



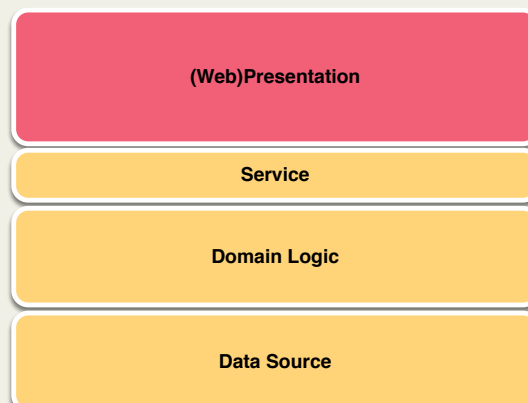
Construção de Sistemas de Software

Camada de Apresentação

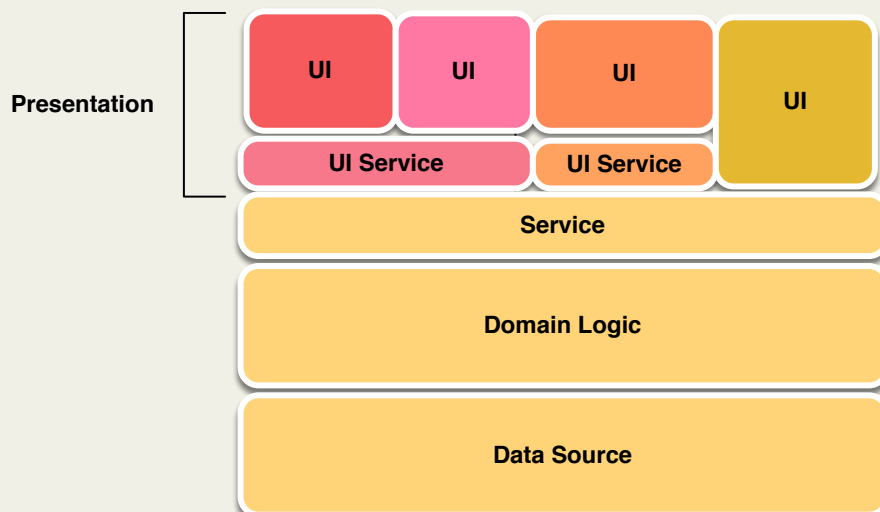
Rich Client Applications

Camada de Apresentação: caso particular

- Até agora estivemos focados na construção da camada de apresentação
 - de aplicações web Java (o cliente é um *browser*)
 - em que todo o código específico da aplicação corre na mesma máquina



Camada de Apresentação: caso geral

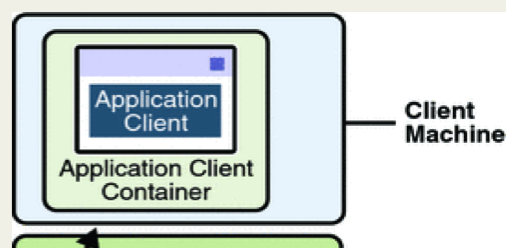
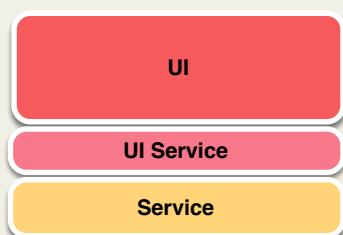


Rich Client Applications

- Aplicações que têm **Rich Client UIs**
 - que permitem experiências de utilização de ricas
 - interativas
 - com grande desempenho
- O termo frequentemente designa apenas a **aplicação cliente** (*Rich Client-Application*), ou seja o componente do UI que
 - corre na máquina do utilizador
 - interage com um outro componente (que presta os serviços da aplicação), numa arquitetura **cliente-servidor**
- O código destes *clientes* faz parte da camada de apresentação da aplicação, a qual contém também o código que expõe os serviços que estão disponíveis para aquela interface (remotamente)

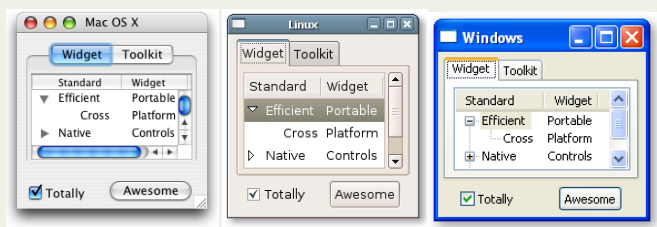
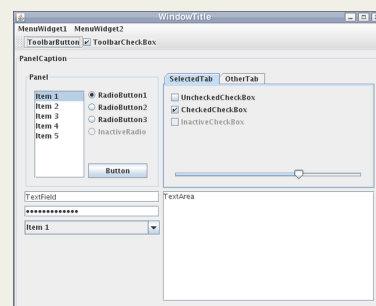
Rich Client Applications

