



UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Laboratorio di Programmazione ad Oggetti

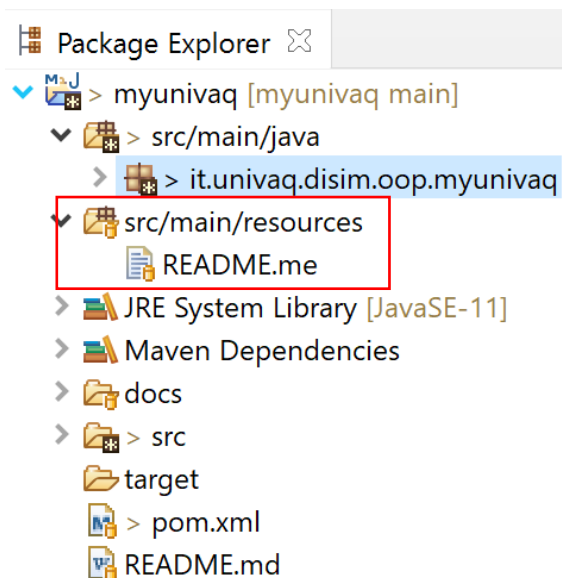
Ph.D. Juri Di Rocco
juri.dirocco@univaq.it
<http://jdirocco.github.io>





Cartella risorse

La cartella `src/main/resources` conterrà tutte le viste, immagini dell'applicazione, ovvero tutte le risorse utilizzate dall'applicazione



File `README.me` presente in quanto GitHub non permette di avere cartelle vuote dentro un repository

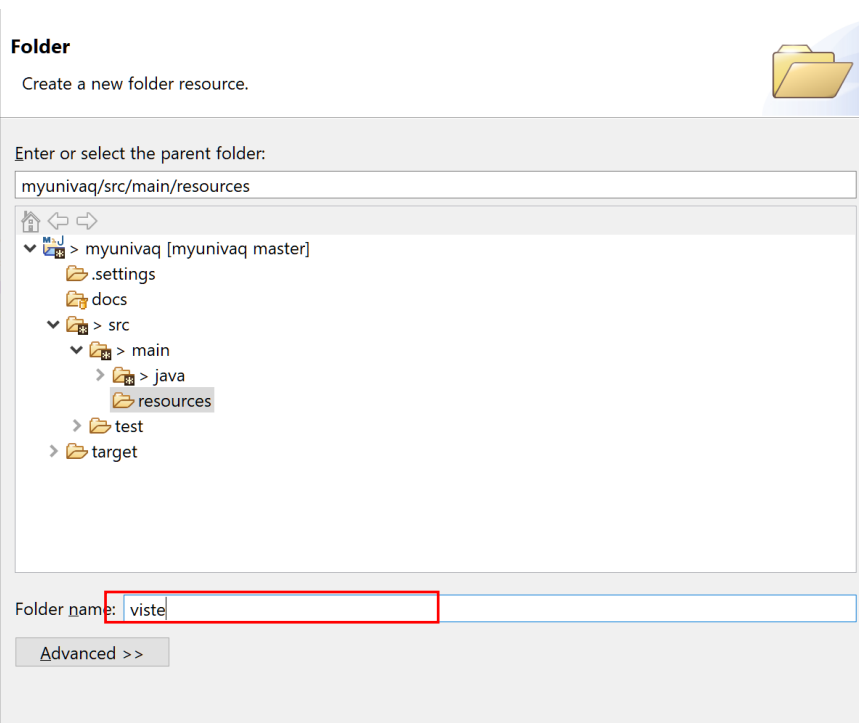
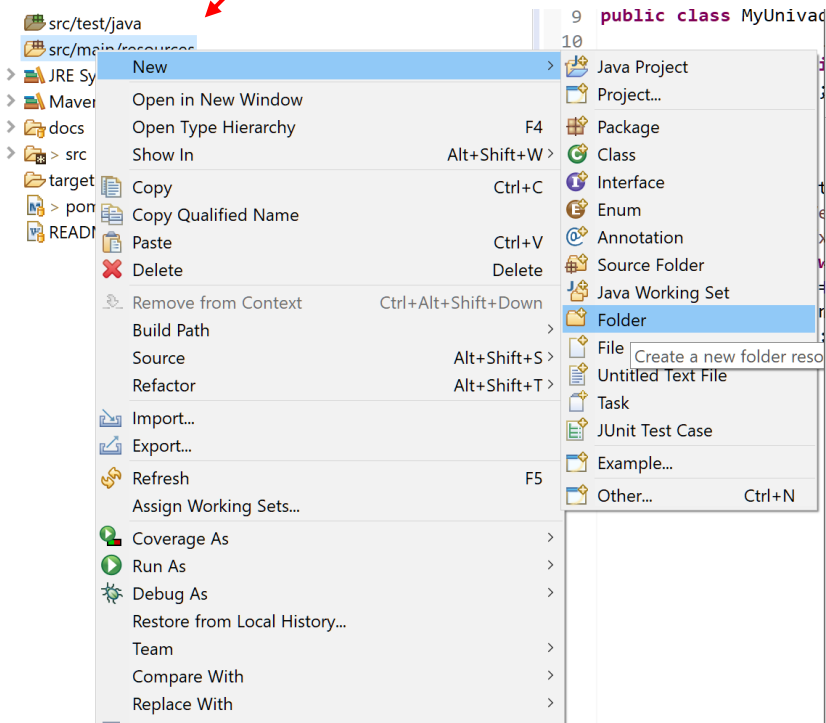


