Ciências ULisboa Faculdade de Ciências da Universidade de Lisboa

2019/2020

Construção de Sistemas de Software

Rich Client Applications

JavaFX

Ciências ULisboa | Informática

CSS (2019/2020)
Caderno de Exercícios
Folha 9

1. *Java Rich Client Applications*

   (a) Estas aplicações são usadas em sistemas que seguem que estilo arquitectural?

   (b) Qual o papel de cada um dos dois (principais) componentes tem no sistema?

   (c) Onde (i.e., em que máquinas) executa cada um dos componentes?

   (d) Que tipo de comunicação existe entre os dois?

   (e) A construção destas aplicações clientes usualmente apoia-se em quê?

2. *JavaFX I*

   (a) Quais os principais objetos envolvidos no UI de uma aplicação programada em JavaFX?

   (b) Explique o conceito de *scene graph* e o seu papel.

   (c) Indique alguns tipos de nós que podem ser colocados num *scene graph* e o que representam.

Exercícios

2. *JavaFX*

   (a) Quais os principais objetos envolvidos no UI de uma aplicação programada em JavaFX?

   (b) Explique o conceito de *scene graph* e o seu papel.

   (c) Indique alguns tipos de nós que podem ser colocados num *scene graph* e o que representam,

   (d) Explique o propósito da classe abstrata *Application* e como deve ser usada.

   (e) O JavaFX permite definir *templates* com a parte estática do UI? Como?

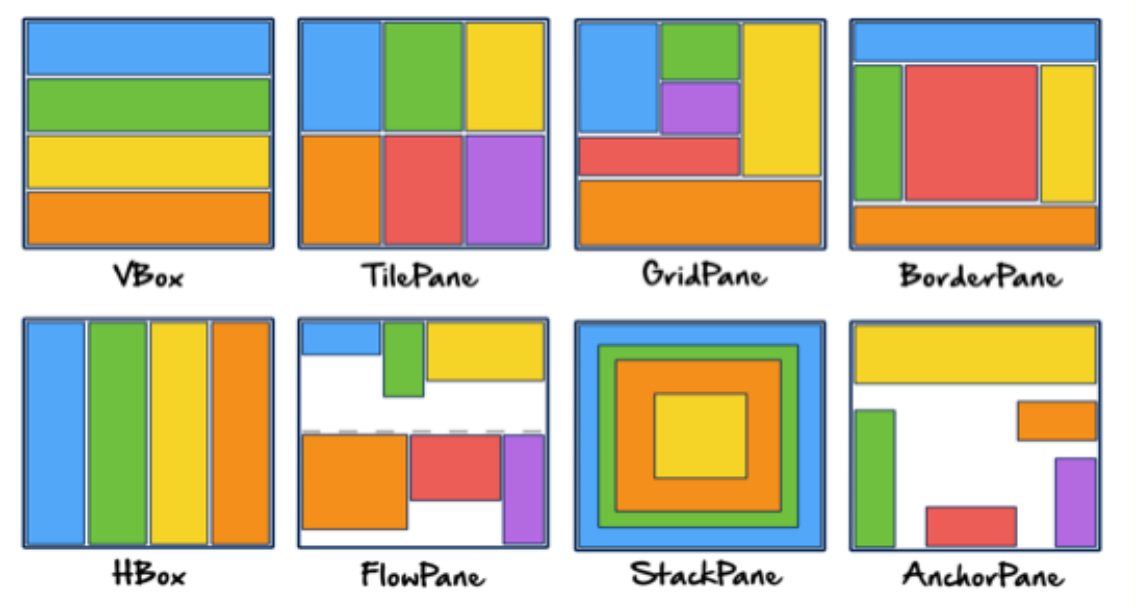
   (f) E como é definida nesse caso a parte dinâmica?

   (g) E como é suportada a definição da aparência do que é apresentado?

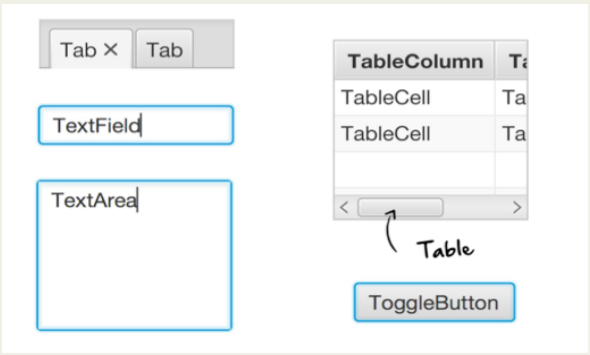
JavaFX API

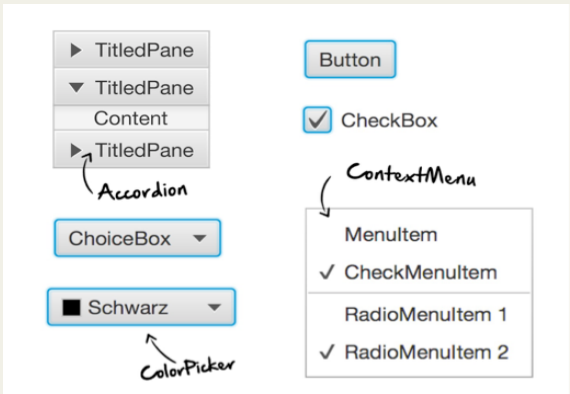
Application	Application class from which JavaFX applications extend.
Stage	The JavaFX `Stage` class is the top level JavaFX container.
Scene	The JavaFX `Scene` class is the container for all content in a scene graph.
Node	Base class for scene graph nodes.
Region	Region is the base class for all JavaFX Node-based UI Controls, and all layout containers.

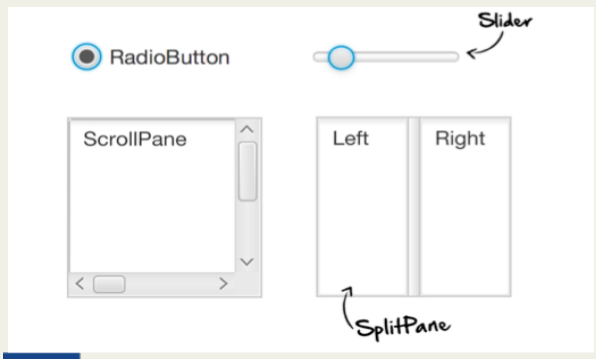
Layout Containers



UI Controls

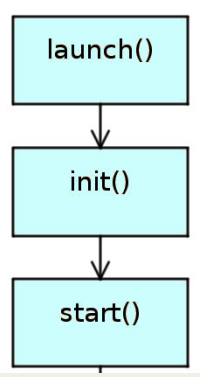




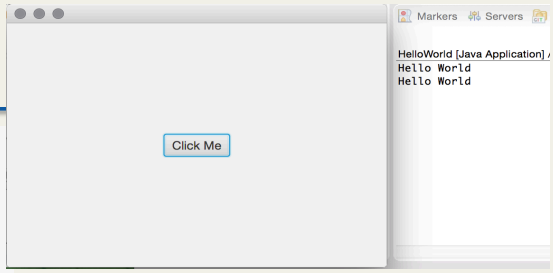


Application (javafx.application.Application)

```
public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {


}
public static void main(String[] args) {
    launch(args);
}
```