- Nas aplicações Java EE estes UIs podem ser construídos como **Application Client Components** que
 - executam dentro de um **Client Container** nas máquinas dos clientes
 - comunicam com os serviços do *backend* recorrendo a **RMI**
 - ganhando acesso aos serviços remotos através de **injeção de dependências**



GUI toolkits

- Existem vários **GUI toolkits** baseados em tecnologia Java
- Podem ser usados para programar os **Application Client Components**
- Entre os mais populares temos
 - **Swing**, Oracle JFC
 - **SWT (Standard Widget Toolkit)**, Eclipse native widgets
 - **JavaFX**, Oracle



JavaFX

“JavaFX 2.0 is the premier platform for Rich Enterprise Client Applications”

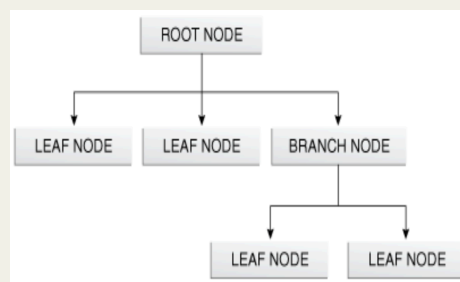
- Facilita a criação de UIs que se comportam de uma forma consistente em diferentes plataformas para *Rich Internet Applications*
- Substitui o **Swing** como o *toolkit* recomendado para as aplicações desktop Java com muitas vantagens
 - design mais consistente
 - mais *features*
 - conjunto rico de APIs de gráficos e media (audio, vídeo e animações)
- Utiliza diferentes *rendering engines* nativos, de acordo com a plataforma de execução (OpenGL, Direct3D)

JavaFX

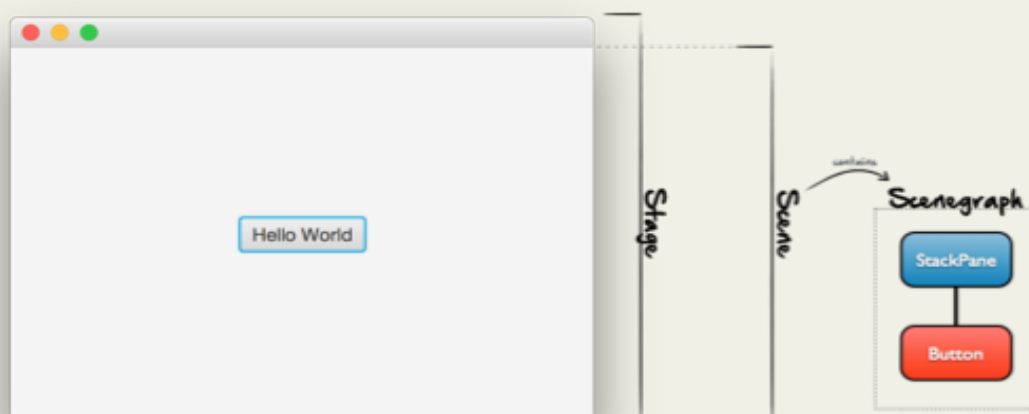
- Permite o desenho de um interface gráfico usando
 - um ficheiro XML para definir o **visual layout** (FXML)
 - um CSS para definir a **aparência gráfica**à semelhança do que acontece nas aplicações web
- IDEs oferecem ferramentas que permitem desenvolvimento rápido
 - **JavaFX Scene Builder**
- A programação do UI é baseada
 - no tratamento de **eventos**
 - em **bindings de propriedades**

Conceitos chave

- **Stage**
 - onde é feita a apresentação da aplicação
- **Scene**
 - contentor dos elementos que compõem uma “página” da aplicação
- **Node**
 - são os elementos de uma **Scene**, com um determinado *layout* e comportamento de interação
 - podem ser contentores de outros nós