Aggiunta cartella viste dentro risorse

Tasto destro mouse su `src/main/resources`

Selezionare Folder

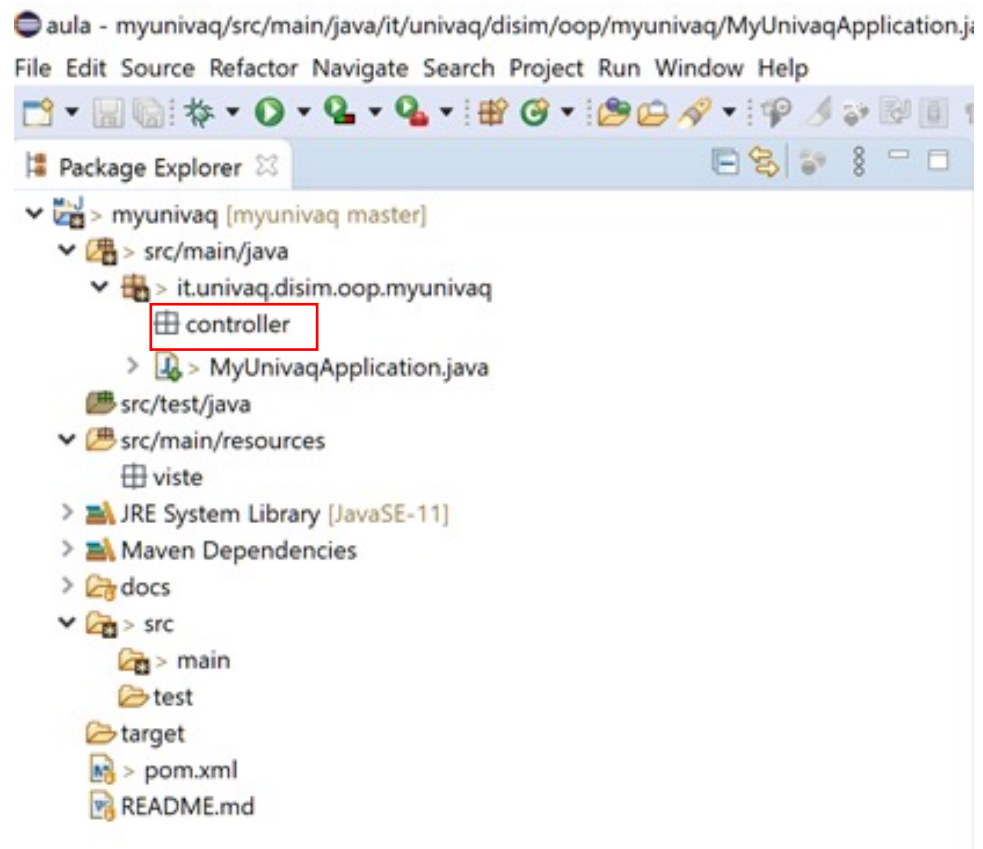
Successivamente inserire il nome **viste**





Creare package controller

- › Creare il package controller dentro il package `it.univaq.disim.oop.myunivaq`
- › Tale package conterrà tutti i controllori associati alle viste





Struttura ad albero package (opzionale)

aula - myunivaq/src/main/java/it/univaq/disim/oop/myunivaq/MyUnivaqApplication.java - Eclipse IDE

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer

- myunivaq [myunivaq master]
 - src/main/java
 - it.univaq.disim.oop.myunivaq**
 - controller**
 - MyUnivaqApplication.java
 - src/test/java
 - src/main/resources
 - viste
 - JRE System Library [JavaSE-11]
 - Maven Dependencies
 - docs
 - src
 - main
 - test
 - target
 - pom.xml
 - README.md

MyUnivaqApplication.java

Top Level Elements

- Select Working Set...
- Deselect Working Set
- Edit Active Working Set...
- 1 Window Working Set
- Filters...
- Package Presentation
 - Flat
 - Hierarchical**
- ✓ Show 'Referenced Libraries' Node
- Link with Editor
- Focus on Active Task

```
17 String javaVersion = Syst
18 String javafxVersion = Sy
19 Label l = new Label("Hell
20 Scene scene = new Scene(n
21 stage.setScene(scene);
22 stage.show();
23
24 }
25
```



