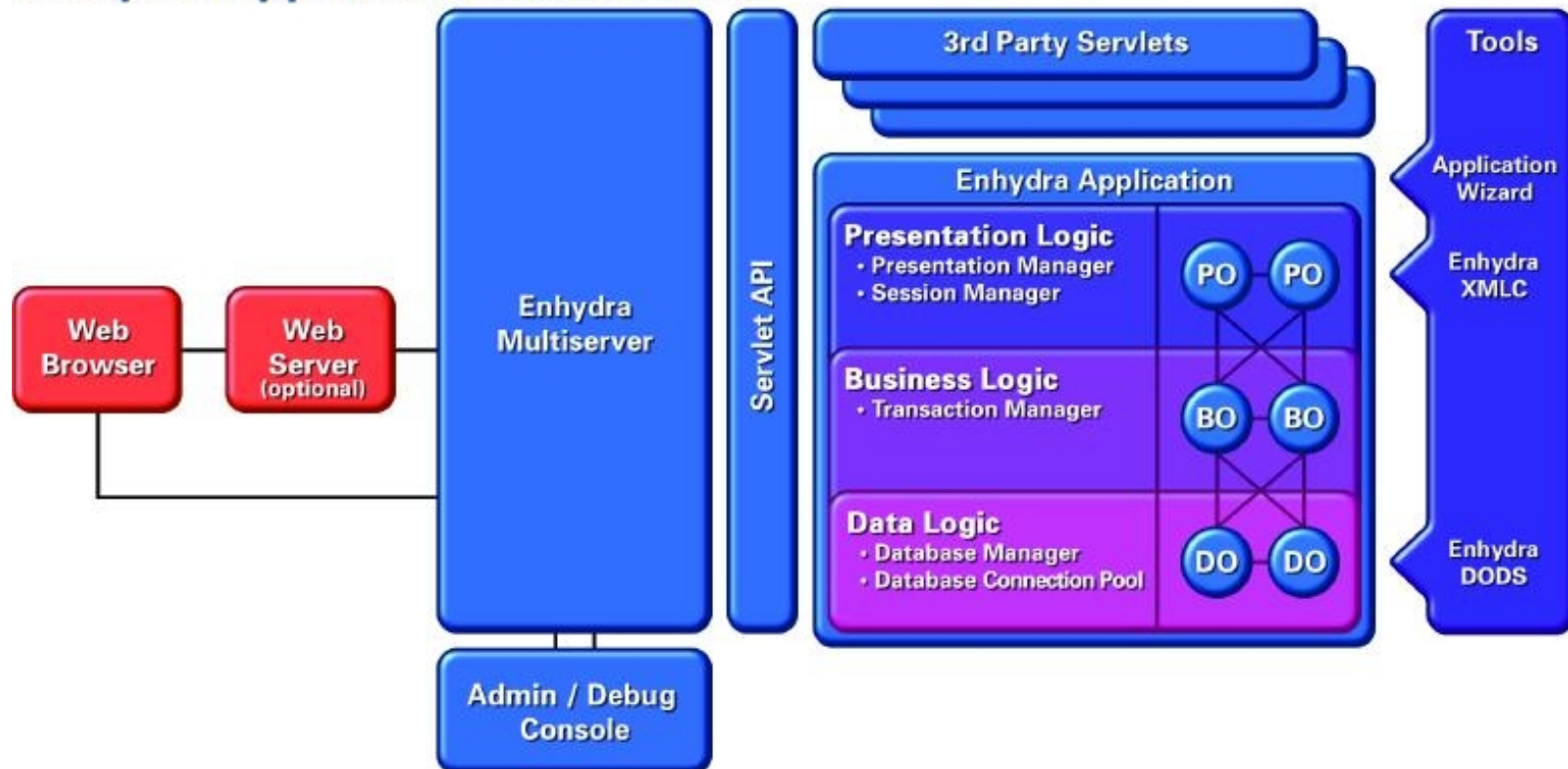
António Nestor Ribeiro

José Creissac Campos

Aplicações Multi-Camada

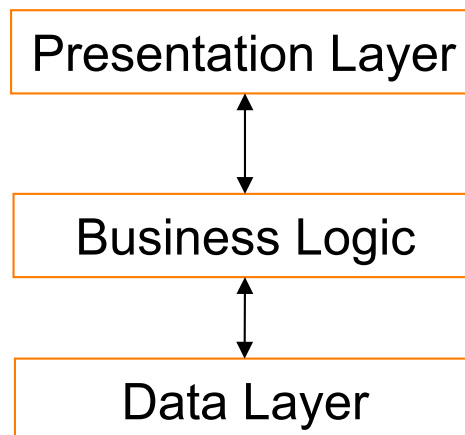
- Diversos tipos de objectos para efectuarem diferentes operações
 - Presentation Objects
 - Business Objects
 - Data Objects