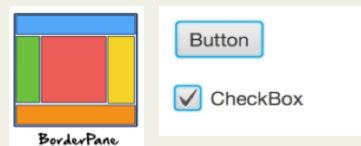


Anatomia de uma aplicação JavaFX: Exemplo

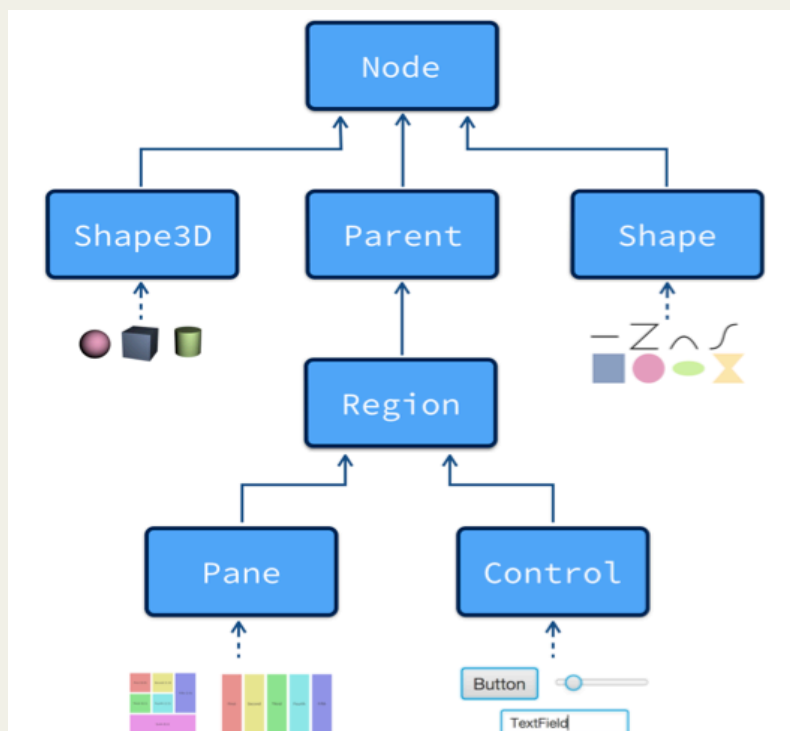


Anatomia de uma aplicação JavaFX

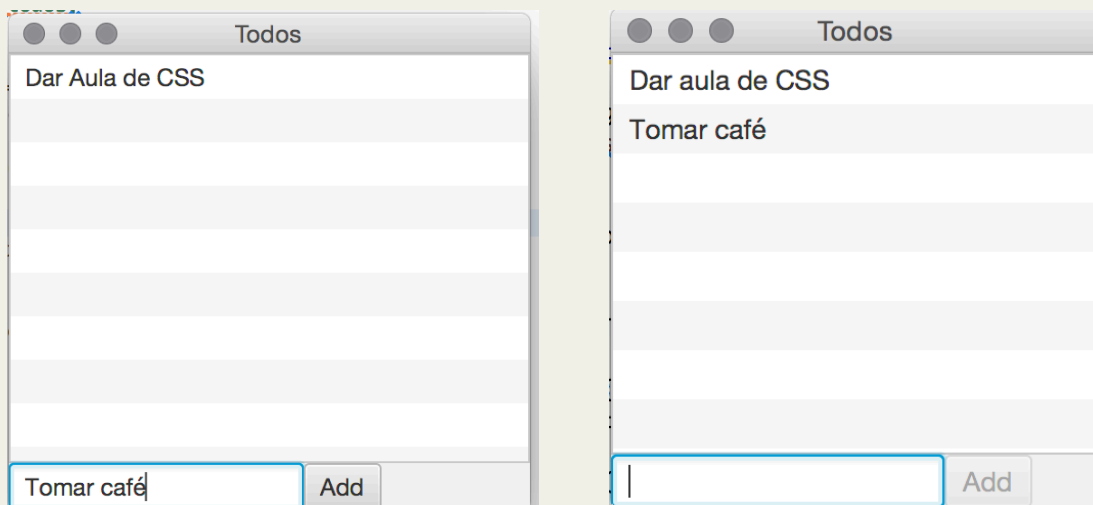
- Uma aplicação JavaFX contém um ou mais **stages**
 - correspondem às janelas
 - o **primary stage** é construído automaticamente
- Cada **stage** tem associada uma **scene**
- Cada **scene** tem associado um **scene graph**
 - os nós são objetos do tipo **node**
 - os arcos representam a relação de composição
- Na hierarquia cuja raiz é **Node** temos tipos que representam
 - **layout containers**: elementos que têm associado um algoritmo de apresentação dos elementos que contêm
 - **UI controls**: elementos com funcionalidades de controlo



Hierarquia de Componentes UI



Anatomia de uma aplicação JavaFX: Exemplo



UI Controls: eventos

- Uma grande variedade de *UI Controls*
 - **Button**
 - *TextArea, TextField, ChoiceBox, ComboBox, ListView, MenuBar, Slider, TreeView,...*
- Ações do utilizador sobre o componente propagadas através do lançamento de eventos
 - **ActionEvent**,
 - *TreeModificationEvent, InputEvent,...*
- Funcionalidade pretendida definida recorrendo à definição de *Event handlers*

```
void setOnAction(EventHandler<ActionEvent> value)
```

```
Button button = new Button("Click Me");  
button.setOnAction (e -> btnClicked());
```

```
private void btnClicked() {  
    System.out.println("Hello World");  
}
```

