



1



2

1

## Quando pensare alla GUI?

**Non subito!**

**La GUI ha bisogno di sapere:**

- Quali sono le operazioni svolte dall'utente
  - diagramma dei CASI D'USO
- Quali oggetti del dominio sono interessati da queste operazioni
  - diagramma delle CLASSI
- Qual è il workflow (la sequenza delle operazioni)
  - diagramma delle ATTIVITA'

**Quindi abbastanza avanti nel progetto**

- **NON** devo avere un diagramma delle classi GUI-based!

3

## Diagramma per UI?

**Non ne esiste uno specifico**

**Gli step sono:**

1. Progettazione del layout
2. **Navigation Map**
3. Gestione eventi: quando, in seguito a quali eventi e di chi si modifica il contenuto di una schermata
4. **Prevedere anche una validazione dell'input**

4

2

## Esempio di navigation map

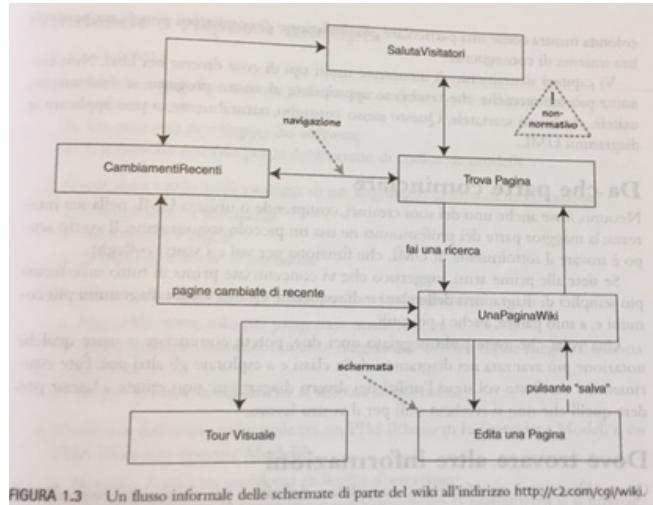


FIGURA 1.3 Un flusso informale delle schermate di parte del wiki all'indirizzo <http://c2.com/cgi/wiki>.

Fonte: UML Distilled, Fowler

5

## GUI IN JAVA

6

3

## GUI in Java – Storia (1)

### **AWT (Abstract Windowing Toolkit):**

- primo toolkit per interfacce grafiche in Java;
- basato sull'idea di *riutilizzo* dei toolkit esistenti;
- offre un'infrastruttura di *accesso a componenti*:
  - esistenti nel sistema operativo ospitante;
  - quindi solo un subset comune ai SO considerati (massimo subset comune)
- definito attorno al concetto di peer:
  - ogni *componente* è un wrapper per un componente del sistema grafico ospitante;

### **Vantaggi:**

- molto efficiente

### **Svantaggi:**

- autolimitato dal ridotto subset di componenti esterne

7

## GUI in Java – Storia (2)

### **Problemi con AWT?**

- componenti limitate (subset comune);
- impossibile realizzare componenti custom;
- chiamate indirette al SO tramite wrapper.

### **Soluzione di Swing:**

- costruire le proprie componenti da zero:
  - all'interno di un `java.awt.Component`;
  - le componenti si disegnano da sé;
  - Swing gestisce eventi di livello più basso;
- vantaggi di questa piccola evoluzione:
  - set ampio di componenti preconfezionate;
  - possibile creare componenti custom.

8

## GUI in Java – Storia (3)

### Java Swing:

- Parte delle Oracle's Java Foundation Classes (JFC) — API per graphical user interface (GUI) di programmi Java
- Offre un aspetto “look and feel” simile a quello di altre piattaforme
- I componenti sono scritti in Java, quindi sono platform-independent

9

## GUI in Java – Storia (4)

### JavaFX

- 2008, dalla versione Java 8, **JavaFX**: uno Swing riscritto, una nuova architettura con dentro tutto quello che serviva, alla pari con la concorrenza
- Punta molto sulla separazione dei concern: GUI vs application logic
- Da dicembre 2019 uno spin-off di Java, va scaricata la libreria a parte
  - Integrazione migliore con NetBeans

