

Alma Mater Studiorum-Università di Bologna

Scuola di Ingegneria

Le basi della programmazione grafica: il framework JavaFX

Corso di Laurea in Ingegneria Informatica

Anno accademico 2021/2022

Prof. ENRICO DENTI

Dipartimento di Informatica – Scienza e Ingegneria (DISI)



PROGRAMMAZIONE GRAFICA

- Sviluppare una *applicazione grafica* è molto diverso da sviluppare un'applicazione tradizionale
 - niente più Read-Eval-Print Loop (REPL)
- L'interazione non avviene più in momenti prefissati, ma *quando l'utente agisce sui controlli grafici*
 - non è il programma che ha il controllo (printf, scanf), ma l'utente
- Si programma *stabilendo come reagire agli eventi* che si verificano quando l'utente agisce sui controlli grafici
 - ciò che accade in un programma grafico è descrivibile *solo a posteriori*, come *interaction history*: non è predicibile
 - .. ed è molto difficile progettare ed effettuare testing e collaudi



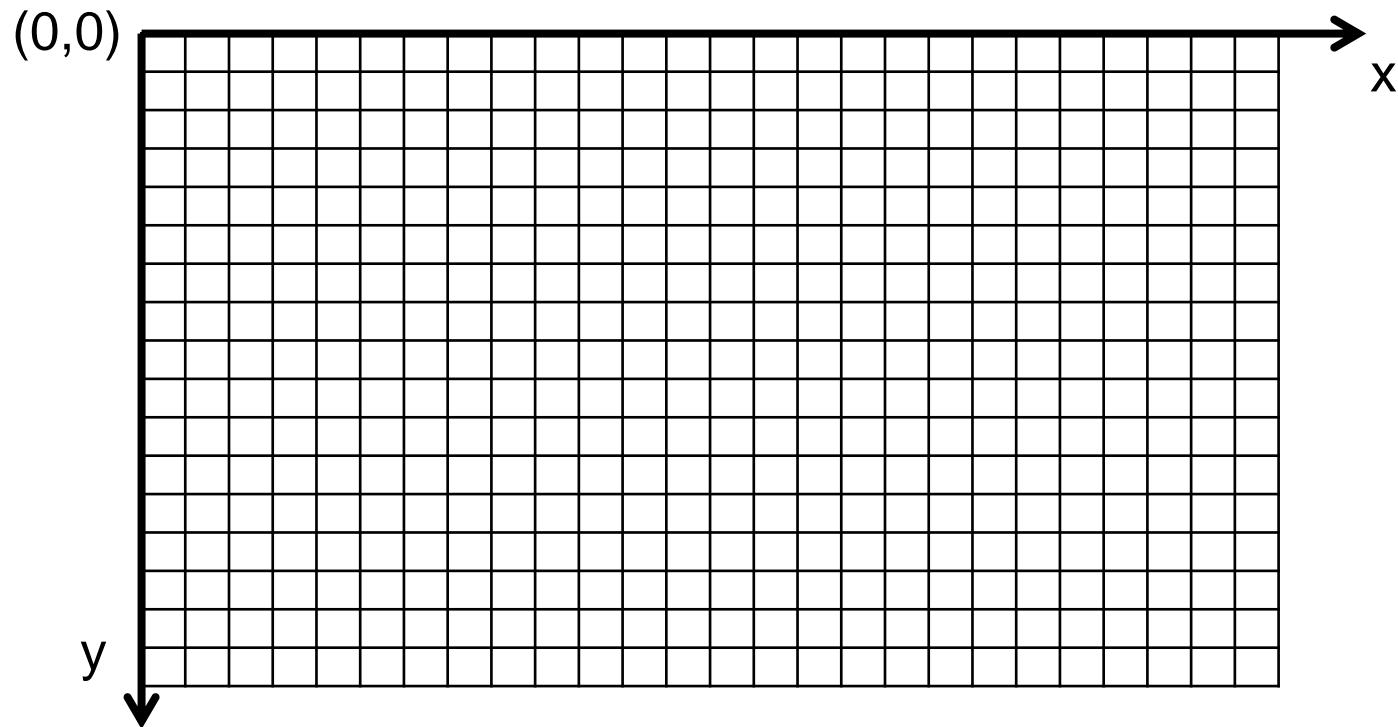
PROGRAMMAZIONE GRAFICA

In una finestra grafica si può operare a due livelli:

- a livello di **singoli pixel**
 - accendendo/spegnendo singoli pixel colorati
 - non c'è consapevolezza di cosa si stia disegnando: la finestra rimane solo una "tela con puntini colorati sopra"
- a livello di **controlli grafici evoluti**
 - collocando in essa *controlli già pronti* secondo le esigenze (bottoni, slider, liste, campi ed aree di testo.. e molto altro)
 - c'è piena consapevolezza dei componenti collocati nella finestra: la finestra in sé funge solo da supporto fisico
 - per farlo, occorre *aggiungere i componenti a un opportuno contenitore* che si relazioni col *modello della finestra*
→ il flusso di controllo è gestito dal framework

LIVELLO PIXEL

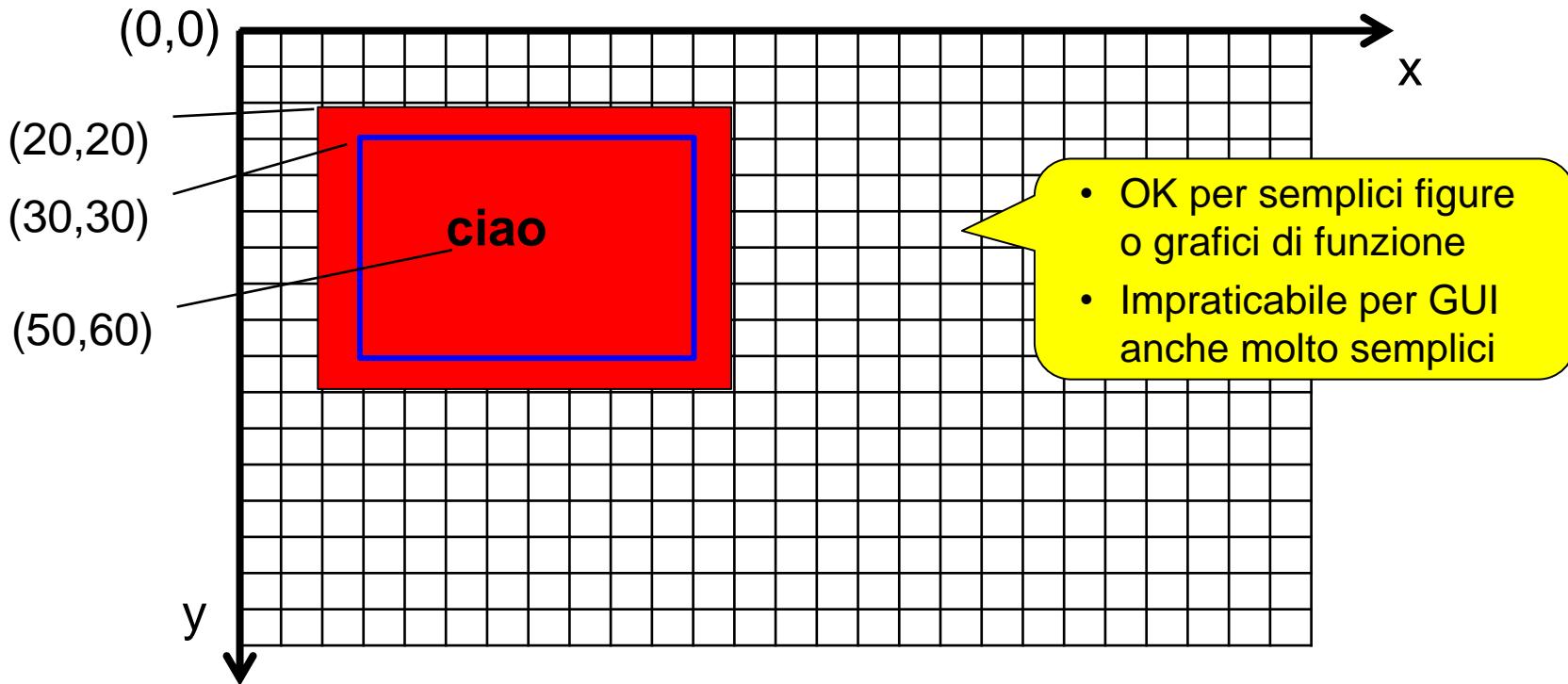
- La finestra è vista come un **piano cartesiano** con *origine in alto a sinistra* e asse verticale orientato verso *il basso*
- Si possono **accendere/spegnere/colorare singoli pixel**, formando così figure e parole



LIVELLO PIXEL – ESEMPI

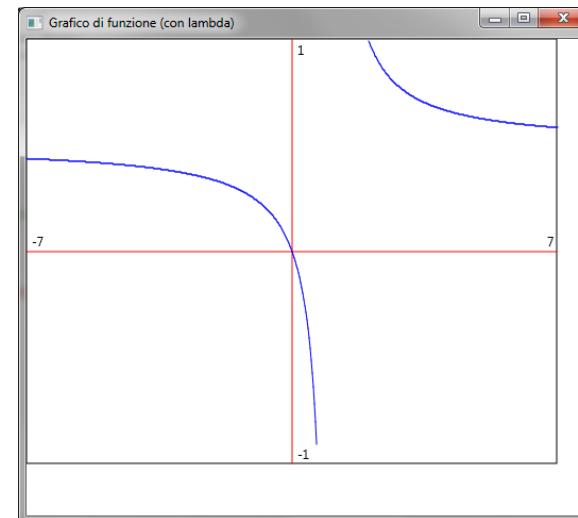
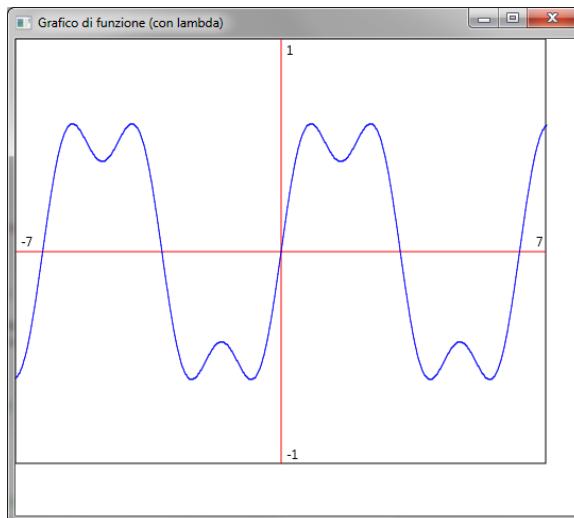
L'esempio [griglia ogni 10 pixel per comodità] mostra:

- un rettangolo rosso pieno, origine a (20,20), dimensioni 100x80
- un rettangolo blu interno, origine a (30,30), dimensioni 80x60
- una scritta "ciao" nera, con origine [angolo basso sinistro] a (50,60)



ESEMPIO: GRAFICO DI FUNZIONE

- Linee orizzontali rosse per assi, scritte nere per estremi x,y
- Pixel blu calcolati uno ad uno secondo la funzione desiderata
- Punti critici:
 - asse y cresce verso il basso → trasformazione di coordinate
 - i valori $x, f(x)$ sono reali, le coordinate dei pixel sono intere
 - ..e se la funzione non esiste in un punto o è discontinua..?





LIVELLO COMPONENTI GRAFICI

- La finestra è vista come **contenitore** su cui **appendere componenti** (*controlli o widget*), come «*post it*»
 - di norma, *non si stabilisce* dove debbano stare tramite *coordinate absolute*, perché nei vari device fisici le finestre hanno dimensioni diverse, si ridimensionano, possono ruotare, etc.
 - si dota piuttosto ogni finestra di un **gestore di layout**, che colloca i *widget* secondo *politiche prestabilite* → *adattamento automatico*
- Ogni **framework grafico** fornisce *decine di widget già pronti*, con comportamento base già predisposto
- Il progettista della GUI deve definire:
 - la **parte strutturale** (quali e quanti *widget*, disposti come)
 - la **parte comportamentale** (come si reagisce agli *eventi*)



ESEMPI BASE

Esempio 9...

Rosso Azzurro

Esempio 9...

Rosso Azzurro

Esempio 10

Scrivere qui il nuovo messaggio

Aggiorna

Esempio 10

Nel mezzo del cammin di nostra vita

Nel mezzo del cammin di nostra vita

Aggiorna

Esempio 20

Rosso Giallo

Add Remove

Esempio 21 bis

0 1 2 3 4 5 6 7 8 9 10

Valore corrente: 5,5 (era: 5,35)

Esempio 25ter - TableView editabile con aggiornamento in tempo reale

Città	Temperatura
Bologna	3.3
Milano	-1.0
Torino	-1.4
Roma	7.5

Riga 1, Milano -1.0 °C

Esempio 10 quater

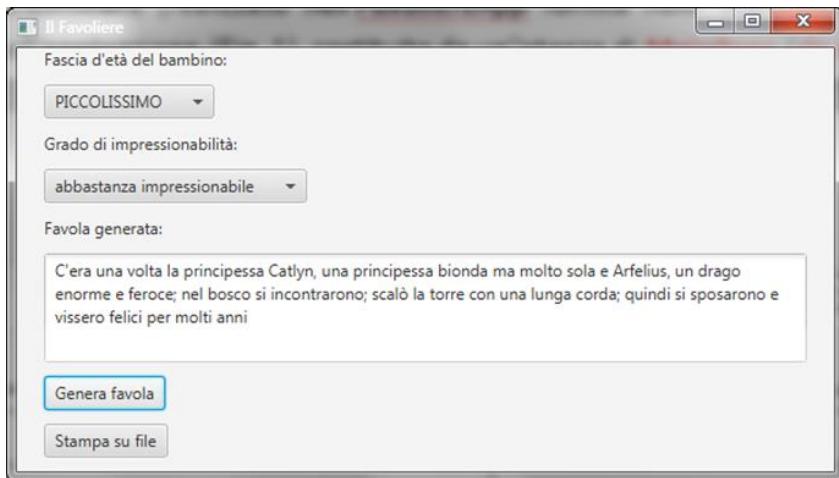
Nel mezzo del cammin di nostra vita,
mi ritrovai per una selva oscura

Caratteri: 69

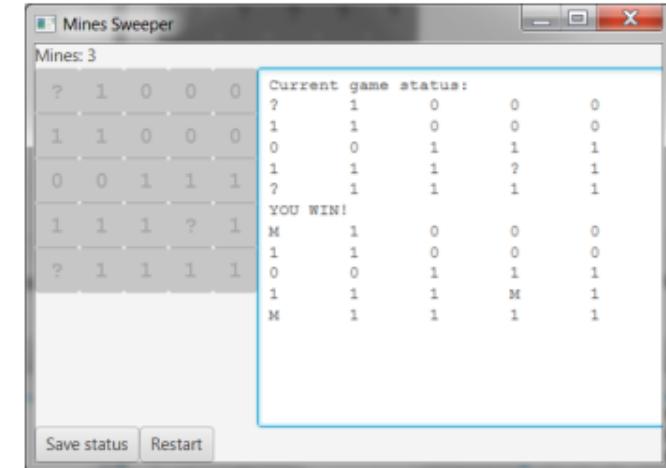


ESEMPI DA COMPITI D'ESAME (2019)

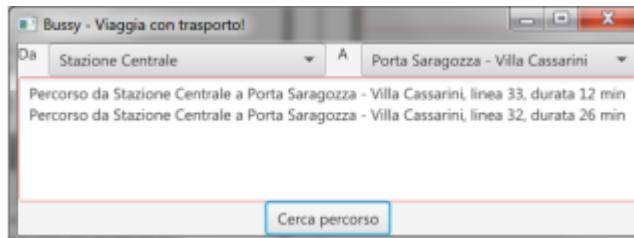
Favoliere



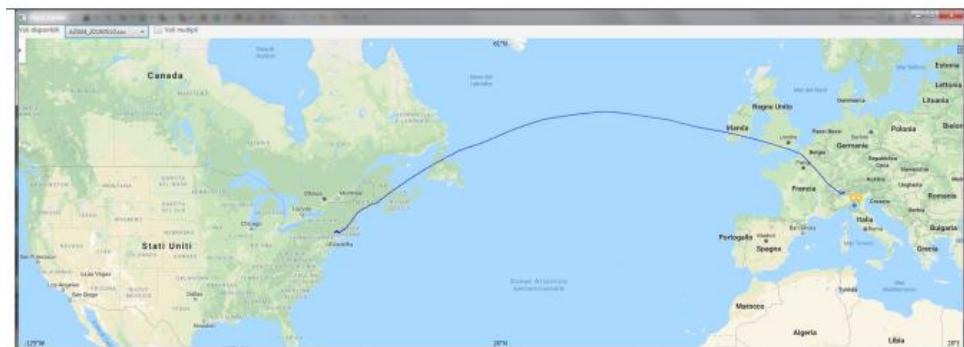
Campo minato



Percorsi bus urbani



Tracciamento voli



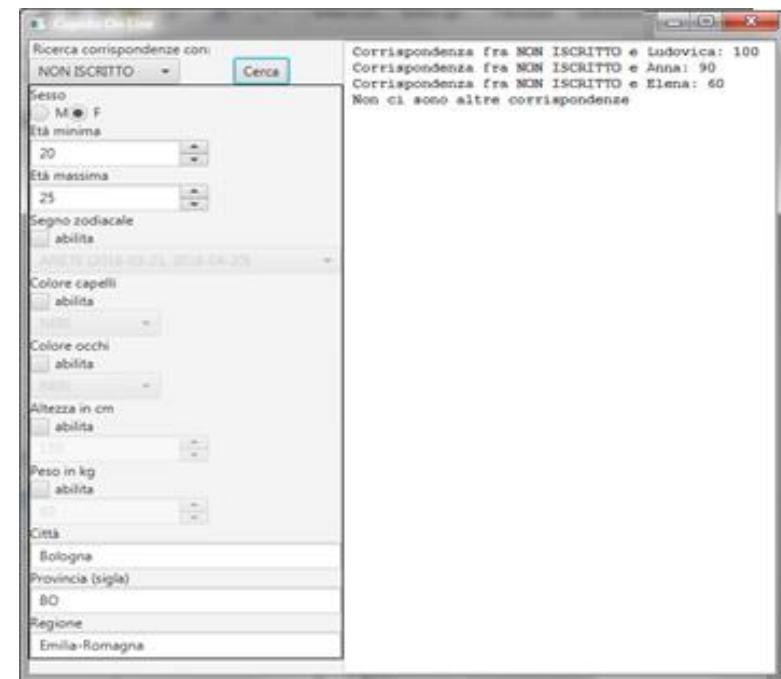


ESEMPI DA COMPITI D'ESAME (2018)

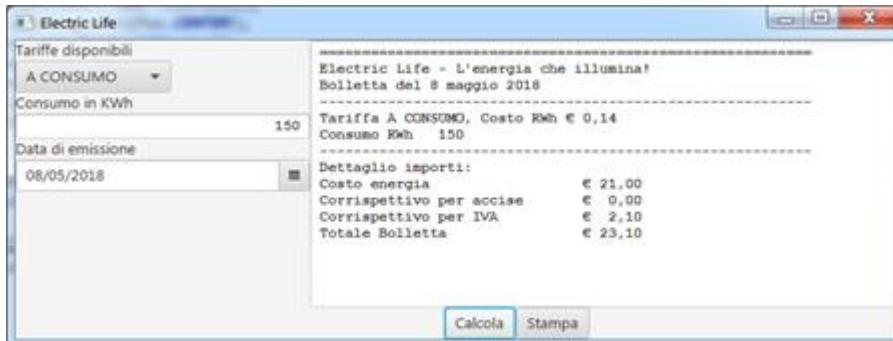
Movimento treni (plastico ferroviario) – (2019)



Anima gemella



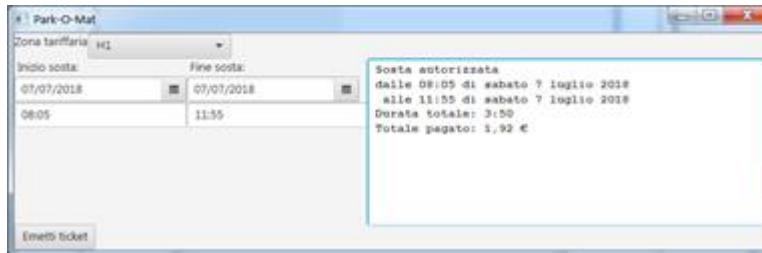
Bolletta elettrica





ESEMPI DA COMPITI D'ESAME (2018)

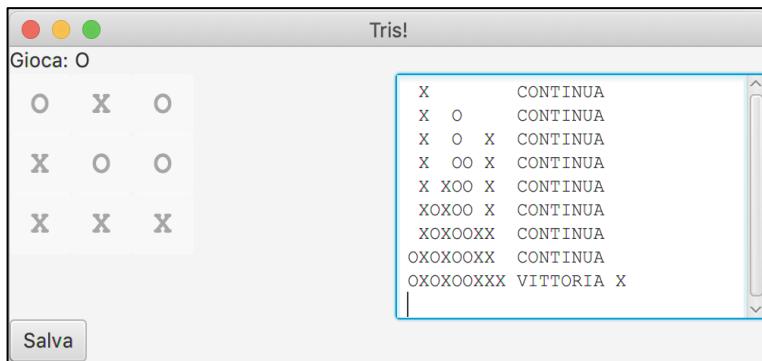
Park-O-Mat



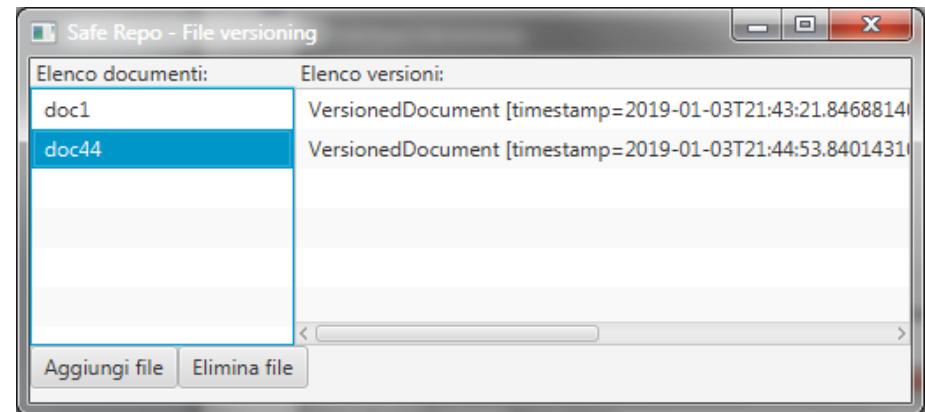
BikeRent



Tris

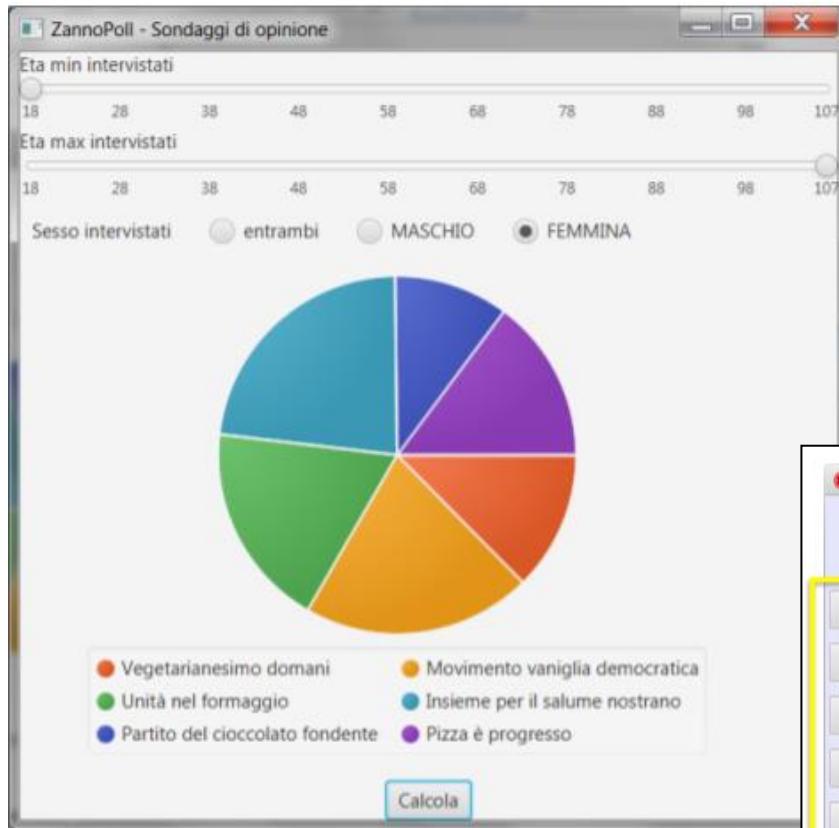


SafeRepo



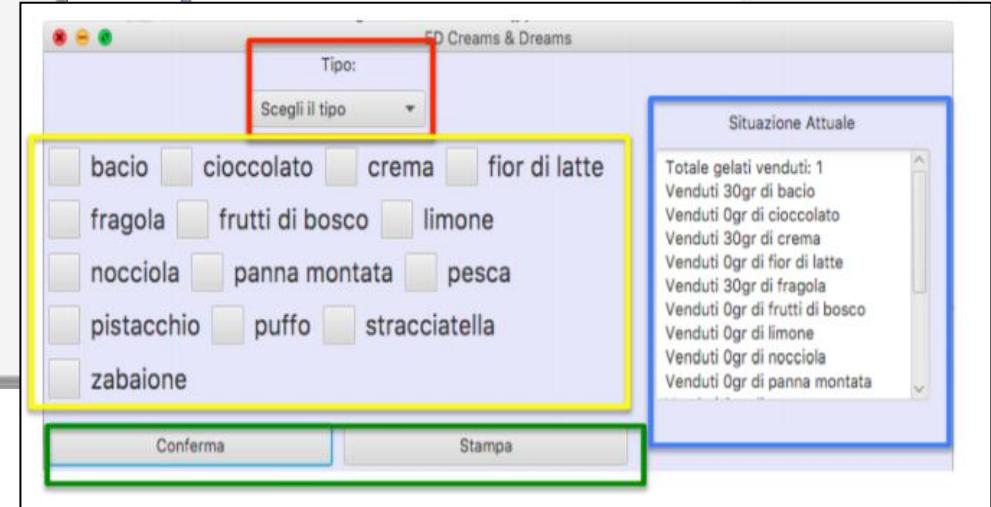
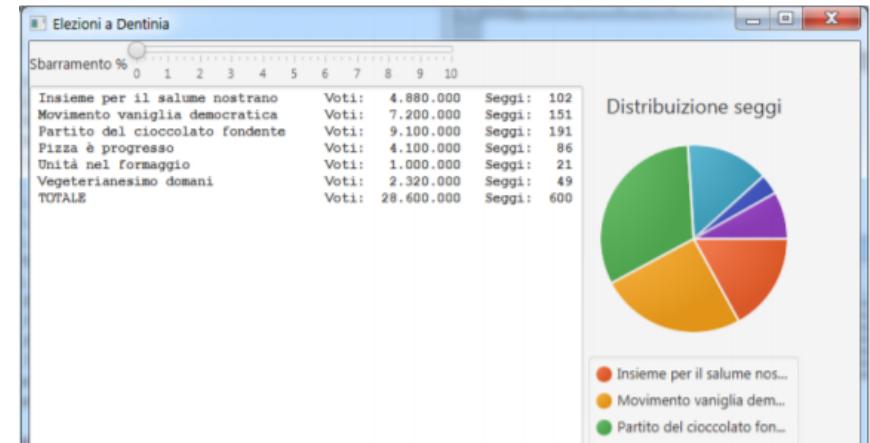
ESEMPI DA COMPITI D'ESAME (2017)

Sondaggi



Gelati, che passione!

Elezioni





ESEMPI DA COMPITI D'ESAME (2017)

Fantacalcio

The screenshot shows the Zanno Fantasy Football application interface. At the top, there is a dropdown menu labeled "Modulo:" with "4-4-2" selected. Below this, the interface is divided into four main sections: Portieri (Goalkeepers), Difensori (Defenders), Centrocampisti (Midfielders), and Attaccanti (Forwards). Each section has a list of player names and a "In campo" button. A green border highlights the Portieri and Difensori sections. A red box highlights the "Modulo:" dropdown. A blue box highlights the results summary on the right.

Zanno Fantasy Football

Modulo:
4-4-2

Portieri
Mirante Antonio
Sarr Mouhamadou Falk

Difensori
M'baya Ibrahima
Masina Adam
Oikonomou Marios
Torousidis Vasilis

Centrocampisti
Pulgar Erick
Rizzo Luca
Taider Saphir
Viviani Federico

Attaccanti
Donsah Godfred
Dzemalii Blerim
Krejci Ladislav
Nagy Adam

In campo

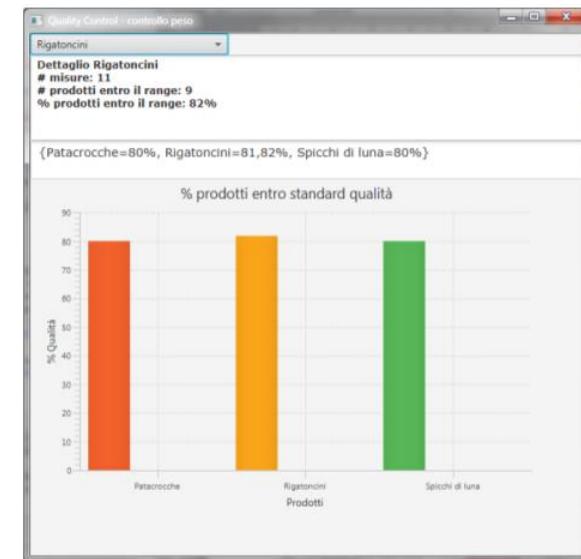
In campo

In campo

In campo

Conferma Formazione
Carica Risultati
Risultato:
Hai totalizzato 87.5 punti che corrispondono a 4 goal

Quality Assurance



Prelievo contanti

The screenshot shows the "Unione Banche di Zannonia" application interface for withdrawing cash. It features two rows of buttons for selecting banknotes and coins. The top row is for "Tagli banconote e monete" and the bottom row is for "Prelievo". The "Importo da prelevare" field contains the value "307".

Unione Banche di Zannonia

Tagli banconote e monete

€ 500 € 200 € 100 € 50 € 20 € 10 € 5 € 2 € 1

Quantità disponibili in cassa

5 25 30 200 200 100 100 50 50

Importo da prelevare Preleva

Tagli banconote e monete

€ 500 € 200 € 100 € 50 € 20 € 10 € 5 € 2 € 1

Prelievo



ESEMPI DA COMPITI D'ESAME (2016)

Diario fitness (2017)

My Fitness Diary

Data: 05/05/2017 Durata: 60

Attività: Allenamento a circuito Intensità: HIGH

Inserisci

Stampa report settimanale Selezione Settimana: 05/05/2017

Stampa

Alienamento di venerdì 05/05/17
Acquagym minuti: 15 calorie bruciate: 60
Allenamento a circuito minuti: 60 calorie bruciate: 480
Minuti Totali Alienamento: 75 Calorie Totali Bruciate: 540

Calcolo imposte

Reali Imposte d...

lavoratori dipendenti	€ 9.000,00
Reddito lordo	€ 40.000,00
Reddito imponibile	€ 31.000,00
Spese mediche	€ 1.500,00
Imposta linda	€ 5.040,00
Imposta netta	€ 4.540,00

Traghetti e porti

Da: QUALSIASI A: Cagliari

Tratte servite: da Livorno a Cagliari
da Civitavecchia a Cagliari
da Napoli a Cagliari
da Palermo a Cagliari

Servizi disponibili:

Dente Appuntito da Livorno a Olbia 07:30-12:30
Zanna Affilata da Livorno a Olbia 09:30-13:30
Canoa a remi da Livorno a Olbia 06:30-19:30

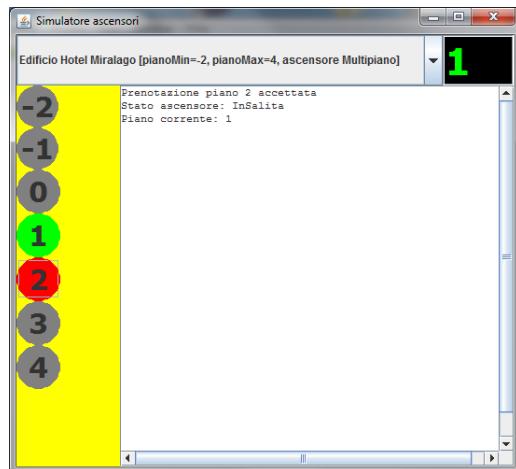
Da: QUALSIASI A: Olbia

Tratte servite: da Livorno a Olbia



ESEMPI DA COMPITI D'ESAME (2015)

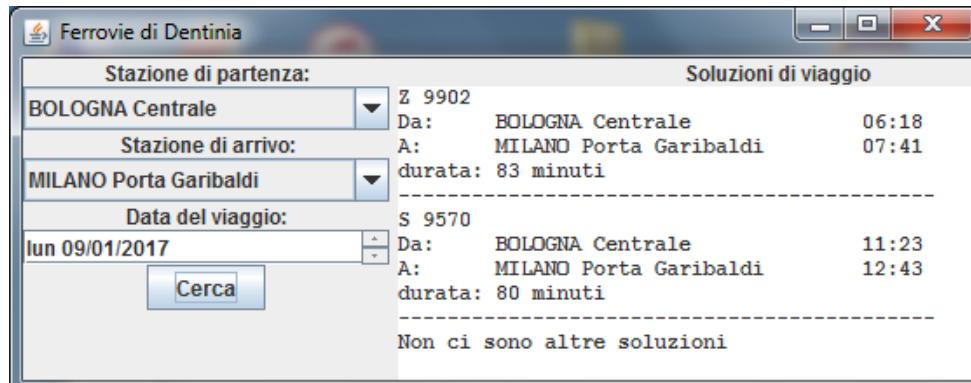
Ascensore



Finanziamento bancari



Ferrovie



Palinsesto TV





ESEMPI DA COMPITI D'ESAME (2015)

Cambiavalute

The screenshot shows a Windows application window titled "Cambiavalute". It has a toolbar with a currency icon, a dropdown menu for "Valuta" set to "USD", and radio buttons for "Acquisto" (Purchase) and "Vendita" (Sale). Below this, there are input fields for "Importo" (Amount) set to "100,00" and "Cambio ufficiale" (Official exchange rate) set to "108,87". A yellow box contains the following text:
Richiesta di compravendita di Dollaro USA
EDbank: 100,00 EUR = 105,96 USD
ZannoChange: 100,00 EUR = 107,20 USD

Dichiarazione dei redditi

The screenshot shows a Windows application window titled "Dentinia Redditi". It displays a tax declaration form with the following data:
Reddito lordo: 35000,00
Tipologia: lavoratori dipendenti € 8.000,00
Imponibile: € 27.000,00
Spese mediche: 1200,00
A yellow box at the bottom shows:
Imposta lorda € 4.660,00
imposta netta € 4.420,00



ESEMPI DA COMPITI D'ESAME (2014)

Oroscopi

Vedo, prevedo e stravedo

Segno zodiacale: ARIETE

Oroscopo mensile del segno:

Amore: grande intimita'
Lavoro: impegnatevi di piu'
Salute: perfetto equilibrio

Stampa annuale

Al ristorante

Dento's Restaurant

Menu disponibili MENU base

ANTIPASTO
Prosciutto e melone € 9...

PRIMO
Linguine all'astice € 9,50

SECONDO
Scaloppina al vino bianco € ...

DESSERT
Mascarpone al cioccolato ...

Prezzo totale € 32,50

Medicinali e farmacie

Cerca Per: Principio Attivo

Chiave: paracetamolo

Filtra

Codice	Nome	Confezione	Principio Attivo	Ditta	Prezzo	Gruppo Equiv...
35313055	DEPALGOS	28 cpr riv 10 ...	Oxicodone/par...	MOLTENI & C....	€ 15,30	OXICODONE+...
35313081	DEPALGOS	28 cpr riv 20 ...	Oxicodone/par...	MOLTENI & C....	€ 15,30	OXICODONE+...
35313028	DEPALGOS	28 cpr riv 5 m...	Oxicodone/par...	MOLTENI & C....	€ 15,30	OXICODONE+...
37021021	CODAMOL	16 cpr riv 500 ...	Paracetamolo/...	ABIOGEN PH...	€ 3,16	PARACETAM...
37351018	PARACETAM...	16 cpr eff 500 ...	Paracetamolo/...	LABORATORI ...	€ 3,16	PARACETAM...
27989033	COEFFERAL...	16 cpr riv 500 ...	Paracetamolo/...	BRISTOL-MYE...	€ 3,96	PARACETAM...
31825060	TACHIDOL	16 cpr riv div 5...	Paracetamolo/...	ANGELINI SpA	€ 3,96	PARACETAM...