UI Controls: dados

- Os dados que os *UI Controllers* precisam de mostrar

eg, numa *ListView<String>*

podem ser obtidos recorrendo a *Observers*

eg, populando a *ListView* com uma *ObservableList<String>*

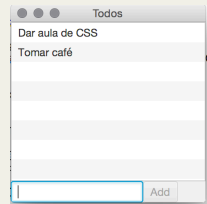
mudanças de estado desta lista são observadas pela *ListView*

```
ObservableList<String> todos
```

```
ListView<String> listView = new ListView<>(todos);
```

estas mudanças podem, por exemplo, ser em reação a uma ação do utilizador sobre *outros UI controllers* como *TextField* e um *Button*

```
addButton.setOnAction(e -> {  
    todos.add(inputField.getText());  
    inputField.setText("");  
    inputField.requestFocus();  
});
```



Ciências
ULisboa | Informática

FXML

- Linguagem baseada em XML que permite **declarativamente** definir a **estrutura** e o **layout** do UI

```
<?xml version="1.0" encoding="UTF-8"?>  
<?import javafx.scene.control.Button?>  
<?import javafx.scene.layout.StackPane?>  
  
<StackPane fx:controller="ComponentController" xmlns:fx="http://javafx.com/fxml/1" >  
    <children>  
        <Button fx:id="button" mnemonicParsing="false" onAction="#sayHello" text="Click Me" />  
    </children>  
</StackPane>
```

- Em alternativa a fazê-lo **programaticamente**

```
Button button = new Button("Click Me");  
button.setOnAction (e -> btnClicked("Ola!"));  
  
StackPane pane = new StackPane();  
pane.getChildren().add(button);
```



Ciências
ULisboa | Informática

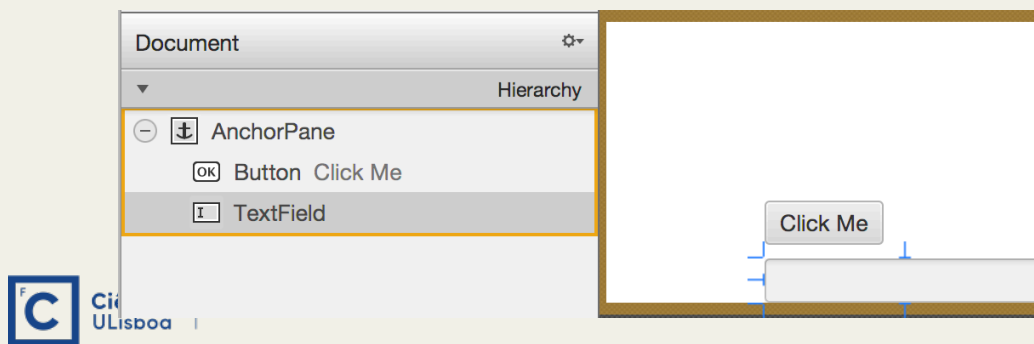
FXML

A **estrutura** e o **layout** do UI podem ser definidas com a ajuda de ferramentas como o **SceneBuilder**

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>

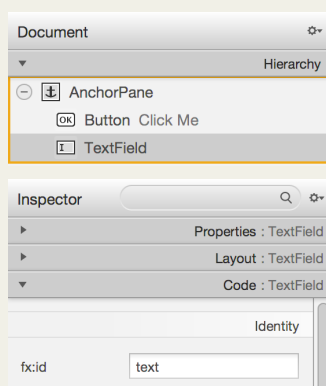
<AnchorPane xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8.0.111"
    fx:controller="ComponentController">
    <children>
        <Button fx:id="button" layoutX="41.0" layoutY="105.0" mnemonicParsing="false"
            onAction="#sayHello" text="Click Me" />
        <TextField fx:id="text" editable="false" layoutX="13.0" layoutY="151.0" />
    </children>
</AnchorPane>
```



FXML

Cada componente definido por **FXML File** + **Controller Class**

- o *fxml* define a **View** do componente e tem uma ligação explícita para a respetiva **Controller Class**
- a *controller class* define o **Controller** e contém
 - o código que é necessário executar quando o utilizador interage com aquele componente do UI
 - atributos onde são automaticamente injetados nós da vista (**injeção de dependências**) marcados com um **fx:id** único no FXML