Application



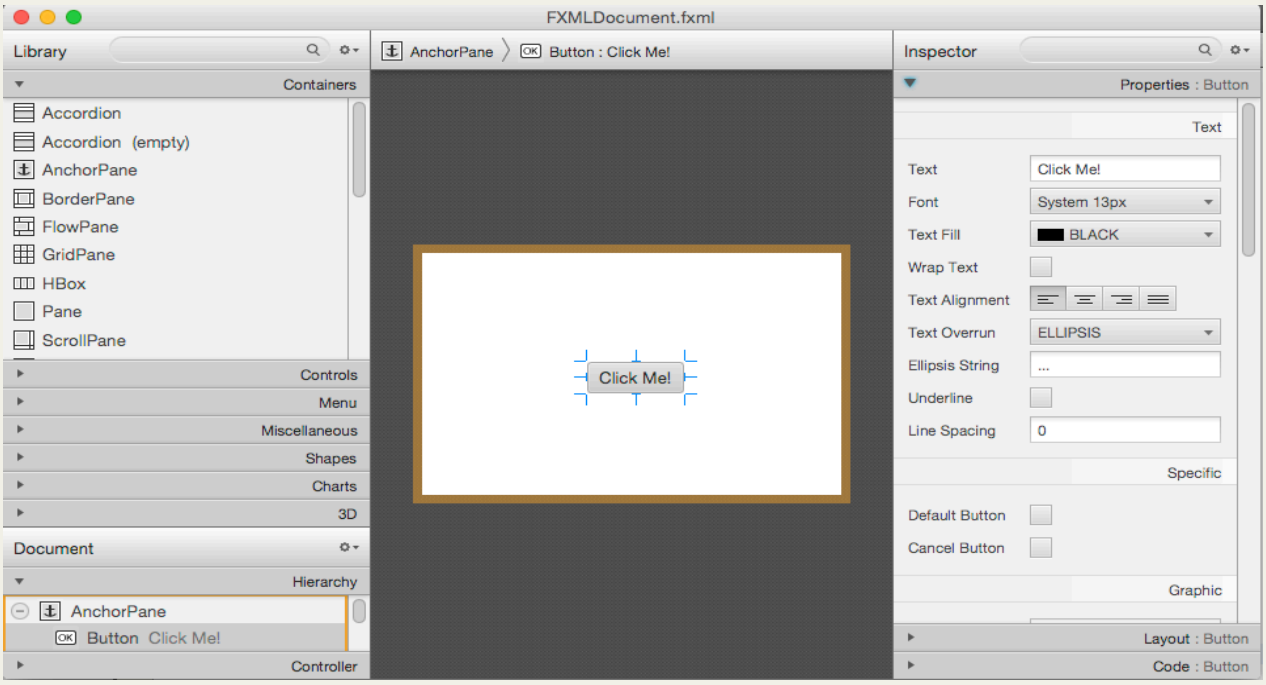

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button button = new Button("Click Me");
        button.setOnAction (e -> btnClicked());

        StackPane pane = new StackPane();
        pane.getChildren().add(button);

        Scene scene = new Scene(pane, 200, 50);

        primaryStage.setScene(scene);
        primaryStage.setWidth(400);
        primaryStage.setHeight(300);
        primaryStage.show();
    }
    private void btnClicked() {
        System.out.println("Hello World");
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

View Templates com FXML

- **FXML File**
  - Define a parte estática da *view*
  - Pode ser gerada com a aplicação **SceneBuilder**
  - Quando é carregado, dá origem a um único nó (**view**) e instancia um único **controller**

- **Controller Class** (*view controller*)
  - tem o código que é necessário executar quando o utilizador interage com componente do UI
  - define a parte dinâmica da view

SceneBuilder



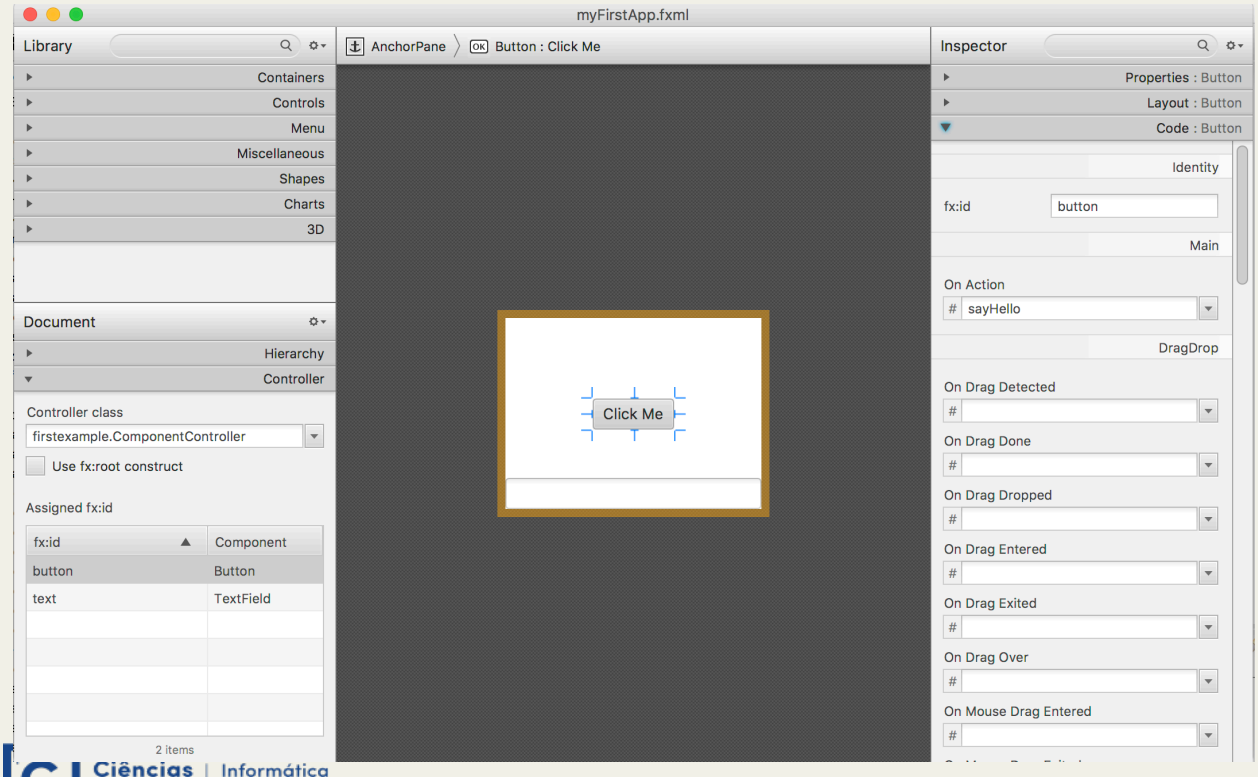
Exemplo (*View Template*)