ESEMPI DA COMPITI D'ESAME (2013)

TeethCollege

TeethCollege

Studenti | 0000987654 Rossi Giuseppe

STUDENTE:	Rossi Giuseppe (matricola: 0000987654)			
27991	Analisi matematica 1 (MAT/05) CFU:9 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28004	Fondamenti di Informatica 1 (ING-INF/05) CFU:12 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
29228	Geometria e Algebra (MAT/03) CFU:6 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
27993	Analisi matematica 2 (MAT/05) CFU:6 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
27996	Fisica 1 (FIS/01) CFU:6 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
28006	Fondamenti di Informatica 2 (ING-INF/05) CFU:12 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
28011	Reti logiche (ING-INF/05) CFU:6 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
26338	Inglese B1 (NOSECTOR) CFU:3 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28012	Calcolatori elettronici (ING-INF/05) CFU:6 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28000	Fisica 2 (FIS/01) CFU:6 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28032	Matematica applicata (MAT/06) CFU:6 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28027	Sistemi informativi (ING-INF/05) CFU:9 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28030	Economia e azienda (ING-IND/35) CFU:6 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
28029	Elettrotecnica (ING-IND/31) CFU:6 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
28014	Comunicazioni (ING-INF/03) CFU:9 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
28020	Sistemi operativi (ING-INF/05) CFU:9 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
28024	Reti di calcolatori (ING-INF/05) CFU:9 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28016	Elettronica (ING-INF/01) CFU:9 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28015	Controlli automatici (ING-INF/04) CFU:9 Ciclo: 1 ^a Tipologia: OBBLIGATORIO			
28021	Ingegneria del software (ING-INF/05) CFU:9 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
17268	Prova finale (NOSECTOR) CFU:6 Ciclo: 2 ^a Tipologia: OBBLIGATORIO			
28072	Amministrazione di sistemi (ING-INF/05) CFU:9 Ciclo: 2 ^a Tipologia: A_SCELTA			
28659	Tecnologie web (ING-INF/05) CFU:12 Ciclo: 2 ^a Tipologia: A_SCELTA			
Totale crediti in carriera: 180				
Cambia insegnamento 17268Prova finale (...) con 17268Prova finale (...) SOSTITUISCI				
Esito operazione: _____				

Elezioni

Elezioni di Dentinia

Partito	Voti	Seggi
Topolini Gialli	7460	4
Gufi Neri	2040	1
Farfalle blu	3482	1
Formiche Rosse	8748	4
Bruchi verdi	8922	4

Elezioni Dentinia – Assegnazione Seggi

Metodo di calcolo: Metodo del quoziente Salva

Elezioni di Dentinia

Partito	Voti	Seggi
Topolini Gialli	7460	4
Gufi Neri	2040	1
Farfalle blu	3482	1
Formiche Rosse	8748	4
Bruchi verdi	8922	5

Elezioni Dentinia – Assegnazione Seggi

Metodo di calcolo: Metodo D'Hondt Salva



ESEMPI DA COMPITI D'ESAME (2013)

Taxi & tassametro

Zann-O-Taxi

Corsa taxi, simulazione costi

Selezione corsa:

Voce	Valore
Corsa	corsa1
Orario	09:40
Distanza km	0,06
Costo	0,50

TeethCollege Exams

Studenti 0000987654 Rossi Giuseppe Mostra: carriera esami

STUDENTE: Rossi Giuseppe (matricola: 0000987654)

27991	Analisi matematica 1	(MAT/05)	CFU:9	Ciclo: 1*	Tipologia: OBBLIGATORIO
28004	Fondamenti di Informatica 1	(ING-INF/05)	CFU:12	Ciclo: 1*	Tipologia: OBBLIGATORIO
29228	Geometria e Algebra	(MAT/03)	CFU:6	Ciclo: 1*	Tipologia: OBBLIGATORIO
27993	Analisi matematica 2	(MAT/05)	CFU:6	Ciclo: 2*	Tipologia: OBBLIGATORIO
27996	Fisica 1	(FIS/01)	CFU:6	Ciclo: 2*	Tipologia: OBBLIGATORIO
28006	Fondamenti di Informatica 2	(ING-INF/05)	CFU:12	Ciclo: 2*	Tipologia: OBBLIGATORIO
28011	Reti logiche	(ING-INF/05)	CFU:6	Ciclo: 2*	Tipologia: OBBLIGATORIO
26338	Inglese B1	(NOSECTOR)	CFU:3	Ciclo: 1*	Tipologia: OBBLIGATORIO
28012	Calcolatori elettronici	(ING-INF/05)	CFU:6	Ciclo: 1*	Tipologia: OBBLIGATORIO
28000	Fisica 2	(FIS/01)	CFU:6	Ciclo: 1*	Tipologia: OBBLIGATORIO

Situazione esami per Rossi Giuseppe

Esami sostenuti: 7
Esami superati: 4
Crediti acquisiti: 33
Media pesata: 20,55
Dettaglio esami sostenuti:
Insegnamento 27991 Voto: 20 Data: 20/06/13
Insegnamento 27993 Voto: 20 Data: 20/06/13
Insegnamento 27996 Voto: 1 Data: 20/06/13
Insegnamento 27996 Voto: 6 Data: 20/06/13
Insegnamento 27996 Voto: 21 Data: 21/06/13
Insegnamento 28004 Voto: 1 Data: 20/06/13
Insegnamento 28004 Voto: 21 Data: 20/06/13
Dettaglio esami superati:
Insegnamento 27991 Voto: 20 Data: 20/06/13
Insegnamento 27993 Voto: 20 Data: 20/06/13
Insegnamento 27996 Voto: 21 Data: 21/06/13
Insegnamento 28004 Voto: 21 Data: 20/06/13

Nuovo esame 17268Prova finale (...) voto 1 Data 20/06/13 INSERISCI



ESEMPI DA COMPITI D'ESAME (2012)

Bolletta elettrica

Bolletta elettrica

Codice Fiscale	=====
MRARSS76H12A944I	ElettroDent SpA - Energia che illumina
Cognome	Sig. Rossi Mario (MRARSS76H12A944I)
Rossi	Bolletta di Maggio 2012
Nome	Tariffa MONORARIA 2, Consumo KWh 400.0
Mario	
Mese	Dettaglio importi:
5	Costo energia entro soglia € 26,78
Anno	Costo energia extra soglia € 60,75
2012	Corrispettivo per accise € 5,68
Tariffa	Corrispettivo per IVA € 9,32
MONORARIA 2	Totale Bolletta € 102,52
Consumo (KWh)	
400	
<input type="button" value="Calcola"/>	

Agenzia viaggi

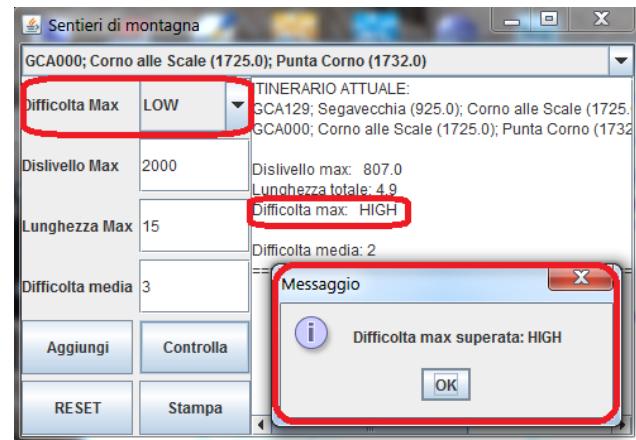
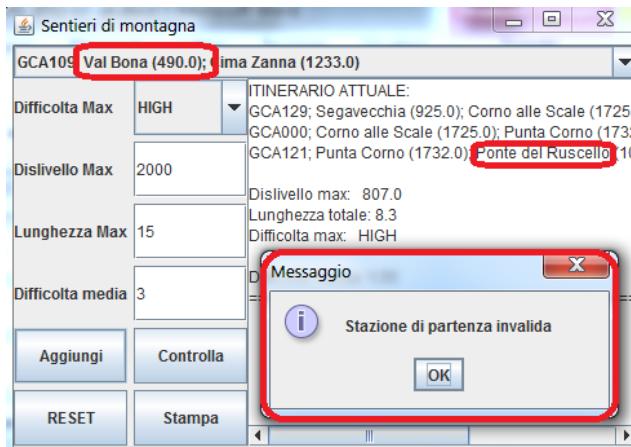
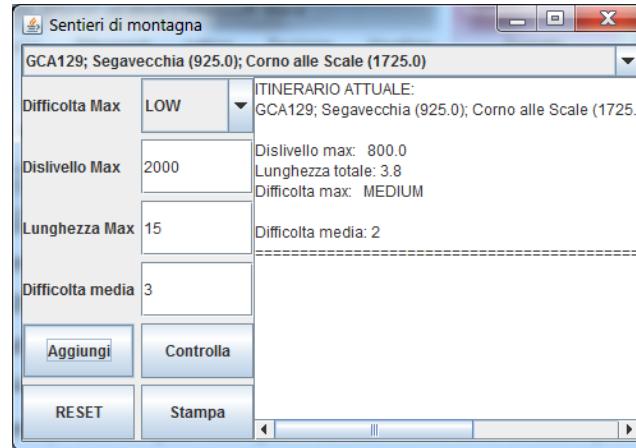
Il gatto e la volpe holidays

KENYA	1000	02/07/2012	02/10/2012
Tipo	COMPLETO	KENYA2356; Dest. :Watamu; Costo: 900.0; Inizio: 03/07/2012; Durata: 9 giorni KENYA2354; Dest. :Watamu; Costo: 936.0; Inizio: 28/07/2012; Durata: 9 giorni KENYA2355; Dest. :Watamu; Costo: 966.0; Inizio: 03/07/2012; Durata: 9 giorni	
Basso costo	<input checked="" type="radio"/>		
Inizio vicino	<input type="radio"/>		
Durata simile	<input type="radio"/> 7		



ESEMPI DA COMPITI D'ESAME (2012)

Gruppo Camminatori Appenninici





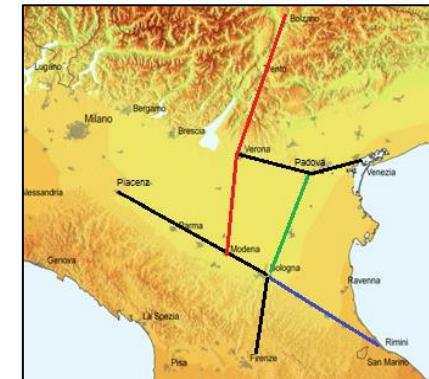
ESEMPI DA COMPITI D'ESAME (2012)

Percorsi autostradali

The screenshot shows a software interface for calculating highway routes. On the left, there are dropdown menus for 'PARTENZA' (Z22, Campogalliano) and 'ARRIVO' (Z4E, Vicenza). A button 'Calcola Percorso' is also present. The main area displays two solutions:

Soluzione 1 (11.1 km, 178.0 cost):
Tratta: Campogalliano, Carpi, Reggiolo, Mantova, Nogarole Rocca
Interscambio fra Z4E Z22

Soluzione 2 (21.2 km, 335.0 cost):
Tratta: Soave, Montebello, Montecchio, Vicenza, Padova Ovest
Interscambio fra Z1 Z22



Bancomat

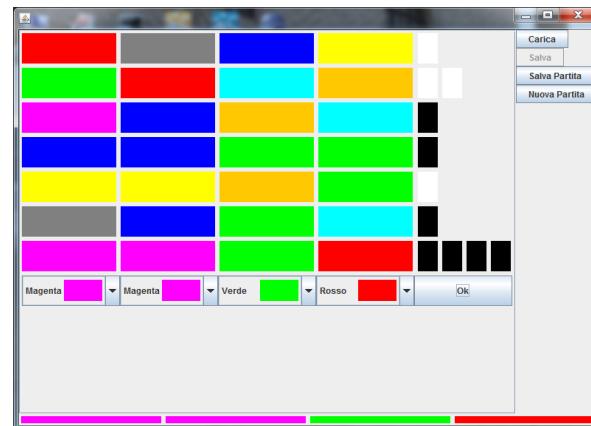
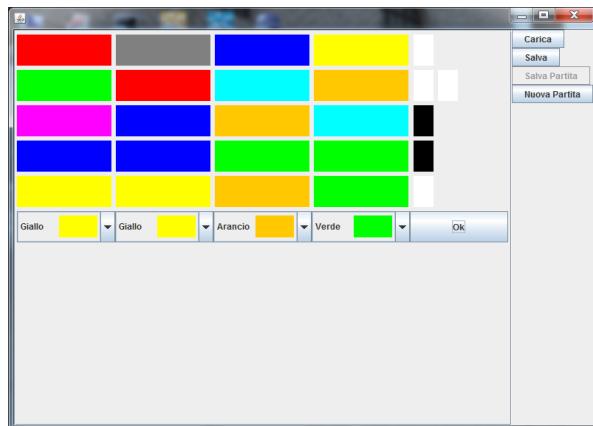
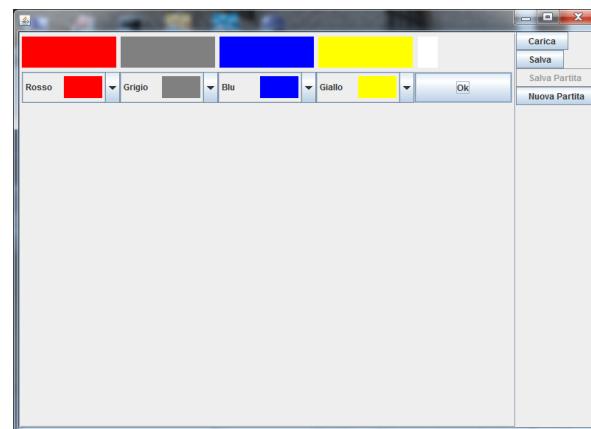
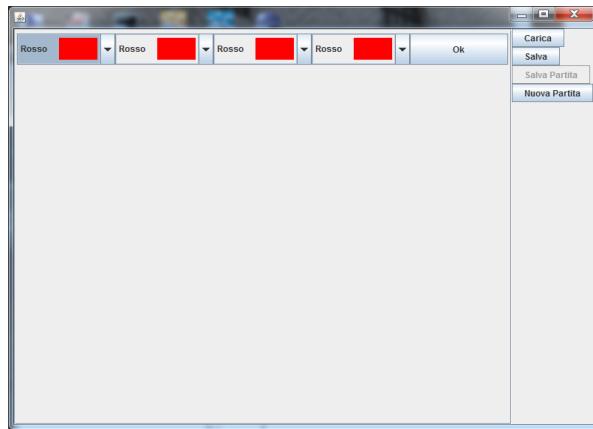
The screenshots show a sequence of windows from a bank machine interface:

- Step 1:** Displays 'Bancomat' and 'Pin' fields. The Pin field contains '3214576'. Buttons 'Ok' and 'Annulla' are at the bottom.
- Step 2:** Shows the selection of 'Operazione' (Prelievo) and 'Ok' or 'Annulla' buttons.
- Step 3:** Shows the 'Importo' field set to '200' and 'Ok' or 'Annulla' buttons.
- Step 4:** Displays the results of the withdrawal: 'Banconote da CINQUANTA: 4' and 'Banconote da VENTI: 0', with 'Ok' and 'Annulla' buttons.
- Step 5:** Shows the final balance: 'Saldo: 1700', with 'Ok' and 'Annulla' buttons.



ESEMPI DA COMPITI D'ESAME (2012)

Master Mind





ESEMPI DA COMPITI D'ESAME (2011)

Ferrovie

Travel Solutions:

Treno: RZ612 da Bologna Centrale a Ferrara Tempo di percorrenza: 0:28
Treno: R 11480 da Bologna Centrale a Ferrara Tempo di percorrenza: 0:55
Treno: R 2230 da Bologna Centrale a Ferrara Tempo di percorrenza: 0:24
Treno: R 11480 da Bologna Corticella a Ferrara Tempo di percorrenza: 0:46
Non ci sono altre soluzioni

Departure City: BOLOGNA Arrival City: FERRARA

Departure Date: mer 01/06/2011

Data Iniziale: Fri 27/05/2011

Data Finale: Mon 27/06/2011

Sorgente	Destinazione	Data	Importo	Causale
esterno	ccGabriele	Jun 27, 2011	123,456	Fatturona
ccGabriele	borsaAM	Jun 27, 2011	1,234	Regalo

Bilancio familiare

Data Iniziale: Fri 27/05/2011

Data Finale: Sun 26/06/2011

Saldo: 0.0

Entrate: 0.0

Uscite: 0.0

borsaAM

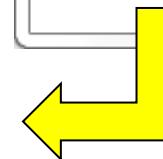
Data Iniziale: Fri 27/05/2011

Data Finale: Sun 26/06/2011

Saldo: 0.0

Entrate: 0.0

Uscite: 0.0





ESEMPI DA COMPITI D'ESAME (2011)

Previsioni meteo

Previsioni Meteo

Città: Bologna
Data: mar 14/06/2011
Tipo di previsione: Puntuale

Previsioni richieste:

Bologna 12/07/11 2.00 temperatura: 18.5° grado di umidità: 91% tempo previsto: pioggia

Bologna 12/07/11 5.00 temperatura: 16.0° grado di umidità: 92% tempo previsto: nuvoloso

Bologna 12/07/11 8.00 temperatura: 20.0° grado di umidità: 87% tempo previsto: sereno

Bologna 12/07/11 11.00 temperatura: 24.0° grado di umidità: 64% tempo previsto: sereno

Bologna 12/07/11 14.00 temperatura: 27.0° grado di umidità: 78% tempo previsto: temporale

Bologna 12/07/11 17.00 temperatura: 15.0° grado di umidità: 58% tempo previsto: nuvoloso

Previsioni Meteo

Città: Bologna
Data: mar 12/07/2011
Tipo di previsione: Globale

Previsioni richieste:

Bologna 12/07/11 temperatura minima: 15.0° temperatura massima: 27.0° tempo previsto: nuvoloso

Agenzia immobiliare

Tipo Annuncio: VENDITA
Tipologia Immobile: MONOLOCALE
Città: BOLOGNA
Zona: SAFFI
Ascensore: No
Posto Auto: No
Data Pubblicazione: 13/09/11
Richiesta: 77000€
Codice: V20688191
Tipologia: MONOLOCALE
Metri Quadri: 25
Descrizione: monolocale interno con angolo cottura

Città: BOLOGNA
Zona: BORGOPANIGALE
Max Richiesta:
Con Ascensore
Con Posto Auto
Periodo (gg):

Cerca

Città: BOLOGNA
Zona: PONTEVECCIO
Ascensore: Si
Posto Auto: No
Data Pubblicazione: 11/09/11
Richiesta: 95000€
Codice: V20783491
Tipologia: BILOCALE
Metri Quadri: 80
Descrizione: ristrutturato con grande open space,
Città: BOLOGNA
Zona: PONTEVECCIO
Ascensore: Si
Posto Auto: No
Data Pubblicazione: 11/09/11
Richiesta: 95000€
Codice: V5489014
Tipologia: BILOCALE
Metri Quadri: 70



FRAMEWORK GRAFICI IN JAVA

Un po' di storia

- in origine: AWT (Abstract Window Toolkit)
 - disponibile da Java 1 [1996], package `java.awt`
 - ormai in disuso (ne sopravvivono piccole porzioni in Swing)
- la star del quindicennio: Swing
 - disponibile da Java 2 [1998], package `javax.swing`
 - scritto in Java, indipendente dalla piattaforma
 - *usato per 15+ anni, amplissima base codice installata*
 - dal 2015 in *maintenance mode* (non più sviluppato, solo bug fix)
- la novità di Java 8: JavaFX
 - disponibile da Java 8 [2014], set di package `javafx.*`
 - *da Java 11, spostato in un modulo da scaricare separatamente*



JavaFX: il supporto



INSTALLARE JavaFX

- Fino a Java 10, JavaFX era incluso nel JDK
- Da Java 11 in poi va invece scaricato e installato a parte

openjfx.io/openjfx-docs/#install-javafx

Getting Started with JavaFX

Introduction
Install Java
Run HelloWorld using JavaFX
Run HelloWorld via Maven
Run HelloWorld via Gradle
Runtime images
JavaFX and IntelliJ
JavaFX and NetBeans
JavaFX and Eclipse
Next Steps

Run HelloWorld using JavaFX

Download an appropriate [JavaFX runtime](#) for your operating system and unzip it to a desired location. For this tutorial, we will be using JavaFX 14.

Add an environment variable pointing to the lib directory of the runtime:

[Linux/Mac](#) **Windows**

```
set PATH_TO_FX="path\to\javafx-sdk-14\lib"
```

You can now compile and run JavaFX applications from the command line using the JavaFX runtime.

Compile the application (e.g. use `HelloFX.java` from this [sample](#)) using:

[Linux/Mac](#) **Windows**

```
javac --module-path %PATH_TO_FX% --add-modules javafx.controls HelloFX.java
```

Note: Additional modules are required for extended functionality. For example, if your application is using `FXML`, you will need to add the `javafx.fxml` module, as shown below:

[Linux/Mac](#) **Windows**

```
javac --module-path %PATH_TO_FX% --add-modules javafx.controls,javafx.fxml HelloFX.java
```



JavaFX: USO

- Dopo aver installato OpenJFX
 - per la compilazione e l'esecuzione da riga di comando si deve specificare il *module path* di OpenJFX
 - per la compilazione e l'esecuzione da dentro Eclipse si deve configurare opportunamente SIA lo strumento SIA ogni progetto!
- Eclipse
 - come per tutte le applicazioni Java, sebbene includa il suo compilatore, ha comunque bisogno del JRE (OpenJDK 17, 18..)
 - inoltre, per i progetti JavaFX occorre aggiungere anche:
 - OpenJFX come *user library*
 - il *module path* alla *Run configuration*
 - altrimenti, non funzionerà nulla! ☹



JavaFX: USO

- Dopo aver installato OpenJFX
 - per la compilazione e l'esecuzione da riga di comando si deve specificare il *module path* di OpenJFX
 - per la compilazione e l'esecuzione da dentro Eclipse si deve configurare opportunamente SIA lo strumento SIA ogni progetto!
- Eclipse
 - come per tutte le applicazioni, ha comunque bisogno di essere configurata la struttura del progetto.
 - inoltre, per i progetti JavaFX occorre aggiungere anche:
 - OpenJFX come *user library*
 - il *module path* alla *Run configuration*
 - altrimenti, non funzionerà nulla! ☹



JavaFX: USO

- JDK: dover impostare **ogni volta** il **module path** a mano è una bella scocciatura
- ECLIPSE: dover impostare **ogni volta** la **libreria ausiliaria** e il **module path nella run configuration** a mano è una **enorme** scocciatura!
- IN ALTERNATIVA, ci si può «**confezionare**» un **JDK personalizzato**, **contenente già OpenJFK**
 - dettagli su <https://openjfx.io/openjfx-docs/>

Nelle prossime slide mostreremo entrambi gli approcci



ECLIPSE & OPENJFX

- In Eclipse, un progetto Java che usi JDK 11+ e anche JavaFX darà luogo a errori:

The screenshot shows a Java code editor window titled "EsJavaFX00.java". The code is as follows:

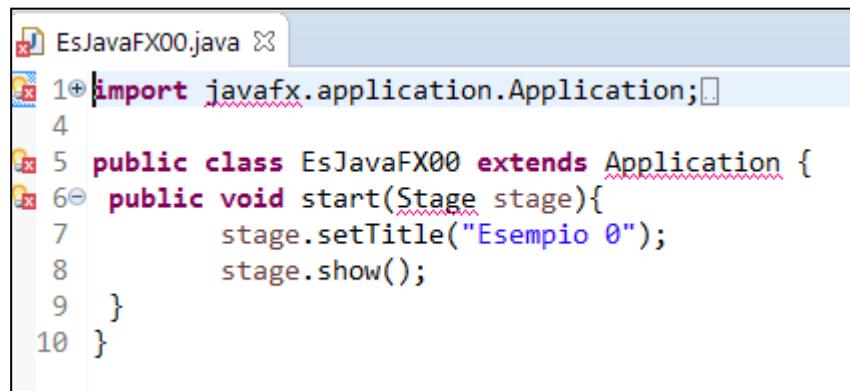
```
1+ import javafx.application.Application;
2
3 public class EsJavaFX00 extends Application {
4     public void start(Stage stage){
5         stage.setTitle("Esempio 0");
6         stage.show();
7     }
8 }
9
10 }
```

The line "import javafx.application.Application;" has a red underline, indicating a compilation error. The code editor interface includes tabs for "File", "Edit", "Search", "Run", and "Help".

- MOTIVO: anche se installato, OpenJFX non è nel *module path* di default e quindi le relative classi non vengono trovate
- SOLUZIONE: aggiungere alle proprietà del progetto l'indicazione del *module path* di OpenJFX
 - la qual cosa richiede però, a sua volta, di definire una *User Library*...

ECLIPSE & OPENJFX

- In Eclipse, un progetto Java che usi JDK 11+ e anche JavaFX darà luogo a errori:



A screenshot of an Eclipse code editor window titled "EsJavaFX00.java". The code is as follows:

```
1+ import javafx.application.Application;
2
3 public class EsJavaFX00 extends Application {
4     public void start(Stage stage){
5         stage.setTitle("Esempio 0");
6         stage.show();
7     }
8 }
9
10 }
```

The line "import javafx.application.Application;" has a red underline, indicating a compilation error. Lines 1 through 10 are numbered on the left.

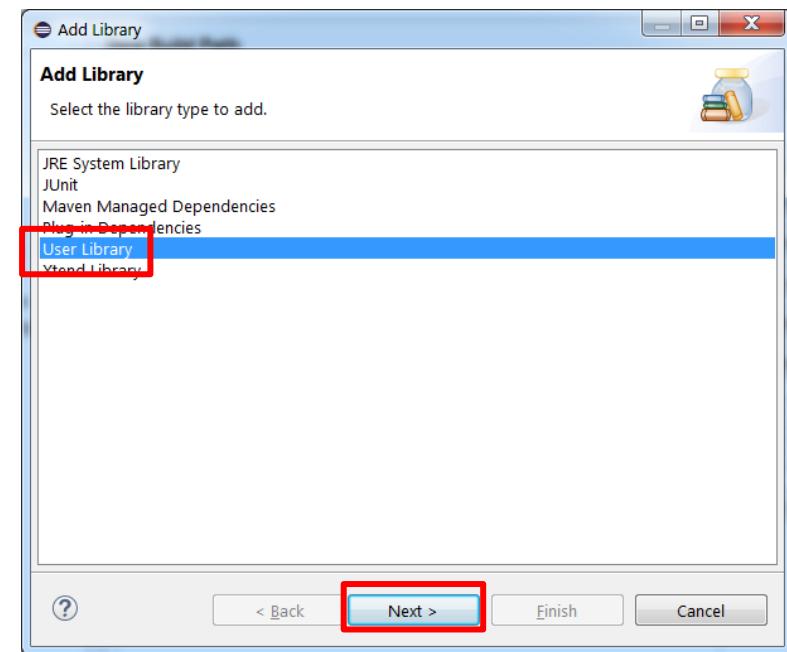
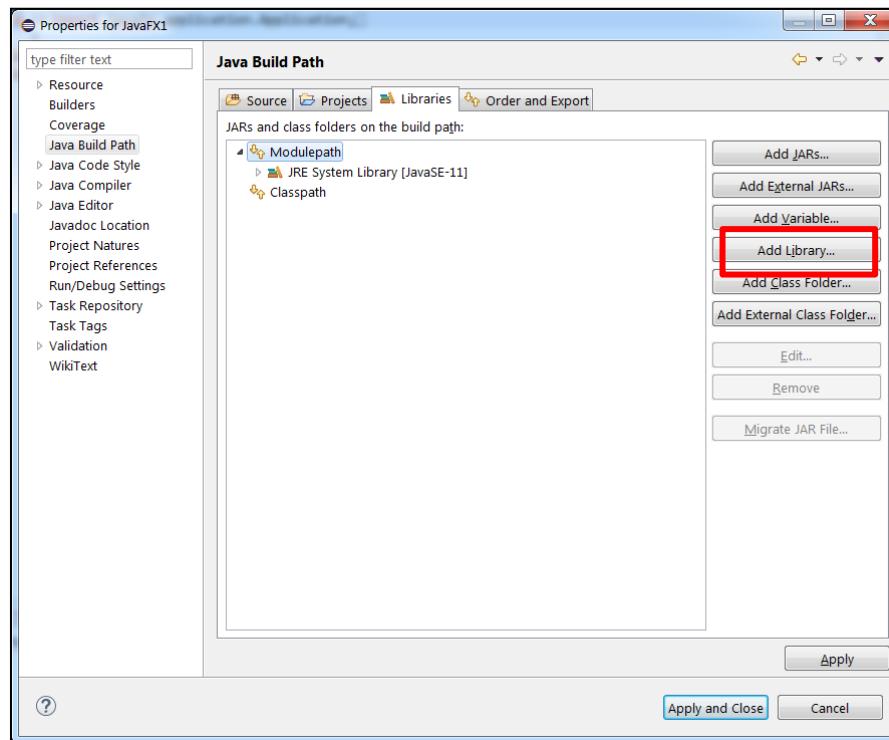
- MOTIVO: anche se installato, OpenJFX non è nel *module path* di default e quindi le relative classi non vengono trovate
- SOLUZIONE 1: aggiungere alle proprietà del progetto l'indicazione del *module path* di OpenJFX
- SOLUZIONE 2: farsi un JDK «personalizzato» e usare quello



SOLUZIONE 1

ECLIPSE & OPENJFX (1)

- Per definire la User Library per JavaFX, apriamo le proprietà del progetto e scegliamo *Add Library*:

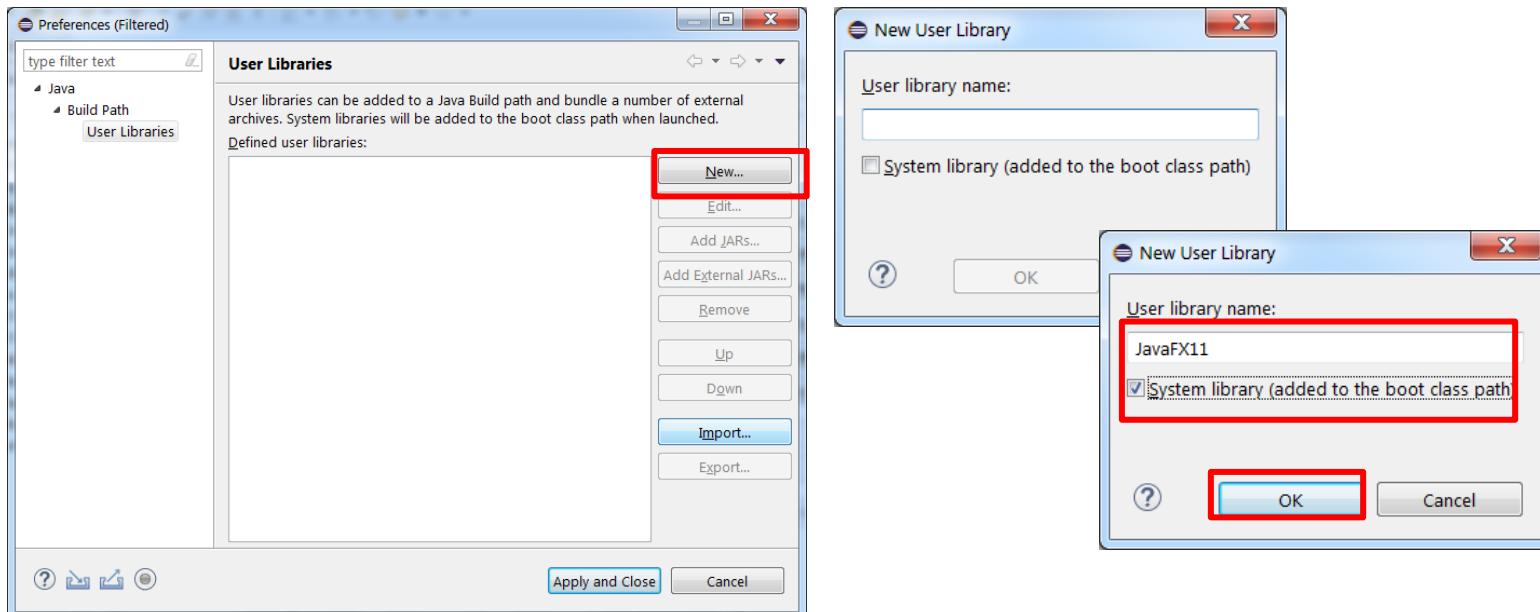




SOLUZIONE 1

ECLIPSE & OPENJFX (2)

- Nella finestra *User Libraries* che si apre, premiamo *New*
- Nella finestra di dialogo che compare, digitiamo il nome (arbitrario) della nuova libreria (ad esempio, "JavaFX17")..
- ... attiviamo l'opzione per renderla una *System Library* e diamo *OK*.

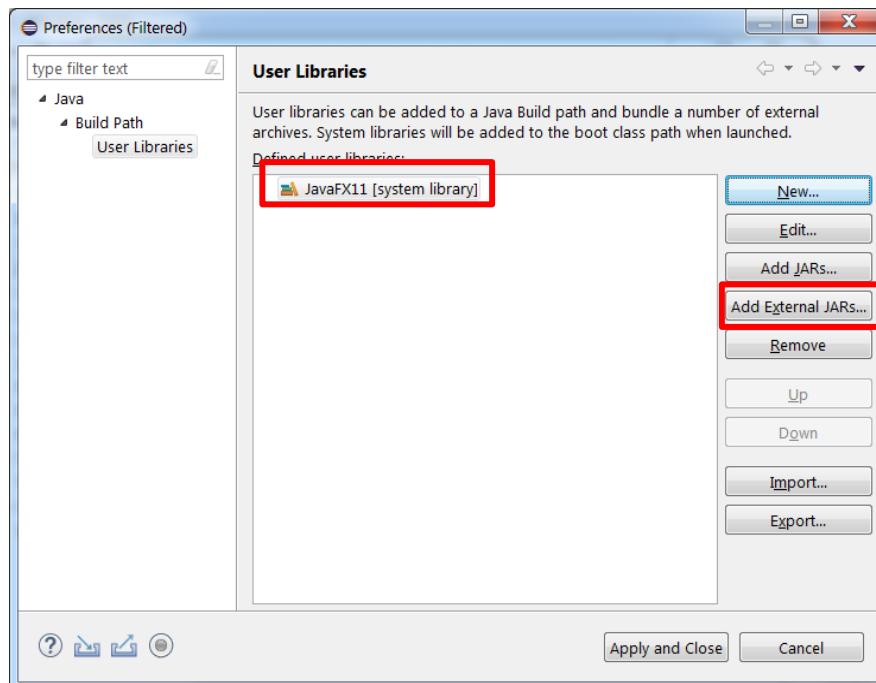




SOLUZIONE 1

ECLIPSE & OPENJFX (3)

- Si ritorna così alla finestra *User Libraries*, dove ora compare anche la nuova libreria ("JavaFX17") da noi creata.
- Essa però al momento è vuota, ossia *non associata ad alcun JAR*: clicchiamo allora su *Add External JARs...*

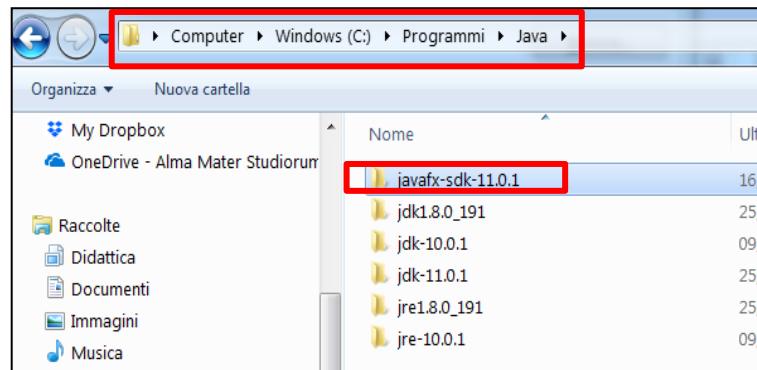




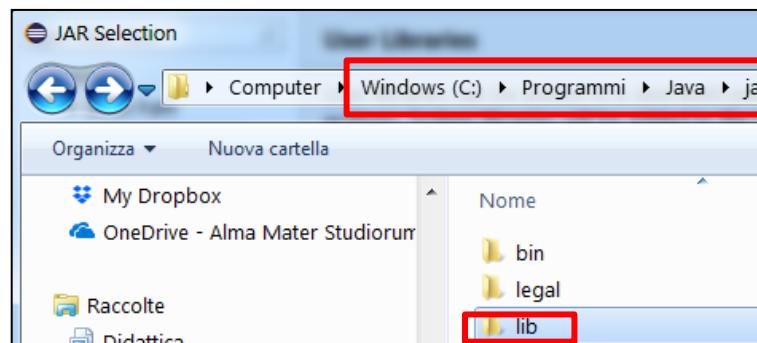
SOLUZIONE 1

ECLIPSE & OPENJFX (4)

- Nel dialogo che compare *navighiamo fino alla cartella del file system dove abbiamo installato JavaFX..*



- .. e scendiamo nella sotto-cartella *lib*:

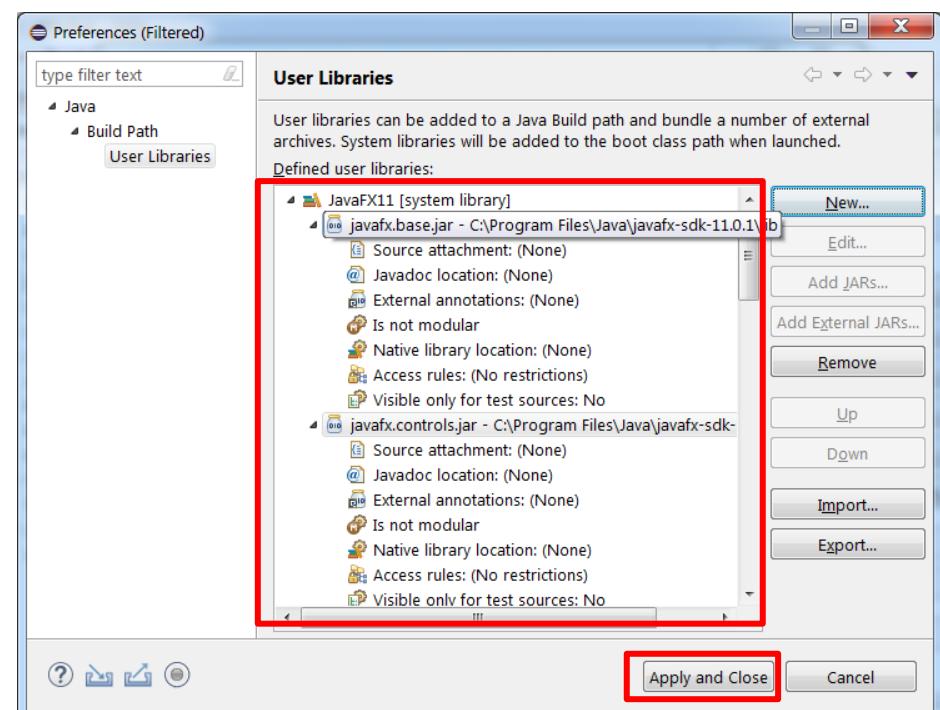
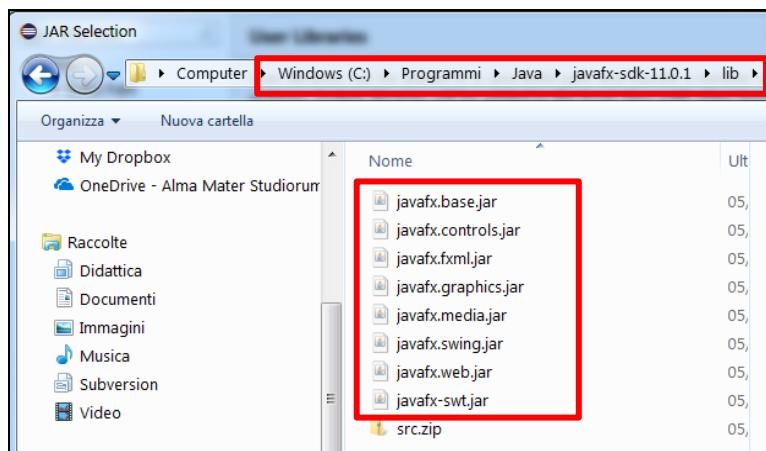




SOLUZIONE 1

ECLIPSE & OPENJFX (5)

- Lì dentro ci sono tutti i JAR che ci servono → selezioniamoli tutti e confermiamo con OK
- Se tutto è andato bene, torneremo alla finestra *User Libraries* e si vedrà la nuova libreria con tutti i JAR. Confermiamo con Apply & Close

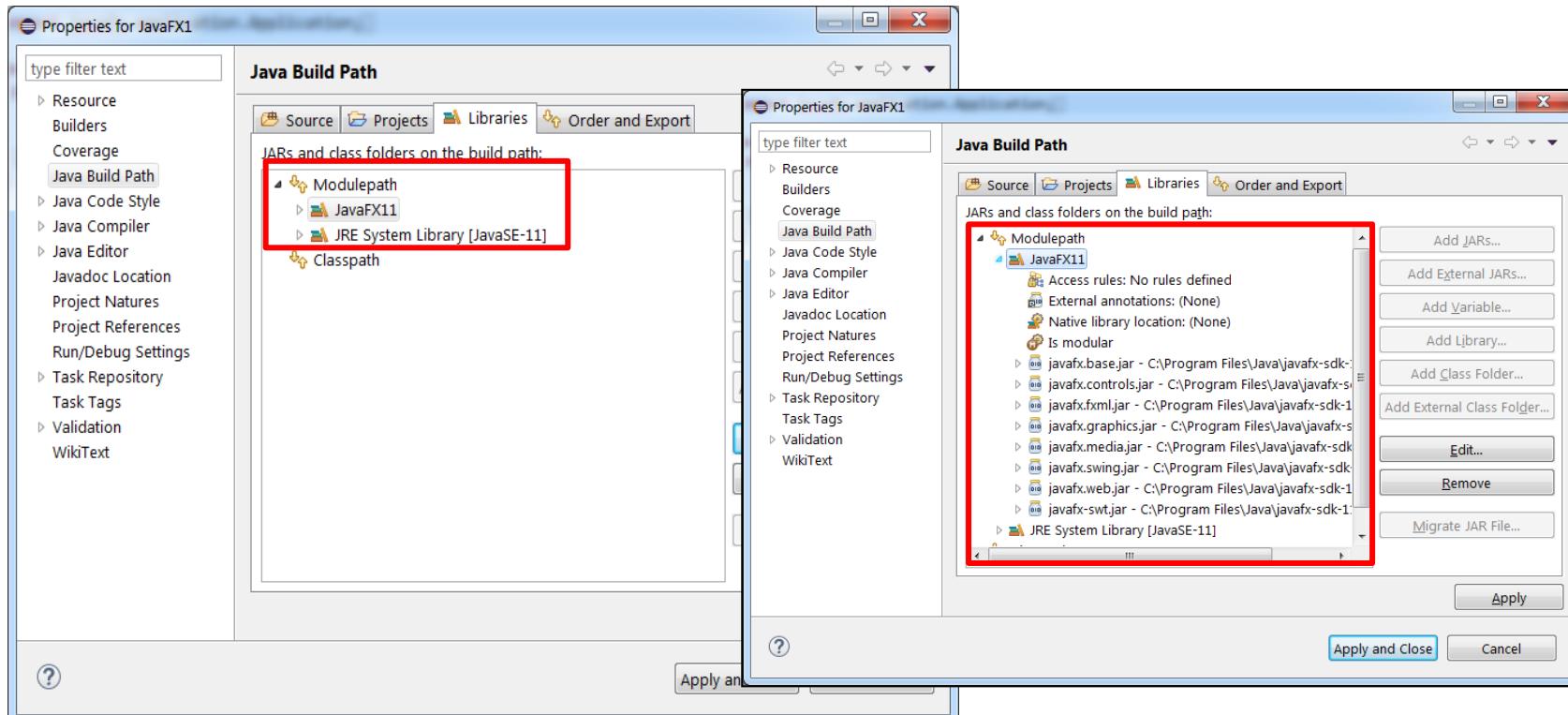




SOLUZIONE 1

ECLIPSE & OPENJFX (6)

- L'ultimo OK ci avrà riportato alla finestra iniziale del Java Build Path
- Potremo osservare con soddisfazione la presenza della nuova libreria accanto al JRE classico (volendo, si può espanderne il contenuto)





SOLUZIONE 1

ECLIPSE & OPENJFX (7)

- Ora finalmente il progetto compila senza errori! ☺ ☺

The screenshot shows the Eclipse IDE interface. The left pane is the Package Explorer, displaying a project structure with packages like BasicElections, BasicElections2, Elections, and JavaFX1. The JavaFX1 package contains a src folder and a (default package). The right pane is the Editor view, showing the code for EsJavaFX00.java:

```
1+ import javafx.application.Application;
2
3 public class EsJavaFX00 extends Application {
4
5     public void start(Stage stage){
6         stage.setTitle("Esempio 0");
7         stage.show();
8     }
9
10
11 public static void main(String[] args) {
12     launch(args);
13 }
14 }
```

- MA provando a eseguirlo non va! ☹ ☹
Motivo: manca l'impostazione della *Run configuration*

The screenshot shows the Eclipse Console window. It displays the output of a terminated Java application named EsJavaFX00. The console output is as follows:

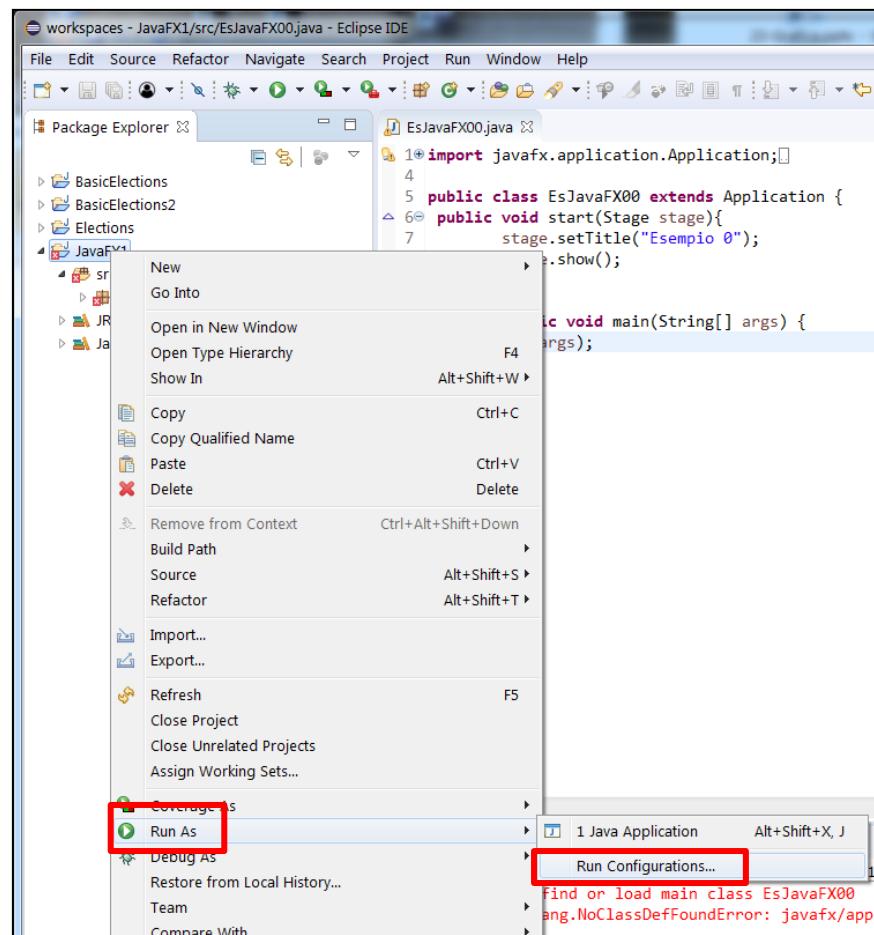
```
<terminated> EsJavaFX00 [Java Application] C:\Program Files\Java\jdk-11.0.1\bin\javaw.exe (27 mar 2019, 18:41:16)
Error: Could not find or load main class EsJavaFX00
Caused by: java.lang.NoClassDefFoundError: javafx/application/Application
```



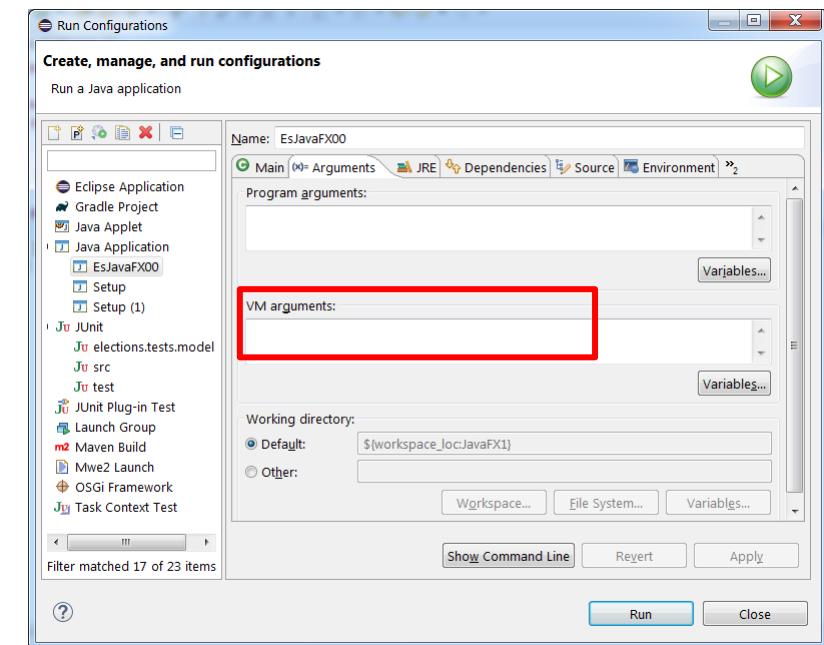
SOLUZIONE 1

ECLIPSE & OPENJFX (8)

- Forza e coraggio: apriamo le *Run Configurations...*



... e posizioniamoci sul tab
Run Arguments

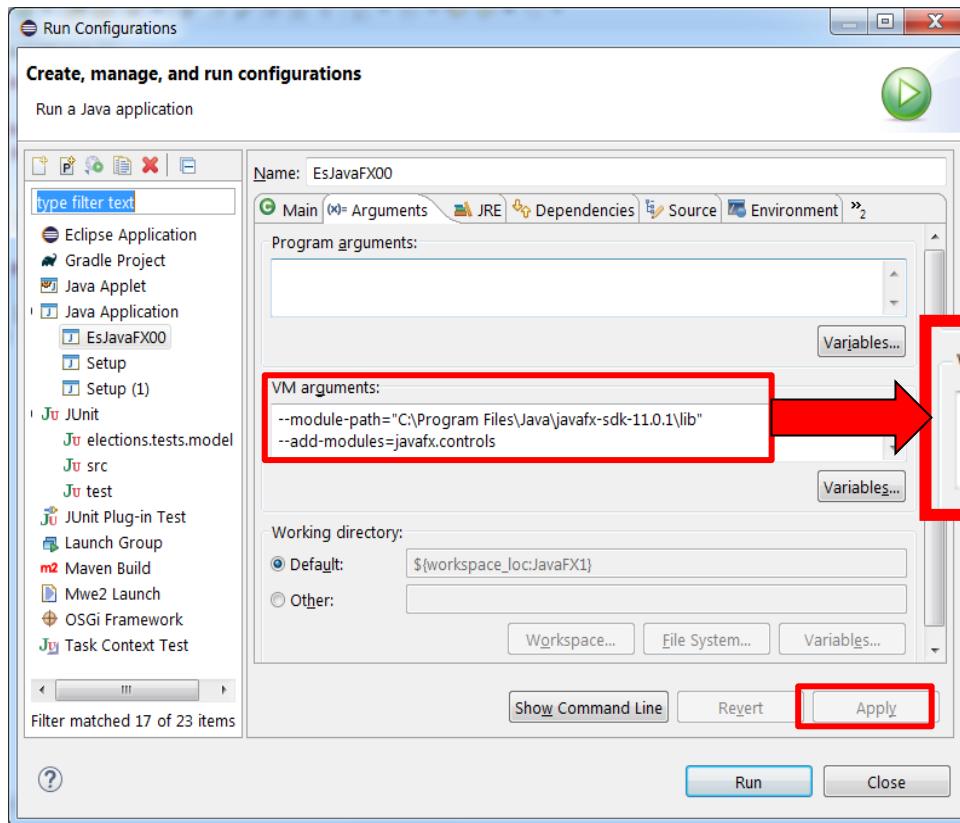




SOLUZIONE 1

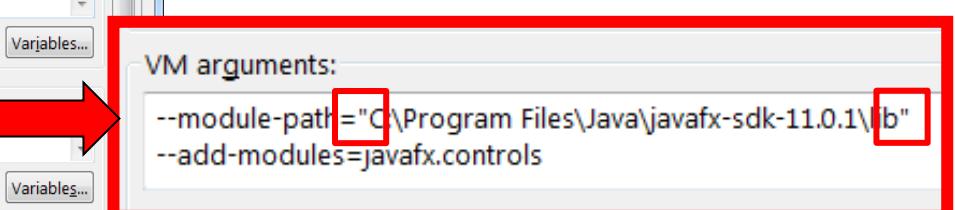
ECLIPSE & OPENJFX (9)

- Inseriamo come *Run Arguments* i giusti valori:



OCCHIO alle virgolette intorno
al module path!