10

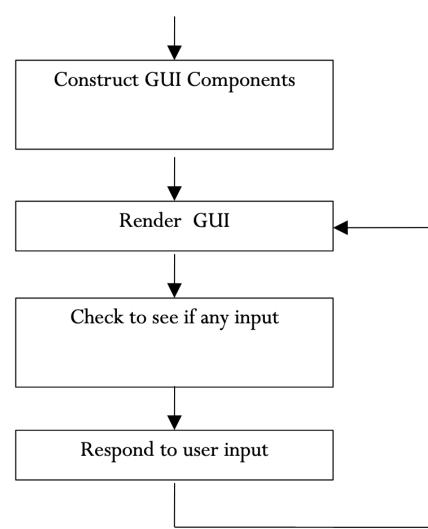
## JavaFX (ma anche Java Swing)

### Componenti principali per funzionamento GUI:

- **Elementi UI:** elementi di base dell'interfaccia che l'utente vede e con i quali interagisce
- **Layout:** elementi che definiscono come gli elementi dell'interfaccia vanno organizzati
- **Behavior:** eventi che accadono quando l'utente interagisce con gli elementi dell'interfaccia e che il programma deve essere in grado di gestire (gestisce l'interazione)

11

## Ciclo di vita di una GUI



12

## JavaFX (1)

**• *Stage*:** la finestra (top-level container)

**• *Scene*:** ogni stage ha (almeno) una scena

- Contiene tutti gli elementi/nodi

**• *Node*:** è un componente grafico, come una forma (*shape*), un'immagine, un control o un pane

13

## JavaFX (2) - Esempio

```

1 package it.unimi.chiara;
2
3
4 import javafx.application.Application;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Label;
7 import javafx.scene.layout.StackPane;
8 import javafx.stage.Stage;
9
10 public class HelloFX extends Application {
11
12     @Override
13     public void start(Stage primaryStage) {
14         String javaVersion = System.getProperty("java.version");
15         String javafxVersion = System.getProperty("javafx.version");
16         Label l = new Label("Hello, JavaFX " + javafxVersion + ",\nrunning on Java " + javaVersion + ".");
17
18         Scene scene = new Scene(new StackPane(l), 400, 300);
19         primaryStage.setScene(scene);
20         primaryStage.show();
21     }
22
23     public static void main(String[] args) {
24         launch();
25     }
26
27 }

```

14

## JavaFX (2) - Esempio

Ogni programma JavaFX è definito in una classe che estende la classe `javafx.application.Application`

```
HelloFX.java 3 Main.java
1 package it.unimi.chiara;
2
3
4 import javafx.application.Application;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Label;
7 import javafx.scene.layout.StackPane;
8 import javafx.stage.Stage;
9
10 public class HelloFX extends Application {
11
12     @Override
13     public void start(Stage primaryStage) {
14         String javaVersion = System.getProperty("java.version");
15         String javafxVersion = System.getProperty("javafx.version");
16         Label l = new Label("Hello, JavaFX " + javafxVersion + ",\nrunning on Java " + javaVersion + ".");
17
18         Scene scene = new Scene(new StackPane(l), 400, 300);
19         primaryStage.setScene(scene);
20         primaryStage.show();
21     }
22
23     public static void main(String[] args) {
24         launch();
25     }
26 }
27 }
```

15

## JavaFX (2) - Esempio

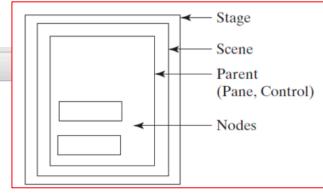
La classe che estende la classe `javafx.application.Application` fa l'override del metodo `start` e viene richiamata dal `main` tramite il metodo `launch`

```
HelloFX.java 3 Main.java
1 package it.unimi.chiara;
2
3
4 import javafx.application.Application;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Label;
7 import javafx.scene.layout.StackPane;
8 import javafx.stage.Stage;
9
10 public class HelloFX extends Application {
11
12     @Override
13     public void start(Stage primaryStage) {
14         String javaVersion = System.getProperty("java.version");
15         String javafxVersion = System.getProperty("javafx.version");
16         Label l = new Label("Hello, JavaFX " + javafxVersion + ",\nrunning on Java " + javaVersion + ".");
17
18         Scene scene = new Scene(new StackPane(l), 400, 300);
19         primaryStage.setScene(scene);
20         primaryStage.show();
21     }
22
23     public static void main(String[] args) {
24         launch();
25     }
26 }
27 }
```