myFirstApp.fxml

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

<?import java.lang.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?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"
        fx:controller="firstexample.ComponentController">
   <children>
      <Button fx:id="button" layoutX="76.0" layoutY="70.0" mnemonicParsing="false"
                onAction="#sayHello" text="Click Me" />
      <TextField fx:id="text" editable="false" layoutY="139.0"
                prefHeight="27.0" prefWidth="223.0" />
   </children>
</AnchorPane>
```

Ciências ULisboa | Informática

Exemplo (View Template)



Ciências ULisboa | Informática

Exemplo (View Controller)

```
package firstexample;
import javafx.event.ActionEvent;

public class ComponentController {

    @FXML
    private Button button;

    @FXML
    private TextField text;

    @FXML
    void sayHello(ActionEvent event) {
        text.setText("Hello World! "+ button.getText()+" button pushed");
    }
}
```


Exemplo (*Application*)

```
package firstexample;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;
import javafx.stage.Stage;

public class HelloWorldFXML extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) throws Exception {

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

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

Exercícios

3. *JavaFX II*

   (a) Explique o papel que têm os eventos e as propriedades na construção de UIs com o JavaFX.

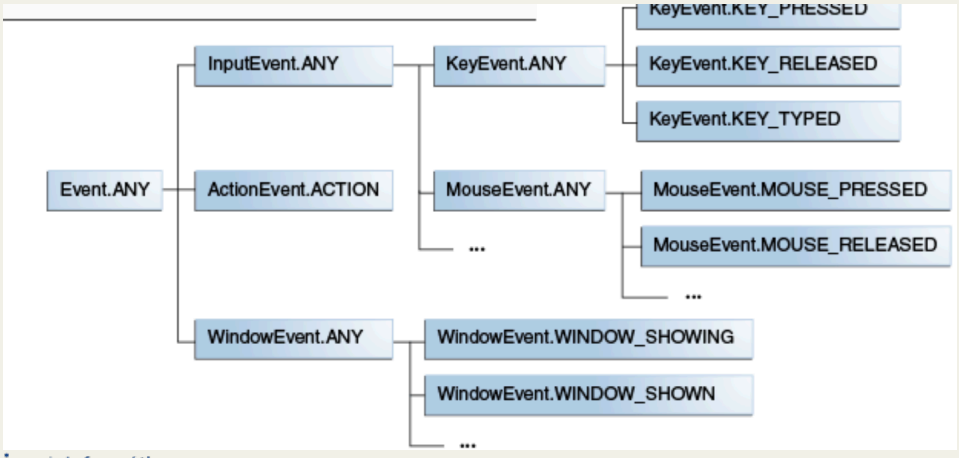
   (b) Explique o conceito de *data binding* e de que caraterísticas das propriedades este depende.

Propriedades: Exemplo

```
import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

class Bill {

    // Define a variable to store the property
    private DoubleProperty amountDue = new SimpleDoubleProperty();

    // Define a getter for the property's value
    public final double getAmountDue(){return amountDue.get();}

    // Define a setter for the property's value
    public final void setAmountDue(double value){amountDue.set(value);}

     // Define a getter for the property itself
    public DoubleProperty amountDueProperty() {return amountDue;}

}
```

Eventos

- São notificações de que alguma coisa de interesse para a aplicação ocorreu
  - Um utilizador pressionou um botão, uma tecla, o rato
- A aplicação pode, ao nível de cada nó, registar **filtros** e **handlers** de eventos que recebem eventos e dão uma resposta



Data binding

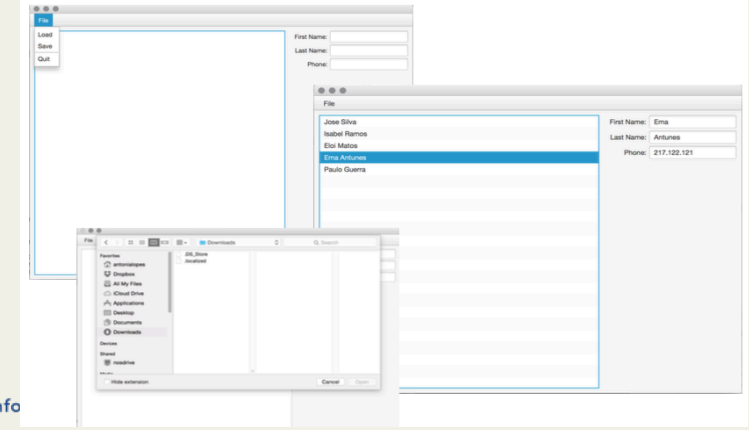
- Serve para ter a sincronização automática dos valores de propriedades

- **Unidirecional** p1.bind(p2)
  - p1 tem sempre o mesmo valor que p2
  - p1 torna-se *read-only*
- **Bidirecional** p1.bindBirectional(p2)
  - mudanças propagadas em ambas as direcções