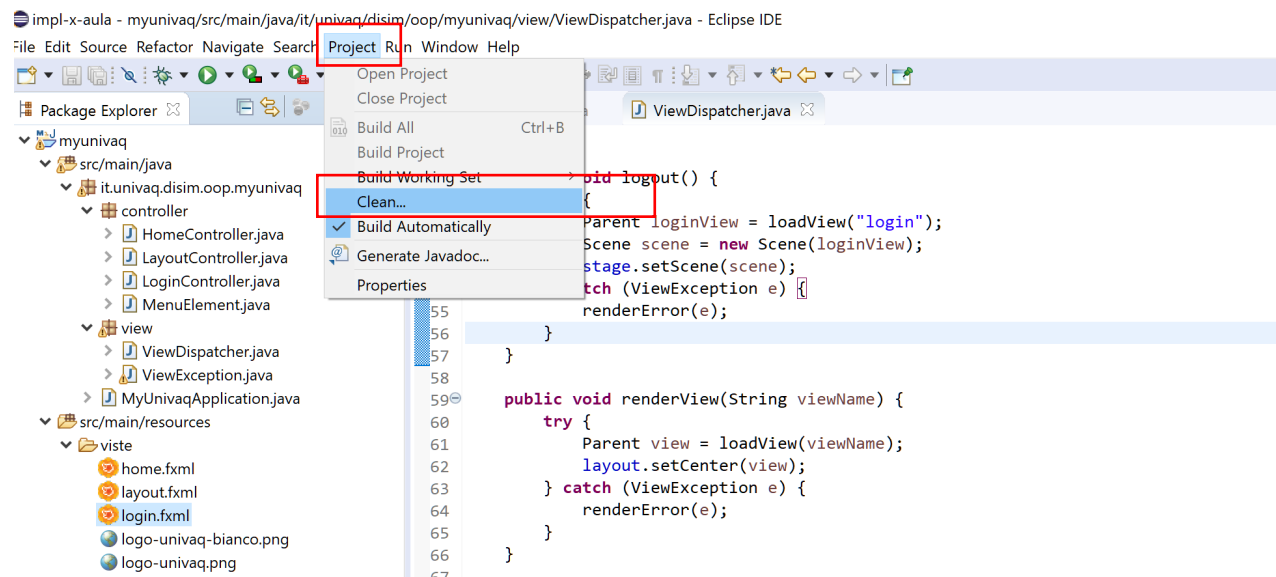
Login (1)

- › Creare dentro cartella **viste** file `login.fxml`
- › Aprire il file `login.fxml` con Scene Builder e costruire la pagina di login
- › Creare classe Controller
- › Caricare la vista con `FXMLLoader`



NOTA IMPORTANTE

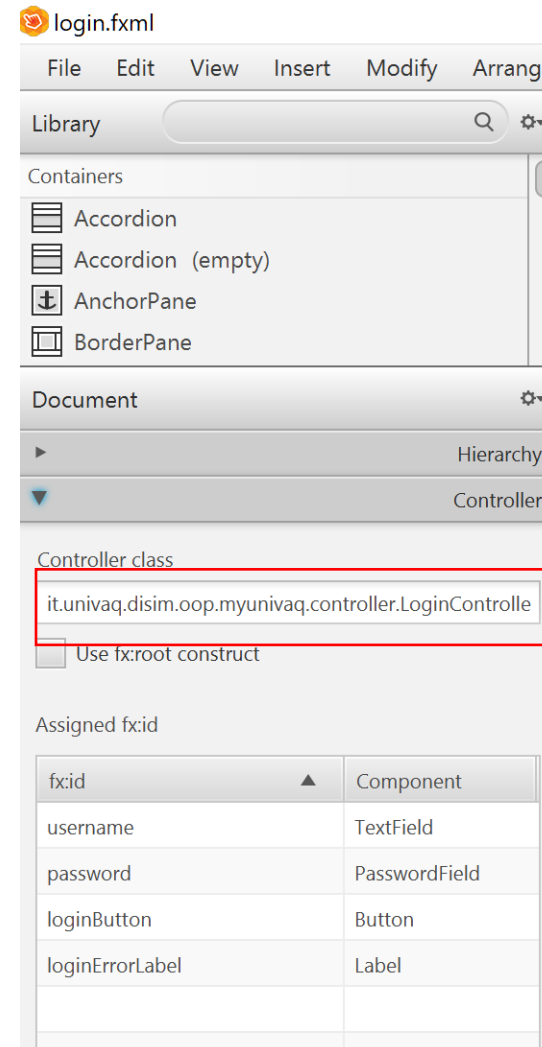
- › Ogni volta che modificate una vista con Scene Builder all'interno di Eclipse dovete cliccare su Project → Clean
- › Tale azione comporta l'aggiornamento della vista dentro la cartella `target/classes` dove effettivamente JavaFX carica le viste durante l'esecuzione