Altrimenti, con gli spazi.. ☹





SOLUZIONE 1

ECLIPSE & OPENJFX (10)

- E finalmente.... FUNZIONAAAA!!!

The screenshot shows the Eclipse IDE interface with the following details:

- File Bar:** File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help.
- Toolbar:** Standard Eclipse toolbar icons.
- Package Explorer:** Shows projects: BasicElections, BasicElections2, Elections, JavaFX1 (selected), src (containing default package), JRE System Library [JavaSE-11], JavaFX11.
- Editor:** Displays the file `EsJavaFX00.java` with the following code:

```
1 import javafx.application.Application;
2
3 public class EsJavaFX00 extends Application {
4     public void start(Stage stage){
5         stage.setTitle("Esempio 0");
6         stage.show();
7     }
8
9 }
10
11 public static void main(String[] args) {
12     launch(args);
13 }
14 }
```
- Output View:** A window titled "Esempio 0" is visible in the bottom right corner, enclosed in a red border. It is a blank white window, indicating the application has run but no content is displayed.



SOLUZIONE 1: RIASSUNTO RUN CONFIGURATION

- IN DEFINITIVA
 - per un'applicazione non modulare (come è il nostro caso) occorre aggiungere alla *Run configuration* gli argomenti:
`--module-path=/lib`
`--add-modules=javafx.controls`
- Il motivo è da ricercarsi nel fatto che:
 - da Java 9, l'intera infrastruttura Java è internamente modulare
 - da Java 11, JavaFX è stata spostata in un modulo a parte, non inserito di default nel JRE standard
 - ergo, è necessario non solo scaricare tale pacchetto extra, ma anche *dire al JRE dove trovarlo e che moduli contiene*
 - altrimenti, le classi di JavaFX non vengono trovate né riconosciute.



SOLUZIONE 2: JDK «PERSONALIZZATO» (1)

- In alternativa a tutto questo, ci si può costruire un JDK «personalizzato» che incorpori già anche OpenJFX

<https://openjfx.io/openjfx-docs/>

Si ricostruisce così, di fatto,
ciò che era lo standard
fino a Java 10.

→ seguire le *istruzioni* per
il proprio sistema operativo

The screenshot shows the 'Getting Started with JavaFX' page. On the left, there's a sidebar with links like 'Introduction', 'Install Java', and 'Runtime images'. The 'Runtime images' link is circled in red. Below it, under 'Modular from CLI', is a link 'Custom JDK+JavaFX image' which is also circled in red. To the right, there's a 'Runtime images' section with instructions for creating a runtime image, including environment variable settings for Linux/Mac and Windows. Below that is a 'Command Line' section with instructions for running or creating a runtime image from the command line, also with Linux/Mac and Windows specific examples.



SOLUZIONE 2: JDK «PERSONALIZZATO» (2)

- Cosa serve
 - scaricarsi preventivamente da Gluon **la particolare versione JMODS di JavaFX** (i JMODS sono i precompilati binari, platform-specific)

Runtime images

If you want to create a runtime image of your JavaFX project follow these instructions.

Download an appropriate [JavaFX runtime](#) and [JavaFX jmods](#) for your operating system and unzip them to a desired location.

<https://gluonhq.com/products/javafx/>

- fare una cartella nuova vuota, da usare come destinazione per il JDK personalizzato da generare (es. “custom-jdk17”)



SOLUZIONE 2: JDK «PERSONALIZZATO» (3)

Downloads

JavaFX version	Operating System	Architecture	Type
17.0.2 [LTS]	[any]	x64	[any]

Include older versions

OS	Version	Architecture	Type	Download
Linux	17.0.2	x64	SDK	Download [SHA256]
Linux	17.0.2	x64	jmods	Download [SHA256]
Linux	17.0.2	x64	Monocle SDK	
macOS	17.0.2	x64	SDK	
macOS	17.0.2	x64	jmods	
macOS	17.0.2	x64	Monocle SDK	
Windows	17.0.2	x64	SDK	
Windows	17.0.2	x64	jmods	
Windows	17.0.2	x64	Monocle SDK	
Javadoc	17.0.2		Javadoc	

JavaFX Roadmap

Release	GA Date	Latest version	Long Term Support
19	September 2022 (planned)	early access	no
18	March 2022	18 (March 2022)	no
17	September 2021	17.0.2 (January 2022)	until September 2026
16	March 2021	16 (March 2021)	no
15	September 2020	15.0.1 (October 2020)	no
14	March 2020	14.0.2.1 (July 2020)	no
13	September 2019	13.0.2 (January 2020)	no
12	March 2019	12.0.2 (July 2019)	no
11	September 2018	11.0.14 (January 2022)	until September 2023



SOLUZIONE 2: JDK «PERSONALIZZATO» (4)

- Procedimento

1. scompattare in tale cartella lo zip coi jmods
→ si ottiene la cartella *javafx-jmods-nn.n.n* che contiene 7 file
javafx.base.jmods, *javafx.controls.jmods*, etc.

Variano con la versione

2. aprire un terminale comandi nella cartella iniziale e settare la directory di lavoro alla directory corrente (cd ...)

3. ora, lanciare lo strumento **jlink** così:

NB: usare la propria
versione (17, 18..)

```
jlink --module-path javafx-jmods-17.0.2  
--add-modules java.se,javafx.fxml,javafx.web,javafx.media  
,javafx.swing --bind-services --output myjdk11_with_fx
```

Si ricorda che la cartella di destinazione non deve già esistere:
se esiste (anche vuota), lui si lamenta

**Verrà creata la classica struttura di un jdk, che si potrà
installare sul pc al posto / a fianco di quello classico**



SOLUZIONE 2: JDK «PERSONALIZZATO» (5)

- Dopo aver installato il nuovo JDK..
- ... finalmente, let's run!
 1. Aprire Eclipse in un workspace nuovo (per pulizia)
 2. Impostare nelle Eclipse → Preferences → Java → Installed JREs *il nuovo jdk personalizzato*
 3. importare (o creare) un progetto Java che usi la grafica
 4. se il progetto è preesistente, *cambiare il build path* cancellando il JRE e l'eventuale JAVAFX precedenti e inserendo come “new JVM” il nuovo jdk personalizzato
 5. FINITO! Ora tutto compila e funziona, anche JUnit, *senza più bisogno di toccare le run configurations ogni volta!* ☺



PROGETTI JAVAFX in ECLIPSE

- Un altro problema è che Eclipse, di suo, *non possiede il concetto di progetto grafico JavaFX*
- Conseguenza: i progetti grafici sono gestiti come progetti Java standard → richiedono il main
 - che invece in JavaFX sarebbe opzionale
- Ci sono quindi tre possibilità:
 1. Far senza ☺
 2. Installare in Eclipse lo *speciale plugin per JavaFX*
 3. [per Eclipse vecchi]: scaricarsi (da Gluon) un Eclipse ad hoc con tutto già dentro («all in one»)



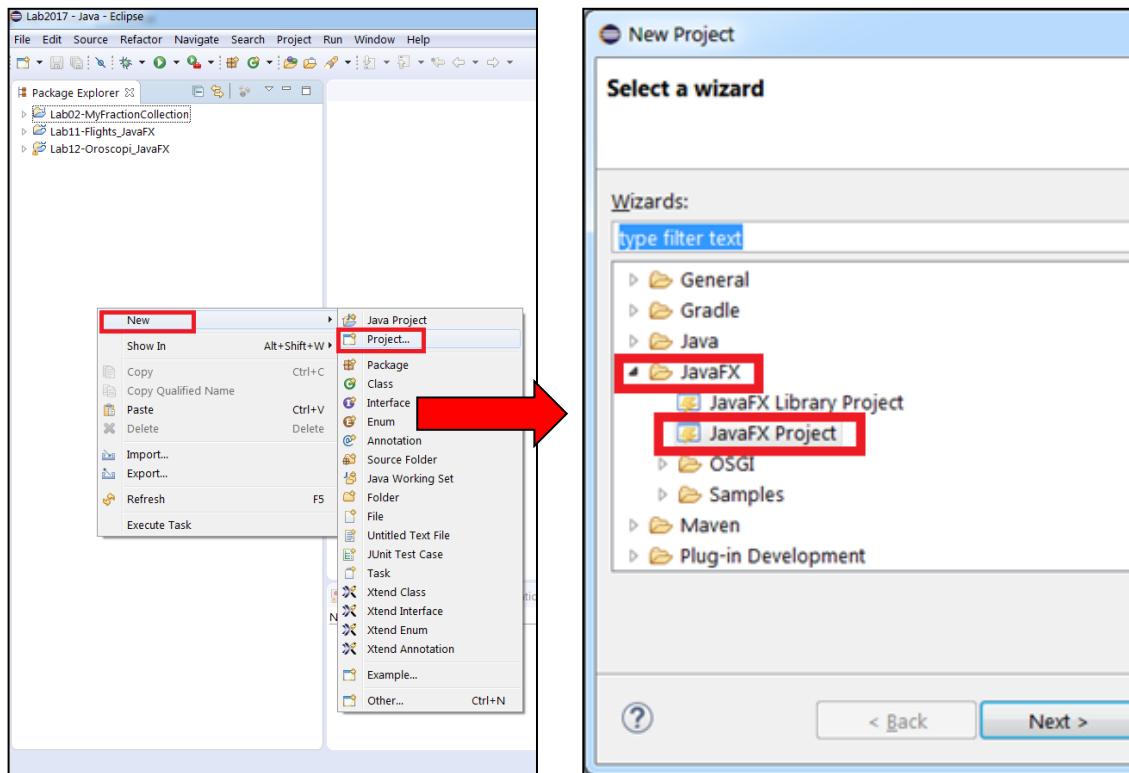
IL PLUGIN PER ECLIPSE: e(fx)clipse

- Il plugin e(fx)clipse per JavaFX
 - aggiunge il concetto di "JavaFX project", con le sue prospettive
 - genera automaticamente gli scheletri delle classi e dei metodi base
 - evita di dover includere il main "standard"
 - e altre comode cose ☺
- PERÒ:
 - non c'è in laboratorio!
 - quindi usatelo, ma *non diventatene dipendenti!*
- Si scarica dal marketplace di Eclipse



ECLIPSE CON JAVAFX

- Con questi strumenti, si rende disponibile il nuovo concetto di *JavaFX Project*, già preconfigurato ☺ ☺





ECLIPSE CON JAVAFX

- Questo progetto nasce con la *giusta struttura JavaFX* e include già lo scheletro della classe base e del main

The screenshot shows the Eclipse IDE interface with the title bar "Lab2017 - Java - Prova/src/application/Main.java - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, Help. The toolbar has various icons for file operations. The Package Explorer view on the left shows a project named "Prova" with a "src" folder containing "application" (which itself contains "Main.java" and "application.css"), "JRE System Library [JavaSE-1.8]", and "JavaFX SDK" (which contains "build.fxbuild"). A yellow callout bubble labeled "Struttura progetto JavaFX" points to the "src" folder in the Package Explorer. The main editor window displays the "Main.java" file with the following code:

```
1 package application;
2
3 import javafx.application.*;
4
5
6 public class Main extends Application {
7     @Override
8     public void start(Stage primaryStage) {
9         try {
10             BorderPane root = new BorderPane();
11             Scene scene = new Scene(root,400,400);
12             scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
13             primaryStage.setScene(scene);
14             primaryStage.show();
15         } catch(Exception e) {
16             e.printStackTrace();
17         }
18     }
19
20     public static void main(String[] args) {
21         Launch(args);
22     }
23 }
24
25
26
27 }
```

A yellow callout bubble labeled "Scheletro classe Main.java generato in automatico, da personalizzare" points to the first few lines of the "Main.java" code.



Da Swing a JavaFX



FRAMEWORK GRAFICI IN JAVA

Un po' di storia

- in origine: AWT (Abstract Window Toolkit)
 - disponibile da Java 1 [1996], package `java.awt`
 - ormai in disuso (ne sopravvivono piccole porzioni in Swing)
- la star del quindicennio: Swing
 - disponibile da Java 2 [1998], package `javax.swing`
 - scritto in Java, indipendente dalla piattaforma
 - *usato per un decennio, amplissima base codice installata*
 - dal 2015 in *maintenance mode* (non più sviluppato, solo bug fix)
- la novità di Java 8: JavaFX
 - disponibile da Java 8 [2014], set di package `javafx.*`
 - da Java 11, *spostato in un modulo da scaricare separatamente*



SWING vs JAVAFX

- Swing e JavaFX condividono molto
 - concetti di *evento* e *ascoltatore degli eventi (event listener)*
 - tipologie di widget e controlli grafici fondamentali
 - impostazione generale dell'applicazione
- ma **JavaFX è costruito su un *approccio più moderno***
 - nuovi componenti e nuovi layout (API di oltre 30 package!)
 - nuovi concetti di *palcoscenico* e *scena*
 - supporto di grafici, effetti speciali e animazioni
 - **nuova struttura dell'applicazione, che deriva da Application**
 - lancio diretto di applicazioni grafiche, anche senza main
 - esecuzione di applicazioni anche in un browser
 - descrizione struttura GUI anche via (F)XML



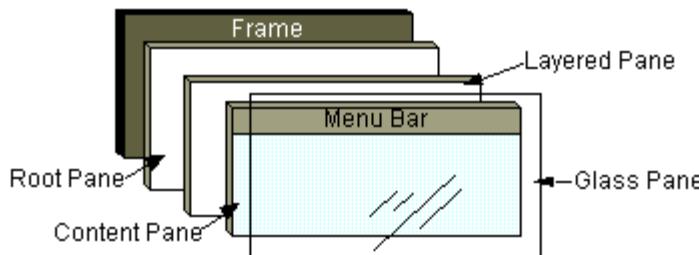
SWING vs JAVAFX

- Un'applicazione Swing richiede la creazione di un **Frame**, in cui va inserito un **Panel**
- Sul pannello si può:
 - disegnare direttamente (grafica a pixel)
 - agganciare componenti (inclusi altri pannelli)
- Organizzazione applicazione *abbastanza libera*

- Un'applicazione JavaFX estende Application (eredita il frame) e si basa anch'essa su un **Pane**
- Sul pannello si può:
 - agganciare una tela (**Canvas**) per disegnare direttamente (a pixel)
 - agganciare componenti (inclusi altri pannelli)
- Organizzazione applicazione *imposta dal framework* secondo *l'approccio teatrale*:
 - il pannello (**Pane**) va inserito in una scena (**Scene**)
 - la scena va resa visibile sul *palcoscenico* (**Stage**)

SWING vs JAVAFX

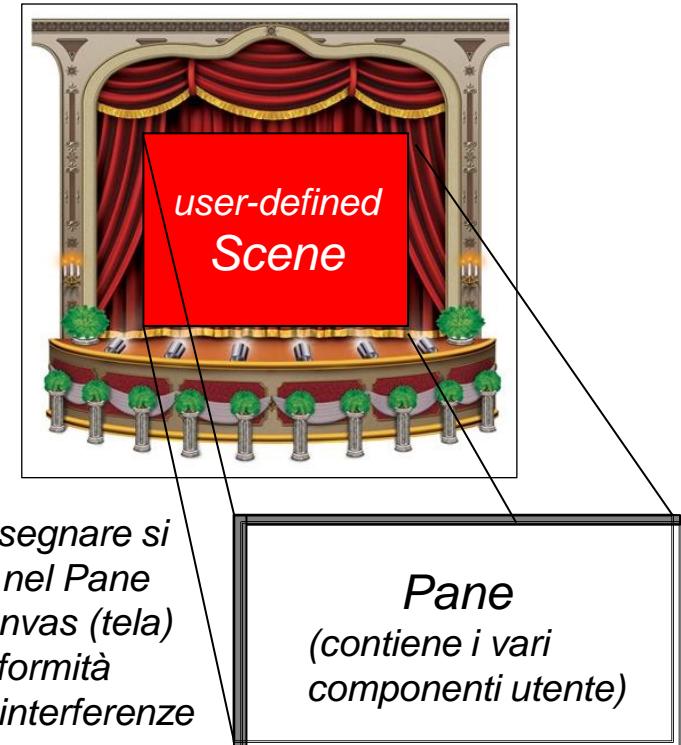
Applicazione Swing



- Sul *content pane* si può:
 - disegnare direttamente (a pixel)
 - agganciare componenti
- Per disegnare si *interferisce nel processo di disegno del pannello*
 - scomodo
 - error-prone
 - disomogeneo rispetto alla grafica a componenti

Applicazione JavaFX

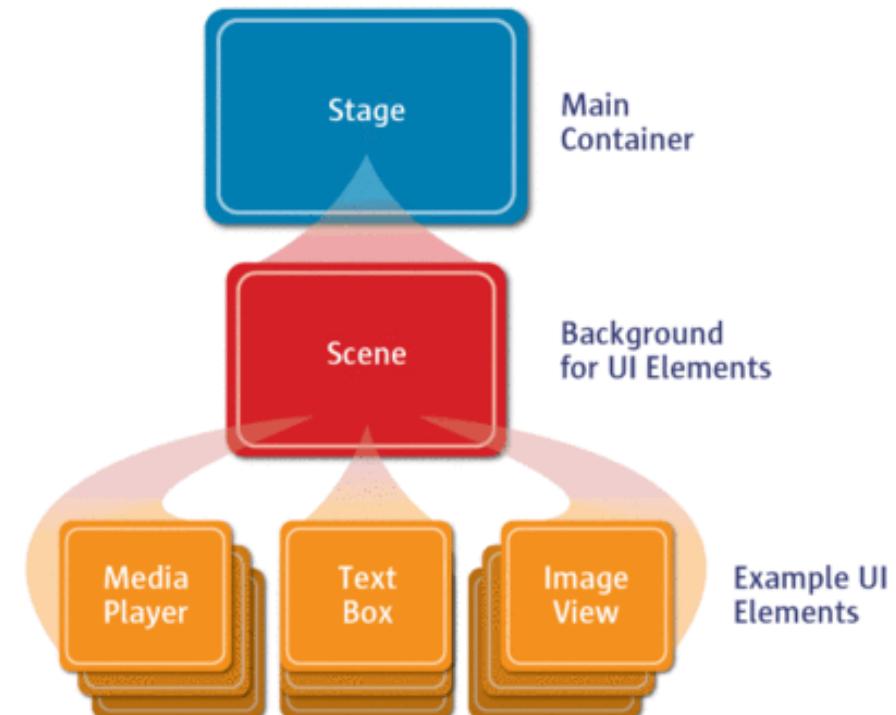
Application (fornisce il frame con lo stage al suo interno)





APPLICAZIONI JAVAFX

- L'applicazione utente estende la classe base **Application**
- Eredita tre metodi fondamentali:
 - **init** per eventuale set-up
 - **start** per creare la GUI
 - **stop** per eventuale shut-down
- L'unico da ridefinire sempre è **start**, il cuore dell'applicazione
 - esso rende disponibile lo **Stage** (già creato) associato alla finestra
- L'utente deve *mettere in scena* il suo spettacolo definendo la sua **Scene**: un opportuno **Pane** deve ospitare i vari componenti.



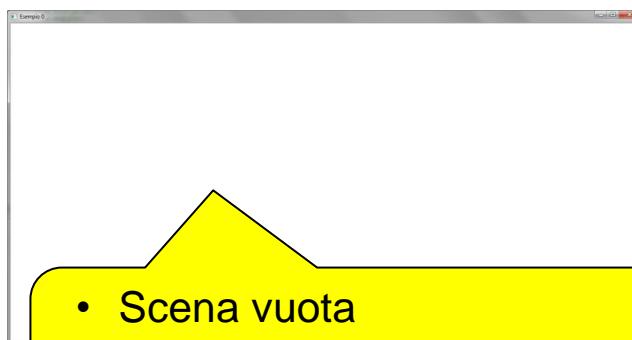


JAVAFX: UN PRIMO ESEMPIO

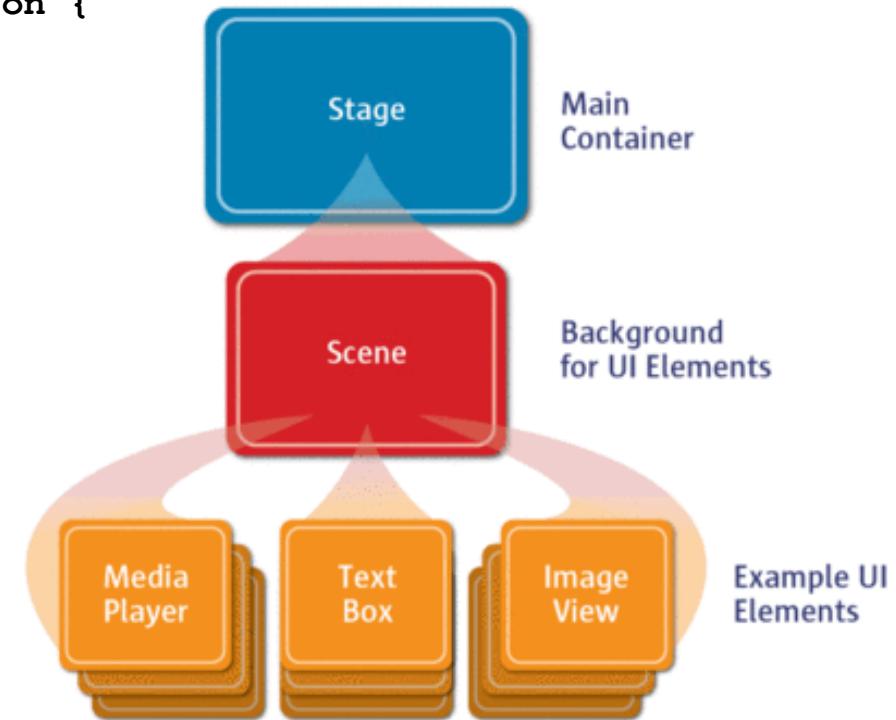
```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.stage.Stage;
```

All'inizio, implementare con
Notepad + riga di comando

```
public class EsJavaFX00 extends Application {  
    public void start(Stage stage){  
        stage.setTitle("Esempio 0");  
        stage.show();  
    }  
}
```



- Scena vuota
- Stage di default, di dimensioni prefissate, a centro schermo





JAVAFX: UN PRIMO ESEMPIO

```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class EsJavaFX00 extends Application {  
    public void start(Stage stage){  
        stage.setTitle("Esempio 0");  
        stage.show();  
    }  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

All'inizio, implementare con
Notepad + riga di comando

- In JavaFX, il compilatore JDK aggiunge automaticamente il main di default, che richiama il metodo ereditato *launch*
- **In Eclipse ciò avviene solo se il progetto è del tipo JavaFX project, previa installazione del plugin**
- Se si fa un progetto Java standard, occorre scrivere a mano questo *main* e aggiungere la libreria JavaFX



JAVAFX: UN PRIMO ESEMPIO

The screenshot shows the Eclipse IDE interface. On the left, the Package Explorer view displays a project structure with a file named `EsJavaFX04.java` under the `src` folder of the `Prova2` project. A red box highlights the `JavaFX SDK` entry in the build path. On the right, the code editor shows the `EsJavaFX04.java` file with Java code for creating a JavaFX application. A red box highlights the main method `public static void main(String[] args)`. Below the code editor, the Java Build Path dialog is open, showing the `Libraries` tab. It lists the `JavaFX SDK` and `JRE System Library [JavaSE-1.8]`. A red box highlights the `Add Library...` button. The properties for the `Prova2` project are also visible on the far left.

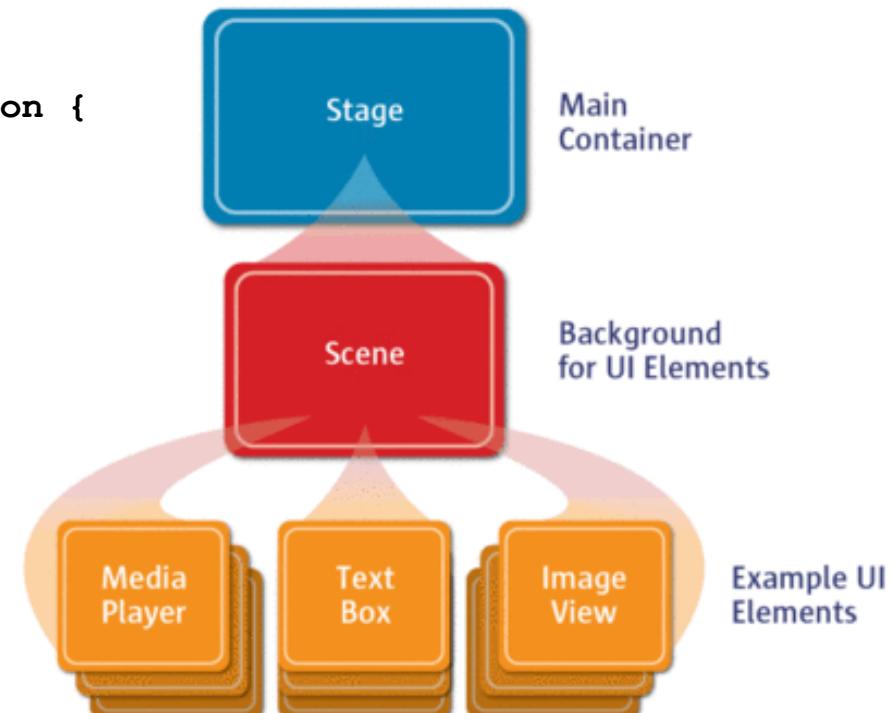
```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.FlowPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.stage.Stage;

public class EsJavaFX04 extends Application {
    public void start(Stage stage){
        stage.setTitle("Esempio 4");
        FlowPane panel = new FlowPane();
        panel.setPrefSize(200,130);
        Canvas canvas = new Canvas(150, 130);
        panel.getChildren().add(canvas);
        GraphicsContext g = canvas.getGraphicsContext2D();
        g.setFont(Font.font("Serif", FontWeight.BOLD, 20));
        g.setFill(Color.RED); g.fillRect(20,20, 100,80);
        g.setFill(Color.BLUE); g.strokeRect(30,30, 80,60);
        g.setFill(Color.BLACK); g.fillText("ciao",50,60);
        Scene scene = new Scene(panel);
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```



PRIMO ESEMPIO (variante)

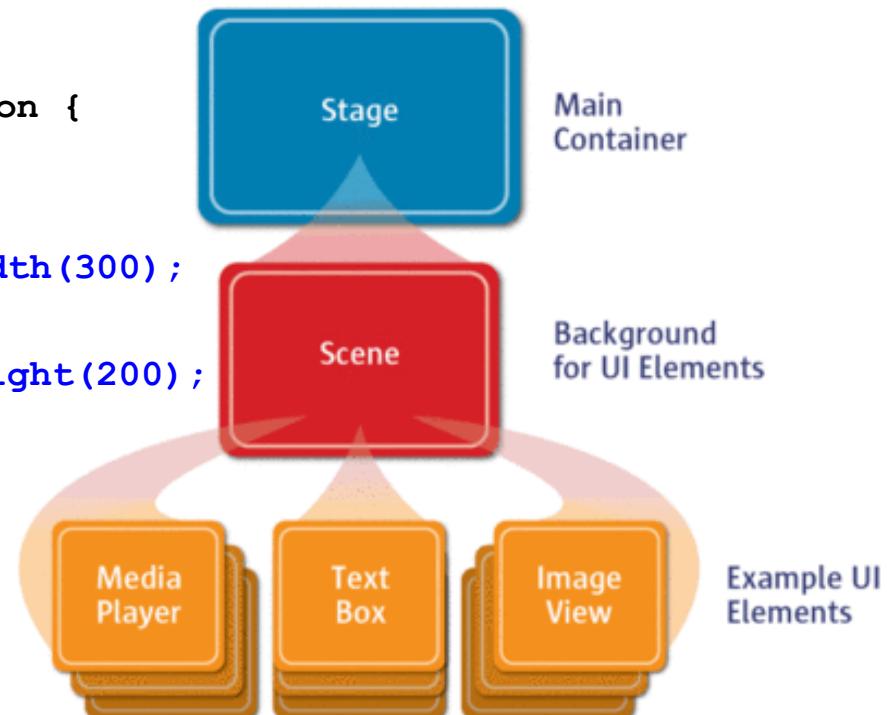
```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class EsJavaFX01 extends Application {  
    public void start(Stage stage){  
        stage.setTitle("Esempio 1");  
        stage.setWidth(200);  
        stage.setHeight(100);  
        stage.setX(300);  
        stage.setY(300);  
        stage.show();  
    }  
}
```



- Scena sempre vuota
- Stage dimensionato e posizionato come desiderato

PRIMO ESEMPIO (variante)

```
import javafx.application.Application;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
  
public class EsJavaFX02 extends Application {  
    public void start(Stage stage){  
        stage.setTitle("Esempio 2");  
        stage.setMinWidth(100); stage.setMaxWidth(300);  
        stage.setWidth(200);  
        stage.setMinHeight(50); stage.setMaxHeight(200);  
        stage.setHeight(100);  
        stage.show();  
    }  
}
```



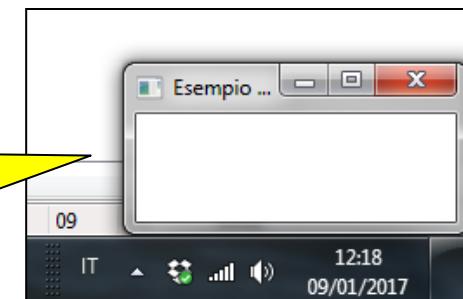
- Stage dimensionato e posizionato come desiderato
- Stage ridimensionabile, ma solo entro i limiti dati

PRIMO ESEMPIO (ulteriore variante)

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class EsJavaFX02bis extends Application {
    public void start(Stage stage) {
        stage.setTitle("Esempio 2");
        stage.setWidth(200);
        stage.setHeight(100);
        javafx.geometry.Rectangle2D screen = Screen.getPrimary().getVisualBounds();
        System.out.println(screen); // per me: 1920 x 1040
        stage.setX(screen.getMinX() + screen.getWidth() - stage.getWidth());
        stage.setY(screen.getMinY() + screen.getHeight() - stage.getHeight());
        stage.show();
    }
}
```

- Si recuperano le dimensioni dello schermo
- Si posiziona lo stage al bordo inferiore destro

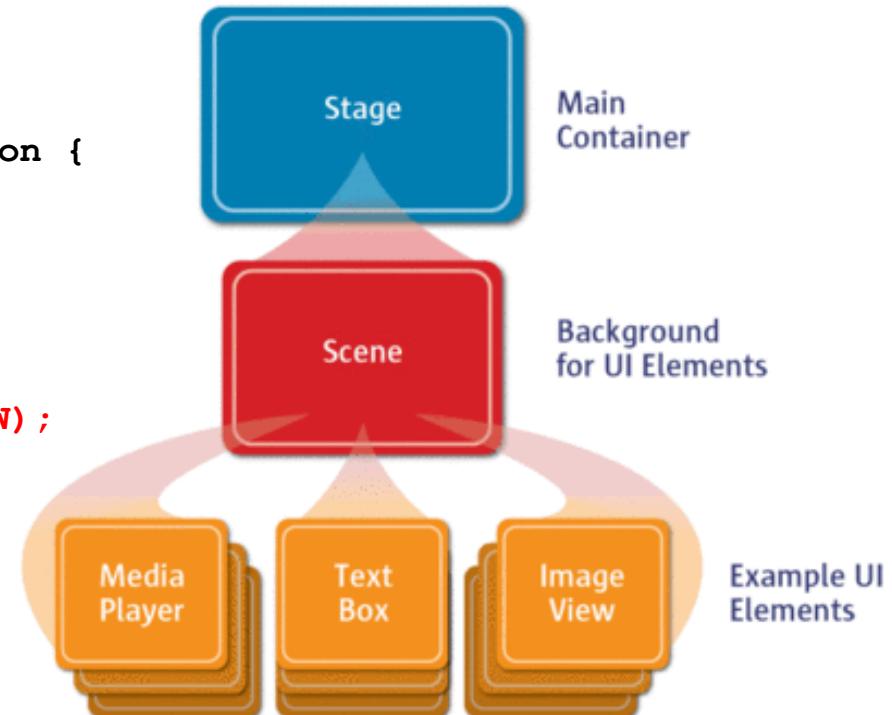




JAVAFX: LA PRIMA SCENA

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class EsJavaFX03 extends Application {
    public void start(Stage stage) {
        stage.setTitle("Esempio 3");
        Pane root = new Pane();
        Scene scene =
            new Scene(root, 300, 50, Color.YELLOW);
        stage.setScene(scene);
        stage.show();
    }
}
```



- Scena gialla, 300x50: occupa tutto lo stage (che quindi non ha dimensione propria)
- Il pannello root (vuoto) tiene tutta la scena



JAVAFX: SCENA con CANVAS

```
public class EsJavaFX04 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 4");  
  
        FlowPane panel = new FlowPane();  
        panel.setPrefSize(200,130);  
  
        Canvas canvas = new Canvas(150, 130);  
        panel.getChildren().add(canvas);  
  
        GraphicsContext g = canvas.getGraphicsContext2D();  
  
        g.setFont(Font.font("Serif", FontWeight.BOLD, 20));  
        g.setFill(Color.RED); g.fillRect(20,20, 100,80);  
        g.setFill(Color.BLUE); g.strokeRect(30,30, 80,60);  
        g.setFill(Color.BLACK); g.fillText("ciao",50,60);  
  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

Pannello di tipo **FlowPane** (a flusso): dispone i suoi componenti da sinistra a destra e dall'alto in basso

Canvas: una tela per grafica a pixel

Aggiunta del **Canvas** alla lista dei figli del pannello

Dal **Canvas** si recupera l'entità (contesto grafico) che serve per disegnare.

Tramite il **GraphicsContext** ricevuto si pilota il disegno, usando i metodi **strokeXXX** (figure con solo contorno) e **fillXXX** (figure piene)





JAVAFX: SCENA con CANVAS

```
public class EsJavaFX04 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 4");  
  
        FlowPane panel = new FlowPane();  
        panel.setPrefSize(200,130);  
  
        Canvas canvas = new Canvas(150, 130);  
        panel.getChildren().add(canvas);  
  
        GraphicsContext g = canvas.getGraphicsContext2D();  
  
        g.setFont(Font.font("Serif", FontWeight.BOLD, 20));  
        g.setFill(Color.RED); g.fillRect(20,20, 100,80);  
        g.setFill(Color.BLUE); g.strokeRect(30,30, 80,60);  
        g.setFill(Color.BLACK); g.fillText("ciao",50,60);  
  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

- Tramite il metodo **setFont** si può impostare il **Font** per le scritte.
- Lo specifico font va creato o con la **new** (sconsigliato) o tramite l'apposita factory internalizzata, **Font.font**

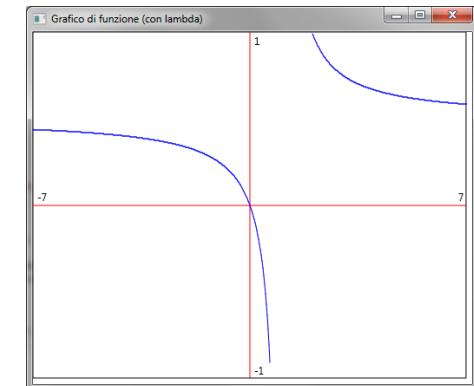
È anche possibile impostare il **Color** da usare per la diverse parti del disegno.