Exercícios

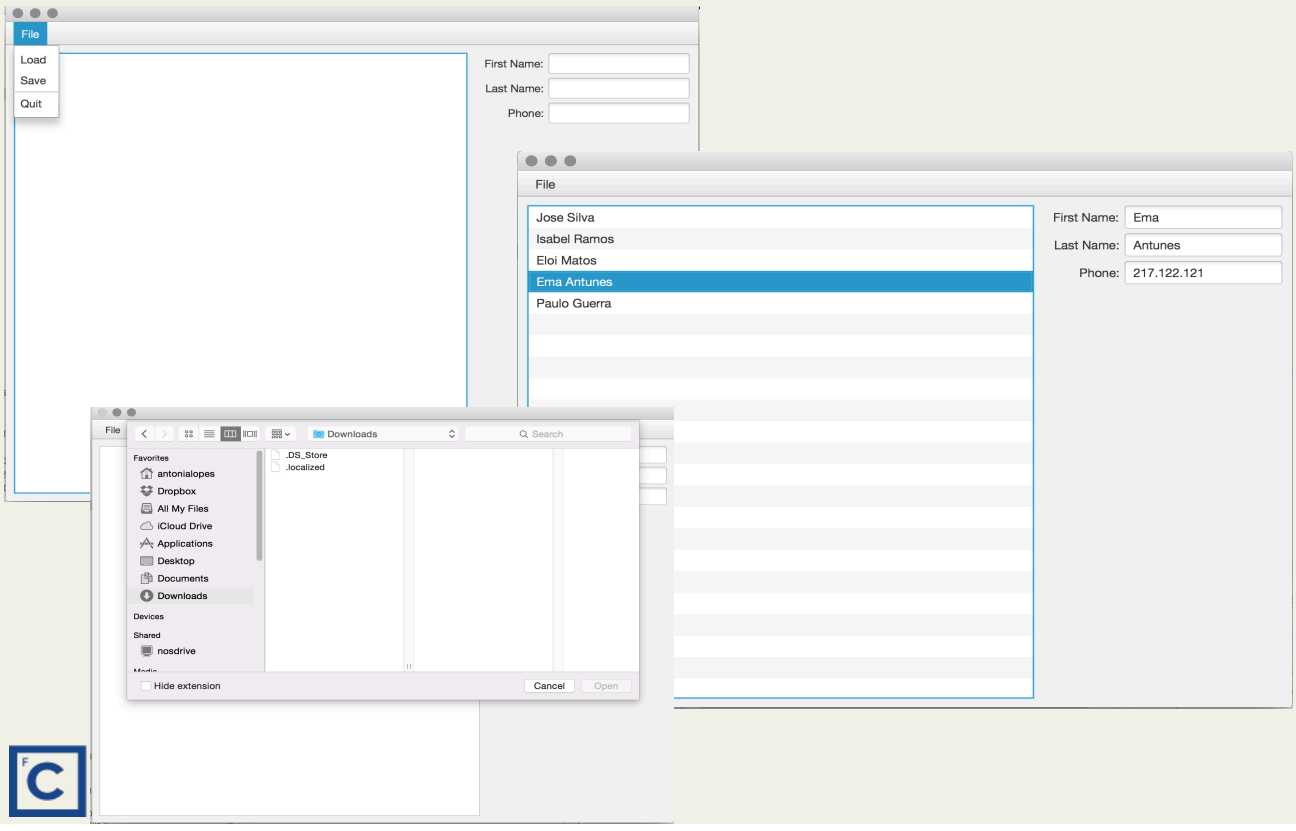
(a) Programe em JavaFX um GUI com as características mostradas abaixo e que permite 1) carregar um conjunto de dados de pessoas a partir de um ficheiro de texto, 2) visualizar as pessoas carregadas numa lista e 3) aceder e modificar os dados das várias pessoas na lista (mais especificamente, os dados da pessoa selecionada).

(b) Identifique os padrões aplicados e os diferentes elementos envolvidos.