Controller (1)

- › Ad ogni vista deve essere **associato** un controllore che ha il compito di gestire la vista ovvero
 - Gestire gli eventi generati dall'utente
 - Aggiornare dinamicamente la vista
 - **Invocare la logica di business** di una specifica vista al verificarsi di un evento





Controller (2)

- › Ha il compito di gestire la vista
- › Classe deve implementare interfaccia `Initializable` ovvero il metodo

```
@Override
```

```
public void initialize(URL url, ResourceBundle rb) {
```

```
}
```

- › Tale metodo viene invocato da JavaFX dopo che è stata caricata la vista dal loader (`FXMLLoader`) e dopo che ha creato il controllore (tramite `new`)
- › Annotazione `@FXML` specificata nelle variabili di istanza **legano** l'elemento grafico presente all'interno della vista con la variabile di istanza specificata
- › Nome della variabile deve essere uguale al valore dell'attributo `fx:id` del componente



Controller (3)

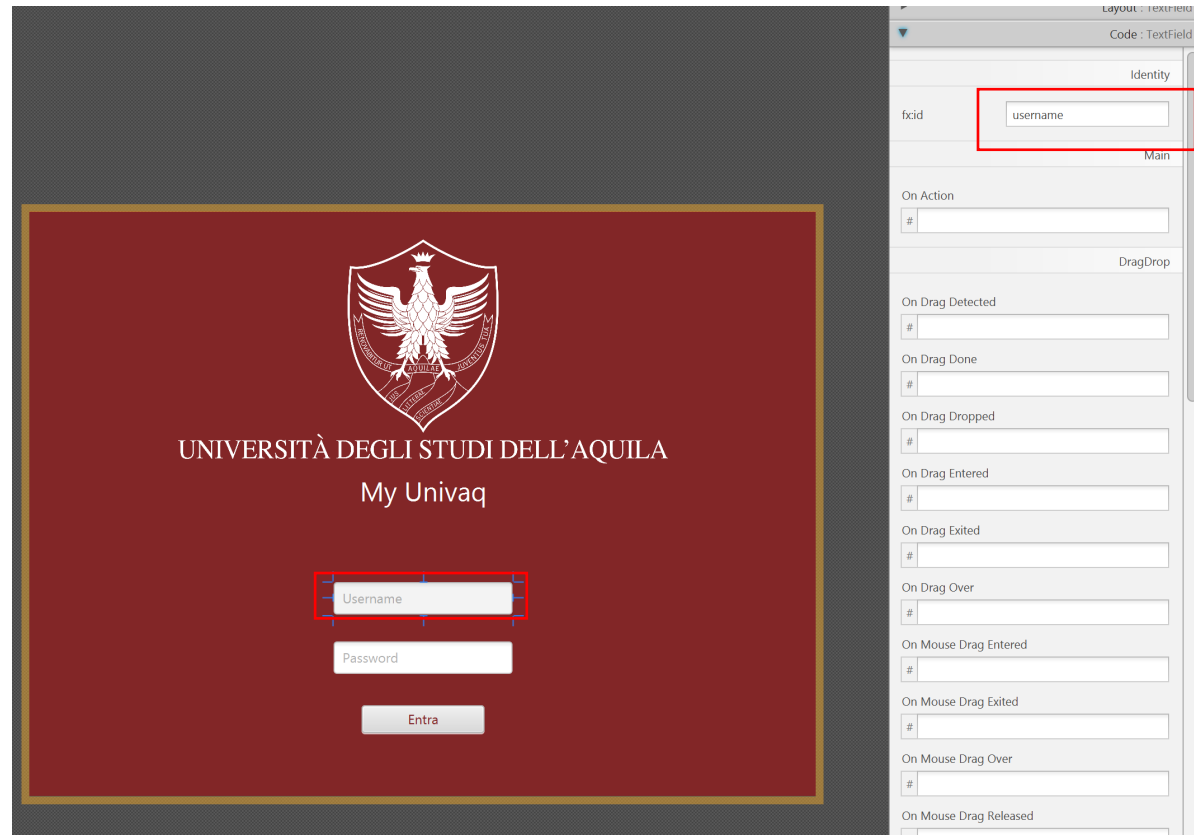
- › Per ogni elemento grafico si deve specificare l'`fx:id` che sarà utilizzato all'interno del controllore per *manipolare* il componente

@FXML

```
private TextField username;
```

- › Ad esempio prendere o modificare il testo della username

```
username.setText("");
```





Controller (4)

- › JavaFX definisce vari tipi di **eventi** di input standard per il mouse, gestures, touch, o chiavi
- › Ognuno di questi tipi di input ha degli *handlers* (gestori) che li processano
- › Annotazione `@FXML` utilizzata nella definizione di metodi per specificare tali handlers
- › Esempio click mouse