JAVAFX: GRAFICO DI FUNZIONE

```
public class EsJavaFX05 extends Application {  
  
    static Function<Float,Float> f1 =  
        x -> (float) (2.0/Math.PI*Math.sin(1.0*x)+2.0/(3.0*Math.PI)*Math.sin(3.0*x));  
    static Function<Float,Float> f2 = x -> 0.5F*x/(x-1);  
  
    private int xAxisMin=-7, xAxisMax=7, yAxisMin=-1, yAxisMax=1;  
    private int larghezza=500, altezza=400;  
    private float fattoreDiScalaX = larghezza/((float)xAxisMax-xAxisMin);  
    private float fattoreDiScalaY = altezza/((float)yAxisMax-yAxisMin);  
  
    public void start(Stage stage){  
        stage.setTitle("Grafico di funzione");  
        FlowPane panel = new FlowPane();  
        Canvas canvas = new Canvas(528, 448);  
        panel.getChildren().add(canvas);  
        GraphicsContext g = canvas.getGraphicsContext2D();  
        doMyGraphics(g, f3); // LA FUNZIONE DA GRAFICARE  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```



Il metodo di utilità **doMyGraphics**
è definito da noi (v. oltre)



GRAFICO DI FUNZIONE: doMyGraphics

```
private void doMyGraphics(GraphicsContext g, Function<Float,Float> f ) {  
    g.setStroke(Color.BLACK);           // cornice nera  
    g.strokeRect(0,0,larghezza-1,altezza-1);  
    g.setStroke(Color.RED);            // assi cartesiani rossi  
    g.strokeLine(0,altezza/2, larghezza-1,altezza/2);  
    g.strokeLine(larghezza/2,0, larghezza/2,altezza-1);  
    g.fillText(""+xAxisMin, 5,altezza/2-5); // graduazione assi  
    g.fillText(""+xAxisMax, larghezza-10,altezza/2-5);  
    g.fillText(""+yAxisMax, larghezza/2+5,15);  
    g.fillText(""+yAxisMin, larghezza/2+5,altezza-5);  
    g.setStroke(Color.BLUE);           // funzione in blu  
    float fXAxisMin = f.apply(new Float(xAxisMin));  
    float xPrev=xAxisMin, yPrev=fXAxisMin;  
    setPixel(g,xAxisMin,fXAxisMin);  
    for (int ix=1; ix<g.getCanvas().getWidth(); ix++) {  
        float x = xAxisMin+((float)ix)/fattoreDiScalaX;  
        float y = f.apply(x);  
        setPixel(g,x,y); // OPPURE: setLine(g,xPrev,yPrev,x,y); SE f CONTINUA  
        xPrev=x; yPrev=y;  
    }  
}
```

La lambda expression
`f` ha come tipo la
functional interface
`Function<U,V>`

Calcolo di $f(x)$
[notare che `f` è una
lambda expression]

I metodi di utilità `setPixel` e `setLine`
sono definiti da noi (v. oltre)



GRAFICO DI FUNZIONE: setPixel / setLine

```
void setPixel(GraphicsContext g, float x, float y){  
    if (x<xAxisMin || x>xAxisMax || y<yAxisMin || y>yAxisMax ) return;  
    int ix = Math.round((x-xAxisMin)*fattoreDiScalaX);  
    int iy = altezza-Math.round((y-yAxisMin)*fattoreDiScalaY);  
    g.strokeLine(ix,iy,ix,iy);  
}
```

Cosa succede togliendola?

Trasformazione di coordinate e accensione singolo pixel (ix,iy)

```
void setLine(GraphicsContext g, float xP, float yP, float x, float y){  
    if (x<xAxisMin || x>xAxisMax || y<yAxisMin || y>yAxisMax ||  
        xP<xAxisMin || xP>xAxisMax || yP<yAxisMin || yP>yAxisMax ) return;  
    int ix = Math.round((x-xAxisMin)*fattoreDiScalaX);  
    int iy = altezza-Math.round((y-yAxisMin)*fattoreDiScalaY);  
    int ixP = Math.round((xP-xAxisMin)*fattoreDiScalaX);  
    int iyP = altezza-Math.round((yP-yAxisMin)*fattoreDiScalaY);  
    g.strokeLine(ixP,iyP,ix,iy);  
}
```

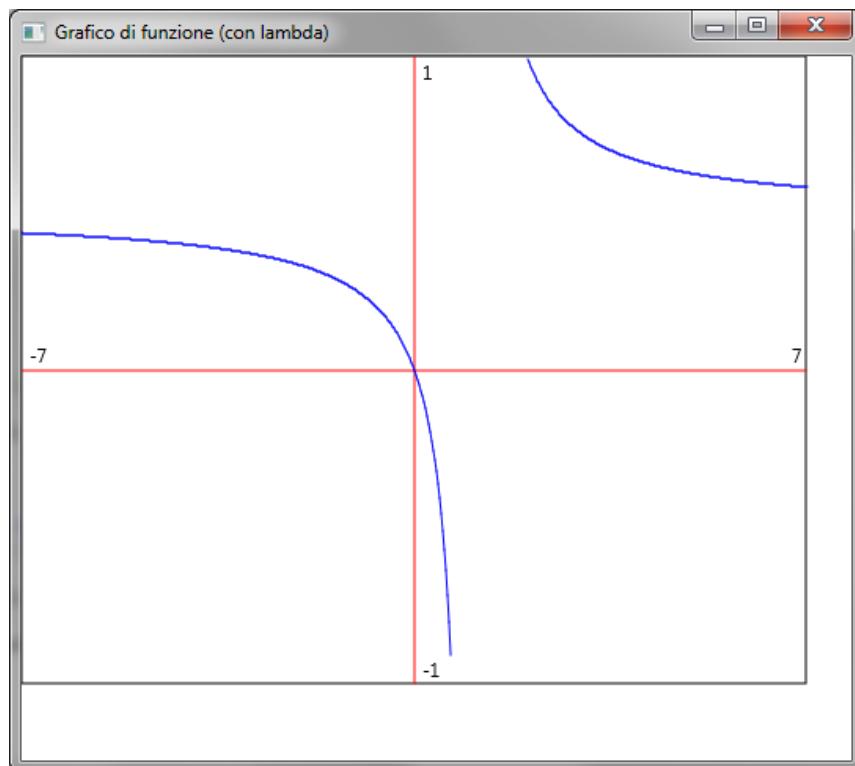
Cosa succede togliendola, se f è discontinua?

Trasformazione di coordinate e disegno segmento dal pixel attuale (ix,iy) al pixel precedente (ixP, iyP)

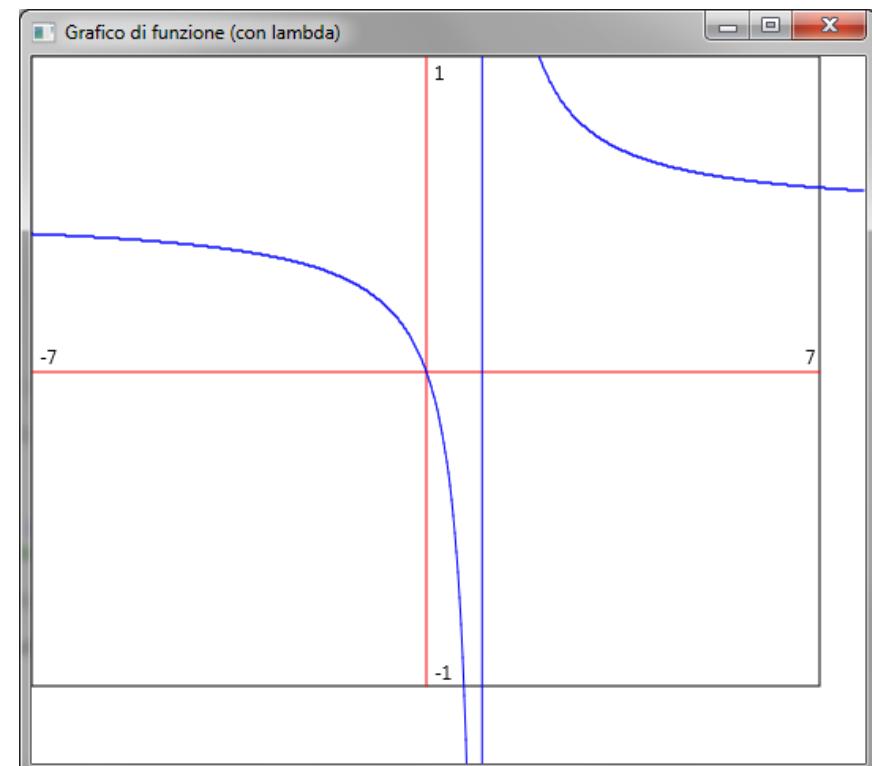


GRAFICO DI FUNZIONE con funzioni discontinue

setLine con controllo



setLine **senza** controllo



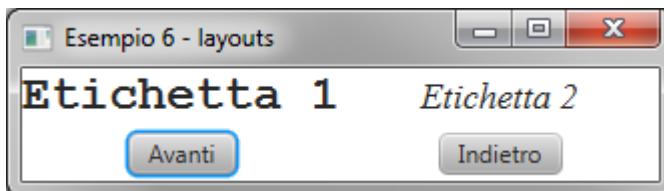


STYLESTHEET

- Per personalizzare il "look" della grafica, JavaFX offre gli *stylesheet*
 - ne sono forniti due, CASPIAN e MODENA (default)
 - metodo `setUserAgentStylesheet` della classe base `Application`
 - altri possono essere definiti dall'utente
 - estendendo quelli preesistenti o impostando un CSS
 - CSS da file: `scene.getStylesheets().add("path/my.css");`
 - CSS da web: `scene.getStylesheets().add("http://.../my.css");`

Caspian

```
setUserAgentStylesheet(STYLESHEET_CASPIAN)
```



Modena

```
setUserAgentStylesheet(STYLESHEET_MODENA)
```

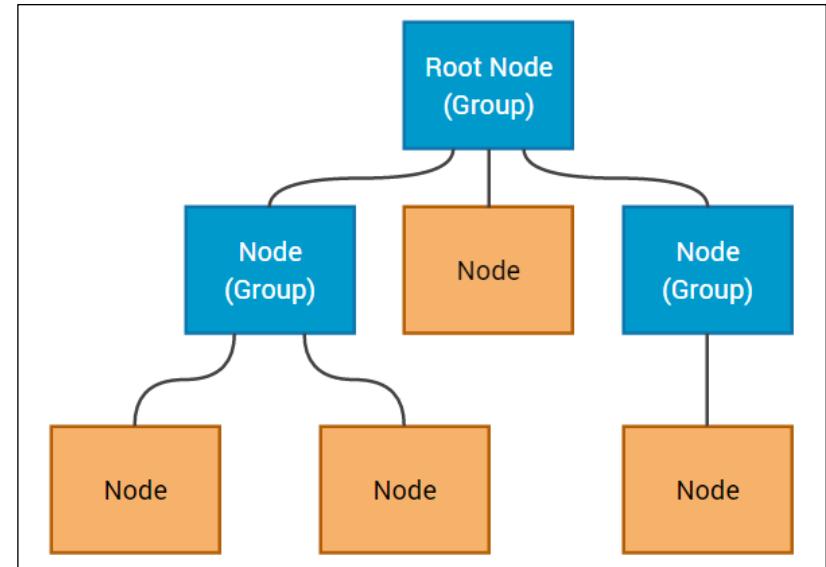




Grafica a componenti (senza gestione di eventi)

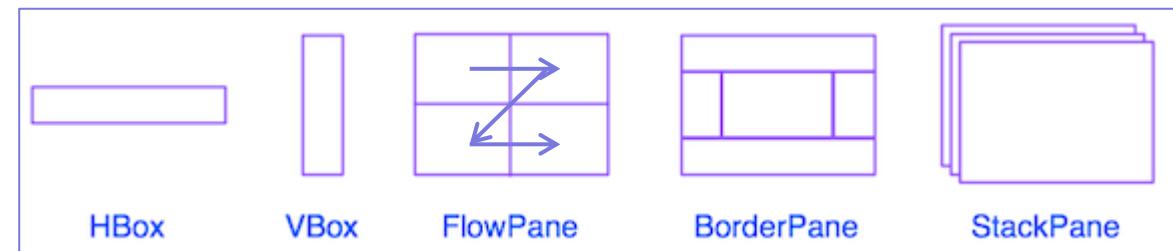
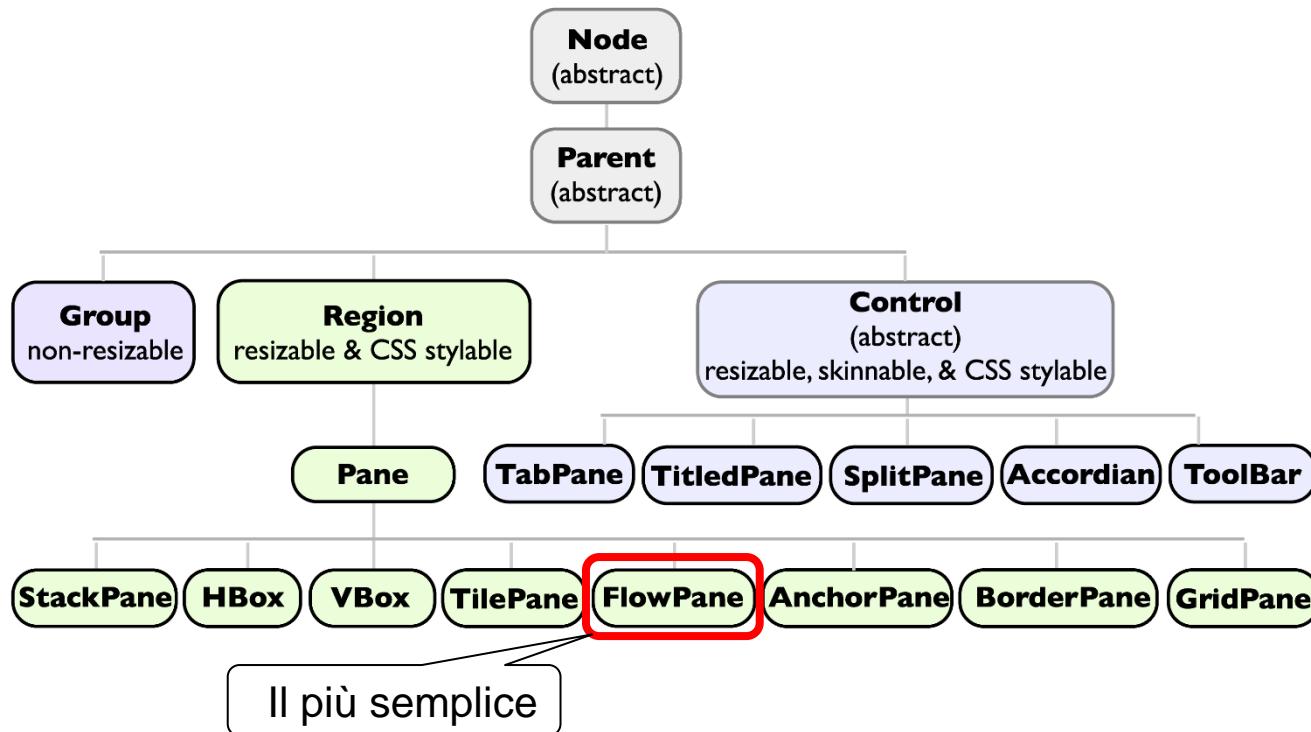
APPLICAZIONI JAVAFX

- Strutturalmente, un'applicazione JavaFX è fatta di *nodi*
- Il *pannello* inserito nella scena è il *nodo radice* (*root node*)
 - ogni altro componente aggiunto al pannello è un nodo-figlio
 - alcuni nodi possono essere *gruppi di altri nodi* (altri pannelli)
- Il pannello dispone i propri componenti secondo un certo *layout*
 - non si usano coordinate assolute, perché nei vari device fisici le finestre hanno dimensioni diverse, si ridimensionano, possono ruotare...
 - ogni pannello incorpora un *gestore di layout* che colloca i vari widget secondo una ben precisa *politica*, così da garantire l'adattamento *automatico* anche se finestra e pannelli vengono ruotati, ridimensionati, etc.





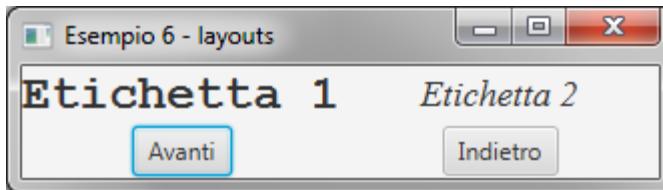
LAYOUT PANES





LAYOUT PANES: CONFRONTO

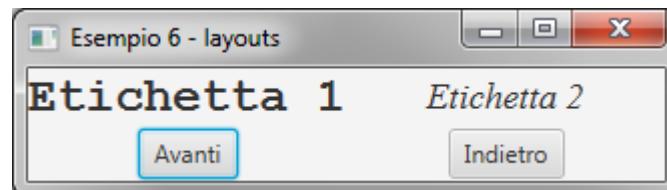
FlowPane



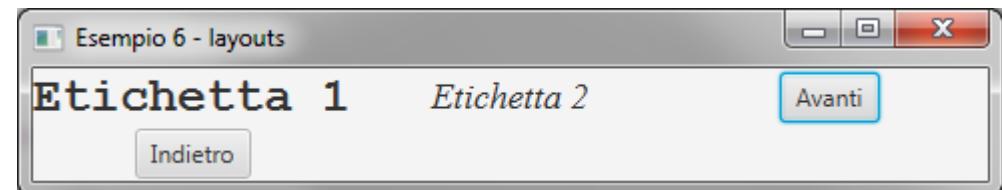
HBox



TilePane a due colonne...



... e a tre colonne



BorderPane



GridPane (griglia 3x3)



VBox





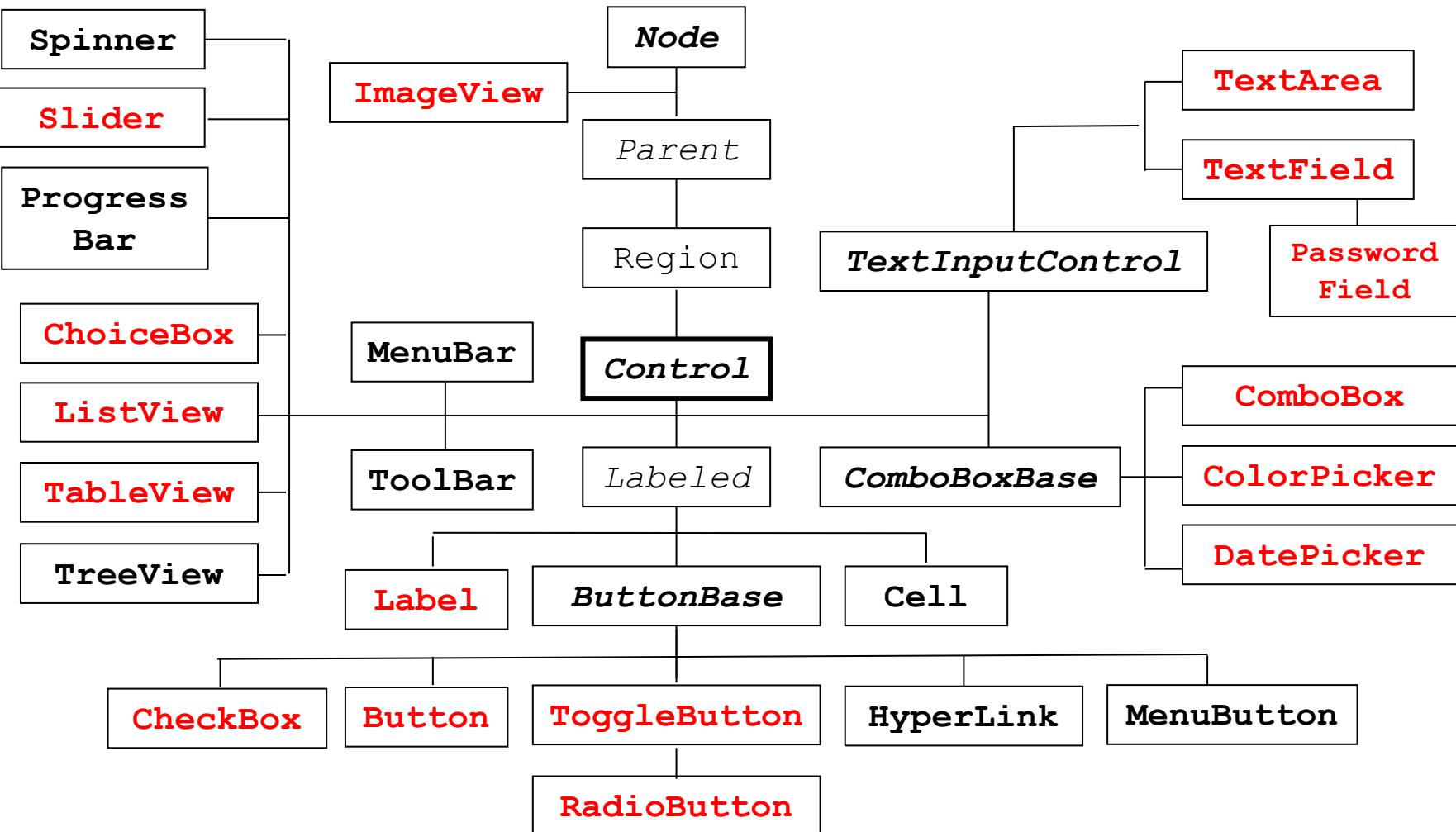
CONTROLLI JAVAFX: PANORAMICA

- *Bottoni*
 - pulsante, interruttore on/off, casella di opzione, bottone radio
- *Componenti di testo*
 - etichetta*, campo a riga singola, password, area di testo a più righe
 - (*) unico componente di solo output, non interattivo
- *Liste e tavelle*
 - lista classica, casella pop-up, casella combinata, tabella di valori
- *Regolazioni*
 - barra di regolazione, selettore di date, selettore di colore
- *Dialoghi*
 - finestra di input, conferma, warning, apertura / salvataggio file
- *Grafici*
 - a linee, istogrammi, a torta..



GERARCHIA DEI CONTROLLI

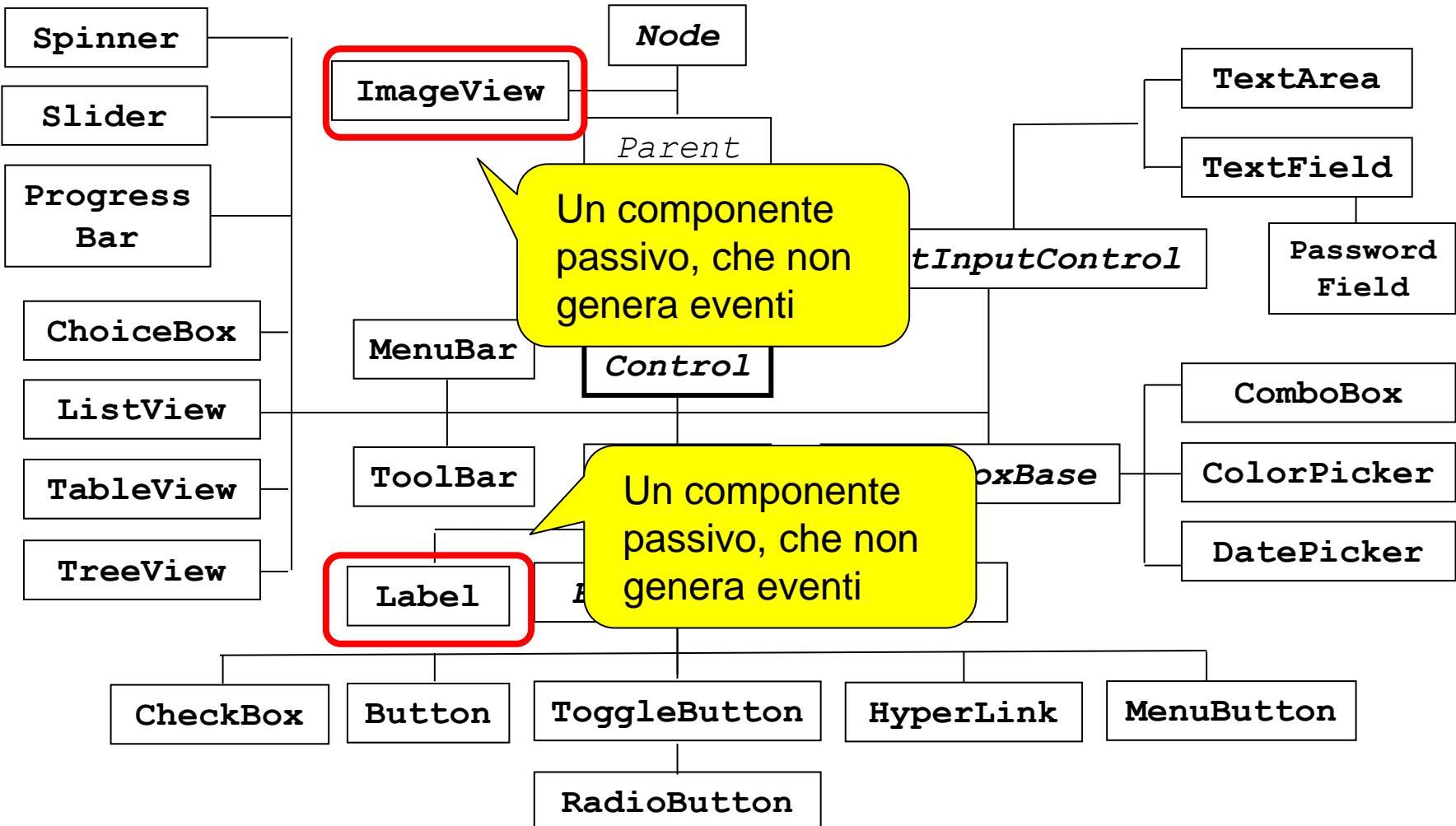
`javafx.scene.control`





GERARCHIA DEI CONTROLLI

`javafx.scene.control`





IL COMPONENTE Label

```
public class EsJavaFX06 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 6");  
        FlowPane panel = new FlowPane();  
        Label l1 = new Label("Etichetta 1");  
        Label l2 = new Label("Etichetta 2");  
        l1.setFont(Font.font("Courier New", FontWeight.BOLD, 24));  
        l2.setFont(Font.font("Times New Roman", FontPosture.ITALIC, 18));  
        panel.getChildren().addAll(l1,l2);  
        Scene scene = new Scene(panel,Color.WHITE);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

...ma si può ridimensionarlo

NB: aggiungere le opportune import

Due Label con Font diverso

Aggiunta delle due Label alla lista dei figli del panel



IL COMPONENTE ImageView

```
public class EsJavaFX07 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 7");  
        BorderPane panel = new BorderPane();  
        panel.setCenter(new ImageView(  
            new Image("alberi.jpg", 400, 300, true, false)));  
        Scene scene = new Scene(panel, Color.WHITE);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

Image

- supporta JPG, BMP, PNG, GIF
- ha vari costruttori (da file, da URL): questo accetta le dimensioni preferite e due boolean (preserva aspetto, effetto smooth)

NB: aggiungere le opportune **import**

Un diverso tipo di pannello

Ogni **ImageView** mostra una **Image**

La **Image** incapsula una immagine (qui, letta da file)



GERARCHIA DEI CONTROLLI

`javafx.scene.control`





COMPONENTI DI TESTO

- Consentono l'immissione di testo
 - a riga singola: **TextField** (e **PasswordField**)
 - su più righe: **TextArea**
- Pilotabili da programma
 - metodi base: **setText** , **getText** , **setEditable**
 - per **TextArea**: **appendText**, **setPrefColumnCount**
- Generano opportuni eventi se
 - si preme ENTER sulla tastiera (tranne **TextArea**)
 - cambia il testo contenuto
- .. ma non è detto che servano sempre!

Ricordate **GraphicApplication** ..? 😊



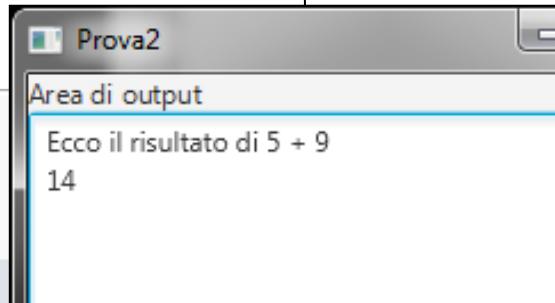
GraphicApplication

Una applicazione grafica con un'area di testo

ed.utils

Class GraphicApplication

Method Summary	
All Methods	Instance Methods
Modifier and Type	Method and Description
void	main() Contiene la logica dell'applicazione.
void	print(java.lang.String txt) Mostra la stringa data nell'area di output della finestra.
void	setTitolo(java.lang.String titolo) Imposta il titolo della finestra.



Ora siamo in grado di capire com'è realizzata



GraphicApplication (1/2)

```
public class GraphicApplication extends Application {  
    protected TextArea ta;  
    private Stage primaryStage;  
    private String titolo;  
  
    public void start(Stage stage) {  
        this.primaryStage=stage;  
        stage.setTitle(titolo);  
        Label l = new Label("Area di output");  
        ta = new TextArea();  
        ta.setPrefColumnCount(25);  
        ta.setEditable(false);  
        FlowPane panel = new FlowPane();  
        panel.getChildren().addAll(l,ta);  
        stage.setScene(new Scene(panel, 320, 220));  
        stage.show();  
        main();  
    }  
    ...
```

Imposta la dimensione
preferita in termini di colonne



GraphicApplication (2/2)

...

// METODI OFFERTI ALL'UTENTE

```
public void main() {
    ta.setText("Welcome");
}
```

Implementazione di default: è destinato a essere sovrascritto dalla logica dell'utente

```
public void setTitolo(String titolo) {
    this.titolo=titolo;
    primaryStage.setTitle(titolo);
}
```

```
public void print(String txt) {
    ta.appendText(txt);
}
}
```

OSSERVA: il contenuto dell'area è *pilotato in toto* da programma.
Non ci sono "reazioni a eventi grafici" dell'utente.



Grafica a componenti con gestione di eventi



EVENT-DRIVEN PROGRAMMING

- I controlli grafici *generano eventi* quando:
 - si compie una qualche *azione* su di essi
 - talune loro *proprietà* cambiano
- Occorre stabilire:
 - *chi* debba essere informato di tali accadimenti
 - *come reagire* a tali accadimenti
- La *parte strutturale* dell'applicazione stabilisce *chi* sia *notificato* dell'accaduto.
- La *parte comportamentale* stabilisce invece *come reagire* all'accaduto.



ARCHITETTURA

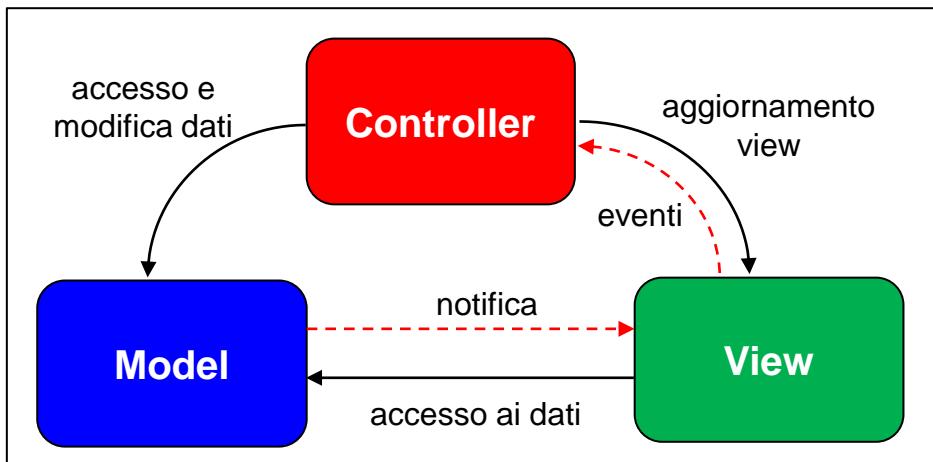
- Ogni framework grafico è progettato secondo un *ben preciso pattern* che implica una ben precisa *architettura* per le relative applicazioni
 - OBIETTIVO 1: massimo disaccoppiamento
 - OBIETTIVO 2: massima *testability*
- Entità fondamentali in gioco
 - il **modello dei dati**: i dati gestiti
 - la/e **vista/e grafica/e** mostrata/e
 - il burattinaio che li controlla e "incolla"
- Approcci tipici: MVC, MVVM (,MVP)

Model

View

Controller
(ViewModel)

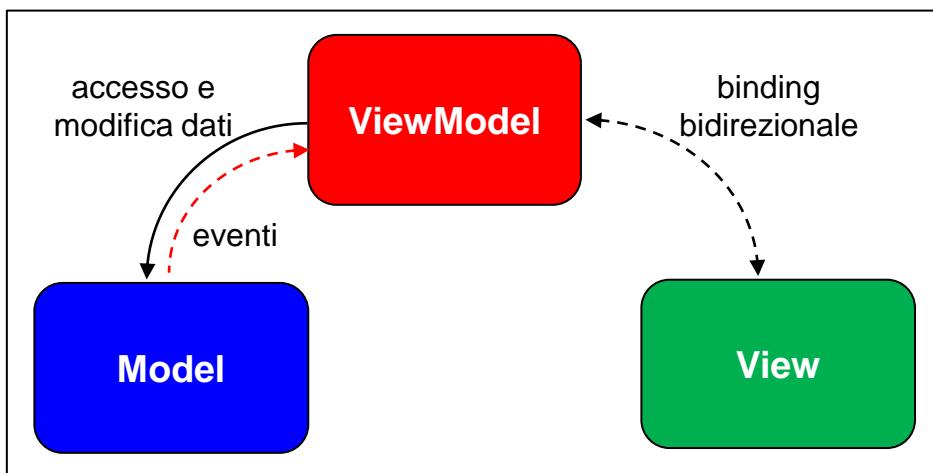
ARCHITETTURA



Schema

Model-View-Controller

- introduce controller esplicito
- disaccoppiamento fra Model e View non totale



Schema

Model-View-ViewModel

- disaccoppia Model e View
- **JavaFX**

Legenda

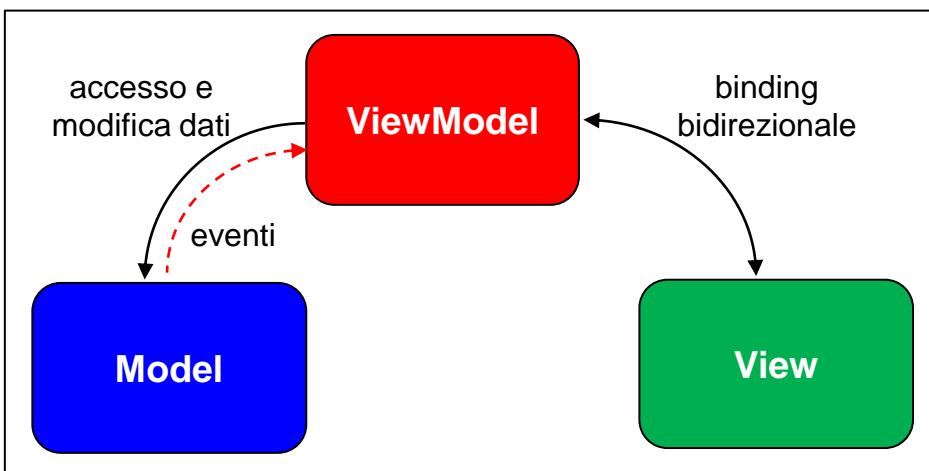
→ associazioni dirette
→ associazioni indirette

ARCHITETTURA

Lo schema è adottato dal framework

- non significa che NOI dobbiamo per forza fare tre classi...
- ..ma dobbiamo capire come LUI è organizzato, per capirne i metodi e la logica di funzionamento

Nelle nostre applicazioni, potremo anche usare MVC



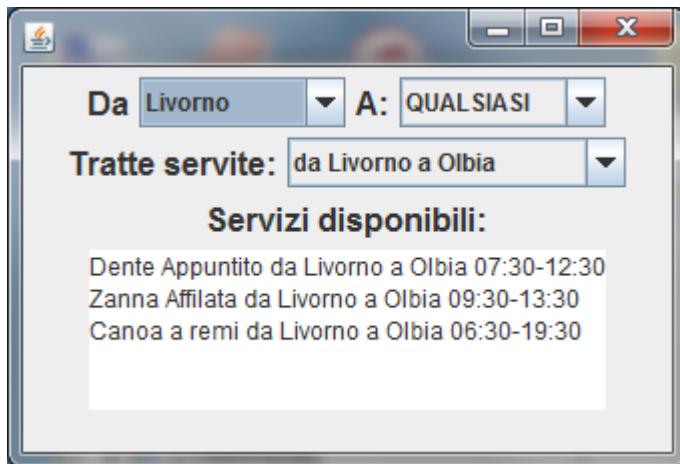
Schema

Model-View-ViewModel

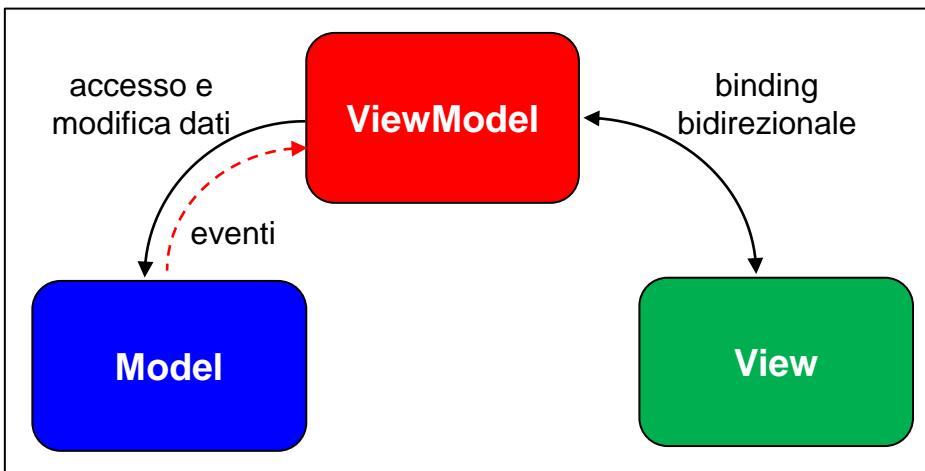
- disaccoppia totalmente Model e View
- **JavaFX**



ESEMPIO: TeethFerries



- Il *model* rappresenta i *porti*, le *tratte* e i *servizi* svolti.
- La *view* è la finestra grafica coi controlli che vedete.
- Il "*burattinaio*" governa *l'interazione*: su richiesta della view, cerca i servizi, fornisce l'elenco delle tratte, etc.



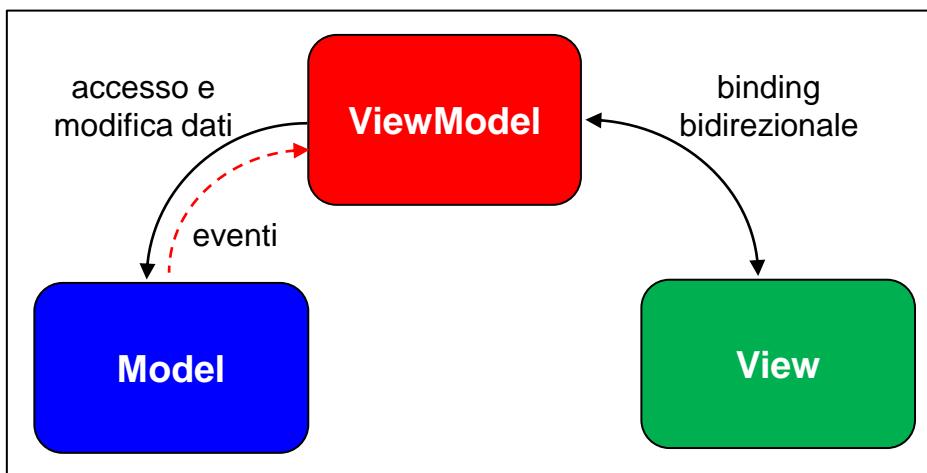
Schema
Model-View-ViewModel

- disaccoppia totalmente Model e View
- **JavaFX**

EVENTI & GESTIONE

Gli *eventi* sono il modo di trasmettere informazione fra i widget e il "burattinaio"

- Quando l'utente agisce sul widget, questo genera un *evento*
- L'evento è recapitato *solo al componente preventivamente collegato a esso – l'ascoltatore (listener) dell'evento*



Schema
Model-View-ViewModel

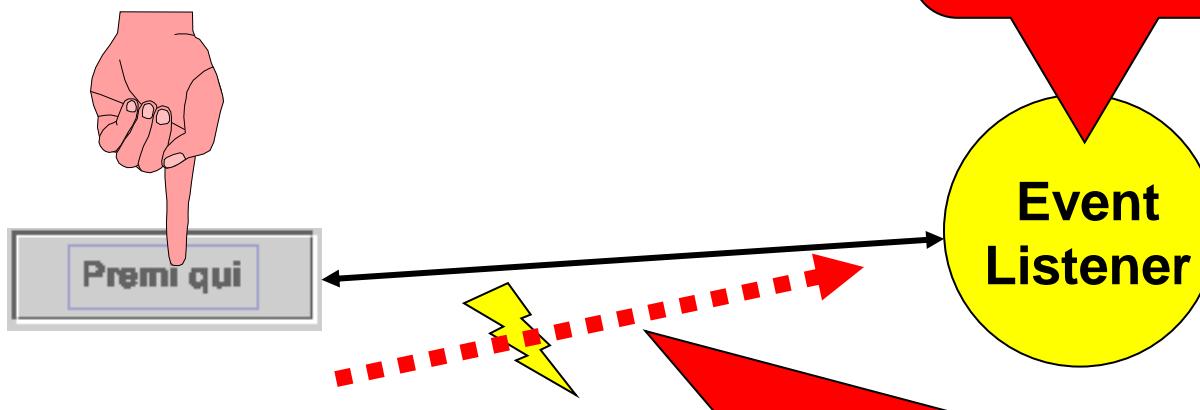
- disaccoppia totalmente Model e View
- **JavaFX**



EVENTI & GESTIONE

Ogni componente viene associato a un **ascoltatore degli eventi** (un oggetto che implementa una interfaccia **Listener**)

2. L'ascoltatore riceve e gestisce l'evento – tipicamente, invocando metodi del burattinaio



L'attività non è più algoritmica (Read-Eval-Print Loop): è **interattiva e reattiva**.

1. Quando si agisce sul componente (ovvero, qui, si preme il pulsante) si genera un evento che è inviato all'ascoltatore



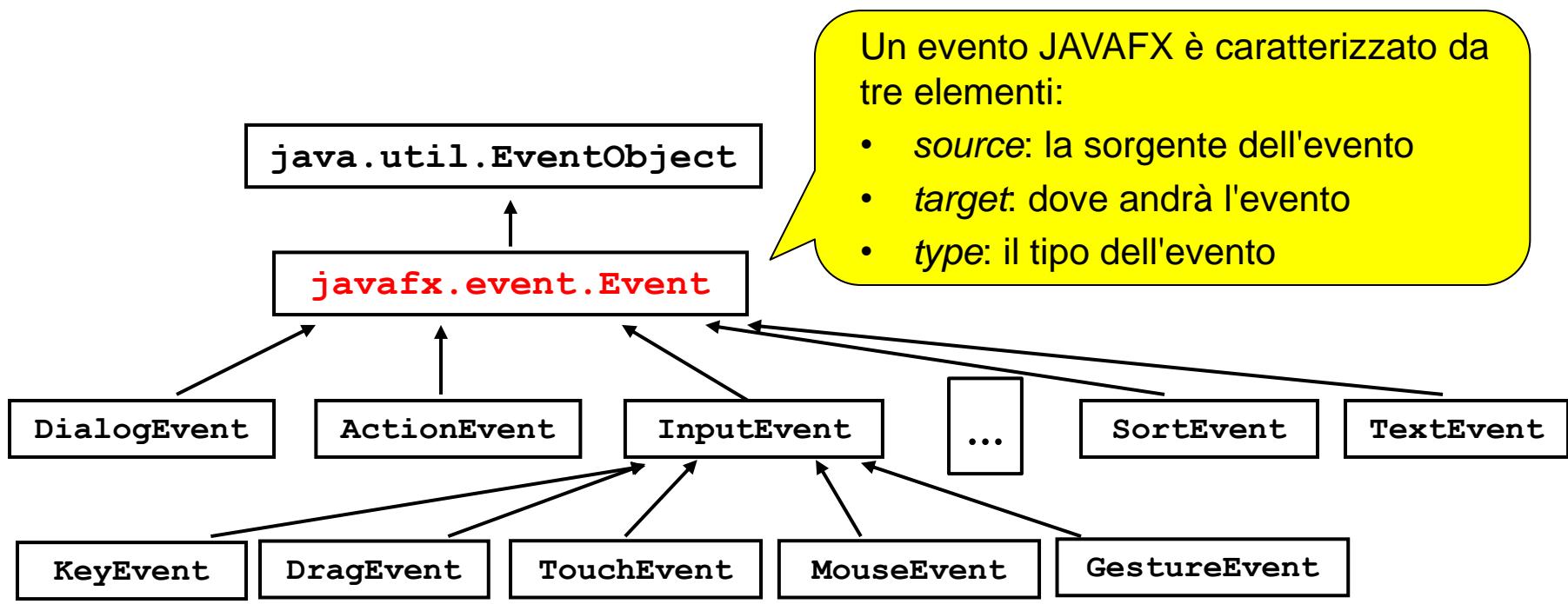
TIPI DI EVENTI

Il **tipo** dell'evento descrive **cosa** è accaduto

- un **bottone** premuto genera l'evento “*azione*”
 - una **lista di voci** genera l'evento “*item selezionato*”
 - un **campo di testo** genera l'evento “*azione*” se viene premuto ENTER sulla tastiera (ma non solo quello..)
- In situazioni diverse, lo stesso widget può generare **eventi diversi**
 - ad esempio, il bottone radio genera tre tipi di eventi: *l'azione*, *l'item selezionato* e il *cambiamento*
 - si sfrutta il più appropriato al caso concreto.

EVENTI IN JAVAFX

Un **evento** è un oggetto, istanza di una qualche sottoclasse di `java.util.EventObject`

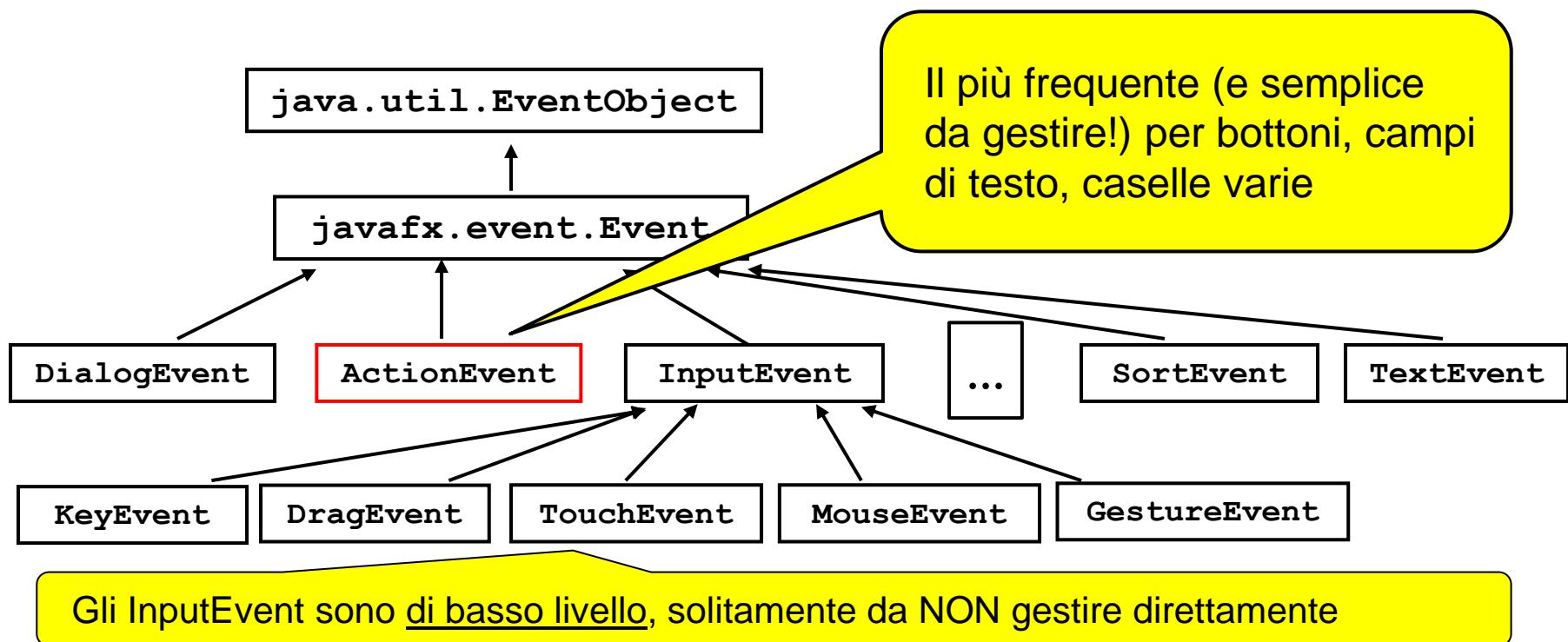


..e molti, molti altri.



EVENTI IN JAVAFX

Un **evento** è un oggetto, istanza di una qualche sottoclasse di `java.util.EventObject`





IL COMPONENTE Button

- Un **Button** premuto **genera un *evento di azione***, ossia un'opportuna istanza di **ActionEvent**
 - se nessun ascoltatore è stato preventivamente associato a quel bottone, l'evento andrà perso e non accadrà nulla
- L'evento **viene notificato all'ascoltatore** che era stato in precedenza **associato a quel bottone**
 - l'ascoltatore dev'essere del giusto tipo
 - il tipo è espresso dall'interfaccia **EventHandler<ActionEvent>** che dichiara il metodo **void handle(ActionEvent ev)**
 - l'ascoltatore può essere un'istanza di una qualunque classe (anche anonima o lambda) che implementi tale interfaccia



ESEMPIO CON Button

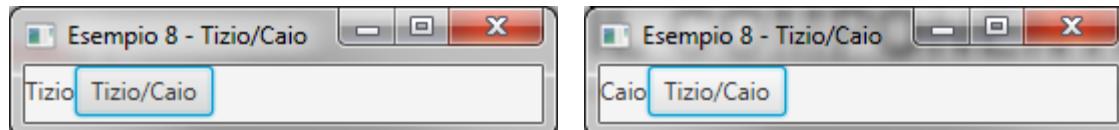
```
public class EsJavaFX08 extends Application {  
    private Label l;  
    public void start(Stage stage){  
        stage.setTitle("Esempio 8");  
        FlowPane panel = new FlowPane();  
        l = new Label("Tizio");  
        Button b = new Button("Tizio/Caio");  
        b.setOnAction(...);  
        b.setTooltip(new Tooltip("Premere per commutare"));  
        panel.getChildren().addAll(l,b);  
        Scene scene = new Scene(panel,Color.WHITE);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

NB: aggiungere le opportune **import**

Estratta da **start** per poterla riusare..