Enhydra Application Architecture



Três camadas

- A camada de interface, *Presentation Layer*, permite isolar a interface com o utilizador por forma a que o resto da aplicação esta não esteja dependente de uma interface concreta.
- A camada de negócio, *Business Logic*, implementa a lógica da aplicação.
- A camada de dados, *Data Layer*, permite isolar o acesso aos dados, por forma a que o resto da aplicação não esteja dependente da origem ou estrutura sob a qual os dados estão armazenados.

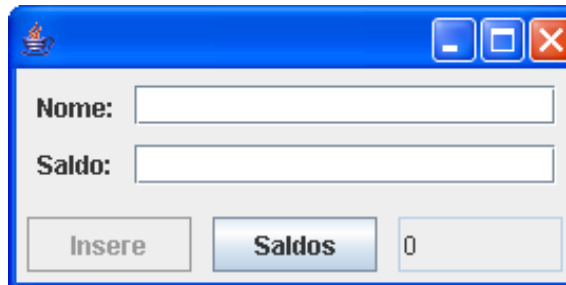


Camada de Apresentação

- Programação Orientada ao Evento
 - Controle da aplicação pode estar na camada de interface
 - Aplicação limita-se a responder a eventos:
 - clicar do rato num botão
 - inserir um carácter num campo de texto
 - ...
 - Durante a inicialização da aplicação são registados os métodos que serão chamados quando ocorrerem determinados eventos.

Camada de Apresentação

- Programação Orientada ao Evento
 - Que eventos podemos/temos de tratar neste exemplo?

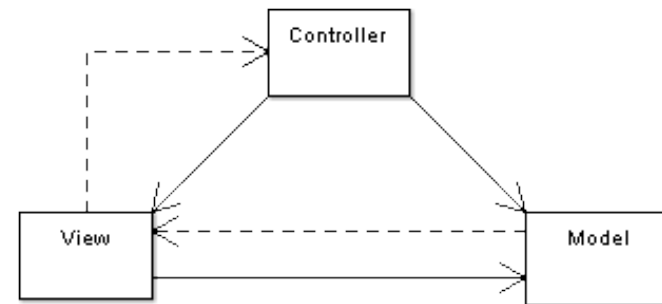


Nome:

Saldo:

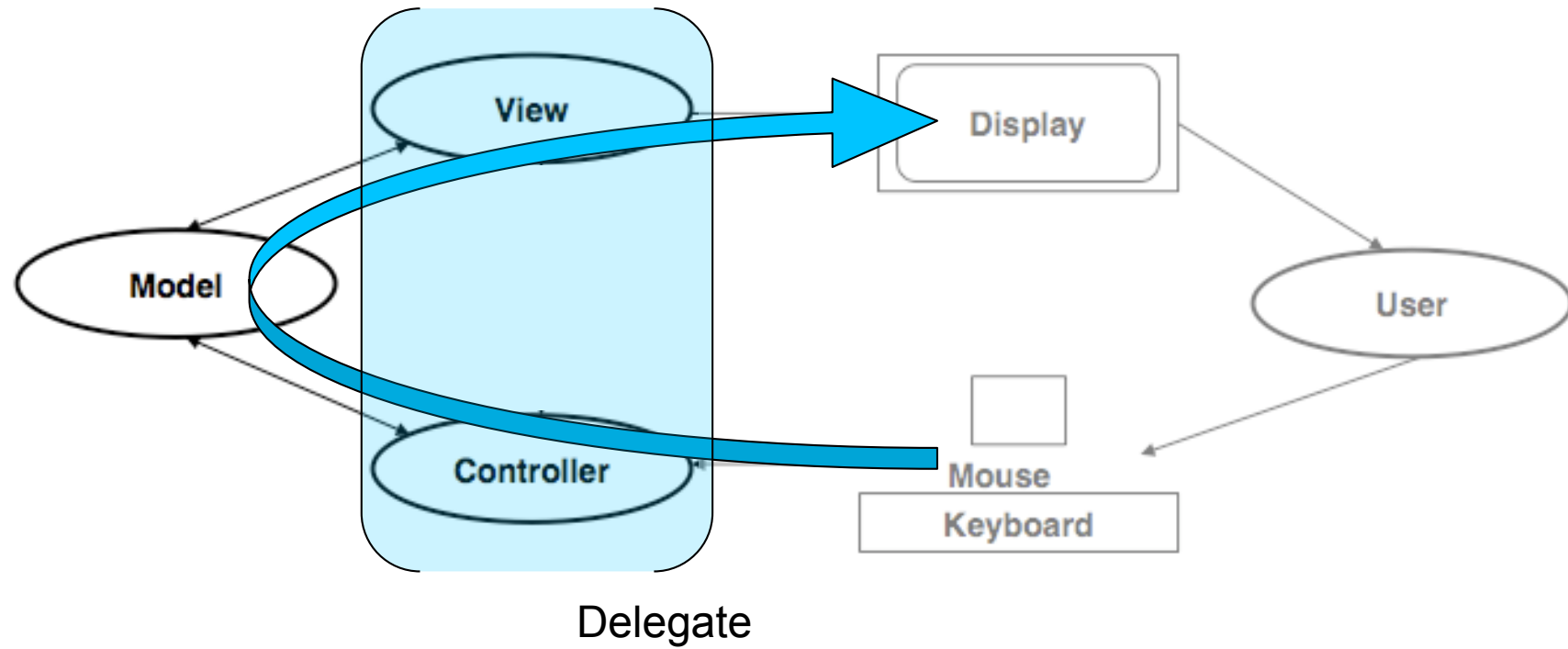
Camada de Apresentação

- Model - View - Controller *pattern*

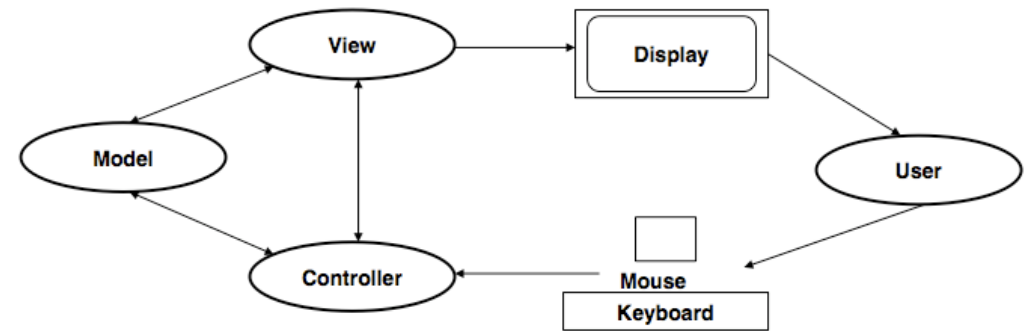


- Model: Business Logic
 - core program functionality and data
- View: A apresentação dos dados da aplicação. Desenho da interface e seus componentes
- Controller: Processa e responde aos eventos
 - Handles user input
 - Translates interface events into program functions

MVC Pattern



MVC Behavior



- Each Controller registered with a View through the Observer pattern
 - Translate event into operations on the Model
- Each View registered with a Model through the Observer pattern
 - When the Model changes all Views are automatically updated consistently
- Possible to have more than one View per Model

MVC Example: Java

- Model
 - Classes Java standard
- View
 - Componentes GUI (Graphical User Interface)
 - Swing:
<http://java.sun.com/docs/books/tutorial/ui/features/components.html>
- Controller
 - Colecção de componentes de escuta (*listeners*) dos elementos da GUI (actionPerformed, mouseClicked)

Camada de Apresentação

- Observer *pattern*
 - The **observer pattern** (sometimes known as [publish/subscribe](#)) is a [design pattern](#) used in computer programming to observe the state of an [object](#) in a [program](#). (www.wikipedia.org).
 - Através deste padrão podemos fazer com que a camada Business Logic gere eventos para a camada de interface, ou gerar interacções entre componentes.