```
public class ComponentController {

    @FXML
    private Button button;

    @FXML
    private TextField text;

    @FXML
    void sayHello(ActionEvent event) {
        text.setText(button.getText()+" Button Pressed!");
    }

}
```

FXML

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.layout.AnchorPane?>

<AnchorPane xmlns:fx="http://javafx.com/fxml/1" xmlns="http://javafx.com/javafx/8.0.111"
    fx:controller="ComponentController">
    <children>
        <Button fx:id="button" layoutX="41.0" layoutY="105.0" mnemonicParsing="false"
            onAction="#sayHello" text="Click Me" />
        <TextField fx:id="text" editable="false" layoutX="-13.0" layoutY="151.0" />
    </children>
</AnchorPane>
```

```
public class ComponentController {

    @FXML
    private Button button;

    @FXML
    private TextField text;

    @FXML
    void sayHello(ActionEvent event) {
        text.setText(button.getText()+" Button Pressed!");
    }

}
```



Ciências
ULisboa | Informática

FXML: *Startup*

- Integração dos vários componentes é definida na **Startup Class**
 - classe que estende **javafx.application.Application**
 - carrega cada ficheiro FXML, o que cria um nó e um controlador
 - define como se organizam os vários nós criados
 - usa o nó raiz para definir uma **scene** e coloca-a no **primary stage**
 - lança a aplicação

```
AnchorPane root = FXMLLoader.load(getClass().getResource("example.fxml"));
Scene scene = new Scene(root, 300, 275);

primaryStage.setScene(scene);
primaryStage.setWidth(400);
primaryStage.setHeight(300);
primaryStage.show();
```

Propriedades

- Um interface definido pelo JavaFx

```
javafx.beans.property  
Interface Property<T>
```

que serve para embrulhar tipos existentes, adicionado-lhe a funcionalidade de notificação de mudança.

- Uma **propriedade**
 - tem métodos **get()** and **set()** para obter e mudar o valor da propriedade
 - o valor da propriedade é **observável** no sentido em que emite eventos quando é mudado
- Existem implementações *default* para todos os tipos de propriedades
 - SimpleIntegerProperty implements Property<Integer>**
 - SimpleStringProperty implements Property<String>**
 - ...
 - SimpleObjectProperty<T> implements ObjectProperty<T>**

Data Bindings

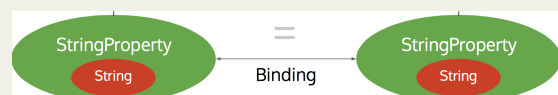
```
javafx.beans.property
```

```
Interface Property<T>
```

```
All Superinterfaces:
```

```
Observable, ObservableValue<T>, ReadOnlyProperty<T>, WritableValue<T>
```

- Uma **propriedade**
 - é *bindable*, i.e., pode ser ligada a uma outra propriedade do mesmo tipo
 - para isso tem métodos **bind(...)**, **bindBidirectional(...)**
 - as properties ligadas desta forma ficam forçadas a ter o mesmo valor



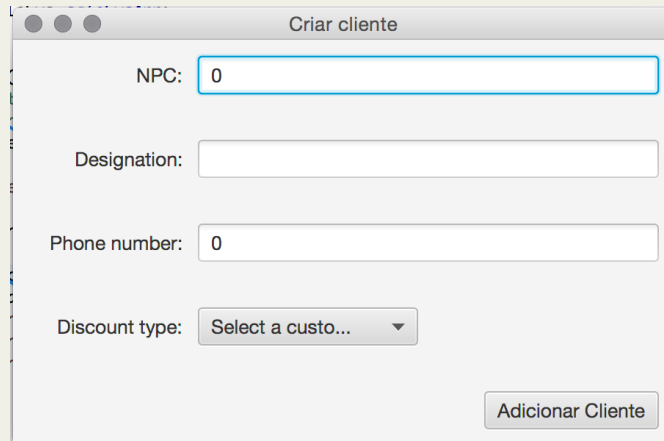
- Data Binding**

- é o mecanismo que permite definir a sincronização automática dos valores de duas propriedades
- pode ser unidirecional ou bidirecional

```
addButton.disableProperty()  
    .bind(Bindings.isEmpty(inputField.textProperty()));
```

Propriedades

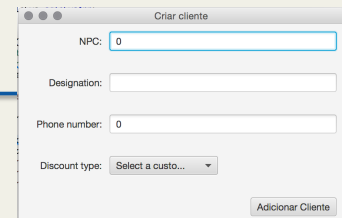
- São um ingrediente importante na definição das classes *ViewModel* usadas para encapsular os dados associados a uma *View*



