

INTRODUZIONE A JAVA FX

Angelo Di Iorio

Università di Bologna

Graphical User Interface - GUI

- Il termine **Graphical User Interface (GUI)** indica la parte di un programma che interagisce con l'utente, attraverso elementi grafici quali finestre, bottoni, menù, etc.
- Terminologia di uso comune e molto diffusa:
 - Finestra (*Window*)
 - Pannello (*Panel*)
 - Menù (*Menu*)
 - Pulsante (*Button*)
 - Etichetta (*Label*)
- Condivisa da applicazioni stand-alone, Web, mobile, etc.

Programmazione a Eventi

- Nelle interfacce grafiche il termine **evento** (**event**) indica un'azione dell'utente sull'interfaccia come la pressione di un bottone, il click del mouse in una determinata area, un'operazione di *drag&drop*, operazioni sulle finestre, etc.
- La **programmazione** di una **GUI** si basa su una corretta e completa **gestione degli eventi**
- La programmazione a eventi è peculiare:
 - Nella maggior parte dei casi il programmatore non sa l'ordine in cui saranno eseguite le varie parti del programma, che dipende appunto dall'ordine degli eventi
 - Il programmatore crea oggetti che generano eventi o che reagiscono ad eventi

Programmazione a Eventi

- Gli eventi generati su un'interfaccia sono moltissimi, a diversi livelli:
 - *Low level*: pressione tastiera, movimento mouse, spostamento cursore, etc.
 - *“Semantic” level*: click su un bottone, doppio click su un oggetto, resize di una finestra, etc.
- Necessario poter distinguere gli eventi e gestirne solo alcuni
- I sistemi che supportano la gestione degli eventi – non solo in Java – si occupano di propagare gli eventi e forniscono gli strumenti per *catturarli e reagire*
- Il programmatore si occupa di indicare quali eventi catturare e quali metodi invocare quando si verificano (non li invoca direttamente!)

Eventi: terminologia

- **Source:**

- Oggetto che genera (o emette) eventi
- Notifica l'evento ai Listener registrati
- Ogni sorgente genera un insieme ben definito di eventi

- **Listener/Handler:**

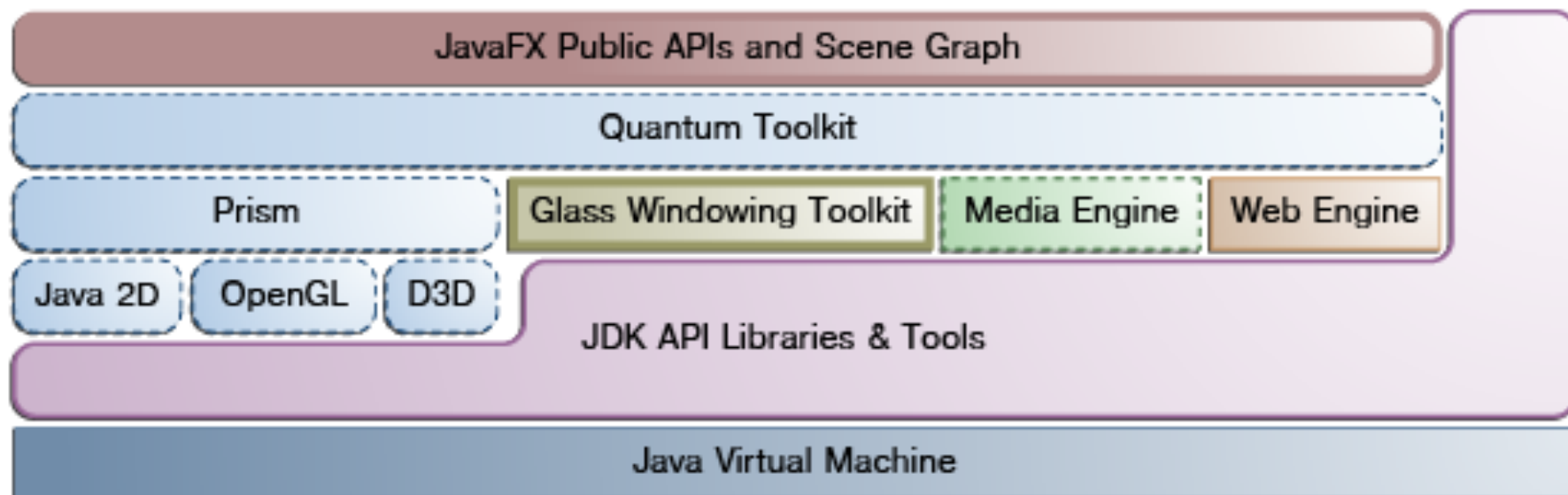
- Oggetto che viene notificato quando si scatena un evento ed esegue azioni in risposta ad un evento
- Si potrebbero separare ulteriormente il Listener dall'Handler ma spesso sono sovrapposti