Aggancio al **Button** il suo ascoltatore

L'ascoltatore implementa il metodo **handle**





ESEMPIO CON Button

```
public class EsJavaFX08 extends Application
    implements EventHandler<ActionEvent>{

    private Label l;
    public void start(Stage stage) {
        stage.setTitle("Esempio 8");
        FlowPane panel = new FlowPane();
        l = new Label("Tizio");
        Button b = new Button("Tizio/Caio");
        b.setOnAction(this);
        panel.getChildren().addAll(l,b);
        Scene scene = new Scene(panel,Color.WHITE);
        stage.setScene(scene);
        stage.show();
    }
    public void handle(ActionEvent event) {
        l.setText(l.getText().equals("Tizio") ? "Caio" : "Tizio");
    }
}
```

Qui l'ascoltatore è la classe stessa

Aggancio al **Button** il suo ascoltatore (l'application stessa)

L'ascoltatore gestisce l'evento



VARIANTE: ASCOLTATORE SEPARATO

```
public class EsJavaFX08bis extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 8 bis");  
        FlowPane panel = new FlowPane();  
        Label l = new Label("Tizio");  
        Button b = new Button("Tizio/Caio");  
        b.setOnAction(new Es8bisListener(l));  
        b.setTooltip(new Tooltip("Premere per commutare"));  
        panel.getChildren().addAll(l,b);  
        Scene scene = new Scene(panel,Color.WHITE);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

Ergo, non è più
necessario estrarre
l'etichetta da `start`

L'ascoltatore implementa il metodo `handle`
ma è necessario passargli gli oggetti su cui
deve operare (qui, l'etichetta `l`), perché non
può accedere a essi direttamente

Aggancio al `Button` il
suo ascoltatore

Qui l'ascoltatore è un
oggetto di una classe
diversa



VARIANTE: ASCOLTATORE SEPARATO

```
class Es8bisListener implements EventHandler<ActionEvent> {  
  
    private Label l;  
  
    public Es8bisListener(Label label) {  
        l=label;  
    }  
  
    public void handle(ActionEvent event) {  
        l.setText(l.getText().equals("Tizio") ? "Caio" : "Tizio");  
    }  
}
```

L'ascoltatore separato

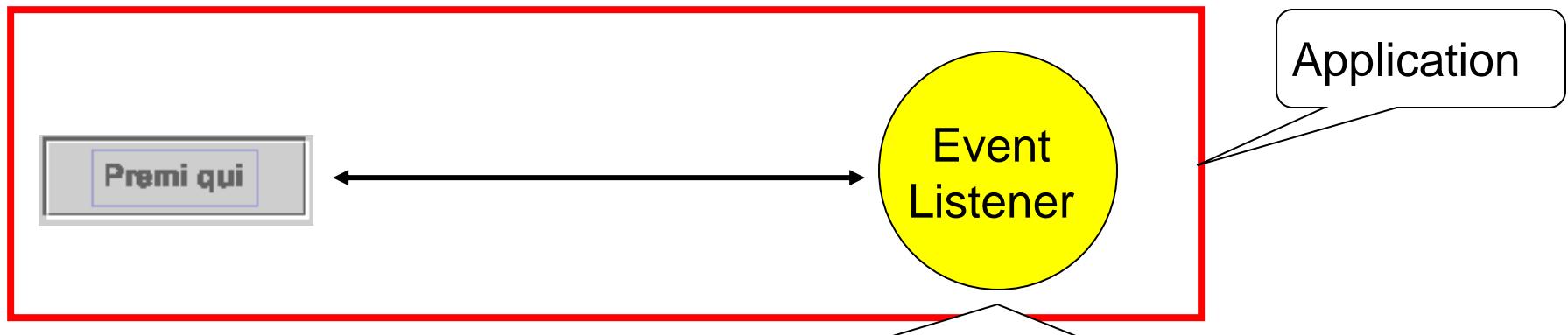
Occorre un costruttore che memorizzi il riferimento alla etichetta su cui deve operare

L'ascoltatore implementa il metodo **handle** (identico a prima)

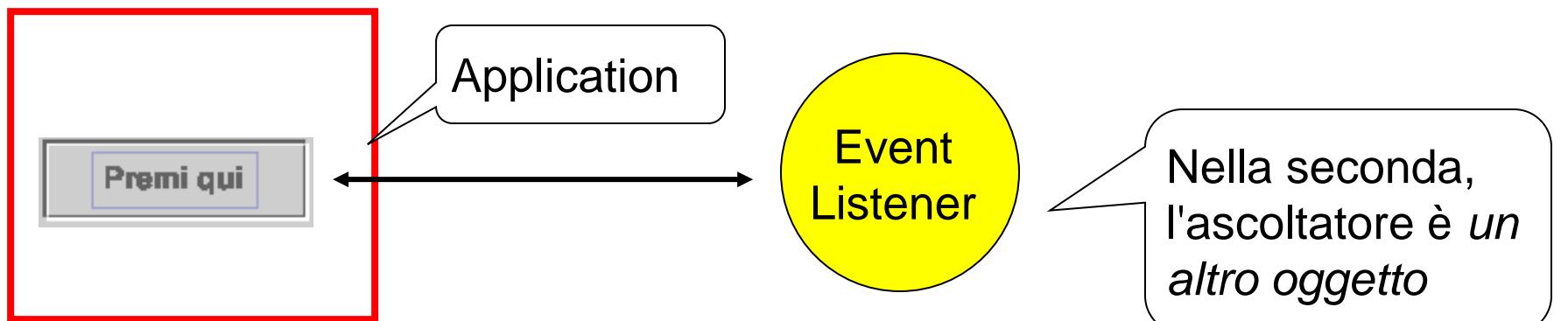




ARCHITETTURE A CONFRONTO



Nella prima versione, l'ascoltatore è *l'application stessa*





ULTERIORE VARIANTE: ASCOLTATORE COME LAMBDA EXPRESSION

L'ascoltatore può essere implementato come *lambda expression* (che sostituisce una classe esterna anonima)

```
public class EsJavaFX08ter extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 8 ter");  
        FlowPane panel = new FlowPane();  
        Label l = new Label("Tizio");  
        Button b = new Button("Tizio/Caio");  
        b.setOnAction(...);  
        b.setTooltip(new Tooltip("Premere per commutare"));  
        panel.getChildren().addAll(l, b);  
        Scene sc = new Scene(panel, 300, 250);  
        stage.setScene(sc);  
        stage.show();  
    }  
}
```

- Il metodo **handle** non è più citato direttamente: *diventa anonimo*, sussunto dalla lambda expression
- RICORDA: la lambda è *tutto ciò che resta* del metodo



ULTERIORE VARIANTE: ASCOLTATORE COME LAMBDA EXPRESSION

L'ascoltatore può essere implementato come *lambda expression*

```
public class E {
    public void start(Stage stage) {
        FlowPane panel = new FlowPane();
        Label l = new Label("Ciao");
        Button b = new Button("Premere per continuare");
        b.setOnAction(this::myHandle);
        b.setTooltip(new Tooltip("Premere per continuare"));
        panel.getChildren().addAll(l,b);
        Scene scene = new Scene(panel,Color.WHITE);
        stage.setScene(scene);
        stage.show();
    }
}
```

- Nel caso più semplice, la gestione può consistere nel richiamare un metodo ausiliario (es. `myHandle`)
- La lambda expression assume allora la forma semplificata di *method reference*: `obj::method`
- Se il metodo ausiliario è nella stessa classe, tale forma diviene banalmente `this::myHandle`

NB: l'ascoltatore sembra interno, ma è in realtà **esterno** in quanto istanza di una classe anonima generata dal compilatore.

```
void myHandle(ActionEvent event) {
    l.setText( l.getText().equals("Tizio")
              ? "Caio" : "Tizio");
}
```



ULTERIORE VARIANTE: ASCOLTATORE COME LAMBDA EXPRESSION

Si può anche fare a meno del metodo ausiliario, implementando la *lambda expression* in loco:

```
public class EsJavaFX08ter extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Lambda Expression");  
        FlowPane panel = new FlowPane();  
        Label l1 = new Label("Tizio");  
        Button b = new Button("Tizio/Caio");  
        b.setOnAction(  
            event -> l1.setText(l1.getText().equals("Tizio") ? "Caio" : "Tizio"));  
        b.setTooltip(new Tooltip("Premere per commutare"));  
        panel.getChildren().addAll(l1, b);  
        Scene scene = new Scene(panel, Color.WHITE);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

La variabile locale **event** (che può chiamarsi anche diversamente) è l'unica cosa che ricorda che "in origine" c'era un metodo **handle**.

L'ascoltatore sembra interno, ma è concettualmente esterno in quanto istanza di una classe anonima generata dal compilatore.

La lambda è molto pratica quando un certo ascoltatore serve *una volta sola*; inappropriata in caso contrario



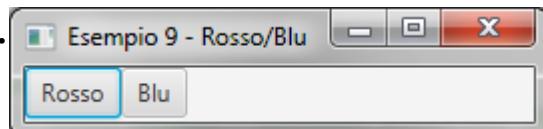
UN ALTRO ESEMPIO APPLICAZIONE "ROSSO/BLU"

```
public class EsJavaFX09bis extends Application {  
  
    private Button b1, b2;  
    private FlowPane panel;  
  
    public void start(Stage stage) {  
  
        stage.setTitle("Esempio 9 - Rosso/Blu");  
        panel = new FlowPane();  
        b1 = new Button("Rosso");  
        b2 = new Button("Azzurro")  
        b1.setOnAction(this::myHandle);  
        b2.setOnAction(this::myHandle);  
        panel.getChildren().addAll(b1,b2);  
        Scene scene = new Scene(panel,Color.WHITE);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

Variabili dichiarate a livello di classe
(non più nel metodo `start`) per
essere visibili anche in `myHandle`

Aggancio a ogni `Button` **lo
stesso ascoltatore** nella
forma di ***method reference***

Qui va implementato il
metodo ausiliario `myHandle`



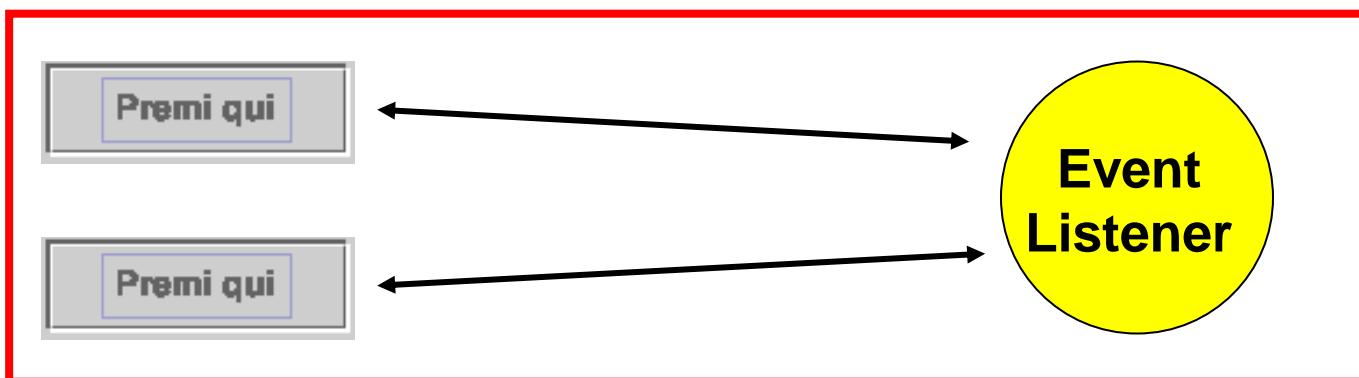


APPLICAZIONE ROSSO/BLU

```
public class EsJavaFX09bis extends Application {  
    ...  
    private void myHandle(ActionEvent event) {  
        if (event.getSource() == b1)  
            panel.setStyle("-fx-background: red;");  
        else  
            panel.setStyle("-fx-background: blue;");  
    }  
}
```

Necessità di *discriminare la sorgente dell'evento*
(spiacevole dover cablare qui il nome della variabile b1)

L'ascoltatore imposta lo stile del pannello (tramite codice HTML / CSS)





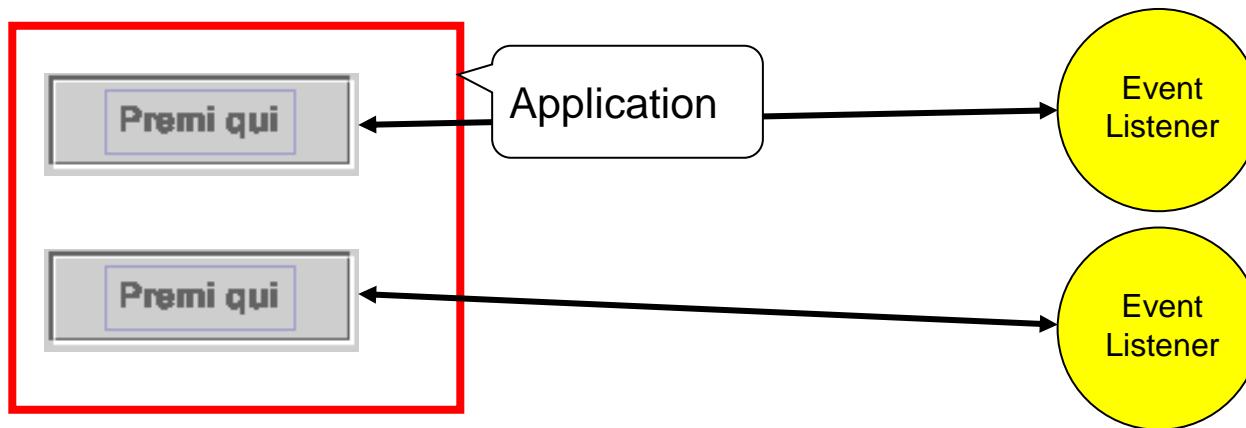
ALTERNATIVA

- Ma fattorizzare la gestione in un listener unico ha senso se le operazioni da fare sono molte e "praticamente uguali"
 - in questo caso, ci ha solo costretto a inserire un `if` per discriminare la sorgente dell'evento
- In alternativa, si può optare per *due listener separati*, ovvero due lambda distinte
 - scompare la necessità di discriminare quale sia il pulsante premuto
 - più semplice usare le lambda expression, poiché ogni lambda equivale a una *istanza singleton* di una *classe anonima diversa* generata dal compilatore
 - non più necessario il metodo ausiliario



APPLICAZIONE "ROSSO/BLU" CON DUE LISTENER SEPARATI

- Applicazione con *due pulsanti* per cambiare il colore di sfondo: un pulsante lo rende rosso, l'altro blu.
- Utilizzando le lambda, è pratico predisporre due ascoltatori diversi* (uno per ogni pulsante) anche se il compito da svolgere è quasi identico
→ si evita di dover *discriminare quale sia* il pulsante premuto



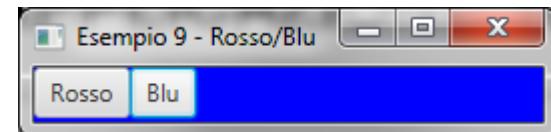
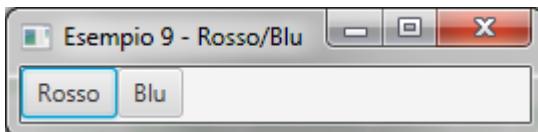


APPLICAZIONE "ROSSO/BLU" CON DUE LISTENER SEPARATI

```
public class EsJavaFX09 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 9 - Rosso/Blu");  
        FlowPane panel = new FlowPane();  
        Button b1 = new Button("Rosso");  
        Button b2 = new Button("Azzurro")  
        b1.setOnAction(event -> panel.setStyle("-fx-background: red;"));  
        b2.setOnAction(event -> panel.setStyle("-fx-background: blue;"));  
        panel.getChildren().addAll(b1,b2);  
        Scene scene = new Scene(panel,Color.WHITE);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

Aggancio a ogni Button
il suo ascoltatore

L'ascoltatore imposta lo
stile del pannello (tramite
codice HTML / CSS)





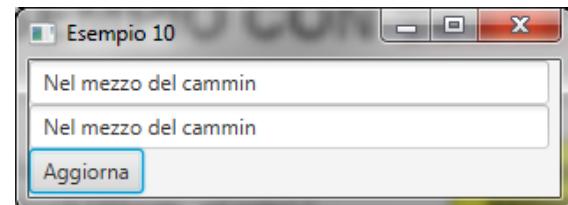
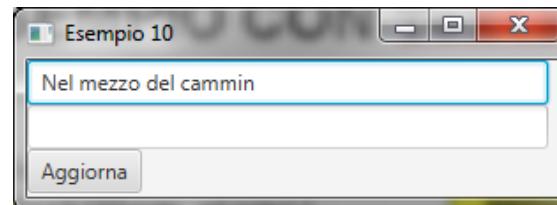
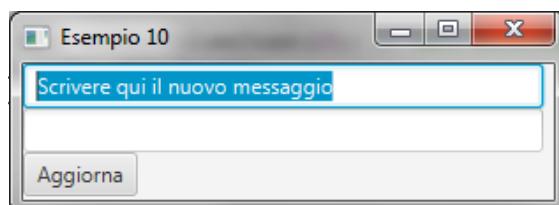
UN ESEMPIO CON TextField (passivo)

```
public class EsJavaFX10 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 10");  
        FlowPane panel = new FlowPane();  
  
        TextField txt1 = new TextField("Scrivere qui il nuovo messaggio");  
        txt1.setPrefColumnCount(25);  
        TextField txt2 = new TextField(); txt2.setEditable(false);  
        txt2.setPrefColumnCount(25);  
        Button b1 = new Button("Aggiorna");  
        b1.setOnAction(event -> txt2.setText(txt1.getText()));  
  
        panel.getChildren().addAll(txt1, txt2, b1);  
        Scene scene = new Scene(panel, Color.WHITE);  
        stage.setScene(scene);  
    }  
}
```

L'unico componente reattivo è sempre il **Button**: i campi di testo sono pilotati da programma (non ci sono ascoltatori su di essi)

Solo output

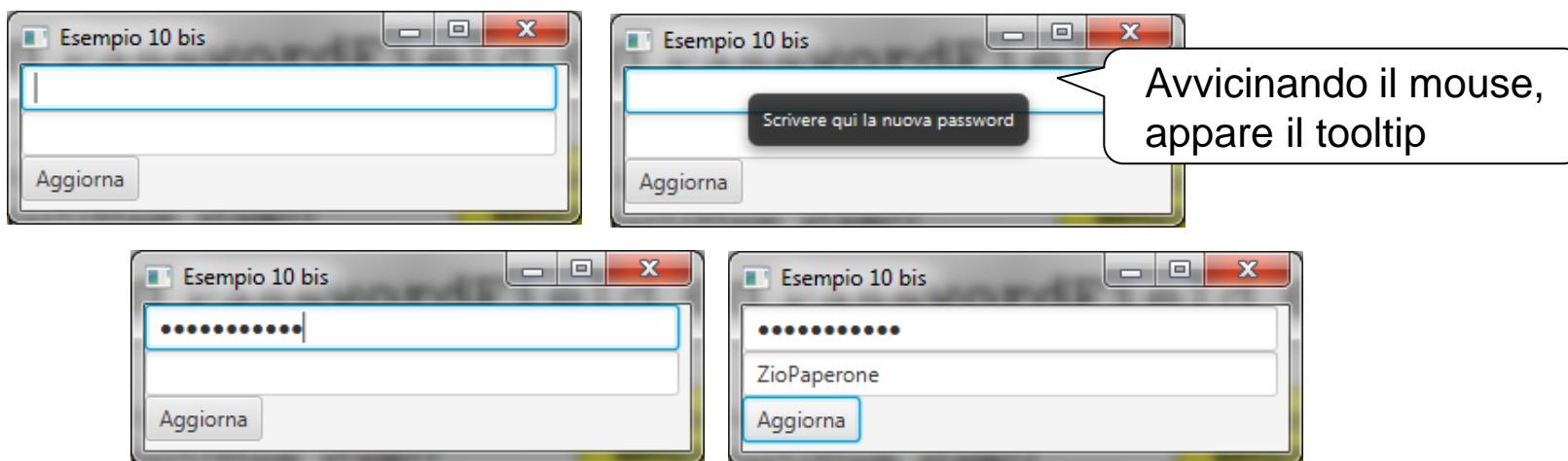
Premendo il pulsante,
il testo sopra è copiato
sotto.





VARIANTE con PasswordField

```
public class EsJavaFX10bis extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 10 bis");  
        FlowPane panel = new FlowPane();  
  
        PasswordField txt1 = new PasswordField(); txt1.setPrefColumnCount(25);  
        txt1.setTooltip(new Tooltip("Scrivere qui la nuova password"));  
        ...  
    }  
}
```





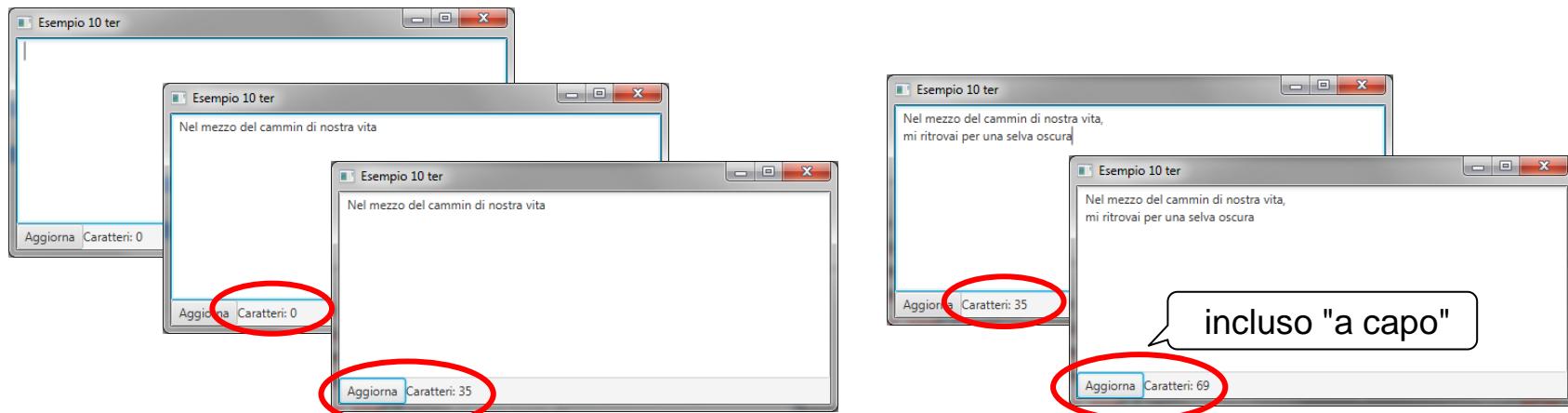
VARIANTE con TextArea

...

```
TextArea area = new TextArea();  
Label label = new Label("Caratteri: 0");  
Button b1 = new Button("Aggiorna");  
  
b1.setOnAction(  
    event -> label.setText("Caratteri: " + area.getText().length()));  
...
```

Label anziché campo di testo non modificabile

La label mostra il numero di caratteri digitati, *ma lo aggiorna solo quando si preme il pulsante*





MVC / MVVM

- Finora il gestore dell'evento è sempre stata l'applicazione stessa o un ente esterno "prescelto"
- Questo però funziona solo per "mini-esempi", con gestione dell'evento altrettanto "mini"
- Quando la logica di gestione dell'evento è complessa e coinvolge molti oggetti, MVC/MVVM suggeriscono che *non è opportuno incapsularla tutta nella GUI*
 - non sarebbe manutenibile
 - non separerebbe chiaramente ambiti e componenti
- I due pattern MVC/MVVM si differenziano per il *modo* e il *livello* in cui gli eventi sono diffusi e percepiti.



MVC

MVC si basa su un *controller esplicito*

- il *gestore dell'evento delega al controller* la gestione vera e propria dei dati
- il *controller* manipola i dati del model (aggiornamento, aggiunta elementi, etc.)

Vantaggio: disaccoppiamento

- se cambia la GUI, non cambia la gestione
- se cambia il modello, la GUI non ne risente
- si possono avere più viste

..come infatti abbiamo già visto in lab in passato!



MVVM

MVVM si basa sull'idea di *proprietà osservabili*, intese come *astrazioni di più alto livello* rispetto ai "semplici eventi"

- unione di una proprietà get/set + evento che scatta a seguito di cambiamenti di stato (set)
- può essere collegata ad altre proprietà, propagando automaticamente i cambiamenti
- può essere agganciata a un *ChangeListener* (ev. lambda)

Vatnaggio: disaccoppiamento con meno "tricky code"

- benefici analoghi a MVC, ma operando a un livello più alto: spesso non si "vedono" neppure direttamente gli eventi
- (...ma talora rende più complicato catturarne il flusso..)



MVC / MVVM

JavaFX permette di adottare entrambi i pattern

- MVC è più conservativo (l'ha usato Swing per 15 anni..)
- MVVM disaccoppia maggiormente, ma richiede di *progettare tutta l'applicazione in termini di proprietà osservabili*
 - scelta da fare fin dall'inizio
 - nei casi di bassa/media complessità, può non valere la pena
- Didatticamente
 - concepire il controller e gestire gli eventi è rilevante
→ main choice: MVC
 - ma lo è anche capire cosa sia una proprietà osservabile
→ cenni di MVVM



MVC IN QUESTO CORSO

Nel seguito del corso adotteremo MVC, però:

- in queste slide **non** ci sarà un controller esplicito, perché l'obiettivo è **presentare i vari widget grafici attraverso mini-esempi didattici**
 - a questo fine è utile che l'esempio sia il più semplice possibile, senza elementi extra (per non confondere)
- nella parte pratica di laboratorio e all'esame invece **ci sarà sempre** un controller esplicito
 - perché il problema è di medio/grande dimensione
 - perché il mondo reale è quello ☺



MVVM IN QUESTO CORSO

Tuttavia, farà capolino anche MVVM

- in queste slide presenteremo il concetto-chiave di *proprietà osservabile* con mini-esempi didattici
 - di base, al di fuori della grafica (per non aggiungere complessità)
 - ma poi, in alcuni esempi, applicati ad alcuni casi in JavaFX
- nella parte pratica di laboratorio e all'esame potrà essere necessario / opportuno gestire alcune proprietà
 - pur mantenendo il modello generale MVC
 - approccio graduale ☺



PROPRIETÀ OSSERVABILI

- Le proprietà osservabili sono *proprietà la cui modifica scatena automaticamente un evento*
 - l'evento potrebbe essere intercettato da un opportuno ascoltatore (un **ChangeListener<T>**)..
 - ..ma può essere anche lasciato agire "sotto traccia", *limitandosi ad agganciare una proprietà ad altre → i cambiamenti si propagano senza dover fare nulla esplicitamente*
- Il concetto di proprietà osservabile è espresso dalla interfaccia **ObservableValue<T>**
 - esistono moltissime proprietà osservabili standard che encapsulano i casi di uso più comune: tipi primitivi, stringhe, liste, read only, ..
 - esistono anche varie sotto-interfacce di **ObservableValue<T>**



PROPRIETÀ OSSERVABILI: GERARCHIA

Interface ObservableValue<T>

Type Parameters:

T - The type of the wrapped value.

All Superinterfaces:

Observable

All Known Subinterfaces:

Binding<T>, JavaBeanProperty<T>, NumberBinding, NumberExpression, ObservableBooleanValue, ObservableDoubleValue, ObservableFloatValue, ObservableIntegerValue, ObservableListValue<E>, ObservableLongValue, ObservableMapValue<K,V>, ObservableNumberValue, ObservableObjectValue<T>, ObservableSetValue<E>, ObservableStringValue, Property<T>, ReadOnlyJavaBeanProperty<T>, ReadOnlyProperty<T>, TextInputControl.Content

All Known Implementing Classes:

BooleanBinding, BooleanExpression, BooleanProperty, BooleanPropertyBase, DoubleBinding, DoubleExpression, DoubleProperty, DoublePropertyBase, FloatBinding, FloatExpression, FloatProperty, FloatPropertyBase, IntegerBinding, IntegerExpression, IntegerProperty, IntegerPropertyBase, JavaBeanBooleanProperty, JavaBeanDoubleProperty, JavaBeanFloatProperty, JavaBeanIntegerProperty, JavaBeanLongProperty, JavaBeanObjectProperty, JavaBeanStringProperty, ListBinding, ListExpression, ListProperty, ListPropertyBase, LongBinding, LongExpression, LongProperty, LongPropertyBase, MapBinding, MapExpression, MapProperty, MapPropertyBase, NumberExpressionBase, ObjectBinding, ObjectExpression, ObjectProperty, ObjectPropertyBase, ObservableValueBase, ReadOnlyBooleanProperty, ReadOnlyBooleanPropertyBase, ReadOnlyBooleanWrapper, ReadOnlyDoubleProperty, ReadOnlyDoublePropertyBase, ReadOnlyDoubleWrapper, ReadOnlyFloatProperty, ReadOnlyFloatPropertyBase, ReadOnlyFloatWrapper, ReadOnlyIntegerProperty, ReadOnlyIntegerPropertyBase, ReadOnlyIntegerWrapper, ReadOnlyJavaBeanBooleanProperty, ReadOnlyJavaBeanDoubleProperty, ReadOnlyJavaBeanFloatProperty, ReadOnlyJavaBeanIntegerProperty, ReadOnlyJavaBeanLongProperty, ReadOnlyJavaBeanObjectProperty, ReadOnlyJavaBeanStringProperty, ReadOnlyListProperty, ReadOnlyListPropertyBase, ReadOnlyListWrapper, ReadOnlyLongProperty, ReadOnlyLongPropertyBase, ReadOnlyLongWrapper, ReadOnlyMapProperty, ReadOnlyMapPropertyBase, ReadOnlyMapWrapper, ReadOnlyObjectProperty, ReadOnlyObjectPropertyBase, ReadOnlyObjectWrapper, ReadOnlySetProperty, ReadOnlySetPropertyBase, ReadOnlySetWrapper, ReadOnlyStringProperty, ReadOnlyStringPropertyBase, ReadOnlyStringWrapper, SetBinding, SetProperty, SetPropertyBase, SimpleBooleanProperty, SimpleDoubleProperty, SimpleFloatProperty, SimpleIntegerProperty, SimpleListProperty, SimpleLongProperty, SimpleMapProperty, SimpleObjectProperty, SimpleSetProperty, SimpleStringProperty, SimpleStyleableBooleanProperty, SimpleStyleableDoubleProperty, SimpleStyleableFloatProperty, SimpleStyleableIntegerProperty, SimpleStyleableLongProperty, SimpleStyleableObjectProperty, SimpleStyleableStringProperty, StringBinding, StringExpression, StringProperty, StringPropertyBase, StyleableBooleanProperty, StyleableDoubleProperty, StyleableFloatProperty, StyleableIntegerProperty, StyleableLongProperty, StyleableObjectProperty, StyleableStringProperty



L'INTERFACCIA E IL LISTENER

- L'interfaccia `ObservableValue<T>` definisce i metodi `addListener`, `getValue`, `removeListener`
- L'ascoltatore appropriato è un `ChangeListener<T>`, che è una interfaccia che dichiara il metodo
`void changed(ObservableValue<? extends T> obs,
 T oldValue, T newValue)`
- Come al solito, l'ascoltatore può essere implementato:
 - come classe standard
 - come classe anonima
 - come *lambda expression*
- Ma la novità è che *potrebbe non essere necessario* ☺



UN ESEMPIO: `StringProperty`

Classe astratta che incapsula un valore `String`

- il nome delle proprietà non coincide col valore incapsulato
- implementazione di default: `SimpleStringProperty`

```
StringProperty docente1 = new SimpleStringProperty(null, "docente 1");
docente1.set("AM");
System.out.println("Proprietà 1: " + docente1.getName());
System.out.println("Valore: " + docente1.get() );
StringProperty docente2 = new SimpleStringProperty(null, "docente 2");
docente2.set("RC");
System.out.println("Proprietà 2: " + docente2.getName());
System.out.println("Valore: " + docente2.get() );
```

```
Proprietà 1: docente 1
Valore: AM
Proprietà 2: docente 2
Valore: RC
```



UN ESEMPIO: StringProperty

Il bello delle proprietà è che si possono *legare* fra loro

- la modifica di una **si riflette automaticamente sull'altra**
- sotto girano eventi, *che però non dobbiamo più gestire a mano!*

Esempio: una nuova proprietà *docenti* = docente1 + docente2

- la modifica di uno qualsiasi dei due docenti implica *l'aggiornamento automatico* della proprietà collegata *docenti*

```
StringProperty docenti = new SimpleStringProperty(null, "docenti");  
docenti.bind( Bindings.concat(docente1, " & ", docente2) );  
  
System.out.println("Proprietà 3: " + docenti.getName());  
System.out.println("Valore: " + docenti.get());
```

Proprietà 3: docenti
Valore: AM & RC

Bindings è una helper class con decine di metodi per legare proprietà di ogni tipo



UN ESEMPIO: StringProperty

La proprietà *docenti* è **legata** a *docente1* + *docente2*

- di conseguenza, ogni modifica a uno qualsiasi dei due docenti implica l'aggiornamento automatico della proprietà *docenti*

La modifica della proprietà *docente2* ...

```
docente2.set("RCalegari");
```

```
System.out.println("Proprietà 3: " + docenti.getName());  
System.out.println("Valore: " + docenti.get());
```

... si riflette automaticamente
sulla proprietà *docenti*

Proprietà 3: docenti

Valore: AM & RCalegari



UN ESEMPIO: StringProperty

Se ci interessa, possiamo *anche* intercettare gli specifici eventi

- basta agganciare normalmente un ChangeListener

```
docente2.addListener( obs, oldValue, newValue) ->  
    System.out.println( "Cambiamento: " + obs +  
                        " da " + oldValue + " a " + newValue)  
);  
docente2.set("RCalegari");
```

Rendiamo visibile l'evento di cambiamento
(che avverrebbe comunque!)

```
System.out.println("Proprietà 3: " + docenti.getName());  
System.out.println("Valore: " + docenti.get());
```

Cambiamento docente 2: StringProperty [name: docente 2, value:
RCalegari] da RC a RCalegari

Proprietà 3: docenti

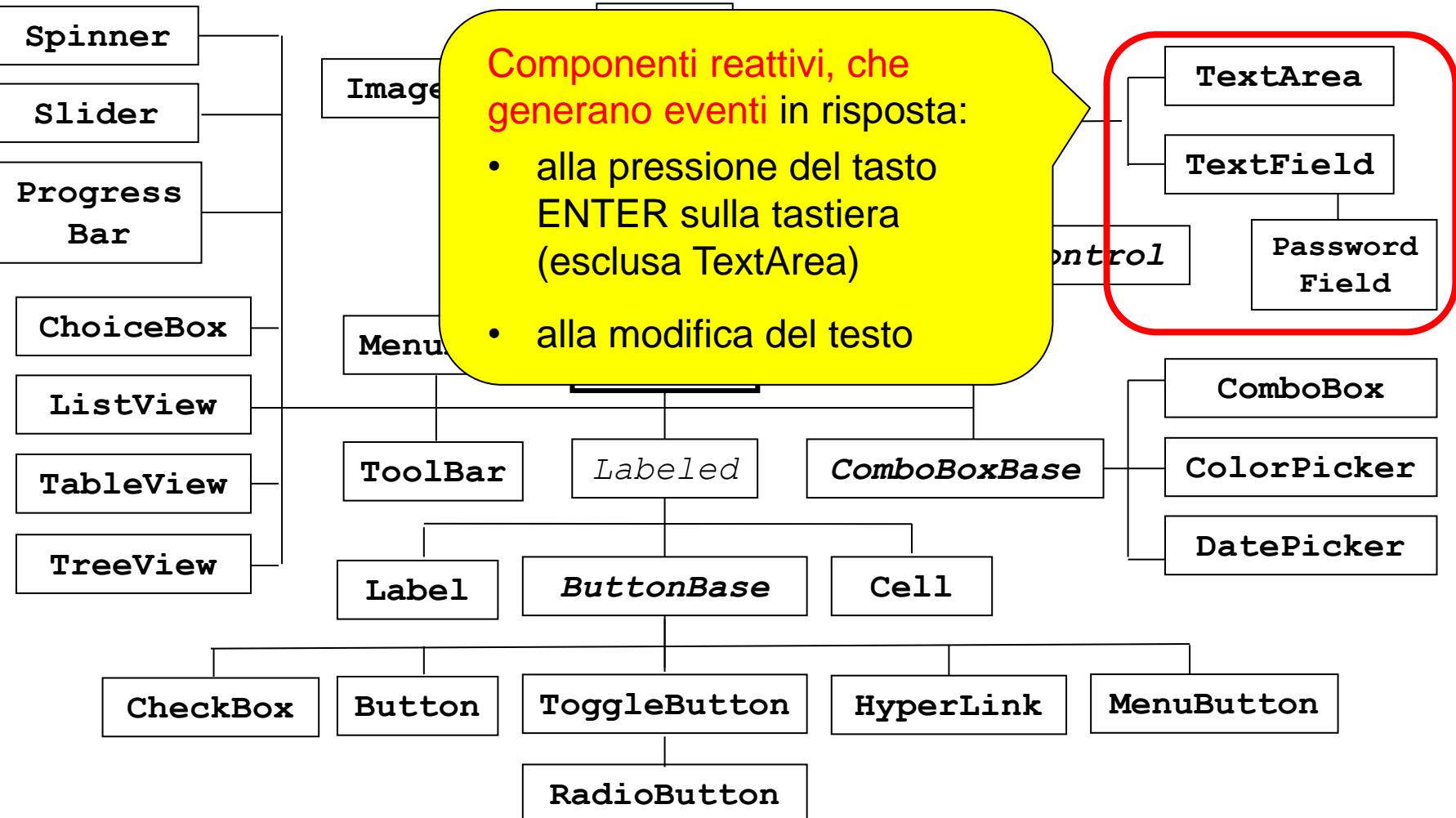
Valore: AM & RCalegari

Nel seguito incontreremo spesso proprietà di vari tipi (stringhe, numeri...)



GERARCHIA DEI CONTROLLI

`javafx.scene.control`





IL COMPONENTE **TextField**

- Un **TextField** genera un *evento di azione*, come un **Button**, quando si preme ENTER sulla tastiera
 - si gestisce l'evento di azione esattamente come in un **Button**
 - il precedente esempio "copia testo" può essere rivisto eliminando il pulsante "Aggiorna", che è ridondante → prossimo esempio
- Se invece si vuole essere *informati in tempo reale* di ogni modifica al testo, occorre **accedere alla proprietà *textproperty*** del model e **agganciarle un ascoltatore**
 - RICORDA: JavaFX segue lo schema MVVM!
Tranne i casi banali (come **Button**), i widget sono *la view* di una entità più complessa, che "dietro le quinte" ha un *model*
 - è al model che si chiedono proprietà e si agganciano listener



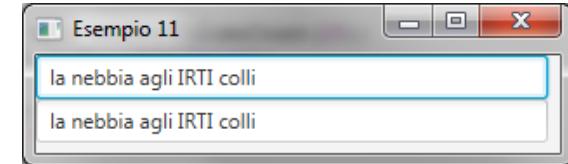
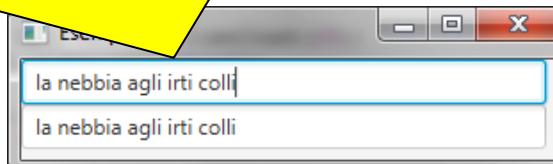
APPLICAZIONE "COPIA TESTO" IN TEMPO REALE

```
public class EsJavaFX11 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 11");  
        FlowPane panel = new FlowPane();  
        TextField txt1 = new TextField("Scrivere qui il nuovo messaggio");  
        txt1.setPrefColumnCount(25);  
        TextField txt2 = new TextField(); txt2.setEditable(false);  
        txt2.setPrefColumnCount(25);  
        txt1.textProperty().addListener(  
            .....  
        );  
        panel.getChildren().add(txt1,txt2);  
        ...  
    }  
}
```

Il primo **TextField** è osservato in tempo reale tramite la proprietà del suo model.