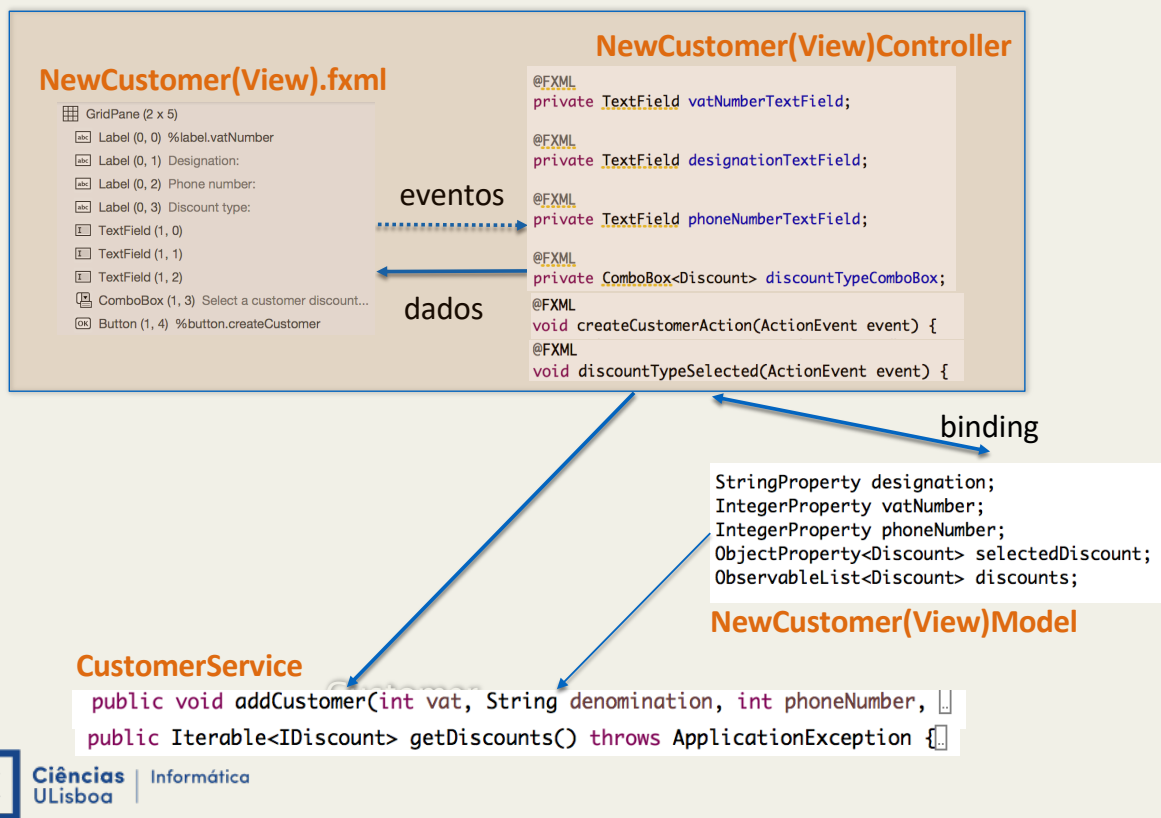
Propriedades

```
public class NewCustomerModel {  
  
    private final StringProperty designation;  
    private final IntegerProperty vatNumber;  
    private final IntegerProperty phoneNumber;  
    private final ObjectProperty<Discount> selectedDiscount;  
    private final ObservableList<Discount> discounts;  
  
}
```

```
    public ObjectProperty<Discount> selectedDiscountProperty() {  
        return selectedDiscount;  
    }  
  
    public final Discount getSelectedDiscount() {  
        return selectedDiscount.get();  
    }  
  
    public final void setSelectedDiscount(Discount d) {  
        selectedDiscount.set(d);  
    }  
  
}
```



JavaFX e os padrões: Variante de MVC + Page Controller



Propriedades: Inicialização

```
public class NewCustomerModel {

    private final StringProperty designation;
    private final IntegerProperty vatNumber;
    private final IntegerProperty phoneNumber;
    private final ObjectProperty<Discount> selectedDiscount;
    private final ObservableList<Discount> discounts;

    public NewCustomerModel(CustomerService cs) {
        designation = new SimpleStringProperty();
        vatNumber = new SimpleIntegerProperty();
        phoneNumber = new SimpleIntegerProperty();
        this.discounts = FXCollections.observableArrayList();
        try {
            cs.getDiscounts().forEach(d -> discounts.add(new Discount(d.getDescription(), d.getId())));
        } catch (ApplicationException e) {
            // no discounts added
        }
        selectedDiscount = new SimpleObjectProperty<>(null);
    }
}
```