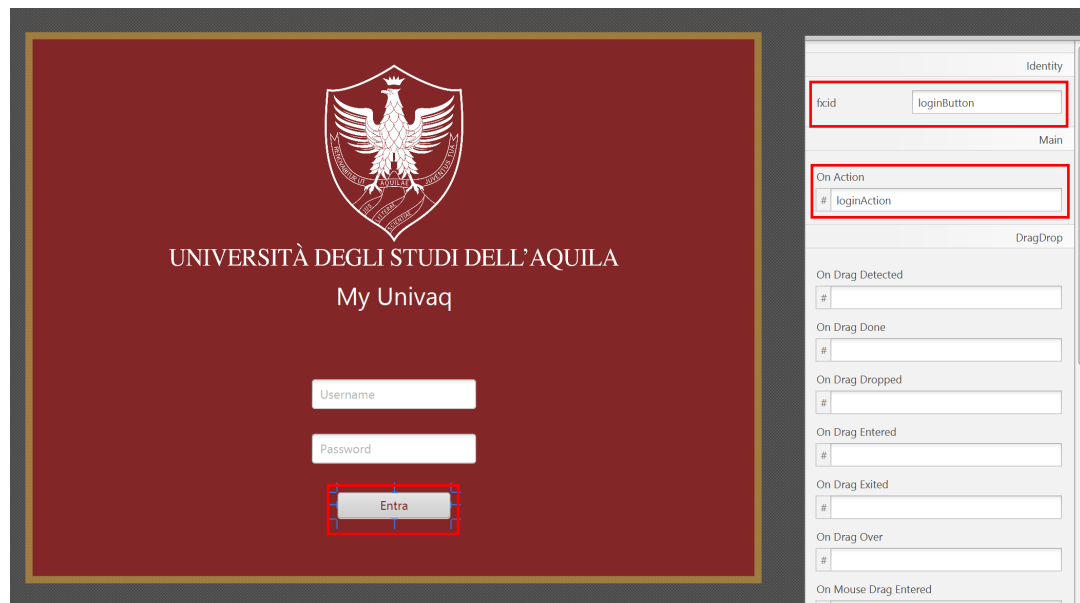
`@FXML`

```
private void loginAction(ActionEvent event) {  
    .....  
}
```



Controller (5)

- › Specificare l'`fx:id` nel componente che sarà utilizzato all'interno del controllore per *manipolare* il pulsante
- › Si può indicare un **metodo** sull'evento **on Action** che verrà invocato da JavaFX quando l'utente clicca sul pulsante
- › Esempio `loginAction`





Proprietà (1)

- › Proprietà JavaFX sono utilizzate in congiunzione con il binding
- › Binding potente meccanismo per esprimere relazioni dirette tra variabili
- › Se gli oggetti partecipano al binding, i cambiamenti di un oggetto si rifletteranno automaticamente sull'altro
- › Esempio: un pulsante diventa cliccabile se un campo di testo è stato popolato dall'utente
- › Proprietà JavaFX sono molto potenti in quanto sono *osservabili* (observables)
 - Osservabile vuol dire che a fronte di un cambiamento (o se diventa non valida) della proprietà, tale cambiamento viene notificato ad alcuni *oggetti*, detti *ascoltatori* (listeners)



Proprietà (2)

- › E' possibile utilizzare il meccanismo di binding built-in per collegare il valore di una proprietà ad un'altra proprietà
- › E' possibile definire delle proprie proprietà JavaFX
- › Per effettuare il **binding** di espressioni o collegare **listeners** a proprietà JavaFX è necessario accedere a tali proprietà
- › Convenzione nome della proprietà in minuscolo seguita dalla parola `Property`
- › Esempio

`fillProperty()` per il riempimento

`opacityProperty()` per l'opacità



Proprietà (3)

- › I listeners di proprietà JavaFX che si applicano alle proprietà degli oggetti (non alle collezioni) sono disponibili in due versioni **invalidation** listeners e **change** listeners
- › Gli **invalidation** listeners si attivano quando il valore di una proprietà non è più valido
- › Quando si ha bisogno di accedere al valore precedente di un observable oltre che al suo valore attuale, utilizzare un **change** listener
 - I change listeners forniscono l'observable e i valori nuovi e vecchi. Cambiare listener può essere più costoso, poiché deve tenere traccia di più informazioni



Proprietà (4)

```
public Slider createSlider() {  
    Slider slider = new Slider(12, 144, 24);  
    slider.setMajorTickUnit(12);  
    slider.setMinorTickCount(3);  
    slider.setShowTickMarks(true);  
    slider.setShowTickLabels(true);  
    slider.setSnapToTicks(true);  
    slider.setPrefWidth(500);  
    slider.setMinWidth(500);  
    slider.setMaxWidth(500);  
    return slider;  
}
```




Proprietà (5)