Il metodo **textProperty** recupera dal model tale proprietà.
È a lei che si aggancia il listener

Non si preme più ENTER: il testo nel primo textfield è copiato istante per istante nel secondo, a ogni modifica.





L'ASCOLTATORE DELLA PROPRIETÀ

Approccio tradizionale:
ascoltatore implementato come classe standard

```
txt1.textProperty().addListener(new MyEs12terListener(txt2));
```

```
import javafx.beans.value.ChangeListener;
javafx.beans.value.ObservableValue;

class MyEs12terListener implements ChangeListener<String> {

    private TextField txt2;

    public MyEs12terListener (TextField t){ txt2=t; }

    public void changed( ObservableValue<? extends String> observable,
                        String oldValue, String newValue) {
        System.out.println("textfield changed from " + oldValue + " to " + newValue);
        txt2.setText(newValue);
    }
}
```

Verboso e poco leggibile



L'ASCOLTATORE DELLA PROPRIETÀ

Approccio alternativo:
ascoltatore implementato come classe *anonima*

```
...
txt1.textProperty().addListener(
    new ChangeListener<String>() {
        public void changed( ObservableValue<? extends String> observable,
                            String oldValue, String newValue) {
            System.out.println("textfield changed from " + oldValue + " to " + newValue);
            txt2.setText(newValue);
        }
    });
panel.getChildren().add(txt1,txt2);
...
```

Meno verboso,
ma ancor meno leggibile



L'ASCOLTATORE DELLA PROPRIETÀ

Approccio moderno e migliore:
ascoltatore implementato come *lambda expression*
(sostituisce la classe anonima)

```
...
txt1.textProperty().addListener(
    (observable, oldValue, newValue) -> {
        System.out.println("textfield changed from " + oldValue + " to " + newValue);
        txt2.setText(newValue);
    }
);
panel.getChildren().add(txt1, txt2);
...
```

Molto meno verboso
e molto più leggibile

Type inference!



L'ASCOLTATORE DELLA PROPRIETÀ

.. e se rinunciamo alla `println` diventa ancora più semplice perché con una singola istruzione non serve più il blocco {...}

```
...
txt1.textProperty().addListener(
    (observable, oldValue, newValue) -> txt2.setText(newValue);
);
panel.getChildren().add(txt1,txt2);
...
```

Breve, conciso, leggibile!

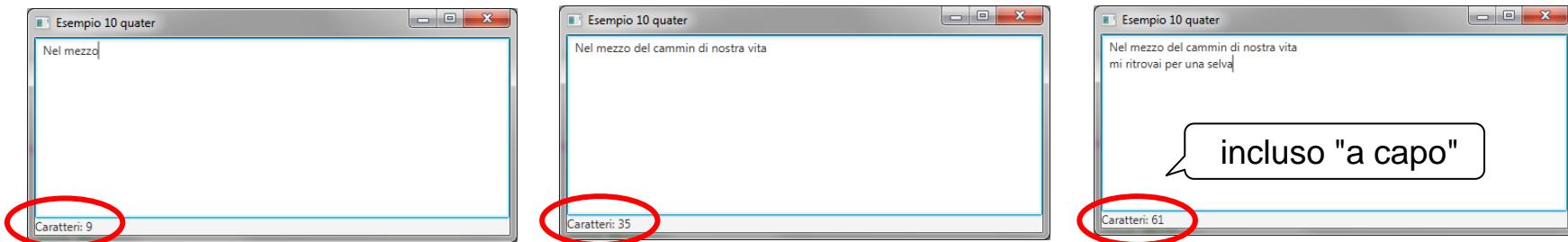


IDEA CON TextArea

...

```
TextArea area = new TextArea();
Label label = new Label("Caratteri: 0");
area.textProperty().addListener(
    (obs, oldV, newV) -> label.setText(
        "Caratteri: " + area.getText().length()
);
...
```

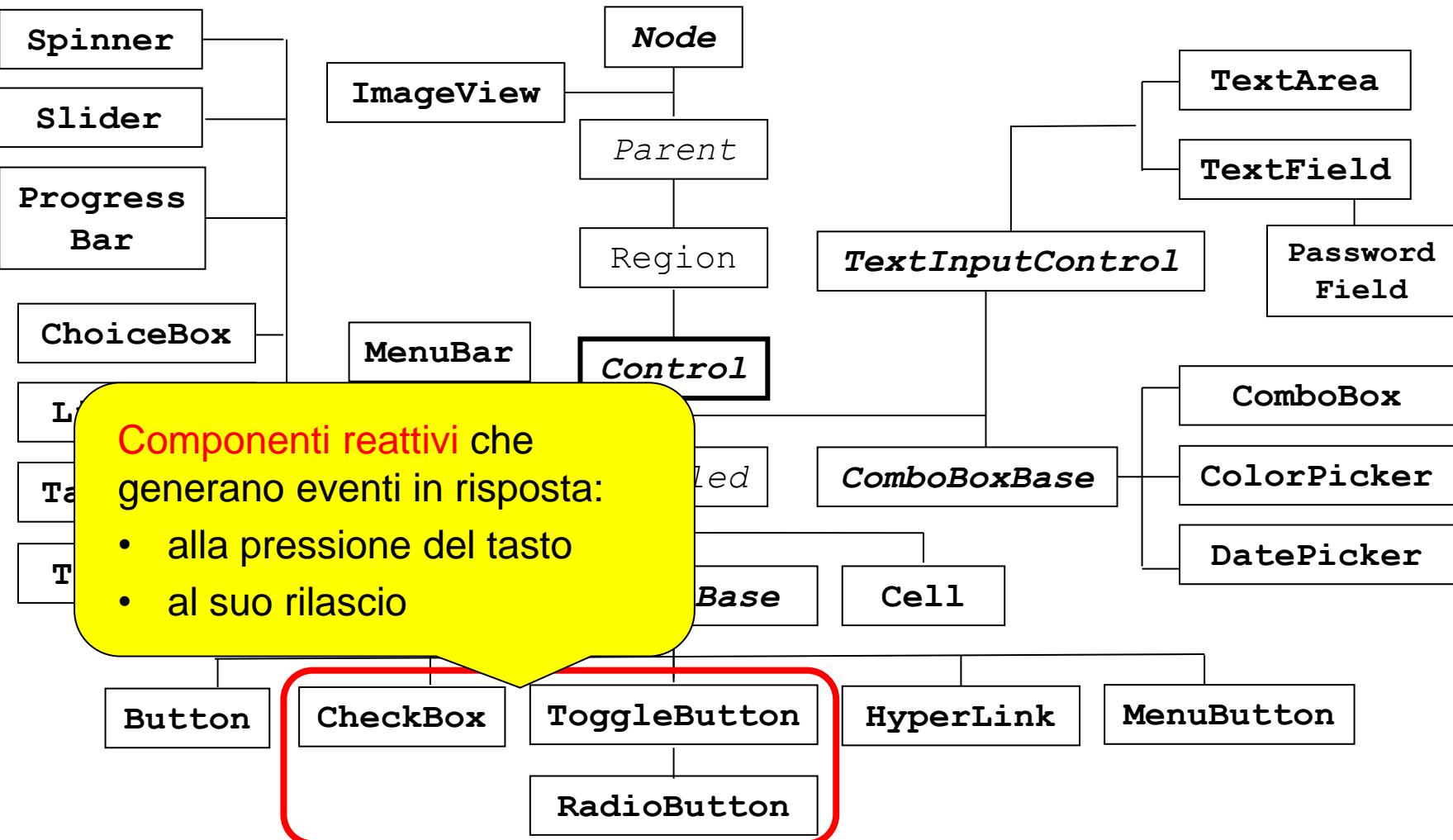
La label mostra il numero di caratteri digitati
e finalmente li aggiorna in tempo reale





GERARCHIA DEI CONTROLLI

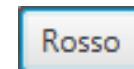
`javafx.scene.control`



THE TOGGLES FAMILY

- Il **Button** base è un **classico pulsante**

- una sola posizione stabile (quella OFF)
- attivo (ON) solo finché lo si mantiene premuto
- appena lo si rilascia, torna nella posizione di riposo



- I **Toggle** sono invece **interruttori a due posizioni**

- premendoli, si attivano e commutano nella posizione ON
- per riportarli nella posizione OFF occorre riprenderli
- ne sono disponibili diversi, per varie esigenze



Toggle



CheckBox



RadioButton



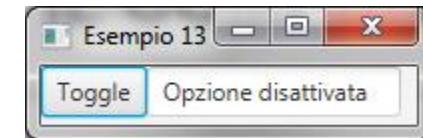
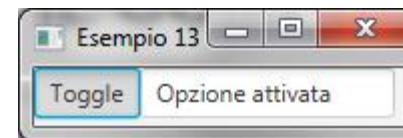
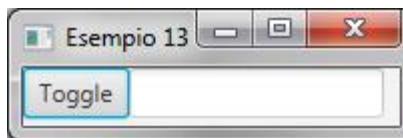
Usati singolarmente (per opzioni
non mutuamente esclusive)

Solitamente usati a gruppi
(opzioni mutuamente esclusive)



IL COMPONENTE Toggle

```
public class EsJavaFX13bis extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 13bis");  
        FlowPane panel = new FlowPane();  
        ToggleButton tg1 = new ToggleButton("Toggle");  
        TextField txt2 = new TextField(); txt2.setEditable(false);  
        tg1.setOnAction( event -> txt2.setText(  
            tg1.isSelected() ? "Opzione attivata" : "Opzione disattivata" ) );  
        panel.getChildren().add(tg1,txt2);  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

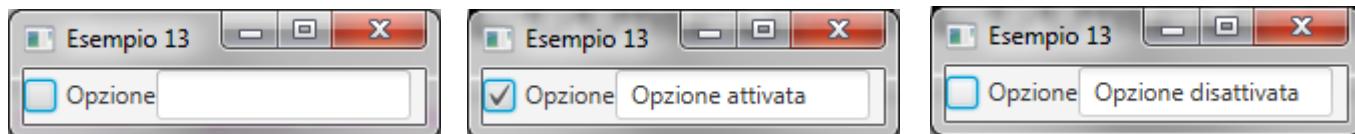


Solito evento di azione,
stesso listener del Button



IL COMPONENTE CheckBox

```
public class EsJavaFX13 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 13");  
        FlowPane panel = new FlowPane();  
        CheckBox ck1 = new CheckBox("Opzione");  
        TextField txt2 = new TextField(); txt2.setEditable(false);  
        ck1.setOnAction( event -> txt2.setText(  
            ck1.isSelected() ? "Opzione attivata" : "Opzione disattivata" ) );  
        panel.getChildren().add(ck1,txt2);  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```



Solito evento di azione,
stesso listener del Button



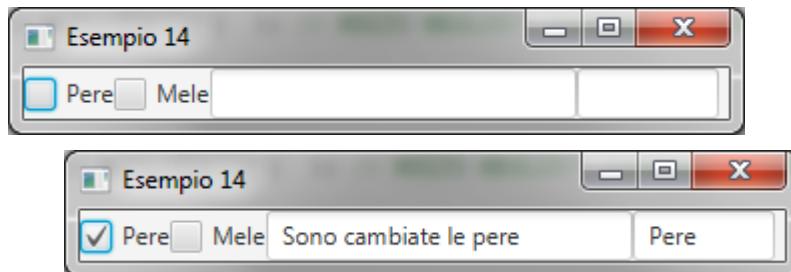
UN CASO CON DUE CheckBox

Obiettivo: gestire due **CheckBox** in modo che:

- un primo **TextField** rifletta l'ultima azione eseguita
- un secondo **TextField** dia invece lo stato complessivo

Gestione eventi:

- un primo **TextField** va gestito per singola checkbox
(due listener separati, uno per ciascuna casella)
- il secondo **TextField** va gestito invece collettivamente
(stesso listener per entrambe le caselle)





UN CASO CON DUE CheckBox

```
public class EsJavaFX14 extends Application {  
    private CheckBox ck1, ck2;  
    private TextField txt1, txt2;  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 14");  
        FlowPane panel = new FlowPane();  
        ck1 = new CheckBox("Pere"); ck2 = new CheckBox("Mele");  
        txt1 = new TextField(); txt1.setEditable(false);  
        txt2 = new TextField(); txt2.setEditable(false);  
        ck1.setOnAction(event -> {  
            txt1.setText("Sono cambiate le pere"); myAuxHandler(); } );  
        ck2.setOnAction(event -> {  
            txt1.setText("Sono cambiate le mele"); myAuxHandler(); } );  
        panel.getChildren().add(ck1, ck2, txt1, txt2);  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
    ...  
}
```

Variabili spostate a livello di classe per essere visibili anche fuori dal metodo **start**

Due listener distinti, ma con una parte comune

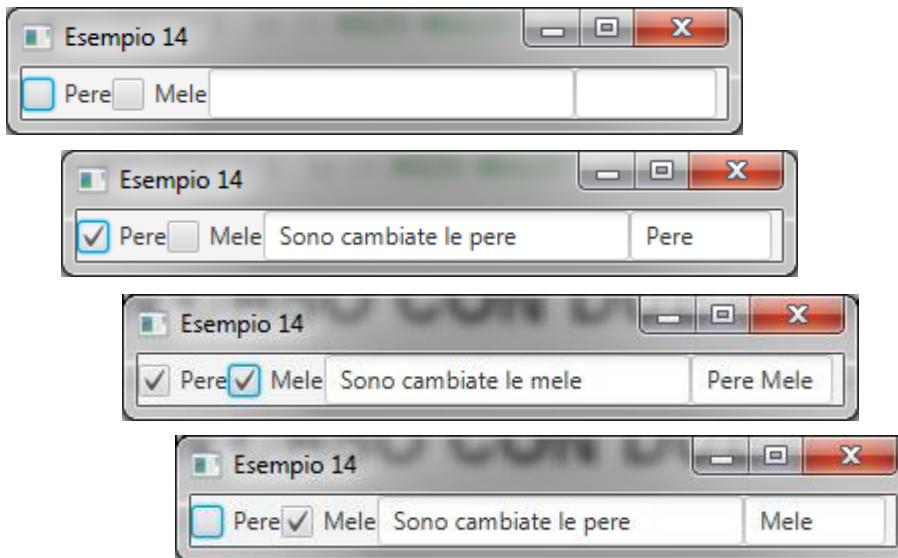
Nostro metodo di utilità per fattorizzare la parte comune ai due casi



UN CASO CON DUE CheckBox

...

```
private void myAuxHandler(){  
    String frase = (ck1.isSelected() ? "Pere " : "")  
        + (ck2.isSelected() ? "Mele" : "");  
    txt2.setText(frase);  
}  
}
```



Sintetizza la frase descrittiva
dello status globale

Il primo textfield riflette
l'ultima azione.
Il secondo riflette lo stato
complessivo delle opzioni



UN CASO CON DUE CheckBox

Alternativa: un listener unico

```
ck1 = new CheckBox("Pere"); ck2 = new CheckBox("Mele");
txt1 = new TextField(); txt1.setEditable(false);
txt2 = new TextField(); txt2.setEditable(false);
ck1.setOnAction(this::myHandle);
ck2.setOnAction(this::myHandle);
...
}
private void myHandle(ActionEvent event) {
    txt1.setText(
        "Sono cambiate le " + (event.getSource() == ck1 ? "pere" : "mele"));
    String frase = (ck1.isSelected() ? "Pere " : "") +
        (ck2.isSelected() ? "Mele " : "");
    txt2.setText(frase);
}
```

Method reference per fattorizzare la gestione in un unico listener

Occorre ispezionare l'argomento **event** per capire quale checkbox è stata premuta per ultima

Sgradevole dover cablare un confronto con una specifica variabile...



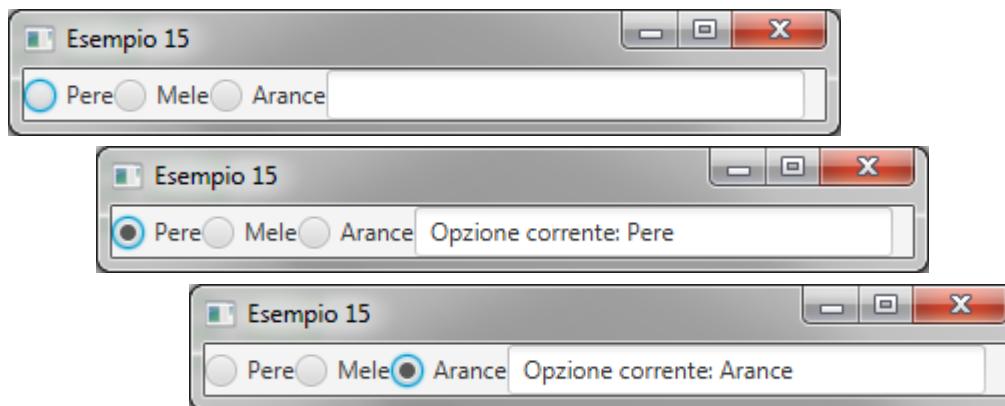
IL COMPONENTE RadioButton

A differenza dei precedenti, il **RadioButton** rappresenta un'opzione *mutuamente esclusiva*

- non è mai usato da solo: ce ne sono sempre almeno due
- il concetto di gruppo è catturato da un'istanza di **ToggleGroup**

Gestione eventi:

- si possono gestire i pulsanti singolarmente (scomodo)...
- ..oppure sfruttare il **ToggleGroup** (più comodo)





IL COMPONENTE RadioButton

```
public class EsJavaFX15 extends Application {  
    private RadioButton b1, b2, b3;  
    private ToggleGroup tg;  
    private TextField txt1;  
  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 15");  
        FlowPane panel = new FlowPane();  
        tg = new ToggleGroup();  
        b1 = new RadioButton("Pere"); b1.setToggleGroup(tg);  
        b2 = new RadioButton("Mele"); b2.setToggleGroup(tg);  
        b3 = new RadioButton("Arance"); b3.setToggleGroup(tg);  
        txt1 = new TextField(); txt1.setEditable(false);  
        b1.setOnAction(this::myHandle);  
        b2.setOnAction(this::myHandle);  
        b3.setOnAction(this::myHandle);  
  
        panel.getChildren().add(b1,b2,b3,txt1);  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

Ogni RadioButton va aggiunto anche al suo gruppo

Method reference per fattorizzare la gestione in un unico listener



IL COMPONENTE RadioButton

```
private void myHandle(ActionEvent event) {  
    txt1.setText("Opzione corrente: " +  
        ( (RadioButton)event.getSource() ) .getText());  
}
```

Occorre ispezionare l'argomento **event** per capire quale radiobutton è stato premuto per ultimo

Necessario il cast a **RadioButton** per poter invocare **getText**

Alternativa: sfruttare (male) il **ToggleGroup**

```
private void myHandle(ActionEvent event) {  
    txt1.setText("Opzione corrente: " +  
        ( (RadioButton)tg.getSelectedToggle() ) .getText());  
}
```

Non ispezioniamo più **event**

Necessario il cast a **RadioButton** per poter invocare **getText**



IL COMPONENTE RadioButton

Alternativa migliore: sfruttare la proprietà
selectedToggleProperty di **ToggleGroup**

- non agganciamo più tanti listener ai singoli pulsanti
- agganciamo un solo **ChangeListener** alla proprietà

...

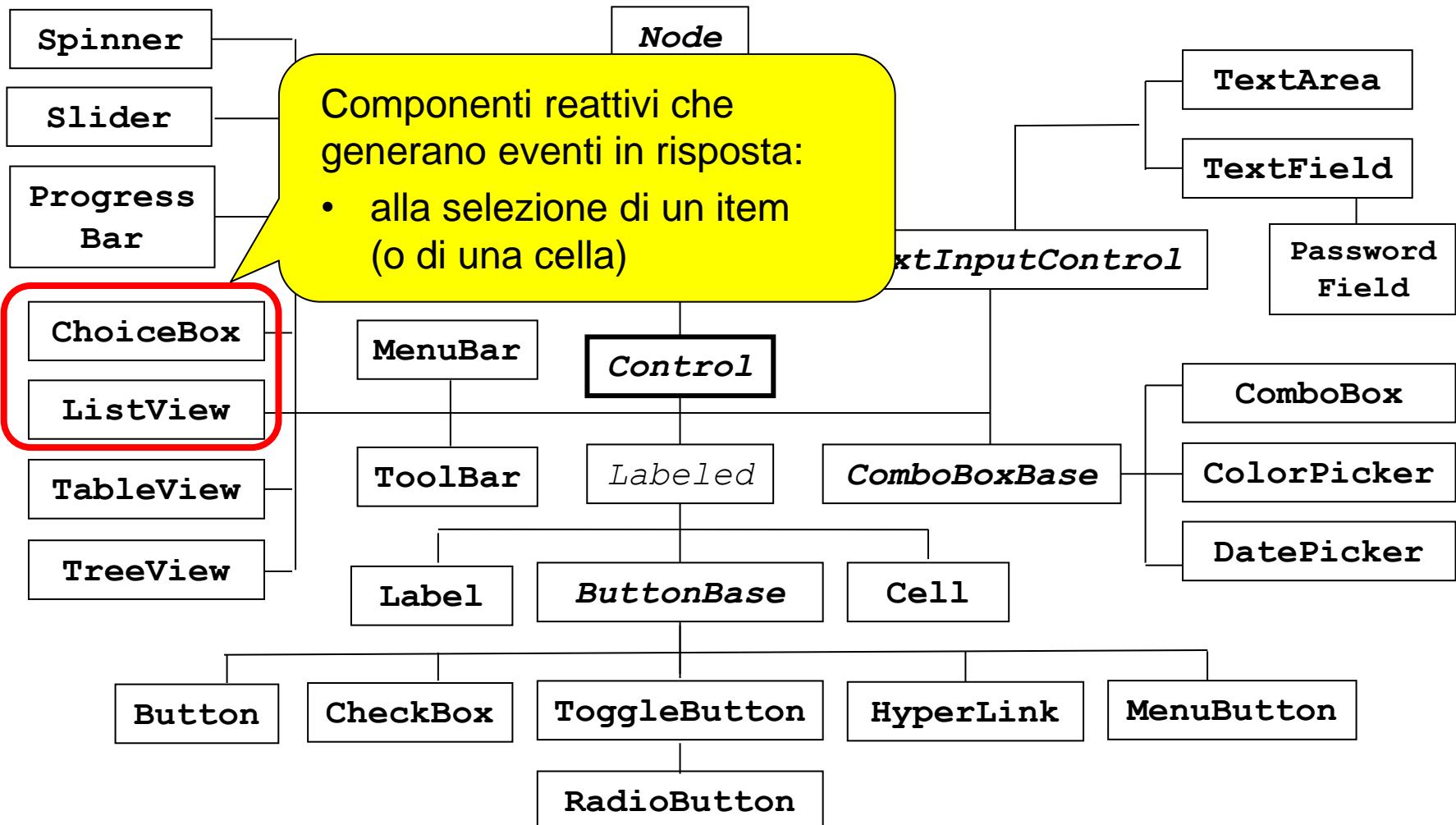
```
tg.selectedToggleProperty().addListener(
    (changed, oldval, newval) ->
    txt1.setText("Opzione corrente: " + ((RadioButton)newval).getText())
);
panel.getChildren().add(b1,b2,b3,txt1);
Scene scene = new Scene(panel);
stage.setScene(scene);
stage.show();
}
```

Necessario il cast a **RadioButton**
per poter invocare **getText**



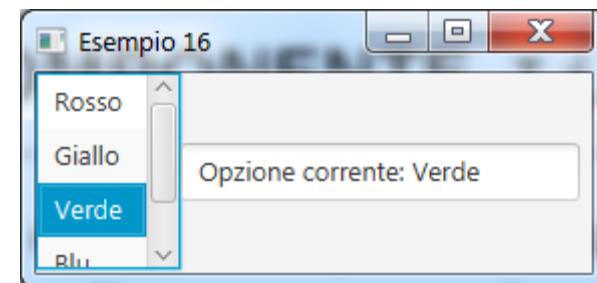
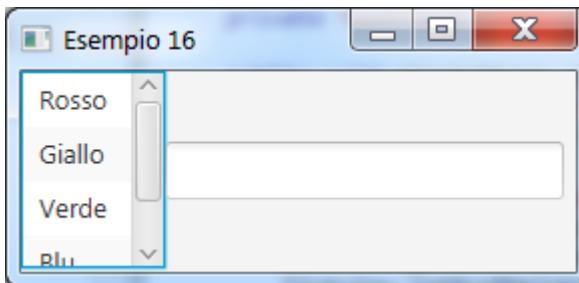
GERARCHIA DEI CONTROLLI

`javafx.scene.control`



IL COMPONENTE ListView

- Un **ListView** è una lista (con eventuali scrollbar) che permette di scegliere una voce da un elenco
 - di default, è abilitata la selezione singola
 - ma si può abilitare la selezione multipla
- Quando una voce viene scelta, nel model associato viene modificata la proprietà **selectedItemProperty**
 - MVVM: se si vuole intercettare l'evento, occorre recuperare la proprietà dal model e agganciarle un **ChangeListener**





IL COMPONENTE ListView

```
public class EsJavaFX16 extends Application {  
    private ListView<String> listview;  
    private TextField txt1;  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 16");  
        FlowPane panel = new FlowPane();  
        listview = new ListView<>();  
        listview.setItems(  
            FXCollections.observableArrayList("Rosso", "Giallo", "Verde", "Blu") );  
        listview.setPrefHeight(listview.getItems().size()*24+ 2);  
        listview.setPrefWidth(72);  
        listview.getSelectionModel().selectedItemProperty().addListener(  
            (obj, oldval, newval) -> txt1.setText("Opzione corrente: " + newval) );  
        txt1 = new TextField(); txt1.setEditable(false);  
        panel.getChildren().add(listview,txt1);  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

È tipizzata: occorre specificare il tipo degli elementi.

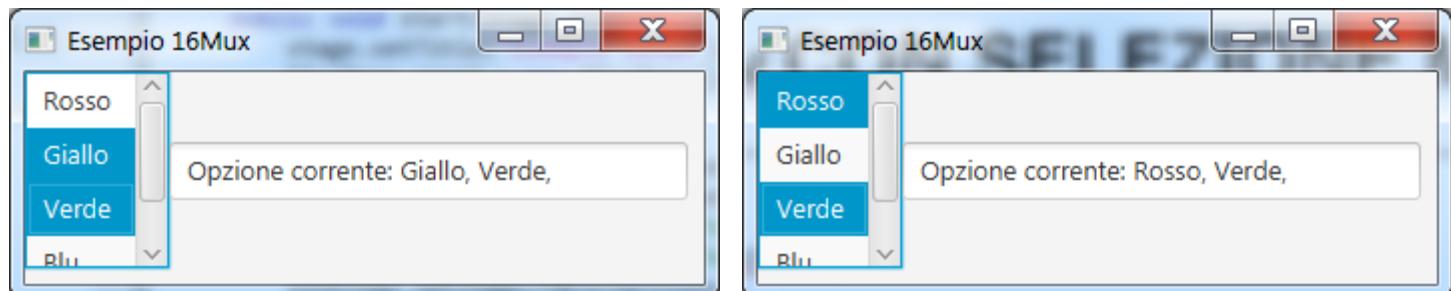
Per popolare la `ListView` usare i metodi factory di `FXCollections`

Di default mostra 10 item (brutto) → impostiamo l'altezza

Gestione evento: `ChangeListener` implementato come lambda expression

ListView CON SELEZIONE MULTIPLA

```
...  
listview = new ListView<>();  
listview.setItems(  
    FXCollections.observableArrayList("Rosso", "Giallo", "Verde", "Blu") );  
listview.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);  
listview.setPrefHeight(listview.getItems().size()*24+ 2);  
listview.setPrefWidth(72);  
listview.getSelectionModel().selectedItemProperty().addListener(  
    (obj, oldval, newval) -> {  
        StringBuilder sb = new StringBuilder();  
        for(String item : listview.getSelectionModel().getSelectedItems())  
            sb.append(item+", ");  
        txt1.setText("Opzione corrente: " + sb) ;  
    }  
);  
...  
Selezione multipla tenendo premuti i  
tasti SHIFT (item adiacenti) o CTRL
```





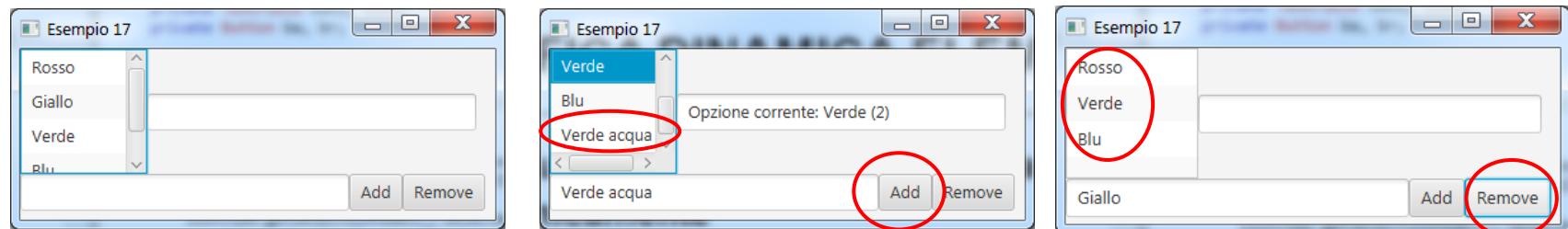
MODIFICA DINAMICA ELEMENTI

È possibile *aggiungere o togliere elementi* da una **ListView** *dinamicamente*

- la lista degli elementi è recuperabile con `getItems()`
- si possono aggiungere/togliere elementi normalmente, come in una qualsiasi lista

Nell'esempio che segue:

- un campo di testo consente di scrivere il nome dell'elemento
- due buttoni Add/Remove effettuano l'aggiornamento richiesto





MODIFICA LISTA ELEMENTI

```
public class EsJavaFX17 extends Application {  
    private ListView<String> listview;  
    private TextField txt1, txt2;  
    private Button ba, br;  
    public void start(Stage stage) {  
        ...  
        txt2 = new TextField(); txt2.setPrefColumnCount(20);  
        ba = new Button("Add");  
        br = new Button("Remove");  
        panel.getChildren().add(listview,txt1,txt2,ba,br);  
        ba.setOnAction(e -> listview.getItems().add(txt2.getText()));  
        br.setOnAction(e -> listview.getItems().remove(txt2.getText()));  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

OCCHIO: `List.add` non controlla la presenza di eventuali duplicati!

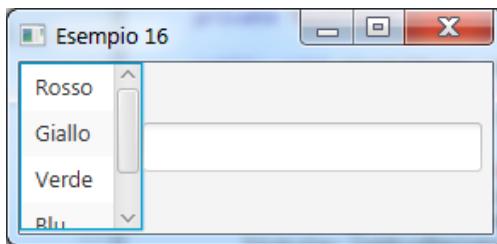
Se elemento non presente non fa nulla (tipico comportamento di `List.remove`)



IL COMPONENTE ChoiceBox

Una **ChoiceBox** è molto simile a **ListView**

- semplicemente, gli elementi sono mostrati *in una lista a discesa pop-up*, che si apre solo cliccandoci sopra, invece che in una lista fissa
- la gestione eventi e le proprietà sono identiche



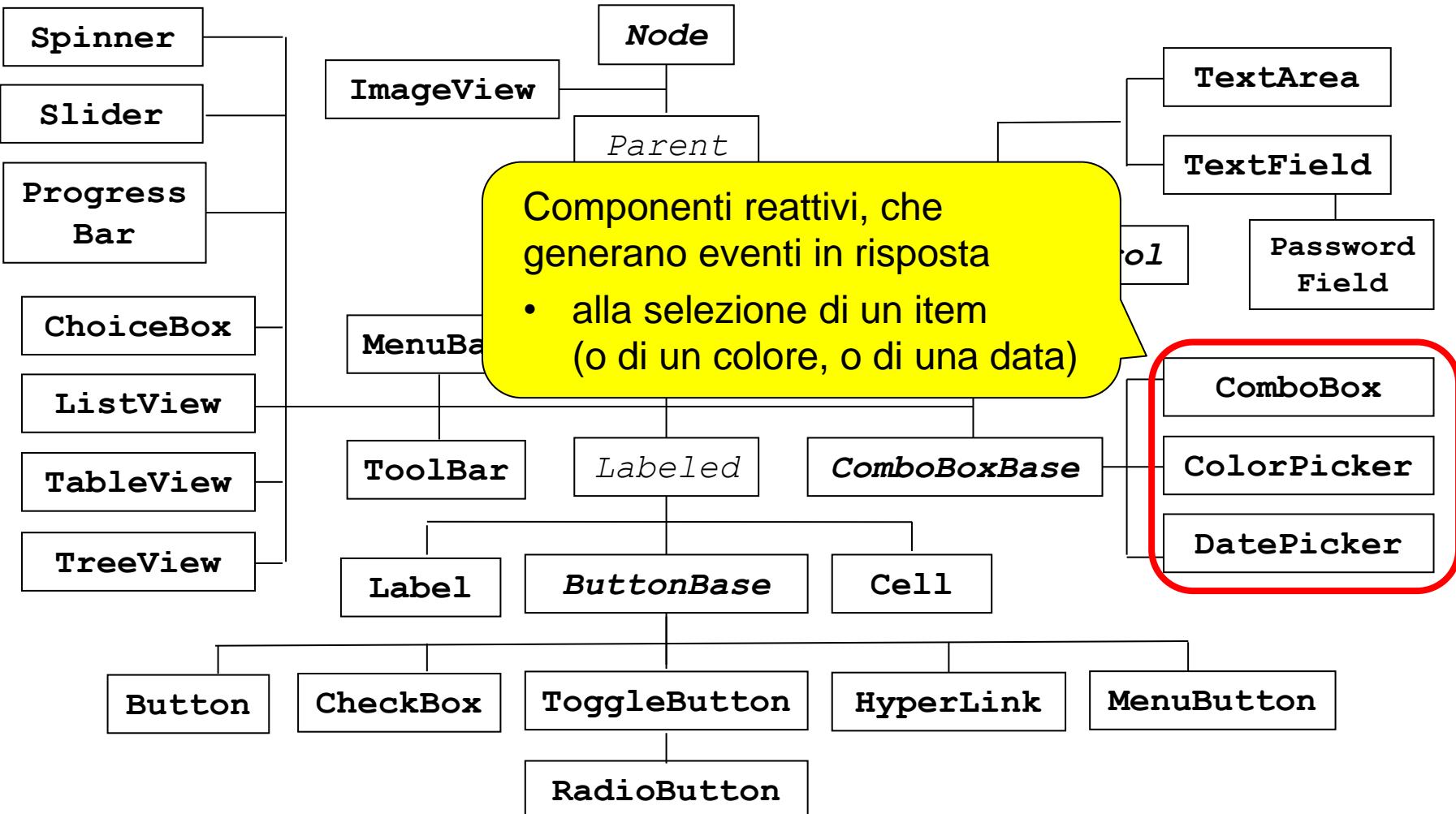
Esercizio lasciato al lettore:

nell'esempio precedente, mettere un **ChoiceBox** al posto di **ListView**



GERARCHIA DEI CONTROLLI

`javafx.scene.control`





IL COMPONENTE ComboBox

Una **ComboBox** unisce le funzionalità di **ChoiceBox** e **TextField**

- si può usare per scegliere elementi da list pop-up...
- ..o anche (se editabile) per scriverne di nuovi, *diversi da quelli proposti nella lista*
- come tutti i figli di ComboBoxBase, *genera un evento di azione* (come un bottone) quando viene «premuto», ossia *quando si sceglie un elemento* → molto pratico da gestire

ComboBox non editabile: funziona come una **ChoiceBox**

ComboBox editabile: si possono scrivere valori non previsti in elenco



IL COMPONENTE ComboBox

```
public class EsJavaFX19 extends Application {  
    private ComboBox<String> cb; Tipizzata  
    private TextField txt1;  
  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 19");  
        FlowPane panel = new FlowPane();  
        cb = new ComboBox<>(); cb.setPrefWidth(100);  
        cb.setItems(  
            FXCollections.observableArrayList("Rosso", "Giallo", "Verde", "Blu")); Stesso approccio di popolamento  
        panel.getChildren().add(cb, txt1);  
        cb.setOnAction(  
            event -> txt1.setText("Opzione corrente: " + cb.getValue()  
                + " (" + cb.getSelectionModel().getSelectedIndex() + ")")  
        );  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

getValue recupera l'elemento selezionato
Per recuperare l'indice occorre invece rivolgersi al model.



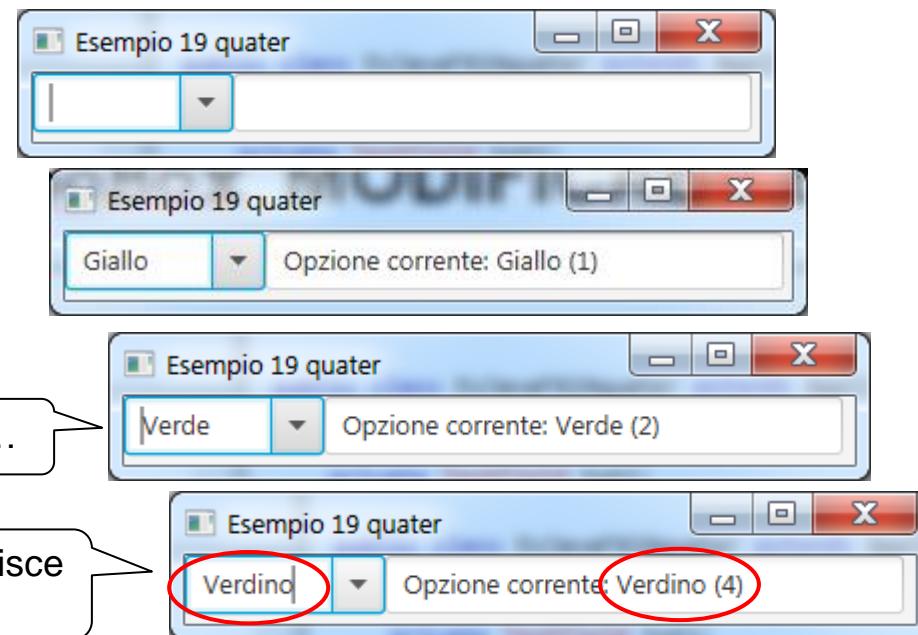
ComboBox MODIFICABILE (1)

```
public class EsJavaFX19bis extends Application {  
    private ComboBox<String> cb;  
    private TextField txt1;  
  
    public void start(Stage stage){  
        stage.setTitle("Esempio 19 bis");  
        FlowPane panel = new FlowPane();  
        cb = new ComboBox<>(); cb.setPrefWidth(100);  
        cb.setEditable(true);  
        cb.setItems(  
            FXCollections.observableArrayList("Rosso", "Giallo", "Verde", "Blu"));  
        panel.getChildren().add(cb,txt1);  
        cb.setOnAction(  
            event -> txt1.setText("Opzione corrente: " + cb.getValue()  
                + " (" + cb.getSelectionModel().getSelectedIndex() + ")")  
        );  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
        stage.show();  
    }  
}
```

ATTENZIONE: nelle combo editabili, se si aggiungono o tolgono elementi *l'indice non riflette in automatico la nuova situazione*
→ Indicazione errata!

ComboBox MODIFICABILE (2)

- In effetti, poter scrivere un elemento non previsto *non significa volerlo necessariamente aggiungere alla lista in modo stabile*
- Se si vuole che il nuovo elemento sia inserito stabilmente in elenco, occorre *aggiornare il model* nella gestione dell'evento



Verde è l'elemento n° 2...