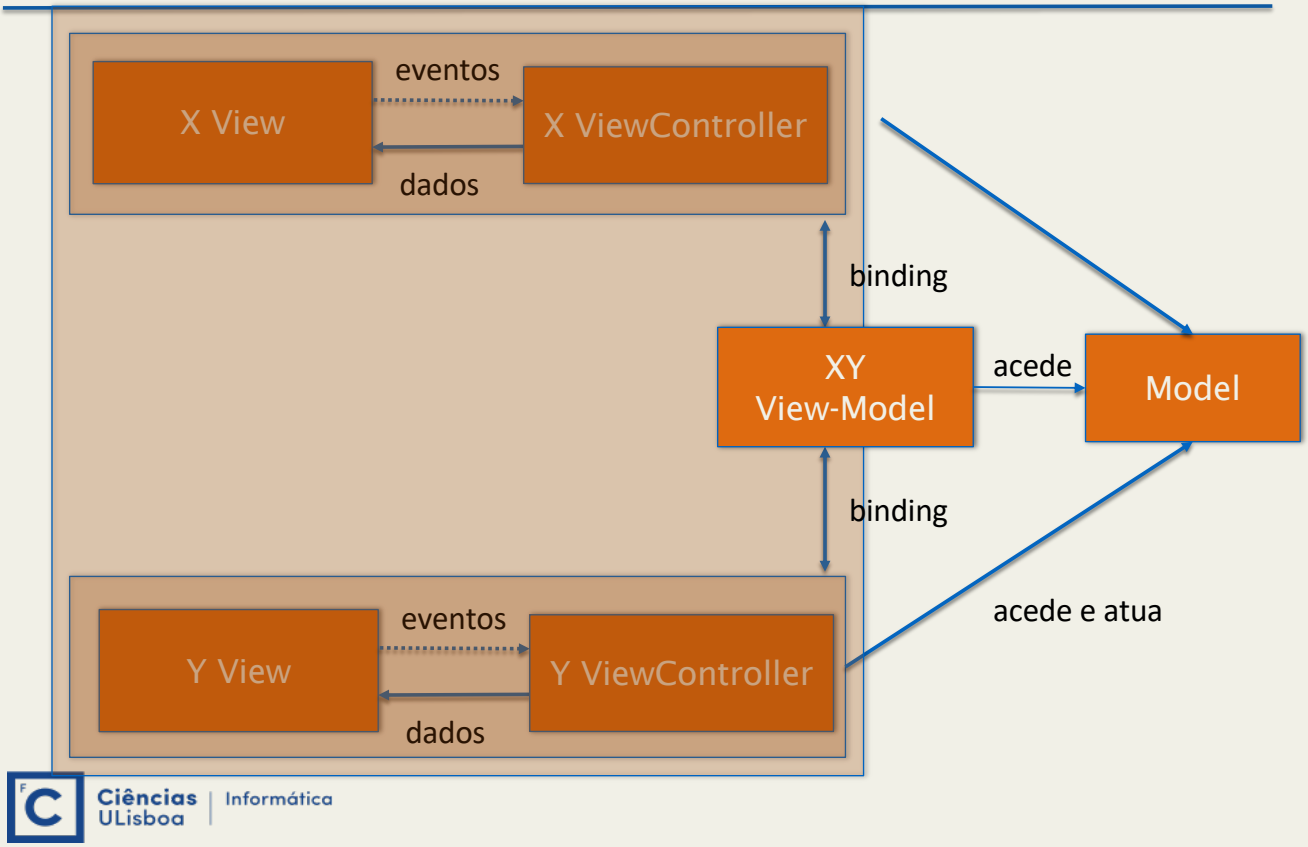


Ciências ULisboa | Info

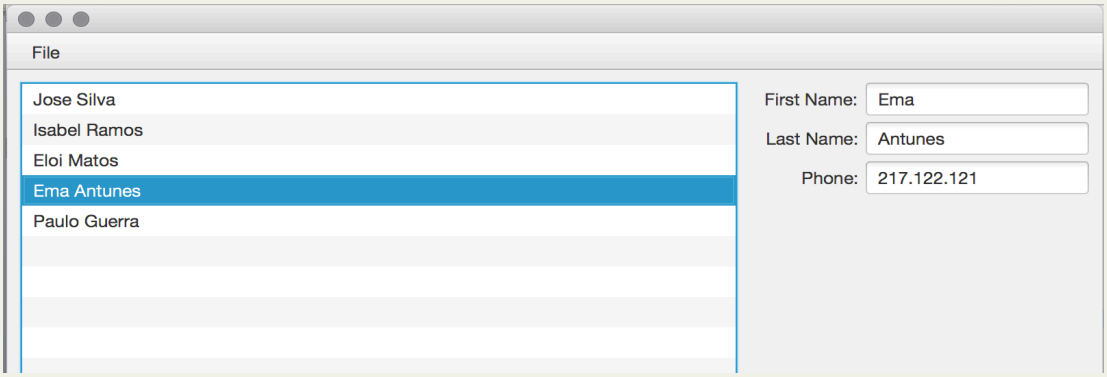
Exercícios



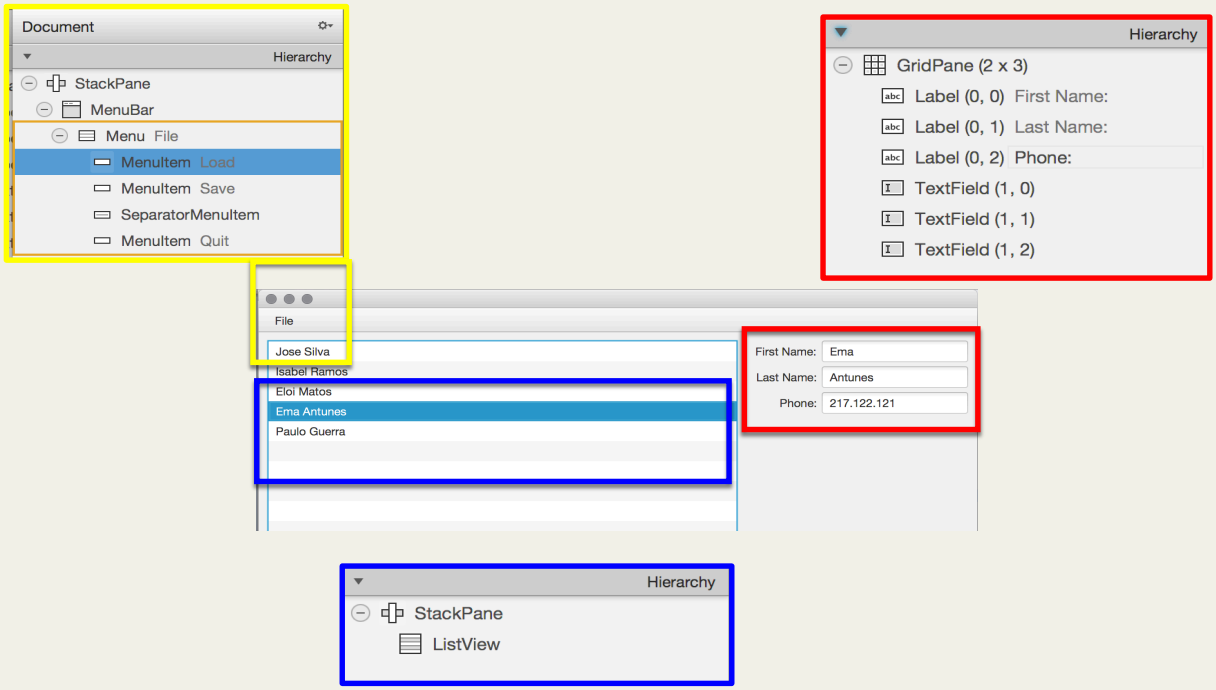
JavaFX e os padrões



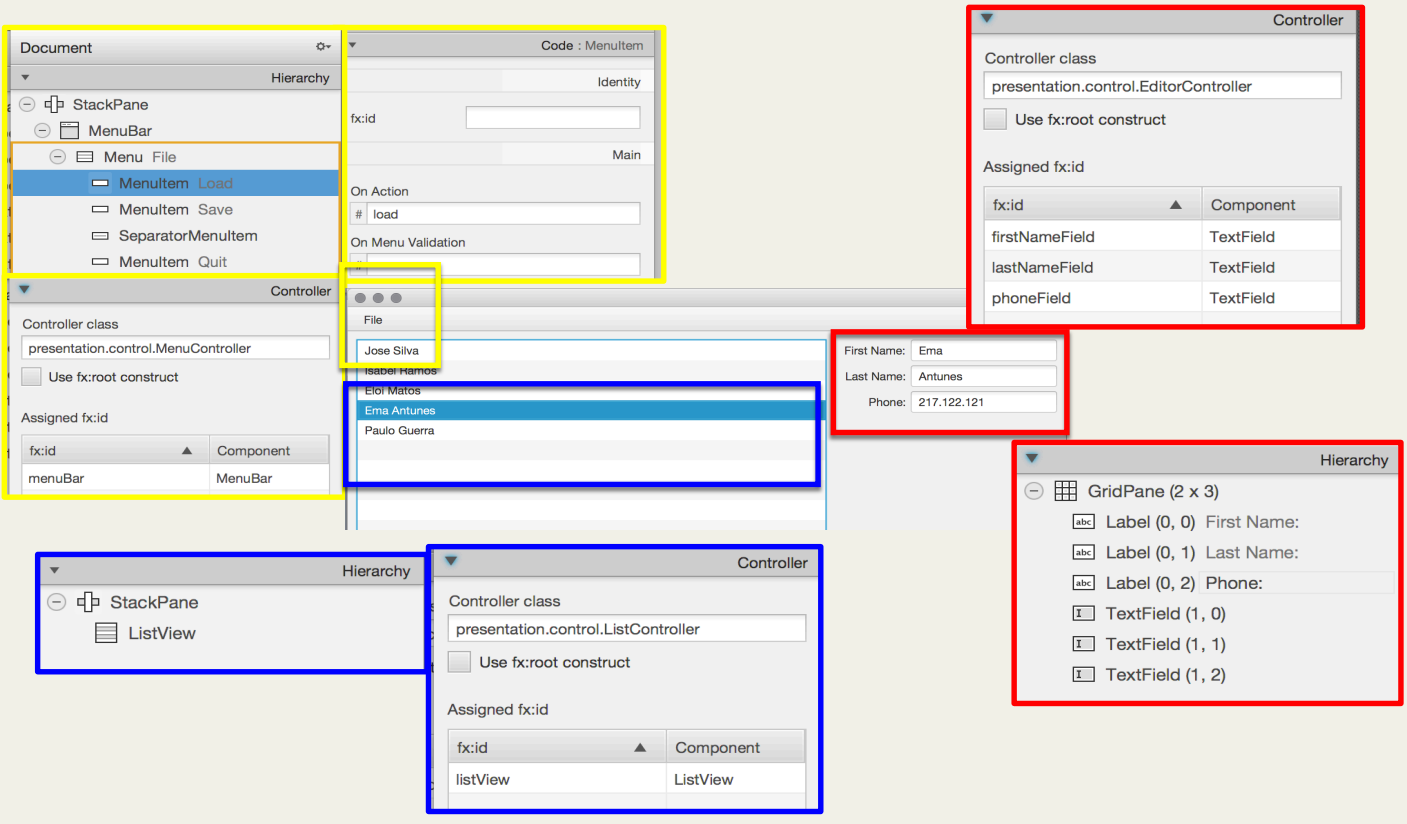
Componentes GUI: Views



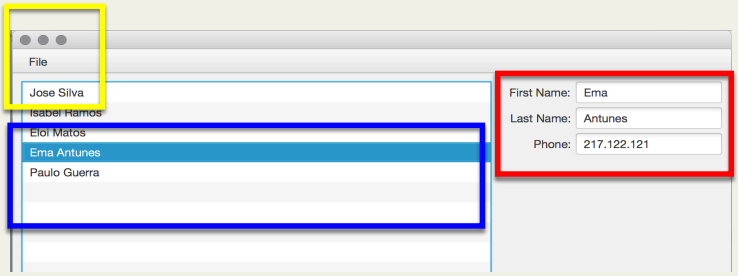
Componentes GUI: Views

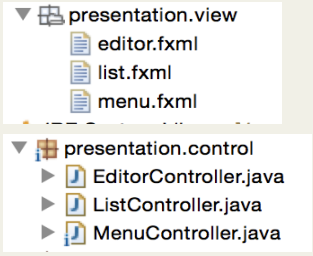


Componentes GUI: Views & View Controllers



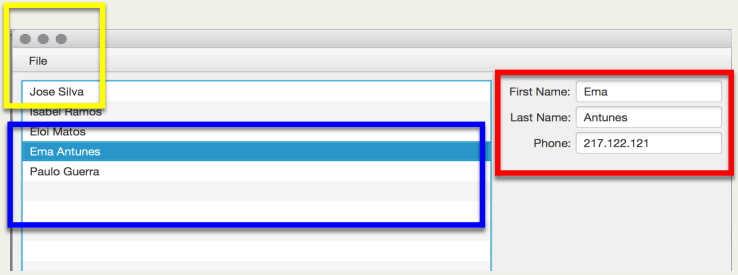
Scene Graph

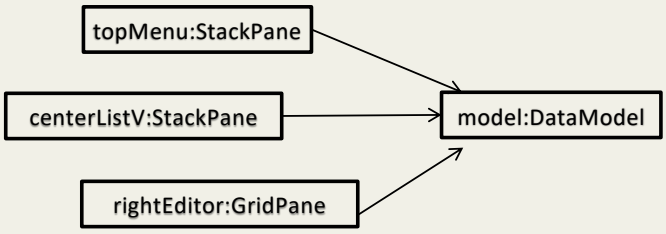




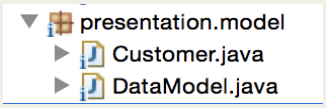

```
BorderPane root = new BorderPane();
FXMLLoader listLoader = new FXMLLoader(getClass().getResource("/presentation/view/list.fxml"));
root.setCenter(listLoader.load());
ListController listController = listLoader.getController();

FXMLLoader editorLoader = new FXMLLoader(getClass().getResource("/presentation/view/editor.fxml"));
root.setRight(editorLoader.load());
EditorController editorController = editorLoader.getController();

FXMLLoader menuLoader = new FXMLLoader(getClass().getResource("/presentation/view/menu.fxml"));
root.setTop(menuLoader.load());
MenuController menuController = menuLoader.getController();
```