› Esempio di invalidation Listener

```
@Override
public void start(Stage primaryStage) {
    try {

        primaryStage.setTitle("ChangeListener Demo");
        BorderPane root = new BorderPane();
        slider = createSlider();
        label = new Label("Font Size: 24");
        label.setFont(Font.font(24));
        vbox = FXUtils.createVBox(slider, label);
        root.setCenter(vbox);

        slider.valueProperty().addListener(new ChangeListener<Number>() {

            @Override
            public void changed(ObservableValue<? extends Number> observable, Number oldValue, Number
newValue) {

                label.setFont(new Font(newValue.doubleValue()));
                label.setText("Font Size: " + newValue.intValue());

            }

        });

        Scene scene = new Scene(root, 1000, 400);
        scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
        primaryStage.setScene(scene);
        primaryStage.show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```



Binding (1)

- › Binding JavaFX è un meccanismo flessibile che elimina la scrittura di listeners in molte situazioni
- › Binding si può utilizzare per collegare il valore di una proprietà JavaFX a un'altra (o più) proprietà
- › Binding delle proprietà può essere **uni-direzionale** o **bi-direzionale**
- › `bind()` per binding uni-direzionale
- › `bindBidirectional()` binding bi-direzionale



Binding (2): uni-direzionale

- › Forma più semplice è collegare il valore di una proprietà con il valore di un'altra
- › Esempio

```
text2.rotateProperty().bind(stackPane.rotateProperty());
```

- › Qualsiasi cambiamento di rotazione allo `stackPane` cambierà immediatamente la proprietà di rotazione di `text2`
- › Questo implica che `stackPane` e `text2` ruotano insieme



Binding (3): bi-direzionale

- › Fornisce una relazione *a due vie* (two-way) tra le due proprietà
- › Quando una proprietà si aggiorna anche l'altra viene aggiornata

```
text2.textProperty().bindBidirectional(text.textProperty());
```

- › Se dentro `text` viene modificato il testo anche dentro `text2` viene modificato e viceversa
- › Binding bi-direzionale non è completamente simmetrico
 - Il valore iniziale di entrambe le proprietà assume il valore della proprietà passata nella chiamata a `bindBidirectional()`



Binding (4): Fluent API e Bindings API

- › Fluent e Binding API permettono di costruire espressioni di bindings quando una proprietà (o più proprietà) è coinvolta nel binding o quando è necessario effettuare qualche calcolo o conversione

- › Esempio

```
text2.textProperty().bind(stackPane.rotateProperty().asString("%.1f"));
```

- › Visualizza nella campo di testo `text2` visualizza l'angolo di rotazione (da 0 a 360 gradi)
- › Proprietà `rotateProperty()` è un `double` mentre `textProperty` è `String`
 - `asString()` converte il `double` in `String` formattando il numero con una sola cifra a destra del punto decimale



Binding (5): Fluent API e Bindings API

- › La proprietà colore (stroke) di `text2` dipende se l'animazione è in esecuzione oppure no

```
text2.strokeProperty().bind(new When(rotate.statusProperty()  
    .isEqualTo(Animation.Status.RUNNING))  
    .then(Color.GREEN)  
    .otherwise(Color.RED));
```



Caso di studio UnivaqFX



Login: Login Controller (1)

```
public class LoginController implements Initializable {  
    @FXML  
    private Label loginErrorLabel;  
    @FXML  
    private TextField username;  
    @FXML  
    private TextField password;  
    @FXML  
    private Button loginButton;  
    @Override  
    public void initialize(URL location, ResourceBundle resources) {  
        loginButton.disableProperty().bind(username.textProperty().isEmpty()  
                                            .or(password.textProperty().isEmpty()));  
    }  
}
```

.....



Login: Login Controller (2)

...

@FXML

```
private void loginAction(ActionEvent event) {  
    if (!("docente".equals(username.getText()) &&  
        ("docente".equals(password.getText()))) {  
        loginErrorLabel.setText("Username e/o password errati!");  
    } else {  
        //Bisogna caricare la vista successiva  
    }  
}  
}
```




Login: Caricamento della vista

```
public class MyUnivaqApplication extends Application {  
    public static void main(String[] args) {  
        launch(args);  
    }  
    @Override  
    public void start(Stage stage) throws Exception {  
        FXMLLoader loader = new  
            FXMLLoader(getClass().getResource("/viste/login.fxml"));  
        Parent login = loader.load();  
        Scene scene = new Scene(login);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```



Login: Risultato finale



UNIVERSITÀ DEGLI STUDI DELL'AQUILA

My Univaq

Entra



Benvenuto (1)

- › E' necessario creare una vista *contenitore* per mantenere coerente il layout dell'applicazione
- › Vi è una vista (chiamata `layout`) che utilizza il layout di `BorderPane` come root dove
 - **Top** vi è il logo di ateneo e a destra il pulsante per uscire dall'applicazione
 - **Sinistra** troviamo le voci di menu che variano a seconda dell'utente (docente o studente)
 - **Centro** vi saranno le diverse viste a seconda delle funzionalità che l'utente sceglie