- Le applicazioni registrano su ciascuna componente (*Sorgente*) solo i *Listeners* del tipo di evento (o più di uno) rilevante/i

Java Swing e Java FX

- In Java il sistema Swing (e la sua evoluzione Java FX) permettono di creare GUI basate un set predefinito di oggetti grafici ed eventi di diversa complessità e sofisticazione
 - Java FX: <https://openjfx.io/>
 - Java Swing: <https://docs.oracle.com/javase/tutorial/uiswing/index.html>
- Parleremo di Java FX che è ormai lo standard per GUI in Java ma i concetti *core* non cambiano e Swing è ancora disponibile e utilizzato
- Concetti e strumenti simili si ritrovano in diversi framework in altri linguaggi di programmazione
 - Es: Qt in C++ (https://wiki.qt.io/About_Qt) o TkInter in Python (<https://wiki.python.org/moin/TkInter>)

Architettura



GUI ed ereditarietà

- La programmazione a eventi di Java FX usa estensivamente l'ereditarietà
- Il programmatore definisce classi derivate da classi della libreria e fa overriding di alcuni metodi
- In particolare si occupa di definire gli handler degli eventi
- La libreria Java FX si occupa di:
 - eseguire l'applicazione
 - gestire le interazioni con gli utenti
 - chiuderla alla fine dell'esecuzione

Hello World (minimale) in Java FX

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

public class HelloWorld extends Application {

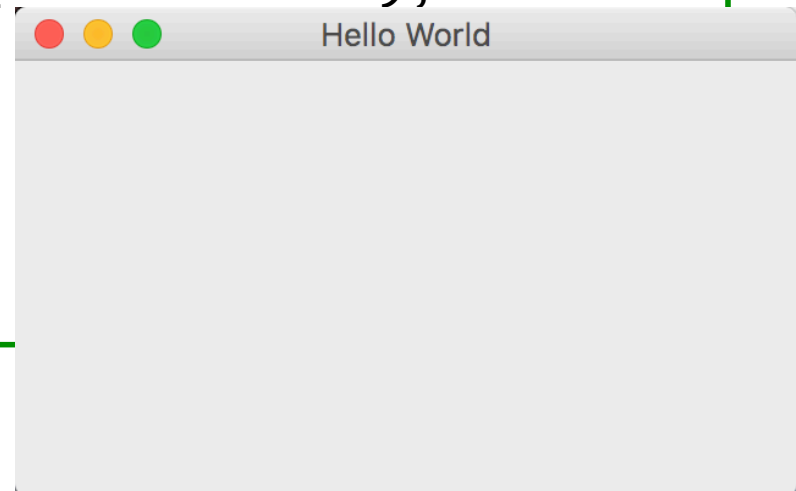
    @Override
    public void start(Stage primaryStage) {

        primaryStage.setTitle("Hello World");

        primaryStage.show();