16

## JavaFX (2) - Esempio

```
HelloFX.java  Main.java
1 package it.unimi.chiara;
2
3
4@ import javafx.application.Application;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Label;
7 import javafx.scene.layout.StackPane;
8 import javafx.stage.Stage;
9
10 public class HelloFX extends Application {
11
12@     @Override
13     public void start(Stage primaryStage) {
14         String javaVersion = System.getProperty("java.version");
15         String javafxVersion = System.getProperty("javafx.version");
16         Label l = new Label("Hello, JavaFX " + javafxVersion + ",\nrunning on Java " + javaVersion + ".");
17
18         Scene scene = new Scene(new StackPane(l), 400, 300);
19         primaryStage.setScene(scene);
20         primaryStage.show();
21     }
22
23@     public static void main(String[] args) {
24         launch();
25     }
26
27 }
```



17

## JavaFX (2) - Esempio

```
HelloFX.java  Main.java
1 package it.unimi.chiara;
2
3
4@ import javafx.application.Application;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Label;
7 import javafx.scene.layout.StackPane;
8 import javafx.stage.Stage;
9
10 public class HelloFX extends Application {
11
12@     @Override
13     public void start(Stage primaryStage) {
14         String javaVersion = System.getProperty("java.version");
15         String javafxVersion = System.getProperty("javafx.version");
16         Label l = new Label("Hello, JavaFX " + javafxVersion + ",\nrunning on Java " + javaVersion + ".");
17
18         Scene scene = new Scene(new StackPane(l), 400, 300);
19         primaryStage.setScene(scene);
20         primaryStage.show();
21     }
22
23@     public static void main(String[] args) {
24         launch();
25     }
26
27 }
```

**javafx.stage.Stage**  
**public final void setScene(Scene value)**  
 Specify the scene to be used on this stage.  
  
**Stage()**  
 Creates a new instance of decorated Stage.  
  
**public final void show()**  
 show this Window by setting visibility to true

**javafx.scene.Scene**  
**Scene(Parent root, double width, double height)**  
 Creates a Scene for a specific root Node with a specific size.

18

## JavaFX (2) - Esempio

```

HelloFX.java  Main.java
1 package it.unimi.chiara;
2
3
4 import javafx.application.Application;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Label;
7 import javafx.scene.layout.StackPane;
8 import javafx.stage.Stage;
9
10 public class HelloFX extends Application {
11
12     @Override
13     public void start(Stage primaryStage) {
14         String javaVersion = System.getProperty("java.version");
15         String javafxVersion = System.getProperty("javafx.version");
16         Label l = new Label("Hello, JavaFX " + javafxVersion + ",\nrunning on Java " + javaVersion + ".");
17
18         Scene scene = new Scene(new StackPane(l), 400, 300);
19         primaryStage.setScene(scene);
20         primaryStage.show();
21     }
22
23     public static void main(String[] args) {
24         launch();
25     }
26
27 }

```

**javafx.stage.Stage**  
**public final void setScene(Scene value)**  
 Specify the scene to be used on this stage.

**Stage()**  
 Creates a new instance of decorated Stage.

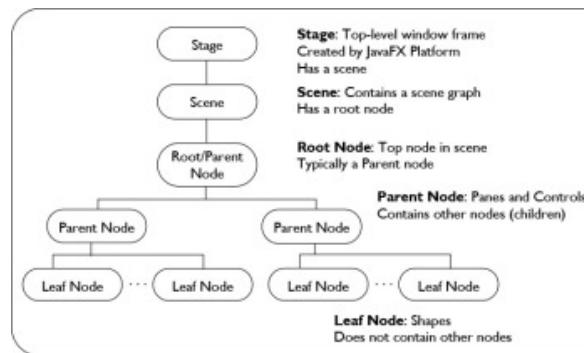
**public final void show()**  
 show this Window by setting visibility to true

**javafx.scene.Scene**  
**Scene(Parent root, double width, double height)**  
 Creates a Scene for a specific root Node with a specific size.