Propriedades: *Data Binding*

```
public class NewCustomerModel {

    private final StringProperty designation;
    private final IntegerProperty vatNumber;
    private final IntegerProperty phoneNumber;
    private final ObjectProperty<Discount> selectedDiscount;
    private final ObservableList<Discount> discounts;

}

public class NewCustomerController extends BaseController {

    @FXML
    private TextField vatNumberTextField;

    @FXML
    private TextField designationTextField;

    @FXML
    private TextField phoneNumberTextField;

    @FXML
    private ComboBox<Discount> discountTypeComboBox;

    public void setModel(NewCustomerModel model) {
        this.model = model;
        designationTextField.textProperty().bindBidirectional(model.designationProperty());
        vatNumberTextField.textProperty().bindBidirectional(model.vatNumberProperty(), new NumberStringConverter());
        phoneNumberTextField.textProperty().bindBidirectional(model.phoneNumberProperty(), new NumberStringConverter());
        discountTypeComboBox.setItems(model.getDiscounts());
        discountTypeComboBox.setValue(model.getSelectedDiscount());
    }
}
```

Tratamento de eventos + *data binding*

```
public class NewCustomerModel {

    private final StringProperty designation;
    private final IntegerProperty vatNumber;
    private final IntegerProperty phoneNumber;
    private final ObjectProperty<Discount> selectedDiscount;
    private final ObservableList<Discount> discounts;

}

public class NewCustomerController extends BaseController {

    @FXML
    private TextField vatNumberTextField;

    @FXML
    private TextField designationTextField;

    @FXML
    private TextField phoneNumberTextField;

    @FXML
    private ComboBox<Discount> discountTypeComboBox;

    @FXML
    void discountTypeSelected(ActionEvent event) {
        model.setSelectedDiscount(discountTypeComboBox.getValue());
    }
}
```