    }

}
```



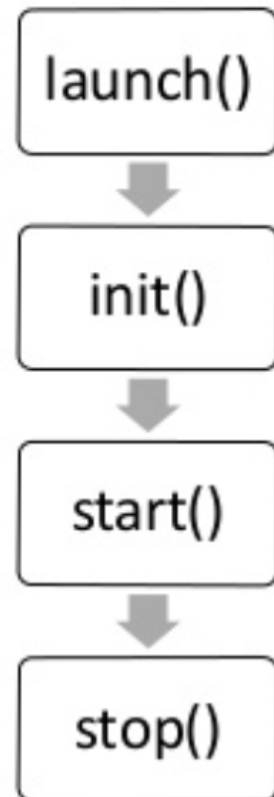
<http://diiorio.nws.cs.unibo.it/asd1819/java/gui/>

Java FX Application e Stage

- Un'applicazione JavaFX estende la classe `javafx.application.Application`
- La classe espone un metodo astratto eseguito al lancio dell'applicazione:
`public void start(Stage primaryStage)`
- Il parametro `primaryStage` indica lo `Stage` (palcoscenico) su cui si svilupperà l'intera applicazione organizzata in scene (`Scene`)
- Ogni applicazione ha uno `Stage` primario su cui è possibile alternare diverse scene
- E' possibile creare nuovi `Stage` e mostrarli come per lo stage primario

Threading, launch() e start()

- L'applicazione viene lanciata anche se non è presente il metodo `main()`. Si può lanciare esplicitamente invocando il metodo statico `launch()`
- JavaFX costruisce un'istanza della classe `Application` e crea un **thread** separato per eseguire `start()`
 - `start()` DEVE essere implementato (è astratto!)
- Altri due metodi possono essere sovrascritti ma hanno già un'implementazione concreta:
 - `init()`: eseguito dopo la creazione dell'oggetto, prima della creazione del thread per cui la GUI non esiste ancora
 - `stop()`: eseguito alla fine dell'applicazione che può essere implicita (es. chiusura finestra) o esplicita (invocazione di `Platform.exit()`)



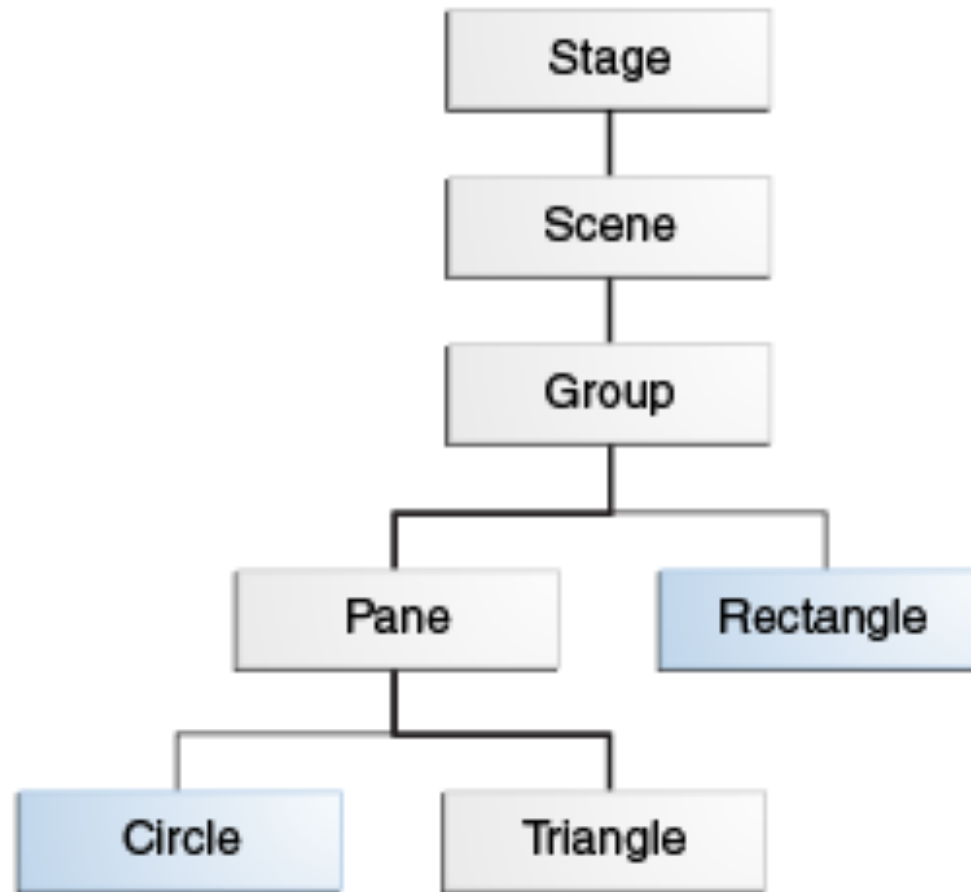
Window

- Uno `Stage`, a meno di disabilitare esplicitamente le opzioni, può essere ridimensionato, spostato, ridotto a icona e chiuso
- `Stage` è infatti una sottoclasse di `Window` che espone i metodi per gestire la finestra ed associare comportamenti specifici ad eventi
- Altri tipi di finestre: `PopupWindow` o `WebView` (che mostra contenuti di pagine Web recuperate tramite `WebEngine`)

Scene graph

- Una scena è organizzata in una **struttura gerarchica di nodi** che rappresentano tutti gli elementi visuali dell'interfaccia
- Questa struttura è il punto di partenza per organizzare e fare il rendering di un'applicazione
- Esistono due tipi di nodi:
 - **Branch node**: nodi intermedi che hanno nodi figlio nella gerarchia
 - **Leaf nodes**: non hanno nodi figlio
- Esiste un nodo radice (**root**) che non ha nodo padre (è diverso da Stage e Scene)