19

## JavaFX (3) – Organizzazione standard di uno stage

### Node:

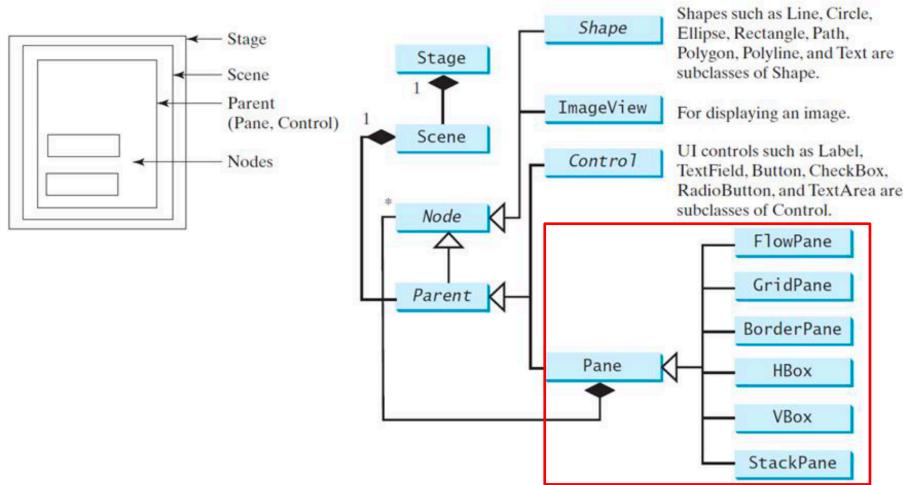


- Label
- Button
- Checkbox
- Radiobutton
- Textfield
- Textarea
- ...

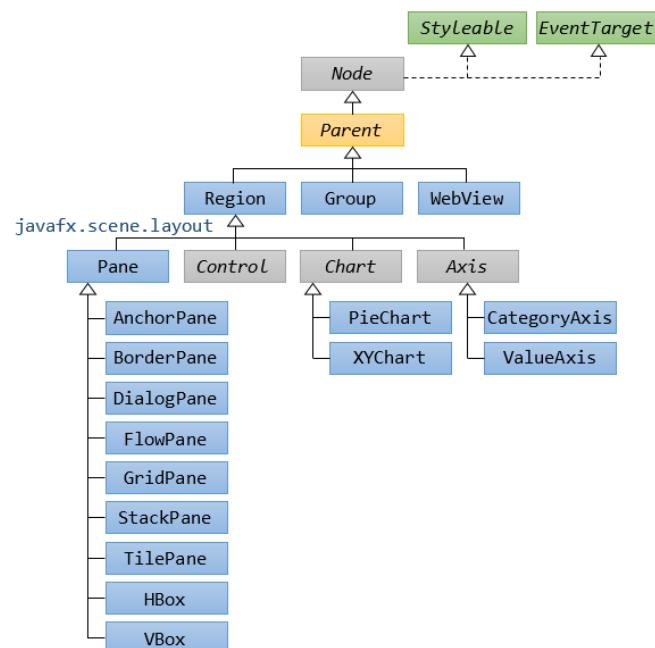
Come organizzarli graficamente??

20

### JavaFX (3) – Struttura standard di uno stage



21



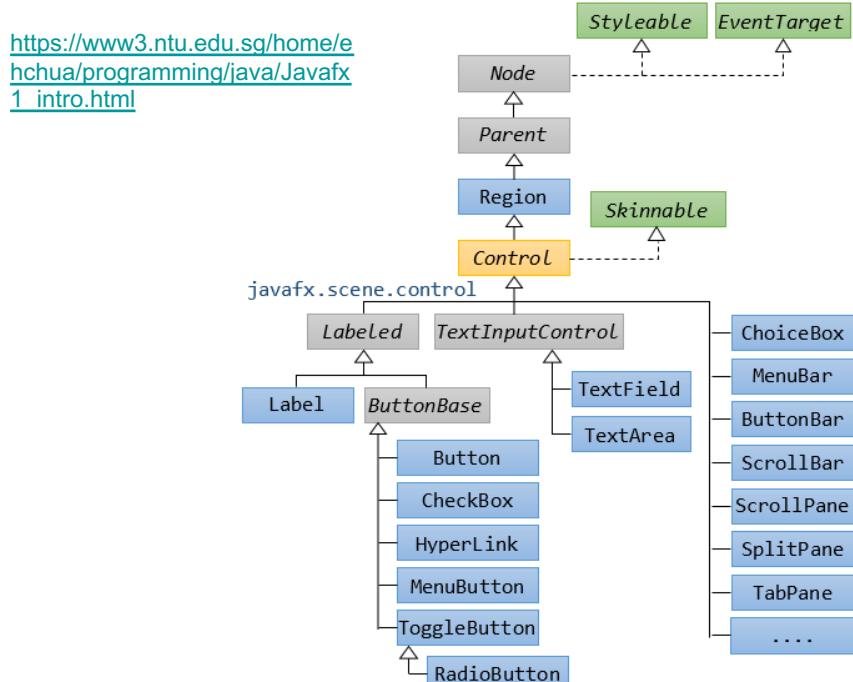
22

## JavaFX (3) - Layout Pane

### Pane per organizzare i nodi in un container

Class	Description
<b>Pane</b>	Base class for layout panes. It contains the <code>getChildren()</code> method for returning a list of nodes in the pane.
<b>StackPane</b>	Places the nodes on top of each other in the center of the pane.
<b>FlowPane</b>	Places the nodes row-by-row horizontally or column-by-column vertically.
<b>GridPane</b>	Places the nodes in the cells in a two-dimensional grid.
<b>BorderPane</b>	Places the nodes in the top, right, bottom, left, and center regions.
<b>HBox</b>	Places the nodes in a single row.
<b>VBox</b>	Places the nodes in a single column.