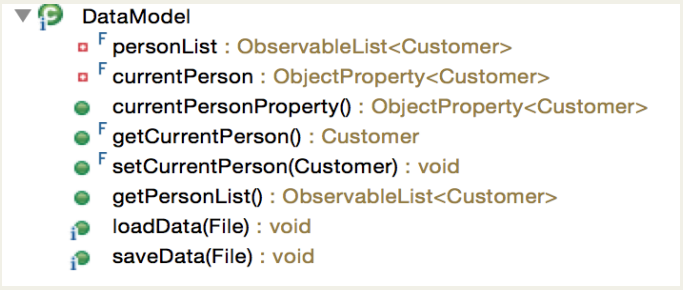
Scene Graph

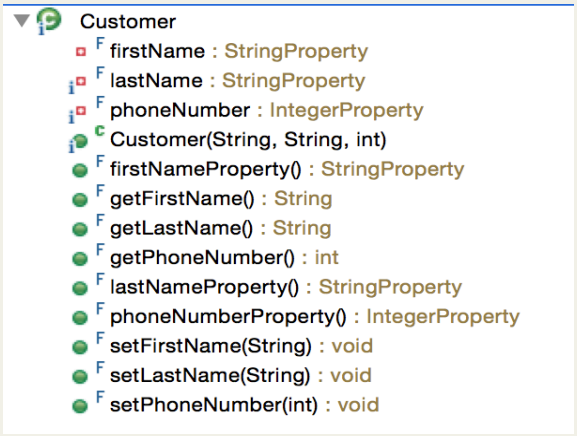


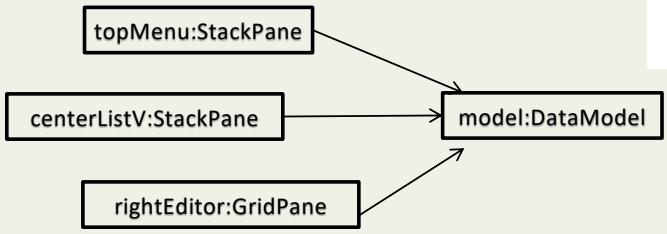


Dados: Model



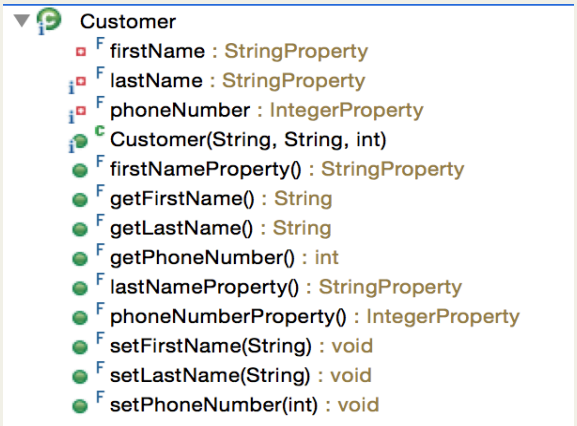


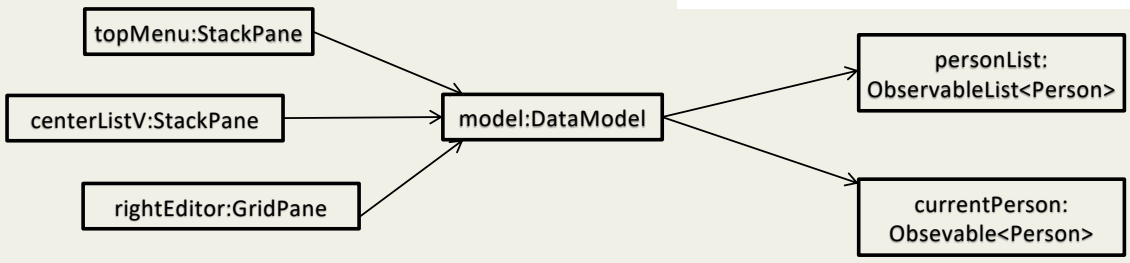




Ciências ULisboa | Informática

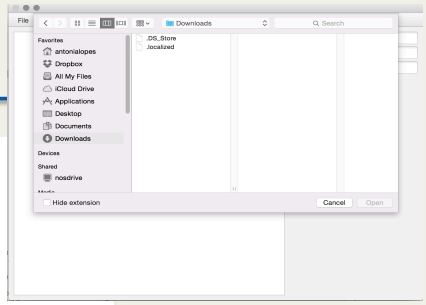
Dados: Model



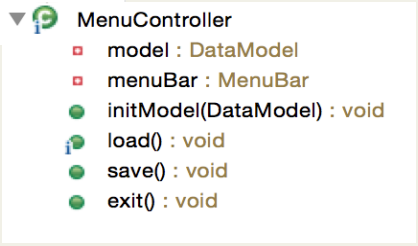


Ciências ULisboa | Informática

Componentes GUI: Controllers



```
public class MenuController {

    private DataModel model;

    @FXML
    private MenuBar menuBar;

    public void initModel(DataModel model) {
        if (this.model != null) {
            throw new IllegalStateException("Model can only be initialized once")
        }
        this.model = model;
    }

    @FXML
    public void load() {
        FileChooser chooser = new FileChooser();
        File file = chooser.showOpenDialog(menuBar.getScene().getWindow());
        if (file != null) {
            try {
                model.loadData(file);
```



Ciências ULisboa | Informática

Componentes GUI: Controllers

```
public class EditorController {

    @FXML
    private TextField firstNameField ;
    @FXML
    private TextField lastNameField ;
    @FXML
    private TextField phoneField ;

    private DataModel model ;

    public void initModel(DataModel model) {
        if (this.model != null) {
            throw new IllegalStateException("Model can only be initialized once");
        }
        this.model = model ;
        model.currentCustomerProperty().addListener((obs, oldCustomer, newCustomer) -> {
            if (oldCustomer != null) {
                firstNameField.textProperty().unbindBidirectional(oldCustomer.firstNameProperty());
                lastNameField.textProperty().unbindBidirectional(oldCustomer.lastNameProperty());
                phoneField.textProperty().unbindBidirectional(oldCustomer.phoneNumberProperty());
            }
            if (newCustomer == null) {
                firstNameField.setText("");
                lastNameField.setText("");
                phoneField.setText("");
            } else {
                firstNameField.textProperty().bindBidirectional(newCustomer.firstNameProperty());
                lastNameField.textProperty().bindBidirectional(newCustomer.lastNameProperty());
                phoneField.textProperty().bindBidirectional(newCustomer.phoneNumberProperty(),
                        new NumberStringConverter(new Locale ("pt", "PT")));
            }
        });
    }
}
```

Componentes GUI: Controllers

```
public class ListController {

    @FXML
    private ListView<Customer> listView ;

    private DataModel model;

    public void initModel(DataModel model) {
        if (this.model != null) {
            throw new IllegalStateException("Model can only be initialized once");
        }

        this.model = model ;
        listView.setItems(model.getCustomerList());

        listView.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) ->
        model.setCurrentCustomer(newSelection));
    }
```