Camada de Apresentação

- Registo dos observadores (Objectos *Observer*)

`x.addObserver(y); // sendo x Observable e y Observer`

- Objecto *Observable* (class `java.util.Observable`)

- Define quando se encontra alterado

`x.setChanged();`

- Pede que os seus *Observers* sejam notificados

`x.notifyObservers(Object arg);`

- Objecto *Observer* (interface *Observer*)

- Recebe notificações de actualização: deve implementar o método

`void update(Observable o, Object arg);`

Camada de Apresentação: tecnologia

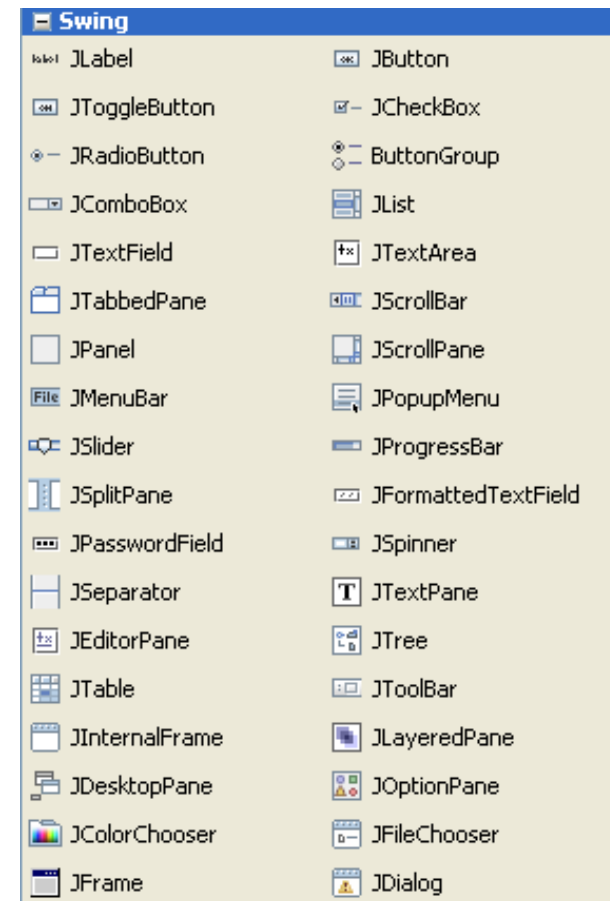
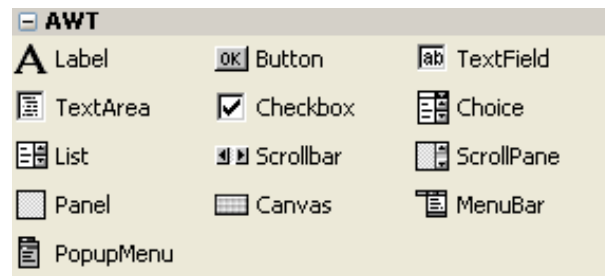
- Java GUI (Graphical User Interface)
 - Ligação entre a aplicação e o GUI nativo do sistema operativo
 - AWT: Abstract Window Toolkit
 - Código C nativo
 - Específico para cada plataforma
 - Limitado em certos aspectos (sem icons, tooltips, ...)
 - Slow and limited feature set
 - Obsoleto – superseded by Swing
 - Swing (funciona sobre o AWT)
 - Java
 - Mais poderoso, sem limitações provocadas pelas plataformas
 - Look and Feel, Acessibilidade

Java GUI components – Swing

- Only top-level containers are heavyweight
- All other components are lightweight
 - 100% Java
 - drawn using Java2D
- Highly customizable (pluggable look-and-feel)
- Platform independent
- Uses AWT!
- Uses MVC (sort of)

Camada de Apresentação

- Lista de componentes disponibilizados pelas duas APIs



Componentes e Listeners

- Pedaço da matriz que associa componentes aos respectivos listeners.

This table lists Swing components with their specialized listeners

Component	Listener							
	action	caret	change	document, undoable edit	item	list selection	window	other
button	✓		✓		✓			
check box	✓		✓		✓			
color chooser			✓					
combo box	✓				✓			
dialog							✓	
editor pane		✓		✓				hyperlink
file chooser	✓							
formatted text field	✓	✓		✓				
frame							✓	
internal frame								internal frame
list						✓		list data
menu								menu
menu item	✓		✓		✓			menu key menu drag mouse
option pane								
password field	✓	✓		✓				
popup menu								popup menu

Camada de Apresentação: Layouts

- Disposição de Componentes na Interface - Layout Manager
 - A colocação de componentes é definida por um gestor de espaço (layout manager)
 - Alguns gestores disponíveis:
 - BorderLayout
 - GridLayout
 - GridBagLayout
 - A posição final dos componentes é ditada pelo gestor (embora o utilizador possa fazer alguns pedidos)
 - É necessário precaver situações como por exemplo o redimensionamento da janela