23



24

## JavaFX (4) – Altro esempio

### 2 novità:

1. Layout più complesso
2. Gestione degli eventi

**Codice leggibile?**

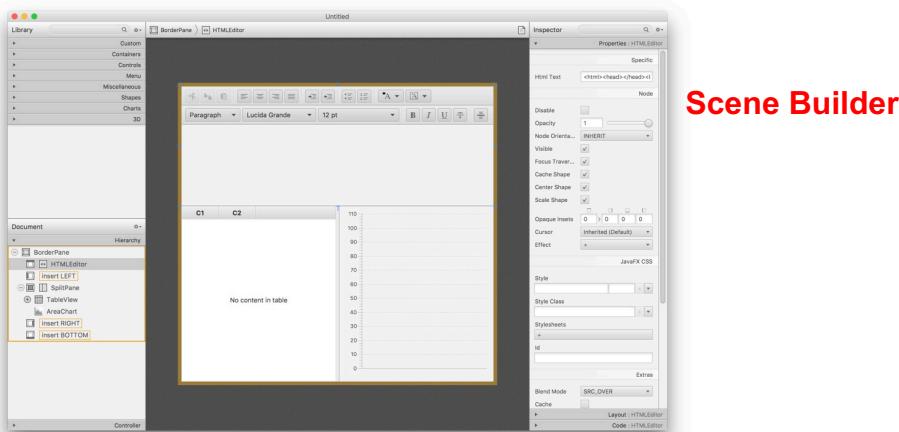
**Codice facilmente scrivibile?**

- Grafica + Application logic + Interazione tutti insieme, molto complicato!!!

25

## JavaFX (5) – Soluzione a complessità GUI

1. Load della grafica da un file XML esterno
2. Il file XML esterno non lo devo "scrivere" io!!



26

13

## JavaFX (6) – SceneBuilder in short

- Interfaccia Drag & Drop
- Separazione dei file che gestiscono la grafica vs quelli che gestiscono la logica dell'applicazione
- Free e open source
- <https://gluonhq.com/products/scene-builder/>
- Esempio di uso
  - HelloWorldFXML
  - Finestra più complessa: la vediamo e la facciamo insieme

27

## JavaFX (7) – Gestione Eventi

- Fino ad ora ci siamo occupati di: **Vista**
- Ora vogliamo gestire il **comportamento**
  - l'applicazione può modificare la grafica o gli elementi testuali presenti
  - l'utente deve poter interagire con alcuni componenti grafici (i control)
- Come fare?
  - Solo tramite la scrittura del codice
  - **SceneBuilder e codice**

28

14

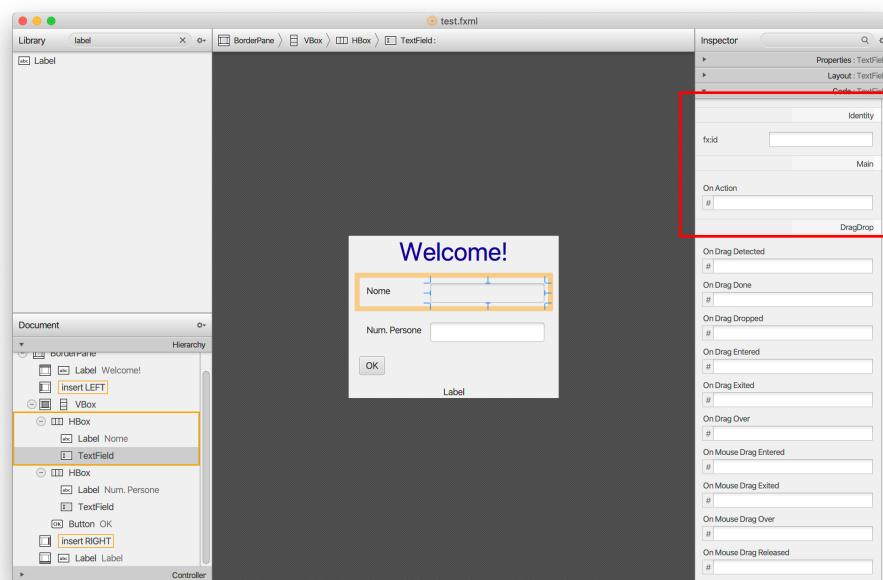
## JavaFX (7) – Gestione Eventi

- Devo specificare:

1. Le parti della GUI che voglio gestire
  - tutti generano eventi, ma a me ne interessano solo alcuni
  - quelli che mi interessano devono essere riferibili dal codice
2. Quali eventi voglio gestire
  - Mouse over? Mouse click? Inserimento testo?
3. Chi deve gestire gli eventi e come
  - Quale classe controller e con quali metodi

29

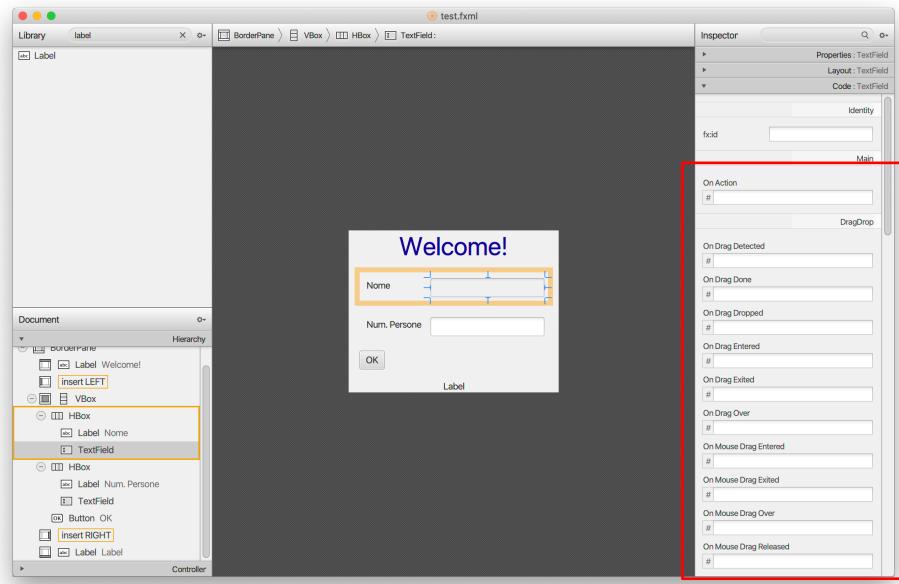
## JavaFX (7) – Gestione Eventi



30

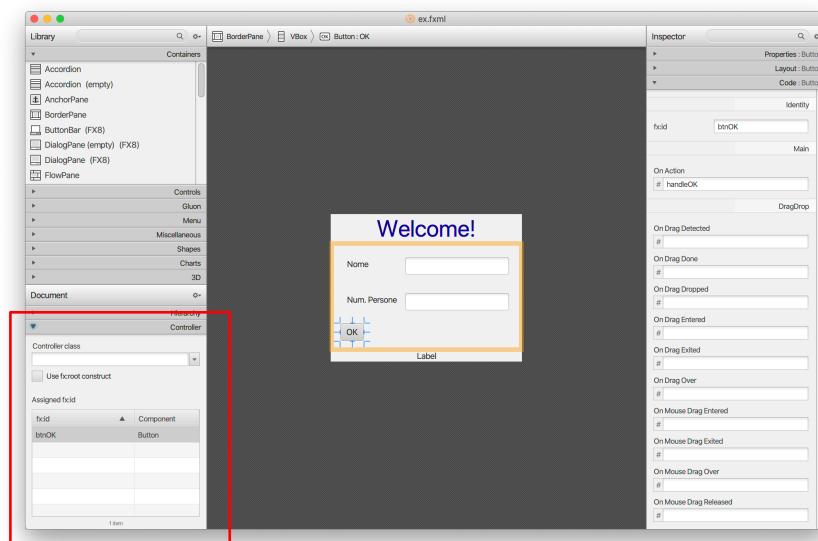
15

## JavaFX (7) – Gestione Eventi



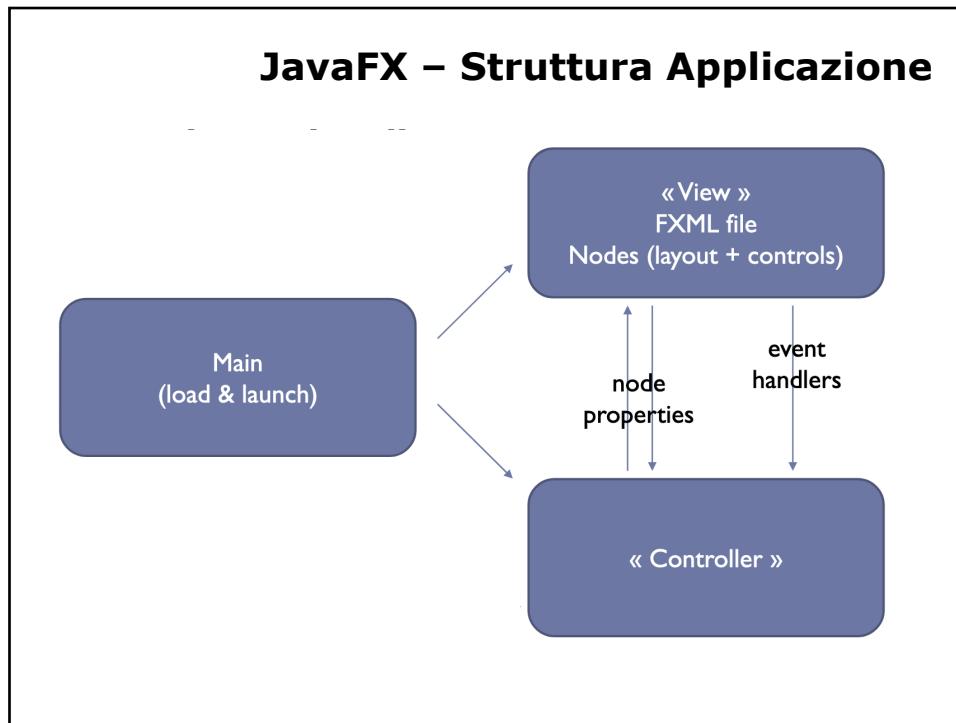
31

## JavaFX (7) – Gestione Eventi

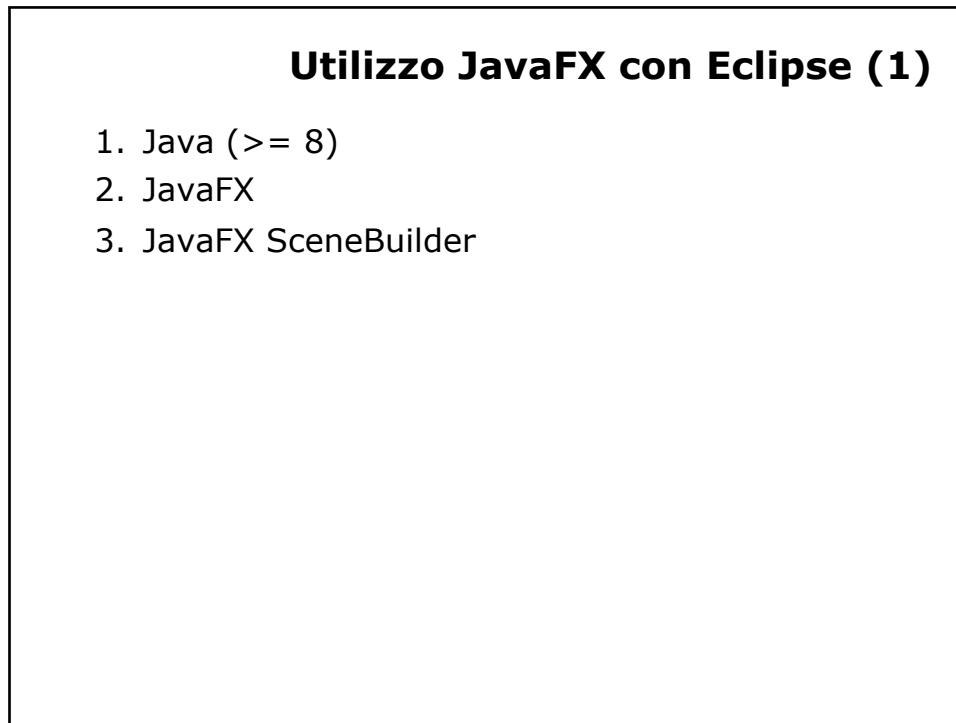


32

16



33



34

## Utilizzo JavaFX con Eclipse (3)

### Scaricare VM

- [https://unimi2013-my.sharepoint.com/:u/g/personal/mario\\_illi\\_unimi\\_it/EOcqRtdieCVCj6LxaDnzOr8BZ5NQciemBpLB0oV8ElmRUw?e=NGzcvC](https://unimi2013-my.sharepoint.com/:u/g/personal/mario_illi_unimi_it/EOcqRtdieCVCj6LxaDnzOr8BZ5NQciemBpLB0oV8ElmRUw?e=NGzcvC)

35

## Documentazione - Download

### JavaFX

- <https://openjfx.io/>
- <https://gluonhq.com/products/javafx/>
  - For more information on JavaFX with Java SE 8, please refer to the [JavaFX Documentation](#).
  - For JDK 11 and later releases, Oracle has open sourced JavaFX. You can find more information at [OpenJFX project](#).