Benvenuto (2)

- › Creare dentro cartella **viste** file `layout.fxml`
- › Aprire il file `layout.fxml` con Scene Builder e costruire la pagina di *struttura* dell'applicazione
- › Creare la classe Controllore ovvero `LayoutController` dentro il package `it.univaq.disim.oop.myunivaq.controller`
- › Il menu che si trova sulla sinistra non viene creato dentro lo Scene Builder ma deve essere creato dinamicamente all'interno del `LayoutController` a seconda del tipo di utente
 - Per ora viene creato sempre quello del docente
 - **Successivamente** provvederemo a creare il menu in base al tipo di utente (docente o studente)



Benvenuto (3)

layout.fxml

File Edit View Insert Modify Arrange Preview Window Help

Library

Containers

- Accordion
- Accordion (empty)
- AnchorPane
- BorderPane
- ButtonBar (FX8)
- DialogPane (empty) (FX8)
- DialogPane (FX8)
- FlowPane
- GridPane
- HBox
- Pane

Document

Hierarchy

Controller

Controller class

it.univaq.dsim.oop.myunivaq.controller.LayoutControl

Use fxmlroot construct

Assigned fcid

fcid	Component
menuBar	VBox
esciButton	ImageView

2 items

Inspector

Properties: VBox

Node

Alignment: TOP_LEFT

Disable: ☐

Opacity: 1

Node Orientation: INHERIT

Visible: ☒

Focus Traversable: ☐

Cache Shape: ☒

Center Shape: ☒

Scale Shape: ☒

Opaque Insets: 0 0 0 0

Cursor: Inherited (Default)

Effect: +

JavaFX CSS

Style

-fx-background-color: #822627

Style Class

Stylesheets

+

Id

Extras

Blend Mode: SRC_OVER

Cache: ☐

Cache Hint: DEFAULT

Depth Test: INHERIT

Insets: 0 0 0 0

Mouse Transpar...: ☐

Layout: VBox

Code: VBox



Benvenuto (4): LayoutController

```
private static final MenuElement MENU_HOME = new MenuElement("Home", "home");

private static final MenuElement[] MENU_DOCENTI = { new MenuElement("Gestione Esami", "insegnamenti"),
    new MenuElement("Approvazione Piani", "piani"), new MenuElement("Sedute Di Laurea", "sedute-laurea"),
    new MenuElement("Laureandi Assegnati", "laureandi"), new MenuElement("Lezioni", "lezioni"),
    new MenuElement("Diario", "diario"), new MenuElement("Conseguimento Titoli", "titoli"),
    new MenuElement("Questionari", "questionari") };

@FXML

private VBox menuBar;

@Override

public void initialize(URL location, ResourceBundle resources) {

    menuBar.getChildren().addAll(createButton(MENU_HOME));

    menuBar.getChildren().add(new Separator());

    for (MenuElement menu : MENU_DOCENTI) {

        menuBar.getChildren().add(createButton(menu));

    }

}
```



Benvenuto (5): LayoutController

```
@FXML
```

```
public void esciAction(MouseEvent event) {
```

```
}
```

```
private Button createButton(MenuElement viewItem) {
```

```
    Button button = new Button(viewItem.getNome());
```

```
    button.setStyle("-fx-background-color: transparent; -fx-font-size: 14;");
```

```
    button.setTextFill(Paint.valueOf("white"));
```

```
    button.setPrefHeight(10);
```

```
    button.setPrefWidth(180);
```

```
    button.setOnAction((ActionEvent event) -> {
```

```
        dispatcher.renderView(viewItem.getVista(), utente);
```

```
    });
```

```
    return button;
```

```
}
```




Benvenuto (6): MenuElement

Classe di utility che contiene il titolo del menu (variabile e nome) e il nome della vista

```
public class MenuElement {  
    private String nome;  
    private String vista;  
    public MenuElement(String nome, String vista) {  
        this.nome = nome;  
        this.vista = vista;  
    }  
    public String getNome() {return nome;}  
    public void setNome(String nome) {this.nome = nome;}  
    public String getVista() {return vista;}  
    public void setVista(String vista) {this.vista = vista;}  
}
```



Benvenuto (6): vista Home

- › Creare dentro cartella **viste** file `home.fxml`
- › Aprire il file `home.fxml` con Scene Builder
- › Creare la classe Controllore ovvero `HomeController` dentro il package `it.univaq.disim.oop.myunivaq.controller`

```
public class HomeController implements Initializable {  
    @FXML  
    private Label benvenutoLabel;  
    @Override  
    public void initialize(URL location, ResourceBundle resources) {  
        benvenutoLabel.setText("Benvenuto docente!");  
    }  
}
```