Tratamento de eventos + *data binding*

```
public class NewCustomerController extends BaseController {

    @FXML
    private TextField vatNumberTextField;

    @FXML
    private TextField designationTextField;

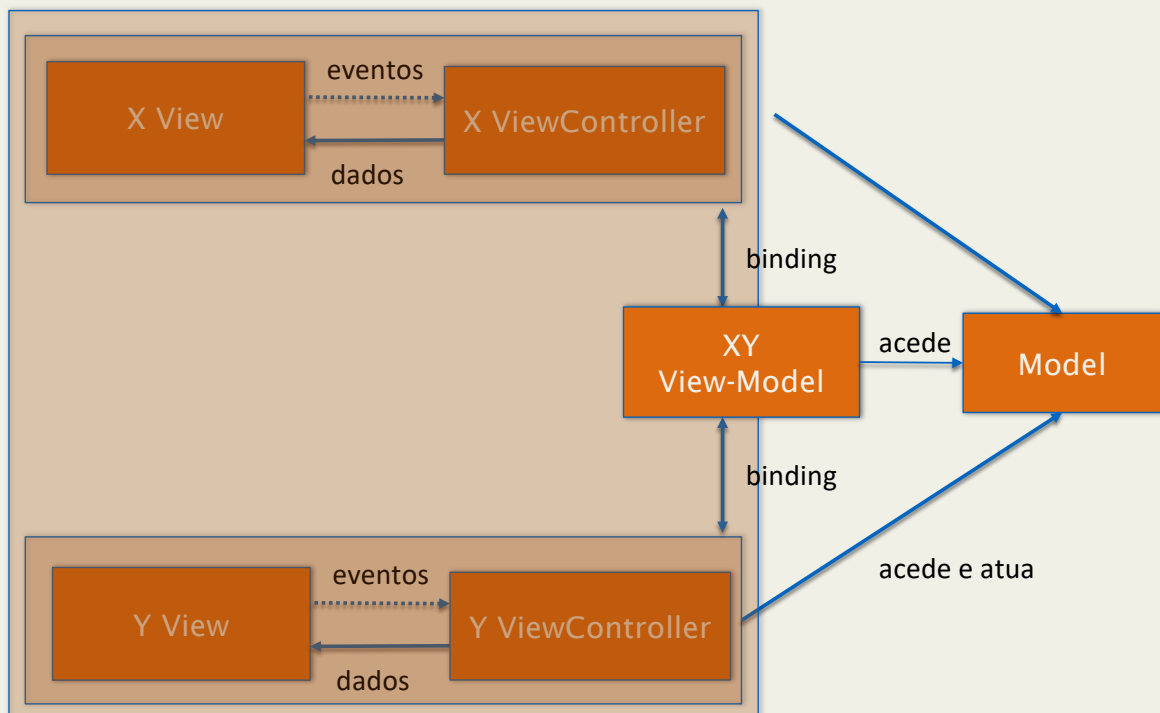
    @FXML
    private TextField phoneNumberTextField;

    @FXML
    private ComboBox<Discount> discountTypeComboBox;

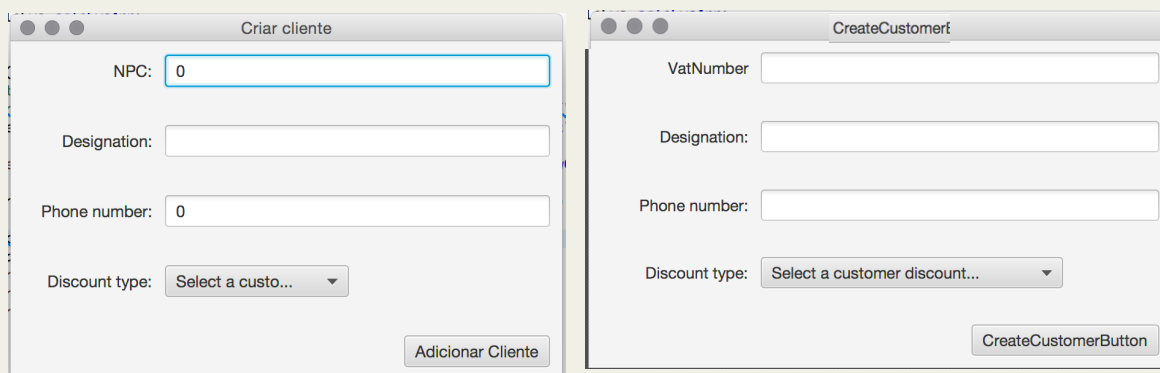
    @FXML
    void createCustomerAction(ActionEvent event) {
        String errorMessages = validateInput();

        if (errorMessages.length() == 0) {
            try {
                customerService.addCustomer(model.getVatNumber(),
                    model.getDesignation(), model.getPhoneNumber(), model.getSelectedDiscount().getDiscountCode());
                model.clearProperties();
                showInfo(i18nBundle.getString("new.customer.success"));
            } catch (VatInvalidException e) {
                showError(i18nBundle.getString("new.customer.error.vatInvalid") + ": " + e.getMessage());
            } catch (ApplicationException e) {
                showError(i18nBundle.getString("new.customer.error.adding") + ": " + e.getMessage());
            }
        } else {
            showError(i18nBundle.getString("new.customer.error.validating") + ":\n" + errorMessages);
        }
    }
}
```

JavaFX e os padrões



Suporte para a internacionalização



```
showInfo(i18nBundle.getString("new.customer.success"));
```

```
ResourceBundle i18nBundle = ResourceBundle.getBundle("i18n.Bundle", new Locale("pt", "PT"));
```

```
FXMLLoader createCustomerLoader = new FXMLLoader(getClass().getResource("/fxml/newCustomer.fxml"), i18nBundle);
```

```
<Label text="%label.vatNumber" GridPane.halignment="RIGHT" />
```

```
Bundle_pt_PT.properties
1 application.title = Criar cliente
2 label.vatNumber = NPC:
3 button.createCustomer = Adicionar Cliente
4
5 error.dialog.title = Erro
6 info.dialog.title = Sucesso
7
```

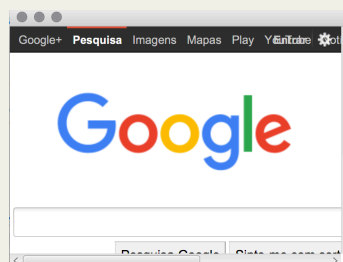


Ciências
ULisboa | Informática

WebView e CSS

- O componente **WebView** permite embeber conteúdo web na aplicação, nomeadamente mostrar páginas definidas por exemplo com HTML5 e CSS

```
StackPane pane = new StackPane();
WebView webView = new WebView();
WebEngine engine = webView.getEngine();
//Load a web page
engine.load("http://www.google.com");
//Add the web view to the JavaFX view
pane.getChildren().add(webView);
```



- É baseado no **WebKit** (*web pages render engine*) e torna possível capitalizar numa aplicação web para fazer uma aplicação desktop, juntando-lhe funcionalidades que só faz sentido ter neste contexto, como por exemplo o acesso ao disco
- Pode se definir o estilo **CSS** a aplicar a um **scene graph** ou a um nó específico