...ma la new entry Verdino finisce
in coda, alla posizione n° 4



ComboBox MODIFICABILE (3)

- In effetti, poter scrivere un elemento non previsto *non significa volerlo necessariamente aggiungere alla lista in modo stabile*
- Se si vuole che il nuovo elemento sia inserito stabilmente in elenco, occorre *aggiornare il model* nella gestione dell'evento

```
ObservableList<String> model =  
    FXCollections.observableArrayList("Rosso", "Giallo", "Verde", "Blu");  
cb.setItems(model);  
cb.setOnAction(  
    event -> {  
        int i = cb.getSelectionModel().getSelectedIndex();  
        if (!cb.getValue().equals(model.get(i))) {  
            model.add(cb.getValue());  
            i = model.size() - 1;  
        }  
        txt1.setText("Opzione corrente: " + cb.getValue() + " (" + i + ")");  
    }  
);
```

Introdotto model esplicito (per poterlo usare dopo)

Nuovo valore aggiunto al model (in coda)

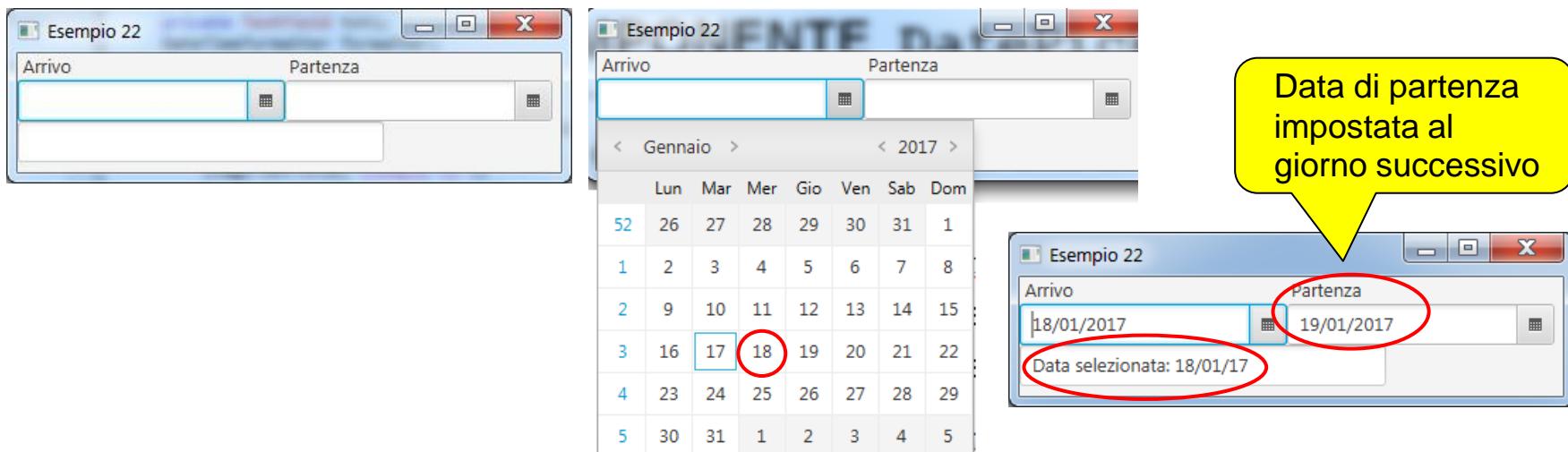
Indice settato di conseguenza a fondo lista



IL COMPONENTE **DatePicker**

Un **DatePicker** consente di scegliere una data dal classico calendarietto pop-up

- il formato dipende dal Locale e dal Formatter impostati
- di default, si posiziona sulla data corrente
- ma è possibile forzarlo su una data diversa
(esempio: arrivo/partenza in hotel)





IL COMPONENTE DatePicker

```
public class EsJavaFX22 extends Application {  
    private DatePicker picker, picker2;  
    private DateTimeFormatter formatter;  
    private TextField txt1;  
  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 22");  
        FlowPane panel = new FlowPane();  
        picker = new DatePicker();  
        picker2 = new DatePicker();  
        picker.setOnAction(  
            event -> {  
                java.time.LocalDate date = (java.time.LocalDate)(picker.getValue());  
                txt1.setText("Data selezionata: " + formatter.format(date));  
                picker2.setValue(picker.getValue().plusDays(1));  
            }  
        );  
    };  
};  
...
```

Vale il **Locale** corrente

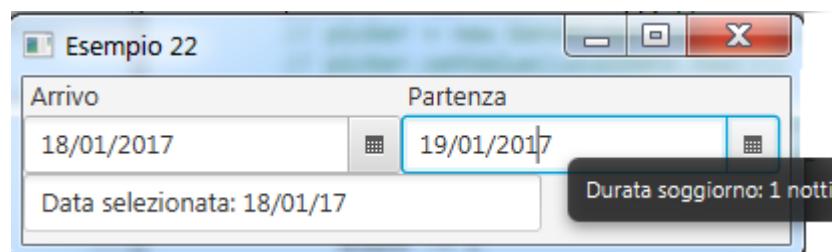
Recupero valore
(data) corrente

Forzatura data corrente
nel secondo picker

IL COMPONENTE DatePicker

```
...  
  
    picker2.setOnAction(  
        event -> {  
            if (picker.getValue() == null) return;  
            long p = java.time.temporal.ChronoUnit.DAYS.between(  
                picker.getValue(), picker2.getValue());  
            picker2.setTooltip(new Tooltip("Durata soggiorno: " + p + " notti"));  
        }  
    );  
    Scene scene = new Scene(panel);  
    stage.setScene(scene);  
    stage.show();  
}  
}
```

Quando si sceglie la seconda data,
si calcola la durata del soggiorno

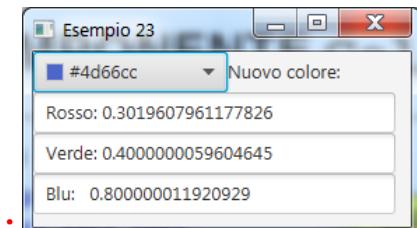
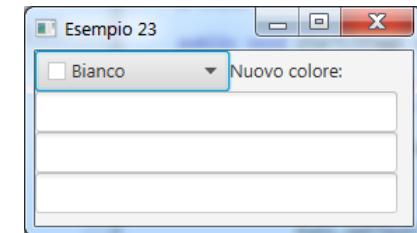




IL COMPONENTE ColorPicker

```
public class EsJavaFX23 extends Application {  
    private ColorPicker picker;  
    private TextField txt1,txt2,txt3;  
  
    public void start(Stage stage){  
        stage.setTitle("Esempio 23");  
        FlowPane panel = new FlowPane();  
        txt1 = new TextField(); txt1.setEditable(false);  
        txt2 = new TextField(); txt2.setEditable(false);  
        txt3 = new TextField(); txt3.setEditable(false);  
        picker = new ColorPicker();  
        picker.setOnAction(  
            event -> {  
                txt1.setText("Rosso: " + picker.getValue().getRed());  
                txt2.setText("Verde: " + picker.getValue().getGreen());  
                txt3.setText("Blu: " + picker.getValue().getBlue());  
            }  
        );  
        ...  
    }  
}
```

ColorPicker è analogo, ma serve per scegliere colori

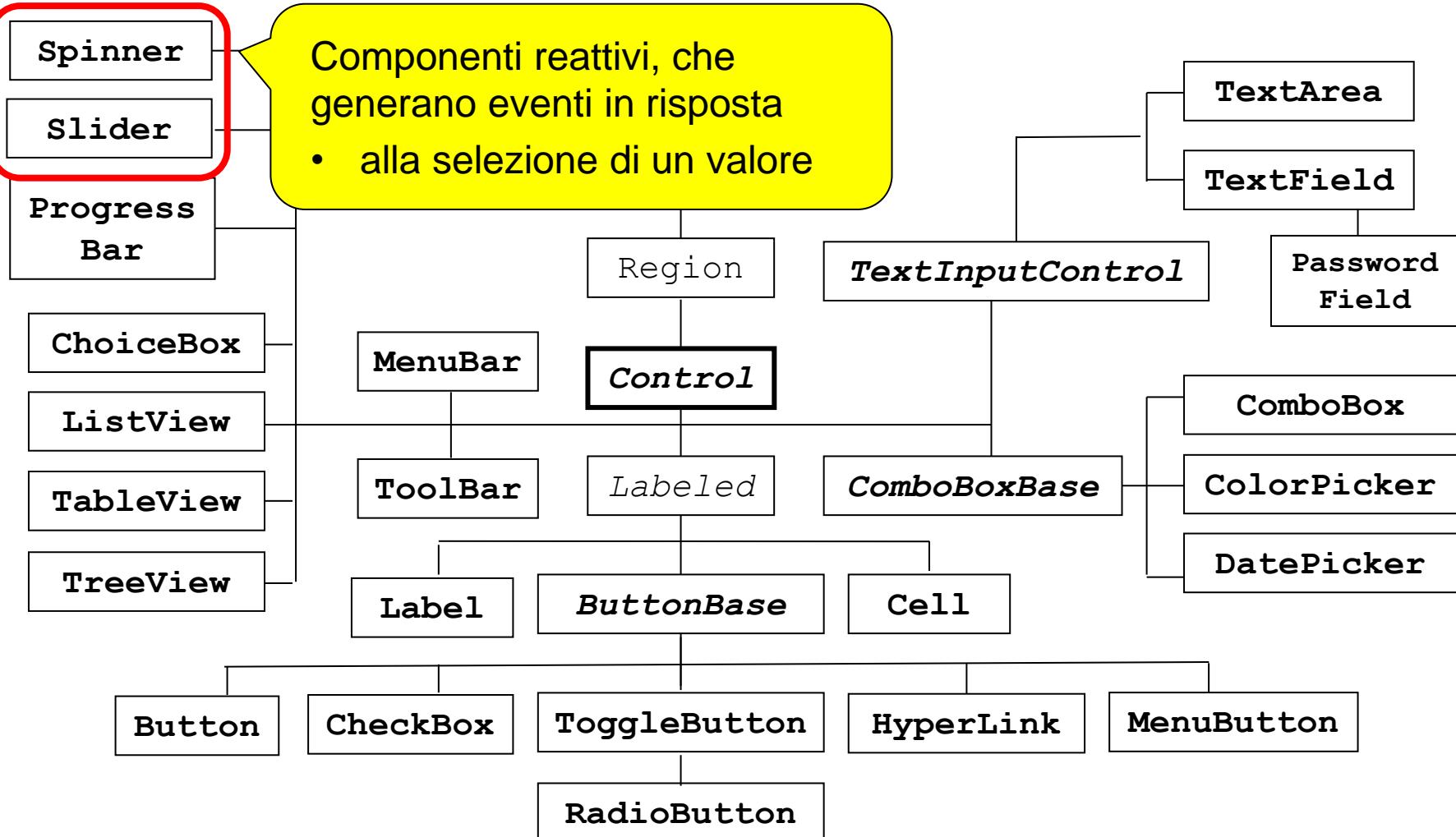


Recupero colore corrente

Recupero valori RGB

GERARCHIA DEI CONTROLLI

`javafx.scene.control`

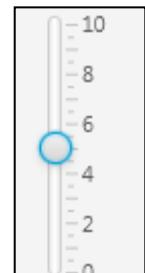
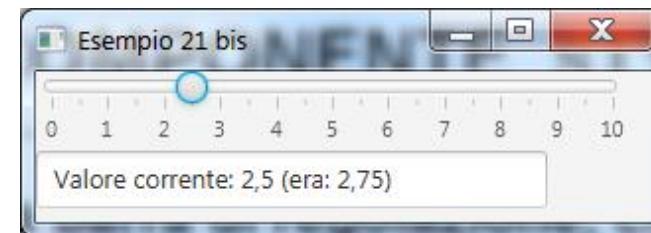
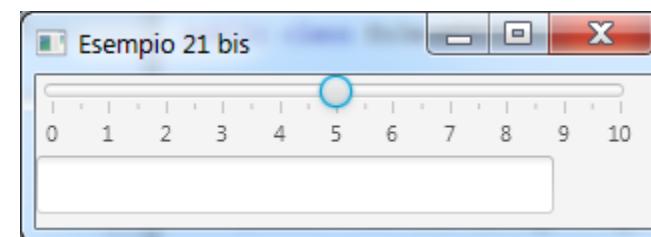
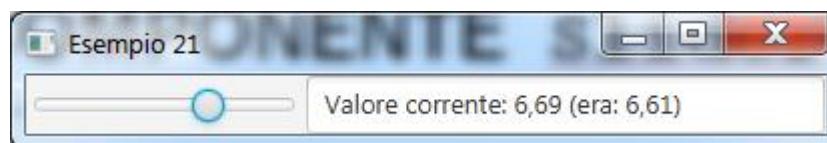
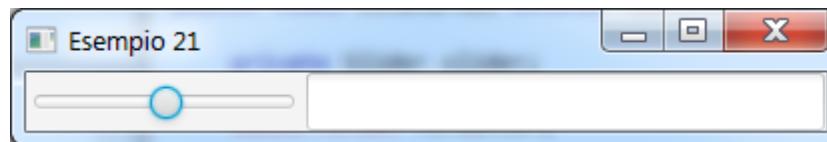




IL COMPONENTE Slider

Lo **Slider** è una barra di regolazione, che consente di scegliere un valore muovendo un cursore su una scala

- sono configurabili sia i limiti min/max, sia la graduazione della scala
- graduazione distingue fra *major tick units* e *minor tick units*
- orientamento orizzontale (default) o verticale
- eventuale listener sulla proprietà osservabile **valueProperty**





IL COMPONENTE Slider

```
public class EsJavaFX21 extends Application {  
  
    private Slider slider;  
    private TextField txt1;  
    private NumberFormat formatter;  
  
    public void start(Stage stage) {  
  
        formatter = NumberFormat.getInstance();  
        formatter.setMaximumFractionDigits(2);  
        stage.setTitle("Esempio 21");  
        FlowPane panel = new FlowPane();  
        txt1 = new TextField(); txt1.setEditable(false);  
        slider = new Slider(0,10,5); // min, max, initvalue  
        slider.valueProperty().addListener( (changed, oldval, newval) ->  
            txt1.setText("Valore corrente: " + formatter.format(newval)  
                + " (era: " + formatter.format(oldval) + ")")  
        );  
        panel.getChildren().add(slider,txt1);  
        Scene scene = new Scene(panel);  
        stage.setScene(scene);  
    }  
}
```



Versione base, senza scala graduata

Argomenti del costruttore:

- valore inizio scala
- valore fine scala
- posizione iniziale cursore



Slider: SCALA GRADUATA

Configurabilità della scala

Major tick

- mostrare, con etichette, ogni 1 unità

Minor tick

- mostrare ogni 1 unità

Spostamento cursore

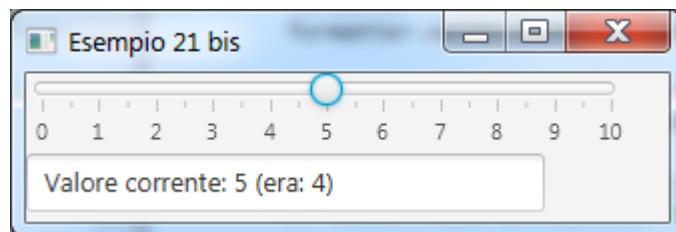
- in caso di movimento tramite tasti freccia:
muovere a blocchi di 1 unità
- forza il cursore a stare sui tick

```
slider.setShowTickMarks(true);  
slider.setShowTickLabels(true);  
slider.setMajorTickUnit(1);
```

```
slider.setMinorTickCount(1);
```

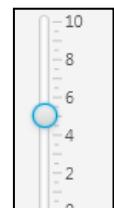
```
slider.setBlockIncrement(1);
```

```
slider.setSnapToTicks(true);
```



Orientamento (eventuale)

```
slider.setOrientation(  
    Orientation.VERTICAL);
```

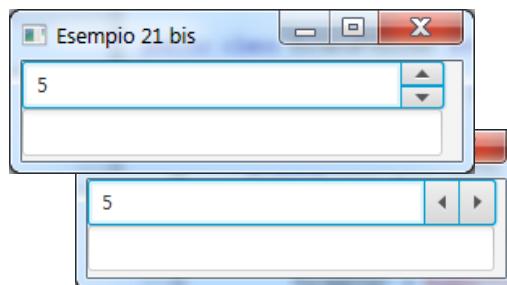




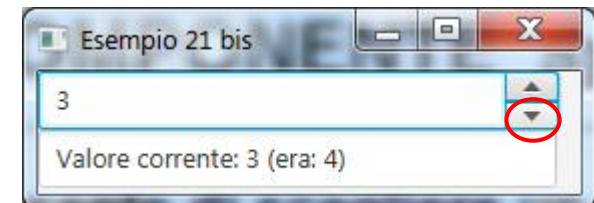
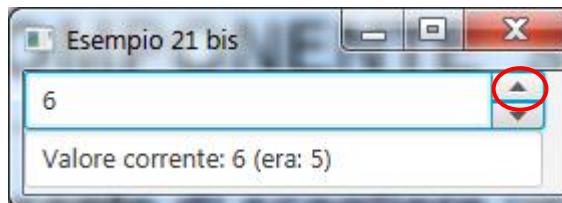
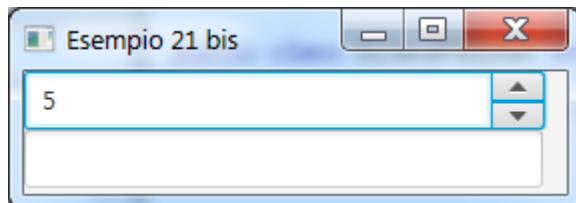
IL COMPONENTE Spinner

Lo **Spinner** consente di scegliere un valore muovendo freccine su/giù (o sx/dx), entro limiti predeterminati

- i valori possono essere interi, reali, liste o user-defined
- necessaria factory per produrre i valori (automatica per numeri e liste)
- eventuale listener sulla proprietà osservabile **valueProperty**



```
Spinner.STYLE_CLASS_ARROWS_ON_LEFT_HORIZONTAL  
Spinner.STYLE_CLASS_ARROWS_ON_LEFT_VERTICAL  
Spinner.STYLE_CLASS_ARROWS_ON_RIGHT_HORIZONTAL  
Spinner.STYLE_CLASS_ARROWS_ON_RIGHT_VERTICAL (default)  
Spinner.STYLE_CLASS_SPLIT_ARROWS_HORIZONTAL  
Spinner.STYLE_CLASS_SPLIT_ARROWS_VERTICAL
```





IL COMPONENTE Spinner

```
public class EsJavaFX21ter extends Application {  
    private Spinner<Integer> spinner; Tipizzato  
    private TextField txt1;  
  
    private NumberFormat formatter;  
  
    public void start(Stage stage) {  
        formatter = NumberFormat.getInstance();  
        formatter.setMaximumFractionDigits(2);  
        stage.setTitle("Esempio 21 ter");  
        FlowPane panel = new FlowPane();  
        txt1 = new TextField(); txt1.setEditable(false); Argomenti come in Slider  
        spinner = new Spinner(0, 10, 5); Eventualmente:  
        spinner.valueProperty().addListener( (changed, oldval, newval) ->  
            txt1.setText("Valore corrente: " + formatter.format(newval)  
                         + " " + formatter.format(oldval) + ")")  
    };  
    panel.c  
    Scene scene = new Scene(panel, 300, 250);  
    stage.setScene(scene);  
    stage.show();  
}
```

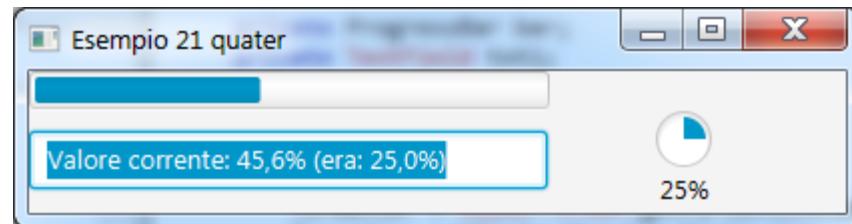
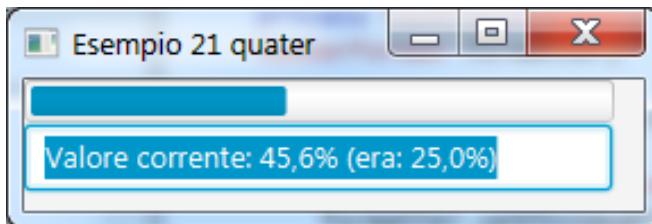


I COMPONENTI `ProgressBar` E `ProgressIndicator`

La `ProgressBar` indica a che punto si è arrivati (%)

- è per sua natura principalmente un dispositivo di output
- eventuale listener sulla proprietà osservabile `progressProperty`

Il `ProgressIndicator` fa lo stesso in un grafico a torta



```
ProgressBar bar = new ProgressBar(0.25);
ProgressIndicator pid = new ProgressIndicator(0.25);
bar.progressProperty().addListener( (changed, oldval, newval) ->
    txt1.setText("Valore corrente: " + formatter.format(newval)
    + " (era: " + formatter.format(oldval) + ")")
);
bar.setProgress(0.456);
```

Scatena aggiornamento proprietà

Formattatore percentuali

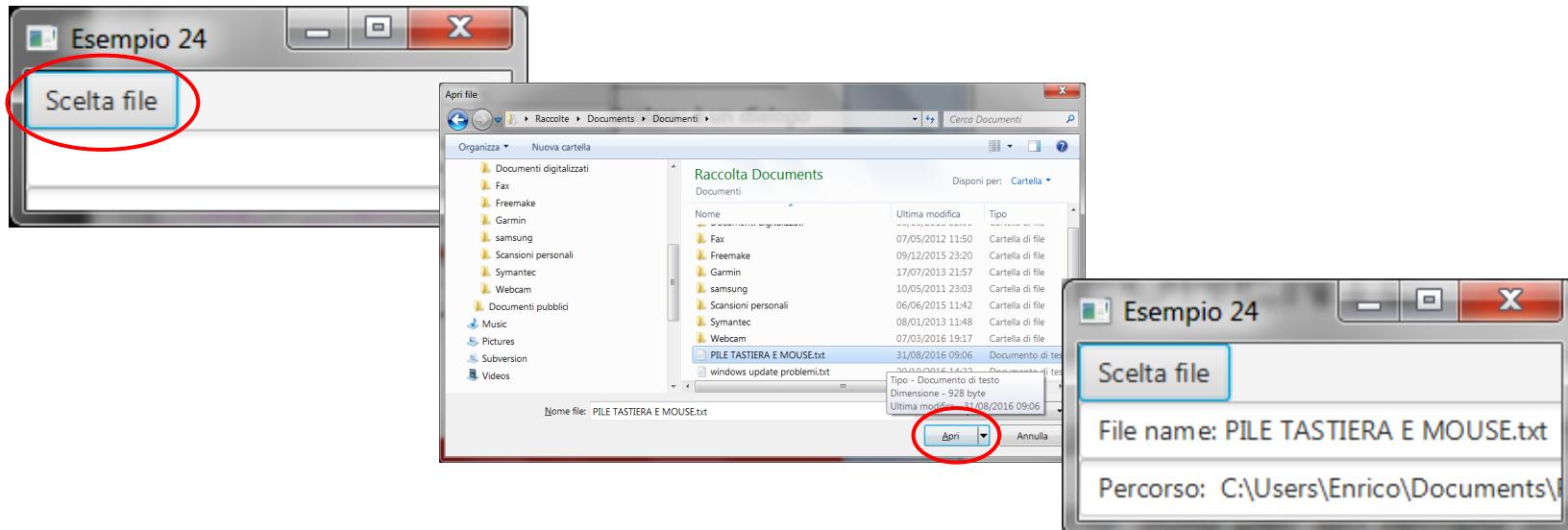


Un componente particolare: il dialogo FileChooser

IL COMPONENTE FileChooser

FileChooser è un componente particolare: è un *dialogo*

- i dialoghi *compaiono solo per un tempo limitato*, per supportare lo svolgimento di una data azione (non restano visibili stabilmente)
- **FileChooser** si usa per aprire/salvare un file
- solitamente lo si fa comparire in seguito alla pressione di un bottone





IL COMPONENTE FileChooser

```
public class EsJavaFX24 extends Application {  
    private FileChooser chooser;  
    private File selectedFile;  
    private Button button;  
    private TextField txt1, txt2;  
  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 24 ter");  
        FlowPane panel = new FlowPane();  
        button = new Button("Scelta file");  
        button.setOnAction( event -> {  
            chooser = new FileChooser();  
            chooser.setTitle("Apri file");  
            selectedFile = chooser.showOpenDialog(stage); // OPPURE showSaveDialog  
            txt1.setText("File name: " + selectedFile.getName());  
            txt2.setText("Percorso: " + selectedFile.getPath());  
        }  
    );  
    ...  
}
```

Si può scegliere fra dialogo di APERTURA o SALVATAGGIO file



IL COMPONENTE FileChooser

Si possono filtrare solo certi tipi di file:

```
chooser = new FileChooser();
chooser.setTitle("Apri file");

chooser.getExtensionFilters().addAll(
    new ExtensionFilter("Text Files", "*.txt"),
    new ExtensionFilter("Image Files", "*.png", "*.jpg", "*.gif"),
    new ExtensionFilter("Audio Files", "*.wav", "*.mp3", "*.aac"),
    new ExtensionFilter("All Files", "*.*"));
```

Si possono accettare selezioni multiple di file:

```
// selezione singola:
File selectedFile = chooser.showOpenDialog(stage);

// selezione multipla:
List<File> selectedFiles = chooser.showOpenMultipleDialog(stage);
```



IL COMPONENTE FileChooser

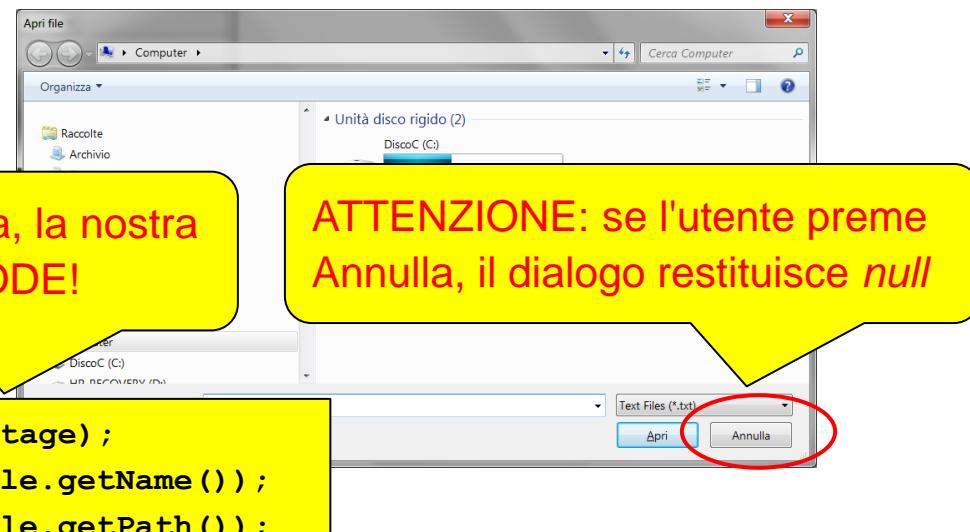
Completamento dell'esempio

```
public class EsJavaFX24 extends Application {  
    ...  
    txt1 = new TextField(); txt1.setEditable(false);  
    txt2 = new TextField(); txt2.setEditable(false);  
    panel.getChildren().addAll(button, txt1, txt2);  
    stage.setScene(new Scene(panel));  
    stage.setMaxWidth(270);  
    stage.setMinHeight(115);  
    stage.show();  
}  
}  
}
```

..e per come è fatta ora, la nostra gestione eventi ESplode!

ATTENZIONE: se l'utente preme Annulla, il dialogo restituisce *null*

```
selectedFile = chooser.showOpenDialog(stage);  
txt1.setText("File name: " + selectedFile.getName());  
txt2.setText("Percorso: " + selectedFile.getPath());
```



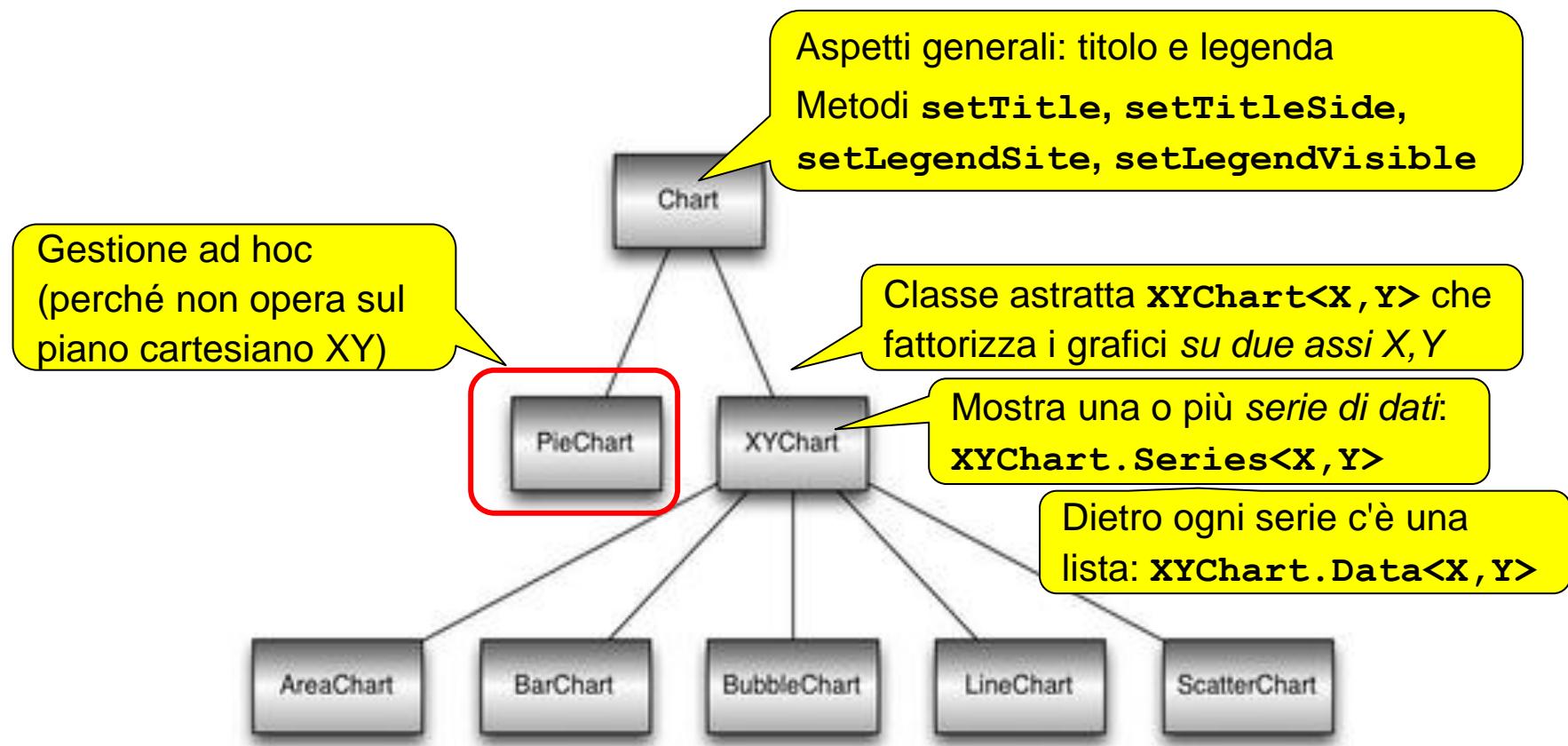


Grafici in JavaFX



THE Chart FAMILY

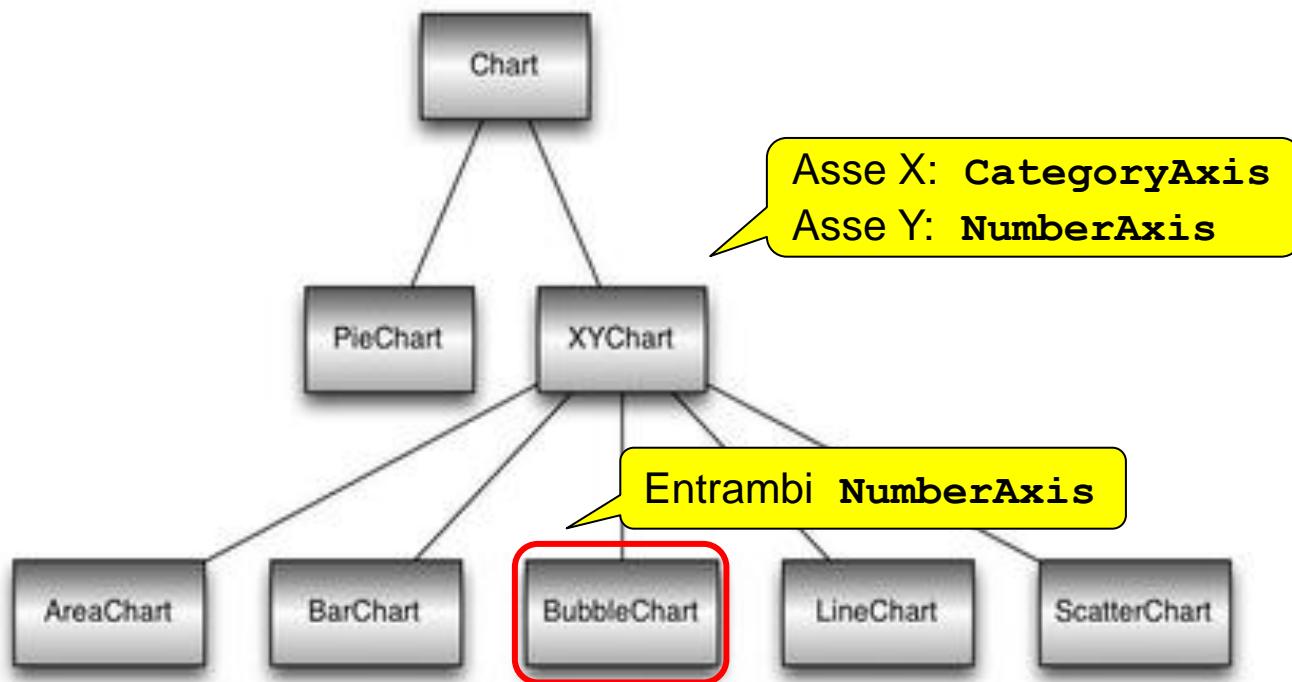
JavaFX offre un ampio ventaglio di grafici 2D



THE Chart FAMILY

Nei grafici XY, i due assi rappresentano

- l'asse orizzontale: categorie (numeri solo nel caso di **BubbleChart**)
- l'asse verticale: numeri





SCHEMA GENERALE

Passi da compiere

1. predisporre gli assi
2. creare l'oggetto Chart con quegli assi e fissarne il titolo
3. predisporre le serie di dati (una o più)
4. popolare ciascuna serie caricando le sue coppie (x,y) di dati
5. aggiungere al grafico le serie

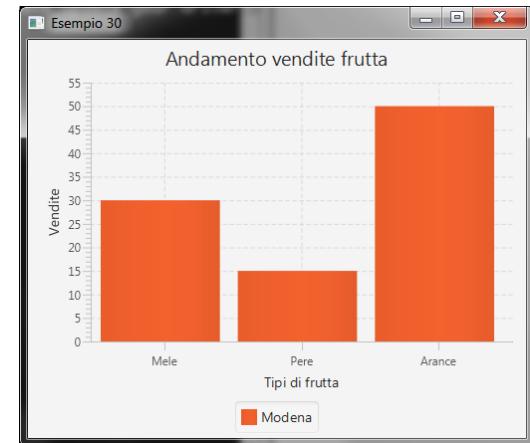
Esempio: vendite di frutta

- asse orizzontale: categorie (tipo di frutta: es. mele, pere, arance)
- asse verticale: valori (vendite di quel tipo di frutta, es. 25, 50, 23)
- una serie per ogni città (es. Modena, Bologna, Imola, Ferrara)
- ogni serie contiene i valori di frutta venduta in quella città
(es. modena → quantità di mele, pere, arance vendute a Modena)



IL COMPONENTE BarChart

```
public class EsJavaFX30 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 30");  
        FlowPane panel = new FlowPane();  
        CategoryAxis asseOrizz = new CategoryAxis();  
        asseOrizz.setLabel("Tipi di frutta");  
        NumberAxis asseVert = new NumberAxis();  
        asseVert.setLabel("Vendite");  
        Tipizzato  
        BarChart<String,Number> chart = new BarChart<>(asseOrizz,asseVert);  
        chart.setTitle("Andamento vendite frutta");  
        XYChart.Series<String,Number> modena = new XYChart.Series<>();  
        Una serie  
        modena.setName("Modena");  
        Popolamento serie  
        coi dati di vendita  
        modena.getData().add( new XYChart.Data<>("Mele", 30));  
        modena.getData().add( new XYChart.Data<>("Pere", 15));  
        modena.getData().add( new XYChart.Data<>("Arance", 50));  
        Aggiunta serie (popolata) al grafico  
        chart.getData().add(modena);  
        panel.getChildren().add(chart);  
        stage.setScene(new Scene(panel));  
    }  
}
```





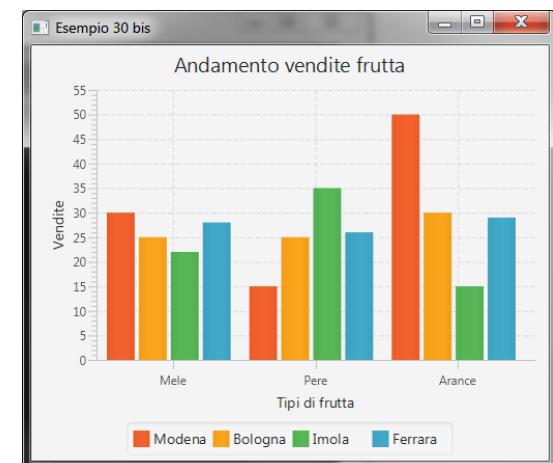
IL COMPONENTE BarChart

Variante con più serie

```
public class EsJavaFX30bis extends Application {  
    public void start(Stage stage) {  
        ...  
        XYChart.Series<String,Number> modena = new XYChart.Series<>();  
        XYChart.Series<String,Number> bologna = new XYChart.Series<>();  
        XYChart.Series<String,Number> imola = new XYChart.Series<>();  
        XYChart.Series<String,Number> ferrara = new XYChart.Series<>();  
        ... // popolamento di tutte le quattro serie  
        chart.getData().add(modena);  
        chart.getData().add(bologna);  
        chart.getData().add(imola);  
        chart.getData().add(ferrara);  
        panel.getChildren().add(chart);  
        stage.setScene(new Scene(panel));  
    }  
}
```

Quattro serie

Aggiunta delle quattro serie al grafico

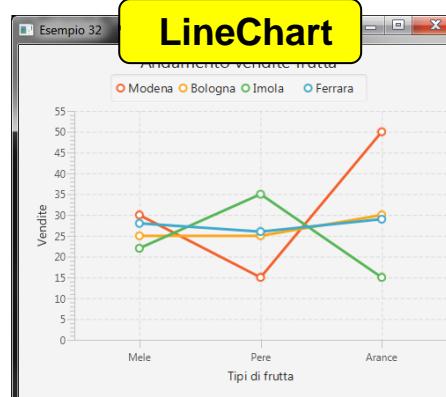
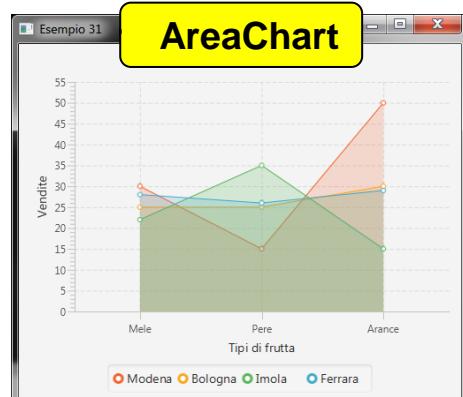
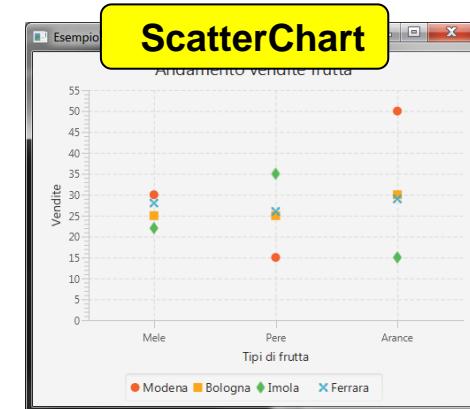
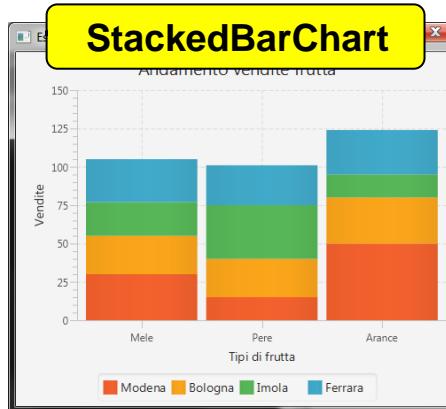
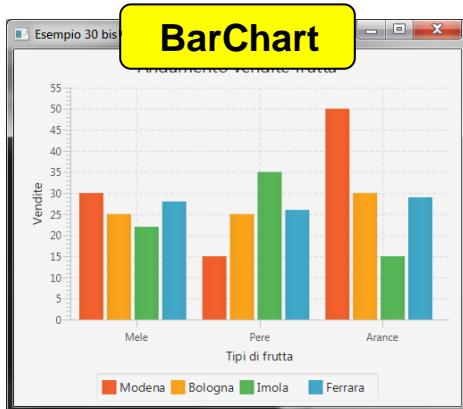




ALTRI TIPI DI GRAFICI

Per cambiare tipo di grafico basta creare un diverso chart

```
BarChart<String,Number> chart = new BarChart<>(asseOrizz,asseVert);
```





IL COMPONENTE BubbleChart

Questo chart richiede assi *numerici* (esempio: funzioni)

```
public class EsJavaFX34 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 34");  
        FlowPane panel = new FlowPane();  
  
        NumberAxis asseX = new NumberAxis(); asseX.setLabel("x");  
        NumberAxis asseY = new NumberAxis(); asseY.setLabel("y");  
        BubbleBarChart<Number,Number> chart = new BubbleBarChart<>(asseX,asseY);  
        chart.setTitle("Funzioni varie");  
  
        XYChart.Series<Number,Number> parabola = new XYChart.Series<>();  
        parabola.setName("parabola");  
        parabola.getData().add( new XYChart.Data<>(0,0));  
        parabola.getData().add( new XYChart.Data<>(10,10));  
        parabola.getData().add( new XYChart.Data<>(15,22));  
        parabola.getData().add( new XYChart.Data<>(20,40));  
        parabola.getData().add( new XYChart.Data<>(30,90));  
  
        ...  
    }  
}
```

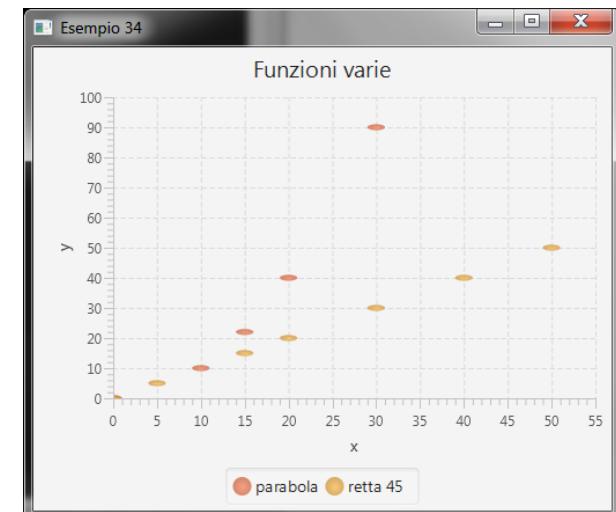


IL COMPONENTE BubbleChart

...

```
retta45.setName("retta 45");
retta45.getData().add( new XYChart.Data<>(0,0));
retta45.getData().add( new XYChart.Data<>(5,5));
retta45.getData().add( new XYChart.Data<>(15,15));
retta45.getData().add( new XYChart.Data<>(20,20));
retta45.getData().add( new XYChart.Data<>(30,30));
retta45.getData().add( new XYChart.Data<>(40,40));
retta45.getData().add( new XYChart.Data<>(50,50));

chart.getData().add(parabola);
chart.getData().add(retta45);
panel.getChildren().add(chart);
stage.setScene(new Scene(panel));
}
```





PieChart: SCHEMA GENERALE

Passi da compiere

1. non ci sono assi: è un grafico a torta!
2. prima si predisponde la serie di dati (una sola)
3. poi si crea l'oggetto PieChart con quella serie, e se ne fissa il titolo

Esempio: le solite vendite di frutta

- una sola serie = una sola città (es. Modena)
- la serie contiene i valori di frutta venduta a Modena



IL COMPONENTE PieChart

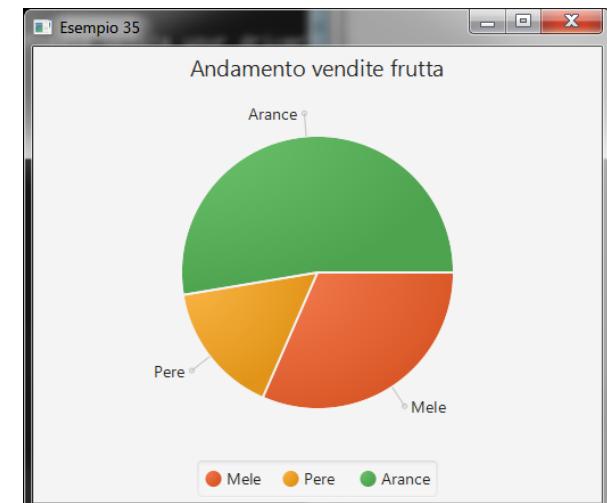
Questo chart richiede assi *numerici* (esempio: funzioni)

```
public class EsJavaFX35 extends Application {  
    public void start(Stage stage) {  
        stage.setTitle("Esempio 35");  
        FlowPane panel = new FlowPane();  
        ObservableList<PieChart.Data> dati =  
            FXCollections.observableArrayList(  
                new PieChart.Data("Mele", 30),  
                new PieChart.Data("Pere", 15),  
                new PieChart.Data("Arance", 50)  
            );  
        PieChart chart = new PieChart(dati);  
        chart.setTitle("Andamento vendite frutta");  
        panel.getChildren().add(chart);  
        stage.setScene(new Scene(panel));  
    }  
}
```

Pre-popolamento serie sotto forma di
ObservableList di PieChart.Data

NON tipizzato

Una serie

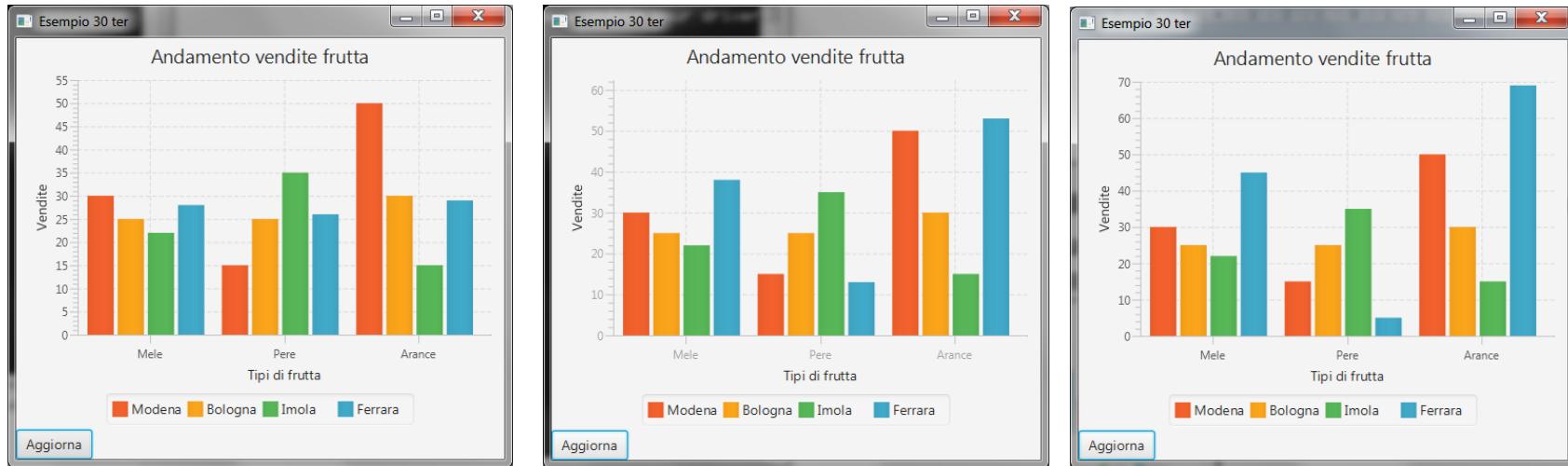




GRAFICI ANIMATI

Le serie dati possono essere modificate e, volendo, le modifiche possono essere *animate*.