Benvenuto (7): vista Home

home.fxml

File Edit View Insert Modify Arrange Preview Window Help

Library No Selection

Containers

- Accordion
- Accordion (empty)
- AnchorPane
- BorderPane
- ButtonBar (FX8)
- DialogPane (empty) (FX8)
- DialogPane (FX8)
- FlowPane
- GridPane
- HBox
- Pane
- ScrollPane
- ScrollPane (empty)
- SplitPane (empty)
- SplitPane (horizontal)

Document

Hierarchy

Controller

Controller class

☐ Use fx:root construct

Assigned fx:id

fx:id	Component
benvenutoLabel	Label

Benvenuto



Benvenuto (6): visualizzare la pagina di benvenuto

- › La vista di benvenuto (home) dovrà essere inserita nella parte centrale della vista di layout dopo che si è effettuato il login
- › La vista di layout dovrà essere la **nuova** scena della applicazione
- › Pertanto all'interno del `LoginController` si dovrebbe
 - Caricare la vista di layout (tramite `FXMLLoader`)
 - Caricare la home (tramite `FXMLLoader`)
 - Sostituire la scena di login all'interno dello stage con la nuova scena layout
- › Quest'ultima operazione non è possibile in quanto `LoginController` **non ha lo stage** → conviene *centralizzare* il caricamento/creazione e il *dispatching* delle viste



Benvenuto (7): visualizzare la pagina di benvenuto

- › A tal fine si è definita una classe `ViewDispatcher` (dentro un nuovo package `view`) che ha il compito
 - Gestire lo stage e di conseguenza cambiare la scena
 - Gestire il caricamento e la visualizzazione di tutte le viste
 - Tale scelta ha il vantaggio di **disaccoppiare** i controllori dalla gestione delle viste



ViewDispatcher (1)

- › Implementata come istanza singola (pattern Singleton)
 - Si è scelto di utilizzare una unica istanza in quanto non servono diversi oggetti di tale classe ma è sufficiente una unica istanza per tutta la applicazione
 - Implementazione
 1. Costruttore privato
 2. Variabile statica `instance` di tipo `ViewDispatcher`
 3. Metodo statico

```
public static ViewDispatcher getInstance()
```
- › Metodi utilizzati
 - `utenti`: Visualizza la scena di login
 - `utente`: Visualizza la scena di benvenuto con il layout della applicazione invocata dal controller `LoginController`
 - `loadView`: Restituisce la view corretta
 - `renderView`: Visualizza una generica vista al centro del layout



ViewDispatcher (2)

```
public class ViewDispatcher {  
    private static ViewDispatcher instance = new ViewDispatcher();  
  
    private ViewDispatcher() {  
    }  
  
    public static ViewDispatcher getInstance() {  
        return instance;  
    }  
}
```

.....



ViewDispatcher (3)

```
private Stage stage;
```

```
public void utenti(Stage stage) throws ViewException {  
    }  
}
```




ViewDispatcher (4)

```
private BorderPane layout;  
  
public void utente() {  
    try {  
        layout = (BorderPane) loadView("layout");  
        Parent home = loadView("utente");  
        layout.setCenter(home);  
        Scene scene = new Scene(layout);  
        stage.setScene(scene);  
    } catch (ViewException e) {  
        e.printStackTrace();  
        renderError(e);  
    }  
}
```



ViewDispatcher (5)

```
public void renderView(String viewName) {  
    try {  
        Parent view = loadView(viewName);  
        layout.setCenter(view);  
    } catch (ViewException e) {  
        renderError(e);  
    }  
}
```



ViewDispatcher (6)

```
private static final String FXML_SUFFIX = ".fxml";
private static final String RESOURCE_BASE = "/viste/";
private void renderError(ViewException e) {
    e.printStackTrace();
    System.exit(1);
}

private Parent loadView(String view) throws ViewException {
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource(RESOURCE_BASE +
                                                                    view + FXML_SUFFIX));

        return loader.load();
    } catch (IOException e) {
        e.printStackTrace();
        throw new ViewException(e);
    }
}
```



ViewException

```
public class ViewException extends Exception {
    public ViewException() {
        super();
    }
    public ViewException(String message, Throwable cause, boolean enableSuppression,
        boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
    }
    public ViewException(String message, Throwable cause) {
        super(message, cause);
    }
    public ViewException(String message) {
        super(message);
    }
    public ViewException(Throwable cause) {
        super(cause);
    }
}
```



MyUnivaqApplication

```
public class MyUnivaqApplication extends Application {  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
    @Override  
    public void start(Stage stage) throws Exception {  
        try {  
            ViewDispatcher viewDispatcher =  
                ViewDispatcher.getInstance();  
            viewDispatcher.loginView(stage);  
        } catch (ViewException e) {  
            e.printStackTrace();  
        }  
    }  
}
```