Layout management

- LayoutManager determines the size and position of components within a container
- Important for platform independence
- Standard Layout Managers
 - null layout manager (absolute positioning)
 - simple - FlowLayout, BoxLayout, BorderLayout, GridLayout, CardLayout
 - general purpose - GridBagLayout, SpringLayout, GroupLayout (Netbeans – Free Design)

Referências bibliográficas

- Creating a GUI with JFC/Swing (aka the Swing Tutorial)
 - <http://java.sun.com/docs/books/tutorial/uiswing/index.html>
- Para saber mais sobre action listeners ver:
 - <http://java.sun.com/docs/books/tutorial/uiswing/events/actionlistener.html>
- Para saber como desenvolver aplicações com NetBeans ver:
 - <http://java.sun.com/docs/books/tutorial/uiswing/learn/settingup.html>

Common requirements on GUIs

- Platform (look and feel) independence
- Scale with size, font, and resolution
- Follow UI guidelines
- Visual consistency
- UI separated from application logic
- Localization independence

Common mistakes

- Using absolute sizes or positions
- Relying on relative proportions of components
- Implicit position dependencies
- Hard coded strings
- Hard coded fonts and colors

Case Study: Turma

- Seja a interface para gestão de uma Turma.

The image shows a user interface for managing a class (Turma). It features the following elements:

- Número:** A text input field.
- Nome:** A text input field.
- Nota Teórica:** A small numeric input field with a spinner.
- Nota Prática:** A horizontal slider ranging from 0 to 20, with a blue diamond marker currently positioned at 10.
- Média:** A horizontal bar representing the average score.
- Buttons:** A vertical stack of five buttons on the right: "Adicionar", "Consultar", "Remover", "Limpar", and "Sair".
- Footer:** The text "Quantos passam? 0" is displayed at the bottom.

- Como associar comportamento ao botão “Adicionar”?

Case Study: Turma

- Declarar o botão

```
adicionar_button = new javax.swing.JButton();
```

- Associar eventos ao botão

```
adicionar_button.setFont(new java.awt.Font("Dialog", 0, 12));
adicionar_button.setText("Adicionar");
adicionar_button.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        adicionar_action(evt);
    }
});
```

Case Study: Turma

- Detectar acção no botão

```
private void adicionar_action(java.awt.event.ActionEvent evt) {  
    // Add your handling code here:  
    if (this.validaDados()) {  
        String num = this.numero.getText();  
        String nome = this.nome.getText();  
        int notaT = Integer.parseInt((String)this.nota_teorica.getSelectedItem());  
        int notaP = this.nota_pratica.getValue();  
  
        this.turma.addAluno(new Aluno(num, nome, notaT, notaP));  
    }  
}
```

Case Study: Turma

- Por vezes é necessário mudar a apresentação (o *View*) em função de mudanças nos dados (o *Model*)
 - por exemplo, a remoção de um aluno obriga a actualizar a interface (número de alunos que passam)
 - invocar o método **notifyObservers**

```
public void delAluno(String num) throws TurmaException {
    if (!this.turma.containsKey(num)) {
        StringBuffer sb = new StringBuffer("Aluno ");
        sb.append(num);
        sb.append(" inexistente!");
        throw new TurmaException(sb.toString());
    }
    this.turma.remove(num);
    this.setChanged();
    this.notifyObservers();
}
```

- A nível da View é preciso codificar o método **update**

```
/**
 * Método necessário para a interface Observer
 */
public void update(Observable observable, Object obj) {
    this.quantos.setText(" "+this.turma.quantosPassam());
}
```


IDEs with GUI builders

- Visual interaction (WYSIWYG)
- Simplified layout design
- Easy manipulation and customization of components
- Quick prototyping
- Code Generation
- Ease of maintenance

Exercício

- Desenvolva uma aplicação que gira uma tabela "músicas" com os seguintes campos:
 - id (chave)
 - album
 - nome
 - intérprete
- A aplicação deverá permitir realizar as seguintes operações:
 - inserir, consultar e remover músicas;
 - obter as músicas de um album;
 - obter os álbuns com participação de um intérprete;
- Neste fase a camada de interface deverá ser desenvolvida em Swing e a camada de dados utilizando Maps e ObjectStreams para persistência.