### SceneBuilder

- <https://www.oracle.com/java/technologies/javase/javafxscenebuilder-info.html> (old)
- <https://gluonhq.com/products/scene-builder/> (new)

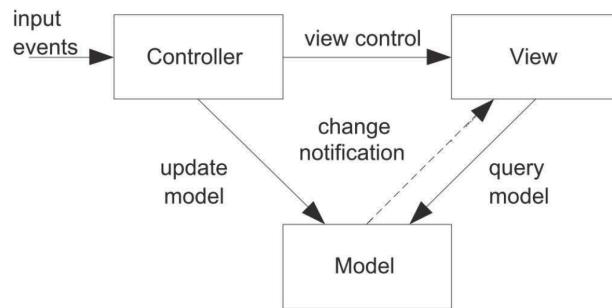
36

18

## Design Pattern MVC (1)

### Model-View-Controller (MVC) pattern

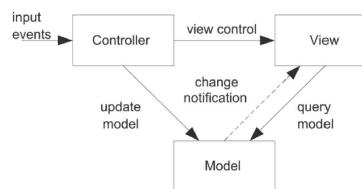
- Il pattern architettonale che aiuta a separare l'interfaccia utente (layer) dalla parte funzionale del sistema
  - Caso speciale di multi-layer



37

## Design Pattern MVC (2)

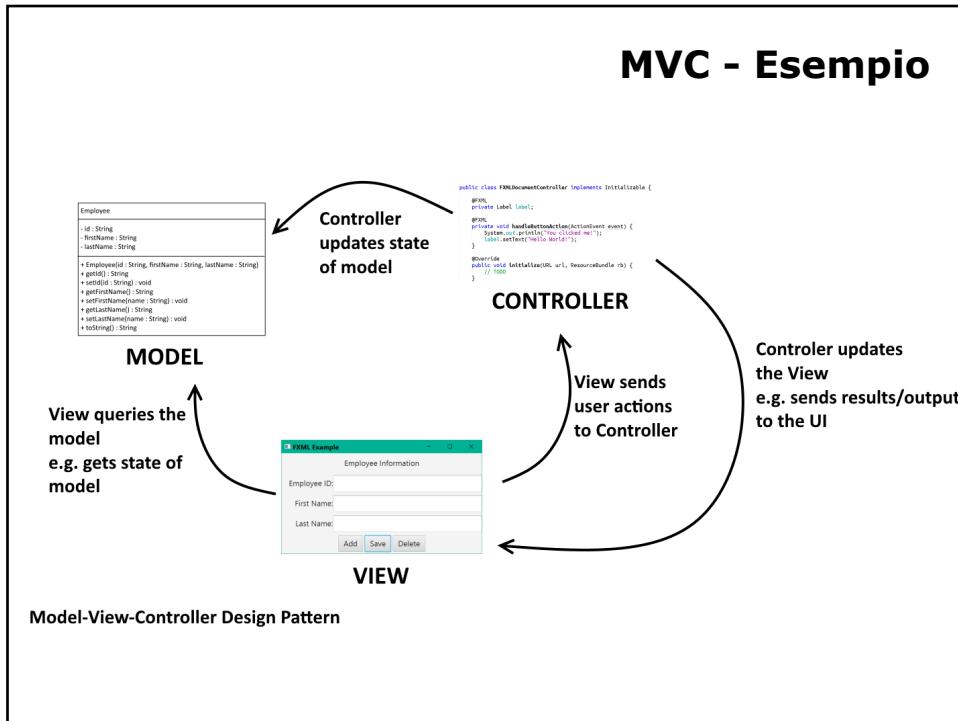
### Model-View-Controller (MVC) pattern



- Model** contiene le classi le cui istanze devono essere visualizzate e manipolate (layer funzionale)
- View** contiene oggetti utilizzati per visualizzare i dati del model all'utente tramite interfaccia
- Controller** contiene gli oggetti che gestiscono l'input dell'utente (a volte non distinta da view)

38

19



39