Per vederlo, aggiungiamo un bottone che cambia i dati di Ferrara:



```
chart.setAnimated(true);

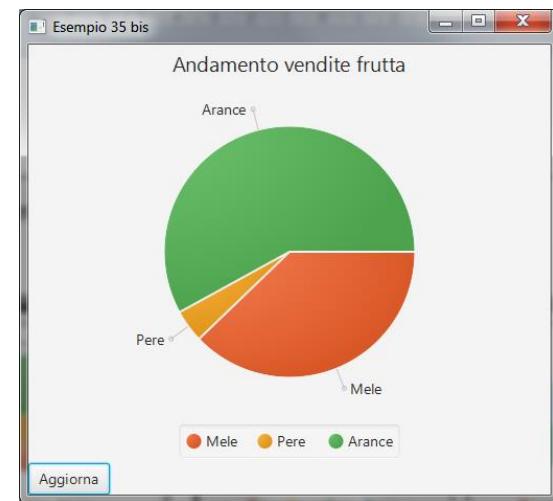
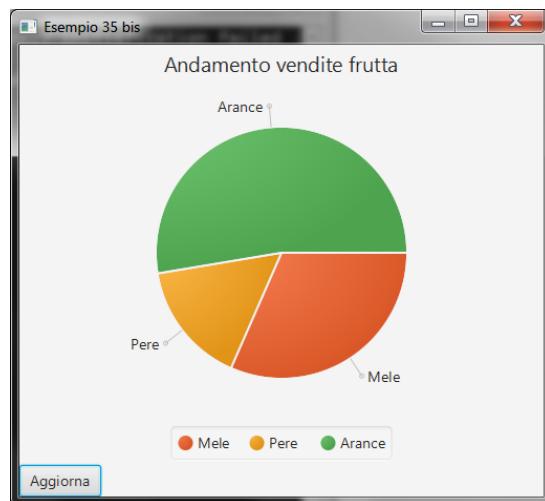
Button button = new Button("Aggiorna");

button.setOnAction(event -> {
    ferrara.getData().get(0).setYValue(45); // era 28
    ferrara.getData().get(1).setYValue(5); // era 26
    ferrara.getData().get(2).setYValue(69); // era 29
});
```



GRAFICI ANIMATI

Analogamente si può fare col grafico a torta:



```
chart.setAnimated(true);  
Button button = new Button("Aggiorna");  
button.setOnAction(event -> {  
    dati.get(0).setPieValue(45); // era 30  
    dati.get(1).setPieValue(5); // era 15  
    dati.get(2).setPieValue(69); // era 50  
});
```



Strumenti per il progetto e la definizione di GUI

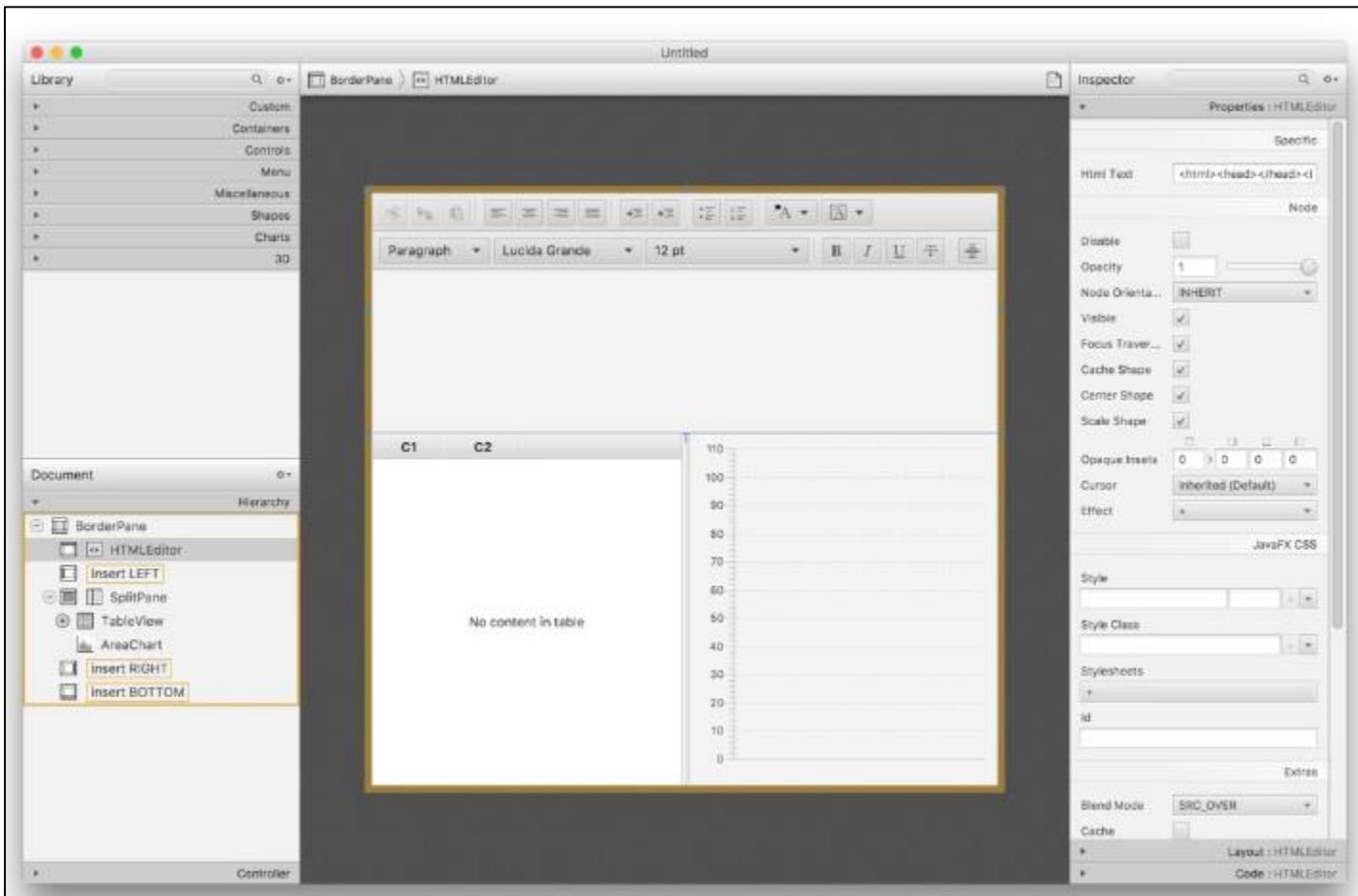


OLTRE IL DISEGNO MANUALE

- In realtà, nessuno disegna a codifica GUI "a mano"
 - i componenti non appaiono come/dove si vorrebbe
 - il layout non è come desiderato
 - *necessità di iterare più volte lo sviluppo*
- Si utilizzano appositi *strumenti visuali*
 - si disegna la GUI e si vede subito "come viene"
 - editing diretto delle proprietà grafiche
 - GUI *descritta tramite file (F)XML anziché cablata nel codice*
- JavaFX è supportato dallo strumento *SceneBuilder*
 - usabile da Eclipse, ma va installato a parte
 - scaricare da <http://gluonhq.com/labs/scene-builder/>
 - poi: www.eclipse.org/efxclipse/install.html



SCENE BUILDER





GUI DESCRITTE VIA FXML

- Questi strumenti però adottano un NUOVO APPROCCIO
 - la *parte strutturale* della GUI non è più "programmata" in Java:
è descritta in un particolare formato testo, chiamato *FXML*
 - il codice Java viene usato solo per la *parte comportamentale*
 - *massima separazione* fra *struttura* e *comportamento*
 - massimo riuso su più piattaforme (Java SE, Android...)
- Architettura dell'applicazione
 - il file FXML **descrive la GUI** in termini di widget → **SceneBuilder**
 - il file FXML **specifica la classe** che funge da **controller**
 - il **controller Java** contiene la *business logic* (gestione eventi)
 - il **main Java** carica la GUI e inizializza il controller.



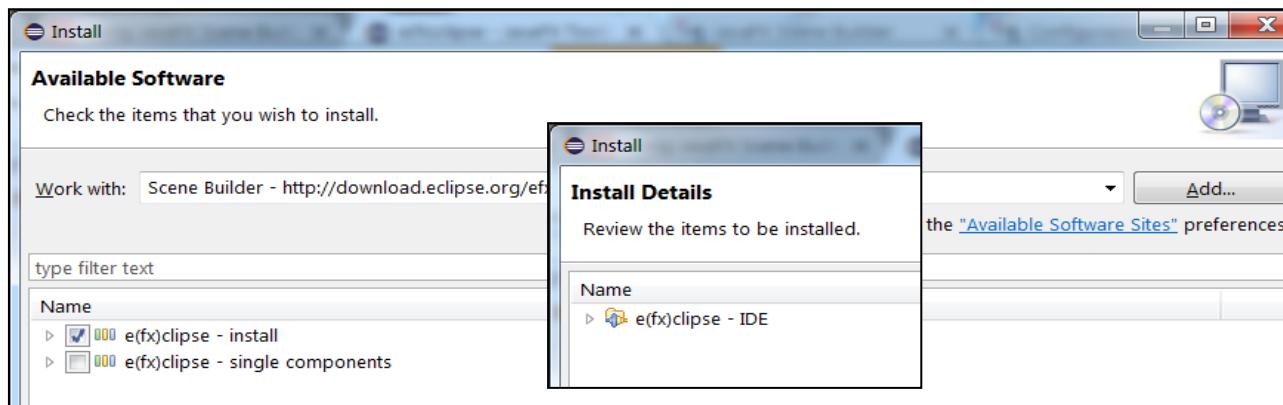
INSTALLAZIONE di SCENE BUILDER

- Scene Builder non è incluso in Eclipse di default
 - va scaricato da Gluon e installato per conto proprio
 - poi, va installato il supporto JavaFX in Eclipse
 - a tal fine: andare in *Eclipse > Help > Install New Software*
 - aggiungere i necessari *update sites*

<http://download.eclipse.org/efxclipse/updates-released/2.4.0/site>

<http://download.eclipse.org/modeling/tmf/xtext/updates/composite/releases/>

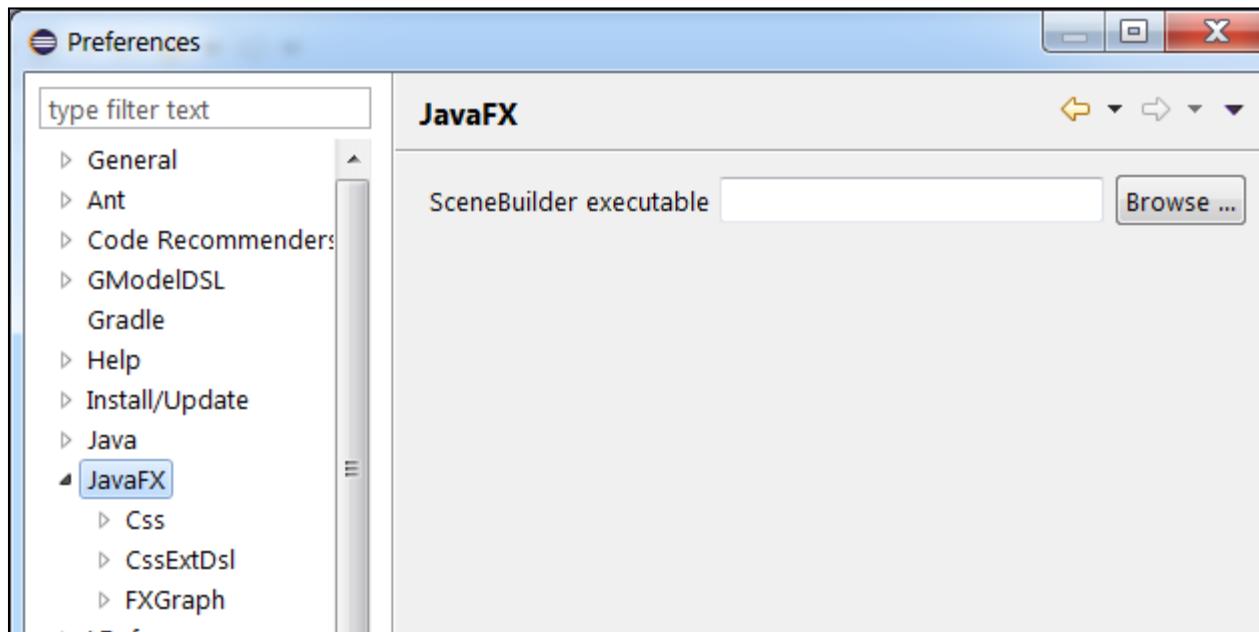
- installare





CONFIGURAZIONE di SCENE BUILDER

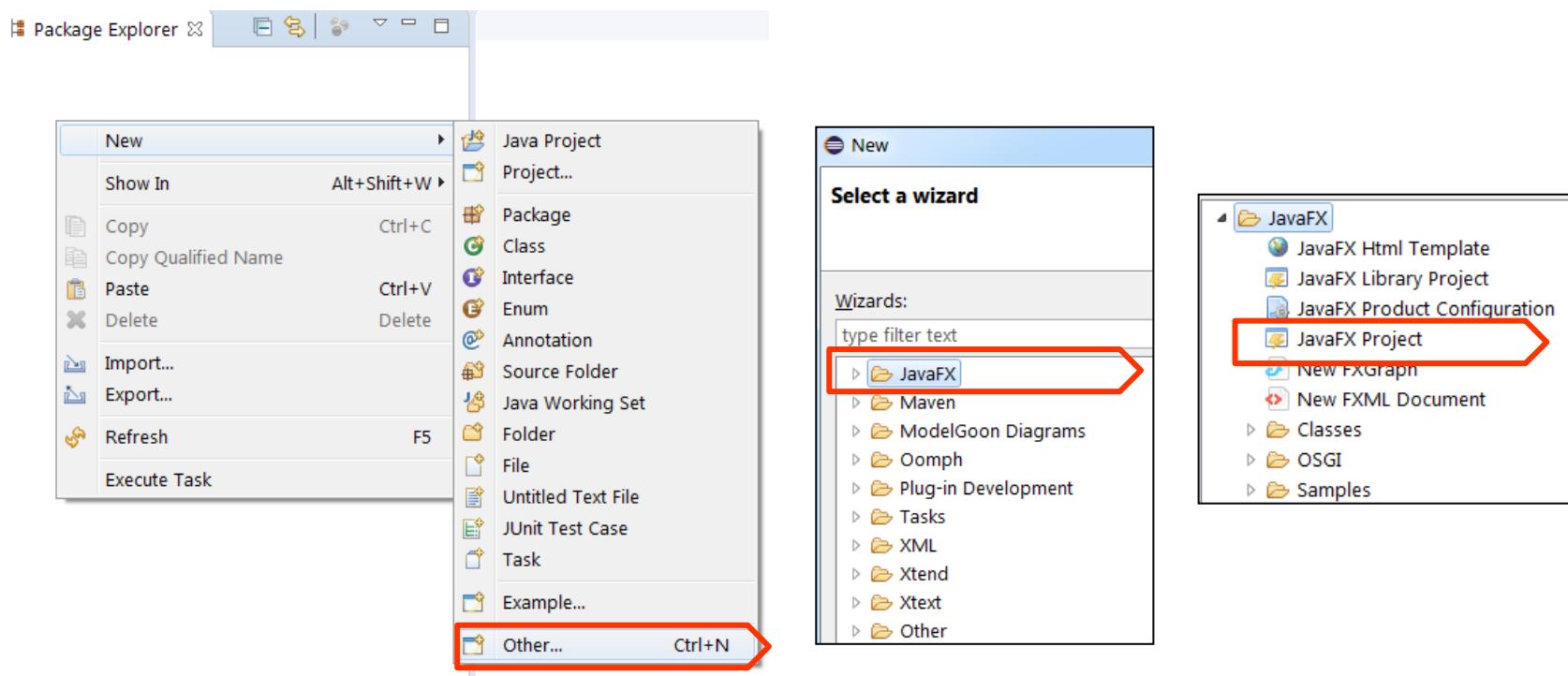
- Prima di essere usato, Scene Builder va configurato
 - *Eclipse >> Window >> Preferences >> JavaFX*
 - sfogliare alla ricerca dell'eseguibile
`C:\Users\Enrico\AppData\Local\SceneBuilder`





SCENE BUILDER (1)

1. New > Java Project
2. New > Other > JavaFX > Application
3. scegliere il nome del package, quello della classe e poi.. via!





SCENE BUILDER (2)

Viene generata l'application, con relativo main (e file di configurazione)



Possiamo sbirciare il main di base,
ma è solo un modello standard
che dovremo modificare.

Il bello deve ancora venire.

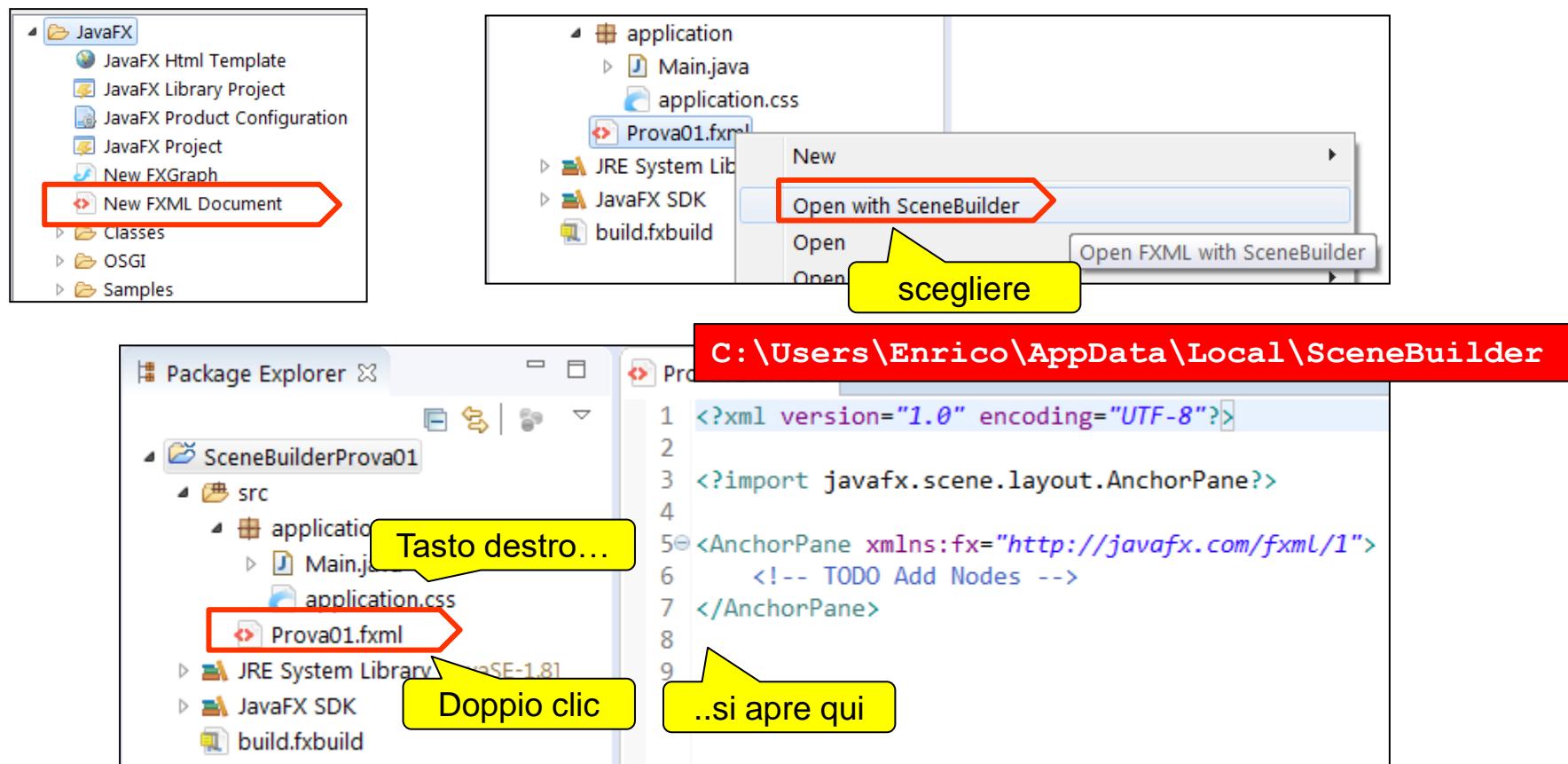
The screenshot shows the Eclipse IDE's code editor with the 'Main.java' file open. The code is as follows:

```
1 package application;
2
3 import javafx.application.Application;
4
5 public class Main extends Application {
6     @Override
7     public void start(Stage primaryStage) {
8         try {
9             BorderPane root = new BorderPane();
10            Scene scene = new Scene(root,400,400);
11            scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());
12            primaryStage.setScene(scene);
13            primaryStage.show();
14        } catch(Exception e) {
15            e.printStackTrace();
16        }
17    }
18}
```



SCENE BUILDER (3)

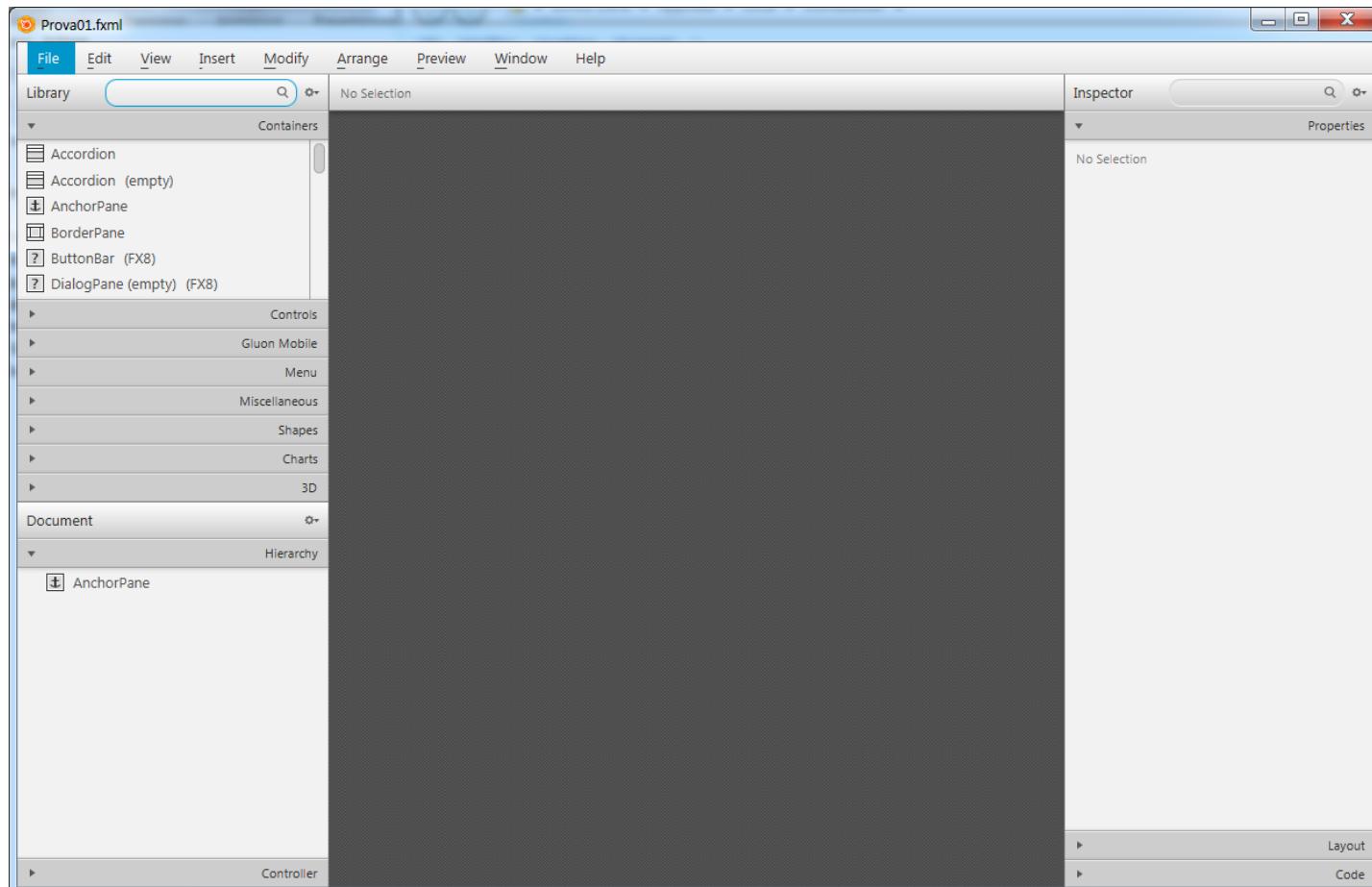
Per usare lo strumento visuale, dobbiamo prima creare il file FXML destinato a contenere la descrizione della GUI. Fatto ciò...





SCENE BUILDER (4)

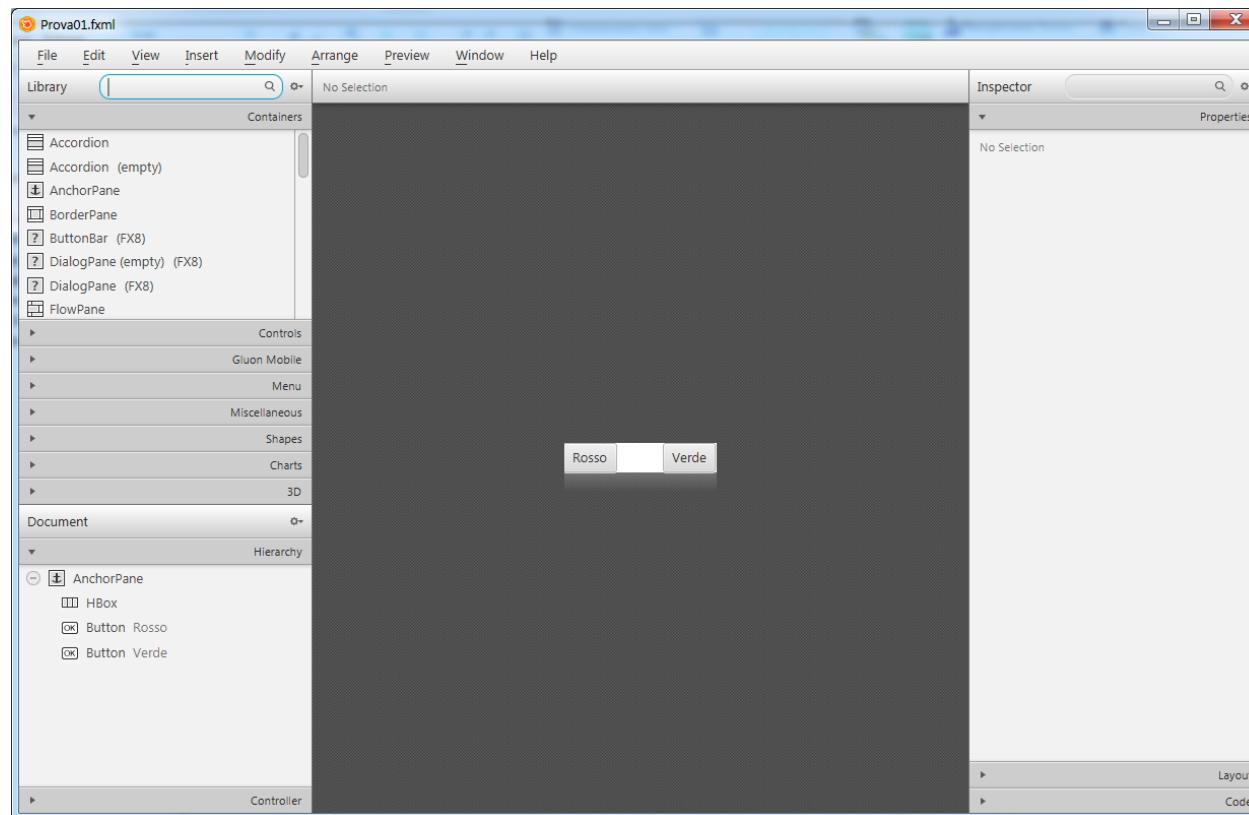
Ora, cliccando col tasto destro sul file FXML, si apre Scene Builder





UN'APP CON SCENE BUILDER (1)

Applicazione Rosso/Verde: costruiamo un **HBox** con due pulsanti e una **Region** (trasparente e vuota) fra loro, per tenerli distanziati:





UN'APP CON SCENE BUILDER (2)

Riaprendolo in Eclipse, vediamo la *descrizione della GUI in FXML*

The screenshot shows the Eclipse IDE interface with the title bar "SceneBuilder - Java - SceneBuilderProva01/src/Prova01.fxml - Eclipse". The menu bar includes File, Edit, Source, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations. The left sidebar is the "Package Explorer" showing the project structure:

- Event Handling Examples
- SceneBuilderProva01 (selected)
 - src
 - application
 - Main.java
 - Main
 - application.css
 - Prova01.fxml
 - JRE System Library [JavaSE-1.8]
 - JavaFX SDK
 - build.fxbuild

The main editor area displays the FXML code for "Prova01.fxml":

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.Region?>
<AnchorPane xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1" >
    <children>
        <HBox minWidth="154.0" prefHeight="29.0" prefWidth="154.0" style="-fx-background-color: white;" >
            <Button layoutX="-1.0" mnemonicParsing="false" text="Rosso" />
            <Region opacity="0.0" prefHeight="22.0" prefWidth="50.0" />
            <Button layoutX="97.0" mnemonicParsing="false" text="Verde" />
        </HBox>
    </children>
</AnchorPane>
```

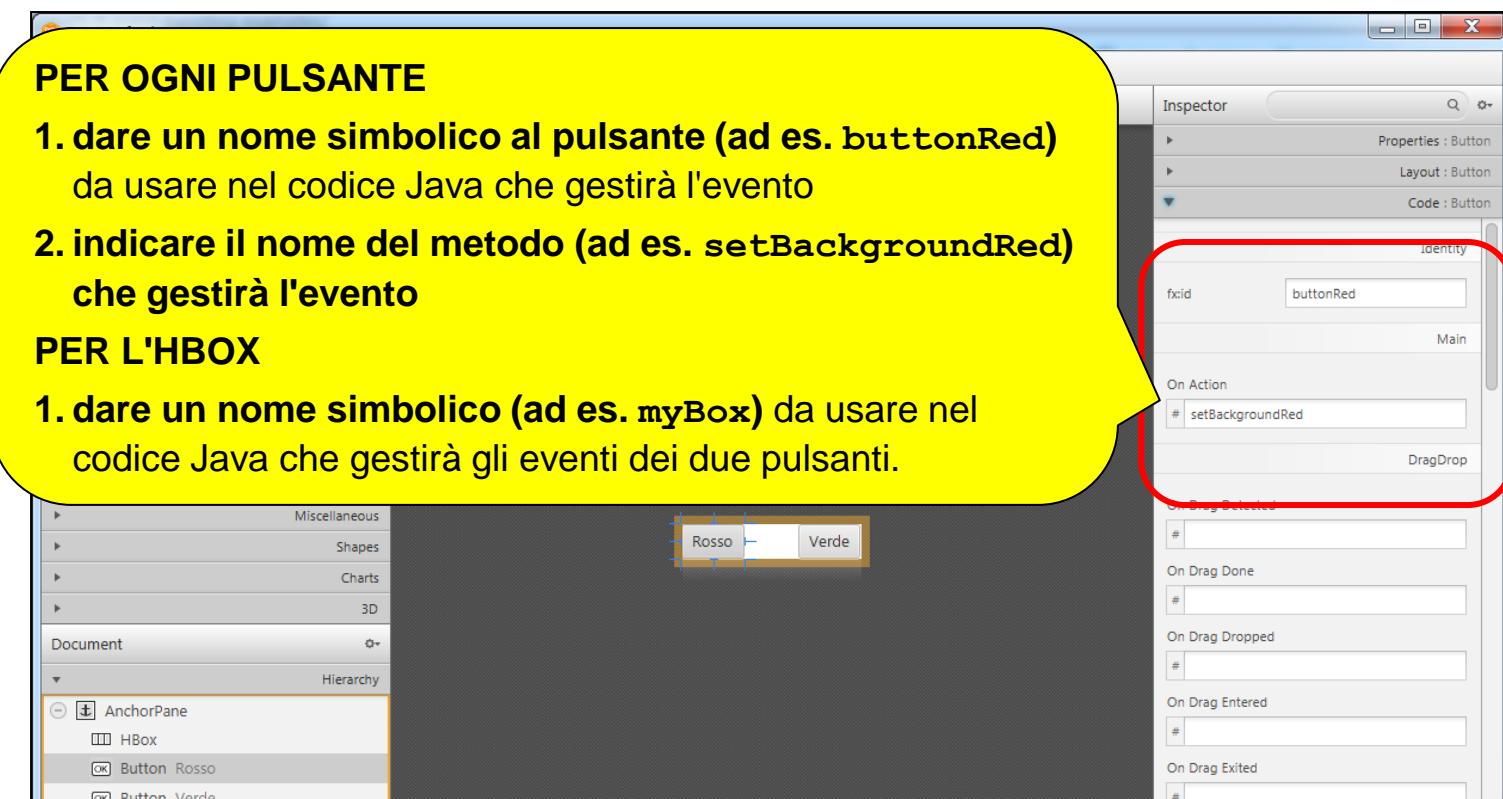
Per poter agganciare alla GUI la *business logic* (gestione eventi), occorre:

- dare un **nome simbolico** ai **singoli componenti**
(in modo da poterli poi referenziare nel codice dell'ascoltatore)
- indicare il **nome del metodo** dell'ascoltatore che gestirà l'evento



UN'APP CON SCENE BUILDER (3)

Per aggiungere i nomi simbolici, fare clic via via sui vari elementi e per ciascuno scegliere (nel menù a destra) la **scheda Code**:





UN'APP CON SCENE BUILDER (4)

Le modifiche fatte in Scene Builder si riflettono nel file FXML:

- i nomi inseriti appaiono come *attributi*, rispettivamente nella forma **fx:id="nome"** per i nomi pulsanti e **onAction="nome"** per i metodi.
- in alternativa, tali nomi si possono aggiungere direttamente come testo del file FXML, senza far uso di Scene Builder.

Questi nomi costituiscono gli agganci (**hook**) fra la *parte strutturale* dell'applicazione (descritta nel file FXML) e la *parte comportamentale* (scritta in Java)

```
<?xml version="1.0" encoding="UTF-8"?>
<AnchorPane xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1">
    <children>
        <HBox fx:id="myBox" minWidth="154.0" prefHeight="29.0" prefWidth="154.0" style="-fx-background-color: white;">
            <Button fx:id="buttonRed" layoutX="-1.0" mnemonicParsing="false" onAction="#setBackgroundRed" text="Rosso" />
            <Region opacity="0.0" prefHeight="22.0" prefWidth="50.0" />
            <Button fx:id="buttonGreen" layoutX="97.0" mnemonicParsing="false" onAction="#setBackgroundGreen" text="Verde" />
        </HBox>
    </children>
</AnchorPane>
```



UN'APP CON SCENE BUILDER (5)

La business logic dev'essere contenuta in una classe *controller*

- la GUI deve sapere chi sia il suo controller: lo specifica l'attributo **fx:controller="nome"** dell'elemento di top-level (**AnchorPane**)
- naturalmente, anche in questo caso l'attributo può essere impostato sia tramite Scene Builder sia direttamente nel testo FXML.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2
3 <?import javafx.scene.control.Button?>
4 <?import javafx.scene.layout.AnchorPane?>
5 <?import javafx.scene.layout.HBox?>
6 <?import javafx.scene.layout.Region?>
7
8<AnchorPane xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1" fx:controller="application.MyController">
9<children>
10<HBox fx:id="myBox" minWidth="154.0" prefHeight="29.0" prefWidth="154.0" style="-fx-background-color: white;">
11<Button fx:id="buttonRed" layoutX="-1.0" mnemonicParsing="false" onAction="#setBackgroundRed" text="Rosso" />
12<Region opacity="0.0" prefHeight="22.0" prefWidth="50.0" />
13<Button fx:id="buttonGreen" layoutX="97.0" mnemonicParsing="false" onAction="#setBackgroundGreen" text="Verde" />
14</HBox>
15</children>
16</AnchorPane>
17|
```

Ora dobbiamo implementare in Java questo controller.



UN'APP CON SCENE BUILDER (6)

Il controller è una comune classe di nostra scelta

- apposite **annotazioni** della forma `@FXML` identificano i widget che il controller *dichiara ma non costruisce*, in quanto specificati via FXML

The screenshot shows the Eclipse IDE interface with the Package Explorer and MyController.java editor tabs. The Package Explorer shows a project structure with a src folder containing application, Main.java, MyController.java, and application.css. A Prova01.fxml file is also present. The MyController.java file contains Java code for a controller class. Annotations @FXML are highlighted with red boxes. A yellow callout points to the annotations in the code.

```
1 package application;
2
3 import javafx.event.ActionEvent;
4 import javafx.fxml.FXML;
5 import javafx.scene.layout.HBox;
6
7 public class MyController {
8
9     @FXML
10    private HBox myBox;
11
12    @FXML
13    public void setBackgroundRed(ActionEvent event) {
14        myBox.setStyle("-fx-background-color: red;");
15    }
16    @FXML
17    public void setBackgroundGreen(ActionEvent event) {
18        myBox.setStyle("-fx-background-color: green;");
19    }
20 }
```

Sono i nomi specificati prima nell'attributo `onAction`

Usa `myBox` senza crearlo



UN'APP CON SCENE BUILDER (7)

Per finire, resta da sistemare il **Main**, che deve:

- caricare il file **FXML** (occhio al path!!)
- recuperare il nodo radice (AnchorPane) e impostarlo sulla scena

```
public void start(Stage primaryStage) {  
    try {  
        FXMLLoader loader = new FXMLLoader(Main.class.getResource("/Prova01.fxml"));  
        AnchorPane page = (AnchorPane) loader.load();  
        Scene scene = new Scene(page);  
        scene.getStylesheets().add(getClass().getResource("application.css").toExternalForm());  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    } catch(Exception e) {  
        e.printStackTrace();  
    }  
}
```

And the winner is... 😊





UN'APP CON SCENE BUILDER (8)

In alternativa, si può non specificare nella GUI il gestore dell'evento, lasciando al metodo **initialise** del controller (chiamato automaticamente da JavaFX) il compito di collegarli:

```
<AnchorPane xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://javafx.com/fxml/1" fx:controller="application.MyController">
    <children>
        <HBox fx:id="myBox" minWidth="154.0" prefHeight="29.0" prefWidth="154.0" style="-fx-background-color: white;">
            <Button fx:id="buttonRed" layoutX="-1.0" mnemonicParsing="false" text="Rosso" />
            <Region opacity="0.0" prefHeight="22.0" prefWidth="50.0" />
            <Button fx:id="buttonGreen" layoutX="97.0" mnemonicParsing="false" text="Verde" />
        </HBox>
    </AnchorPane>
```

MyController.java

```
1 package application;
2
3 import javafx.fxml.FXML;
4
5 public class MyController {
6
7     @FXML
8     private Button buttonRed, buttonGreen;
9
10    @FXML
11    private HBox myBox;
12
13    /**
14     * Initializes the controller class. This method is automatically called
15     * after the fxml file has been loaded.
16     */
17    @FXML
18    private void initialize() {
19        buttonRed.setOnAction( event -> myBox.setStyle("-fx-background: red;") );
20        buttonGreen.setOnAction( event -> myBox.setStyle("-fx-background: green;") );
21    }
22}
23
24
25}
```

Tolti gli attributi **onAction**

Classiche lambda expression



UN'APP CON SCENE BUILDER (9)

Giochino finale:
cambiamo il colore dei pulsanti ogni volta che vengono premuti

```
@FXML  
private void initialize() {  
    buttonRed.setOnAction( event -> {  
        myBox.setStyle("-fx-background-color: red;");  
        buttonRed.setStyle("-fx-background-color: " + changeColorTo());  
    });  
    buttonGreen.setOnAction( event -> {  
        myBox.setStyle("-fx-background-color: green;");  
        buttonGreen.setStyle("-fx-background-color: " + changeColorTo());  
    });  
  
}  
  
private String changeColorTo(){  
    int r,g,b;  
    java.util.Random gen = new java.util.Random();  
    r = gen.nextInt(256); g = gen.nextInt(256); b = gen.nextInt(256);  
    String hexcolor = String.format( "#%02X%02X%02X", r,g,b);  
    System.out.println(hexcolor);  
    return hexcolor;  
}
```

Sintetizza una stringa-colore
RGB in formato esadecimale