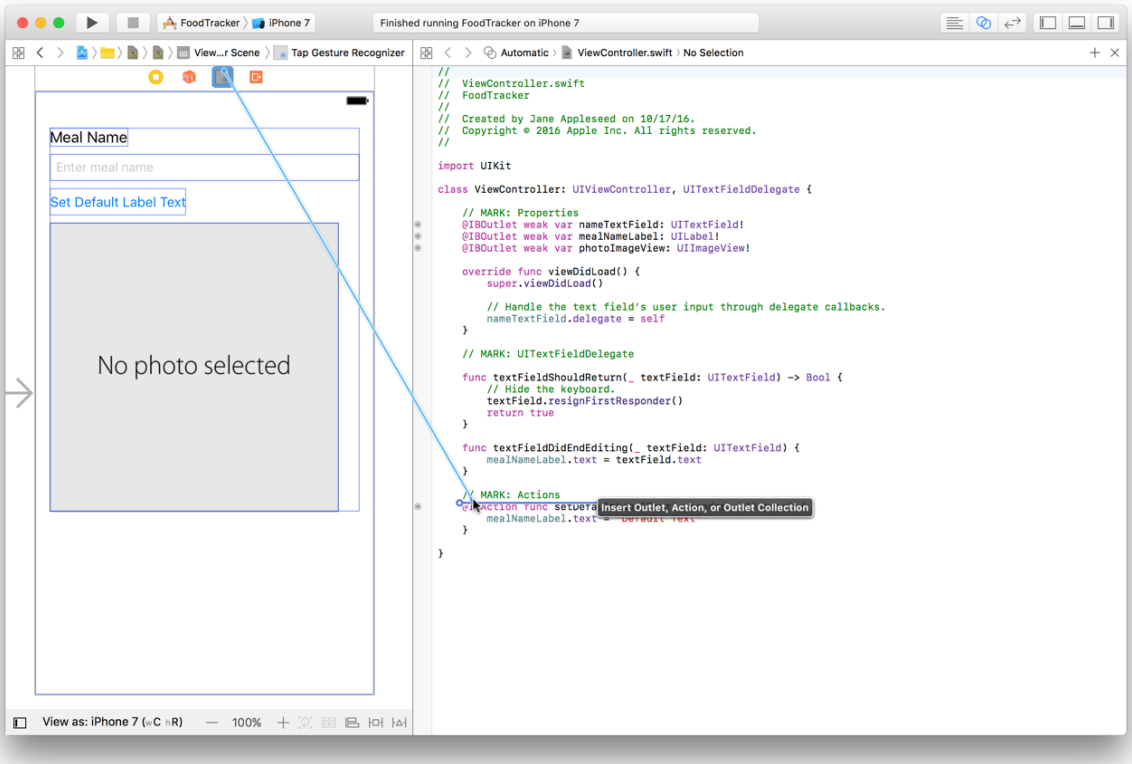


```
public class DataModel {

    /* in this way personList also reports
     * mutations of the elements in it by using the given extractor.
     * Observable objects returned by extractor (applied to each list element) are listened
     * for changes and transformed into "update" change of ListChangeListener.
     * since the phone is not visible, changes in the phone do not need to be propagated
     */
    private final ObservableList<Person> personList =
            FXCollections.observableArrayList(person ->
            new Observable[] {person.firstNameProperty(), person.lastNameProperty()});

    public ObservableList<Person> getCustomerList() {
        return personList;
    }

    private final ObjectProperty<Person> currentCustomer = new SimpleObjectProperty<>(null);

    public ObjectProperty<Person> currentCustomerProperty() {
        return currentCustomer;
    }

    public final Person getCurrentCustomer() {
        return currentCustomerProperty().get();
    }

    public final void setCurrentCustomer(Person person) {
        currentCustomerProperty().set(person);
    }

    public void loadData(File file) {
```

Application

```
public class DemoGUIApp extends Application {

    @Override
    public void start(Stage primaryStage) throws Exception {

        BorderPane root = new BorderPane();
        FXMLLoader listLoader = new FXMLLoader(getClass().getResource("/presentation/view/list.fxml"));
        root.setCenter(listLoader.load());
        ListController listController = listLoader.getController();

        FXMLLoader editorLoader = new FXMLLoader(getClass().getResource("/presentation/view/editor.fxml"));
        root.setRight(editorLoader.load());
        EditorController editorController = editorLoader.getController();

        FXMLLoader menuLoader = new FXMLLoader(getClass().getResource("/presentation/view/menu.fxml"));
        root.setTop(menuLoader.load());
        MenuController menuController = menuLoader.getController();

        DataModel model = new DataModel();
        listController.initModel(model);
        editorController.initModel(model);
        menuController.initModel(model);

        Scene scene = new Scene(root, 800, 600);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```

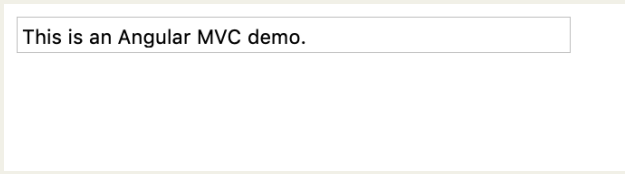
MVVM em iOS/MacOS



MVVM em iOS/MacOS

```swift
class AddArticleViewController : UIViewController {

    // For this example assume they are UITextField
    @IBOutlet weak var titleTextField :BindingTextField!
    @IBOutlet weak var descriptionTextField :BindingTextField!

    var viewModel :AddArticleViewModel! {

        didSet {
            viewModel.title.bind = { [unowned self] in self.titleTextField.text = $0 }
            viewModel.description.bind = { [unowned self] in self.descriptionTextField.text
        }
    }

    @IBAction func AddArticleButtonPressed(_ sender: Any) {

        self.viewModel.title.value = "hello world"
        self.viewModel.description.value = "description"
    }

    override func viewDidLoad() {

        super.viewDidLoad()
        self.viewModel = AddArticleViewModel()
    }

}
```

MVVM em Angular

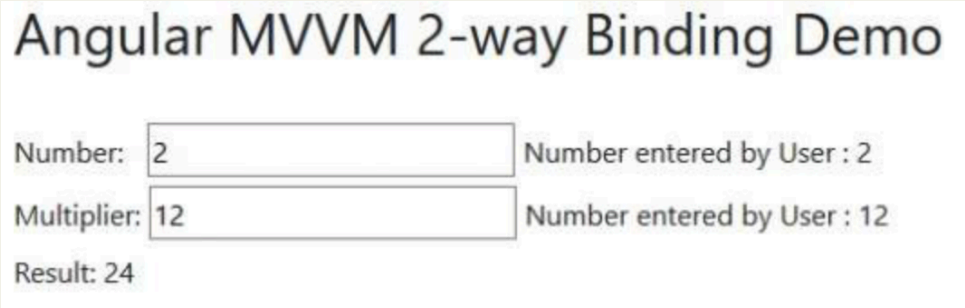



```html
<!DOCTYPE html>
<html ng-app>
<head>
    <title>Angular MVC Demo</title>
    <script type="text/javascript" src="angular.min.js"></script>
</head>
<body ng-controller="TextController">
<p>{{sampleText}}</p>
</body>
<script>
    function TextController($scope) {
        $scope.sampleText = 'This is an Angular MVC demo.';
    }
</script>
</html>
```

Controller da View

Placeholder binded ao view-model

MVVM em Angular (Bidirectional Binding)



MVVM em Angular (Bidirectional Binding)

```
<html ng-app>
<head>
   <title>Angular MVVM 2-way Binding Demo</title>
</head>
<body>
   <h2>Angular MVVM 2-way Binding Demo</h2>
   <br/>
   <form ng-controller="MultiplicationController">
      <label>Number:</label> <input name="number" ng-change="multiply()" ng-model="data.number">
      <label>Number entered by User :</label> {{data.number}} <br>
      <label>Multiplier:</label> <input name="multiplier" ng-change="multiply()" ng-model="data.multiplier">
      <label>Number entered by User :</label> {{data.multiplier}} <br>
      <label>Result:</label> {{data.result}}
   </form>
</body>
<script>
   function MultiplicationController($scope) {
      $scope.data = {number: 0, multiplier: 0, result: 0};
      $scope.multiply = function() {
         $scope.data.result = $scope.data.number * $scope.data.multiplier;
      }
   }
</script>
</html>
```

ViewModel

MVVM em Angular (Architecture)