Struttura gerarchica



Nodi

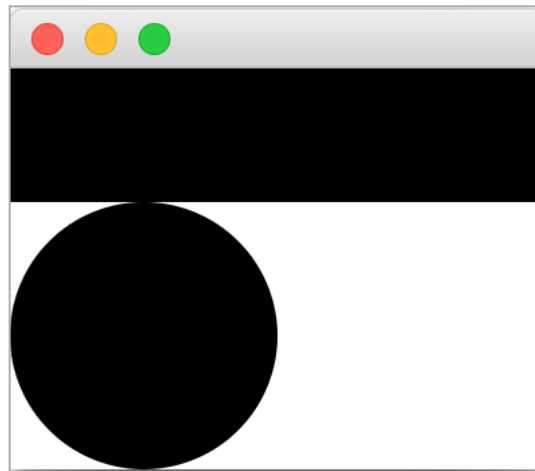
- Quattro tipi di nodo:
 - **Geometrical objects:** forme 2D e 3D, cerchi, rettangoli, poligoni, ecc.
 - **Media elements:** immagini, audio, video
 - **UI controls:** widget per interagire con l'utente come bottoni, checkbox, menu, textarea, ecc.
 - **Groups and Containers:** pannelli, layout orizzontali e verticali, griglie, ecc.

Forme 2D

- Java FX include un set predefinito di classi corrispondenti a forme geometriche bidimensionali:
- Per aggiungere una forma all'interfaccia:
 - Istanziare un oggetto della classe corrispondente
 - Settare le proprietà, alcune condivise altre dipendenti dalla forma
 - Aggiunge ad un gruppo (o un layout, vedremo a breve) che fa parte della scena
- Alcune proprietà:
 - posizione (X, Y) - può essere anche passata al costruttore
 - colori
 - bordi
 - ecc.

Forme 2D

- Modificare l'applicazione `HelloWorld` per ottenere la seguente visualizzazione
 - Rettangolo: 300 x 50
 - Cerchio: $r=50$
 - Colori di default



Forme 2D

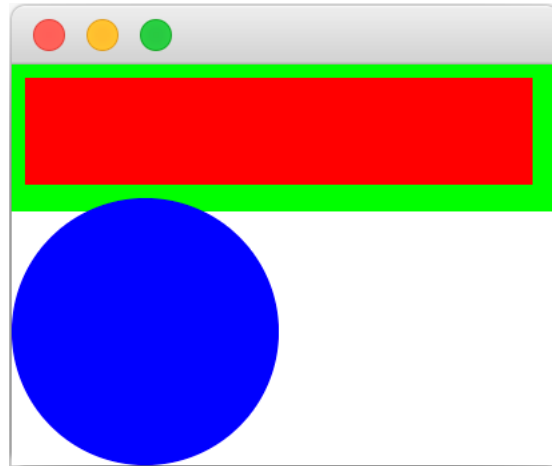
```
public void start(Stage primaryStage) {  
  
    Rectangle r1 = new Rectangle(300, 50);  
  
    Circle c1 = new Circle(50);  
    c1.setCenterX(50);  
    c1.setCenterY(100);  
  
    Group g = new Group();  
  
    g.getChildren().add(r1);  
    g.getChildren().add(c1);  
  
    Scene s1 = new Scene(g);  
  
    primaryStage.setScene(s1);  
    primaryStage.show();  
}
```

Colori

- La classe `Color` è usata per esprimere colori nello spazio RGB (Red Green Blue)
- Un colore può avere anche un valore *alpha* per indicare la trasparenza (da 0 a 1, default 1 nessuna trasparenza)
- Diversi costruttori per esprimere lo stesso colore:
 - Costanti: `Color.RED`, `Color.BLUE`, `Color.ACQUAMARINE`, `Color.OLIVE`, etc.
 - Codice RGB interi: `Color.rgb(0, 122, 122)`
 - Codice RGB esadecimali: `Color.web("0000FF")`
 - ...
- Oggetti `Color` sono passati come parametri per per colorare bordi, sfondi, testo, bottoni, etc.

Forme 2D

- Modificare l'applicazione `HelloWorld` per ottenere la seguente visualizzazione



Forme 2D e colori

```
public void start(Stage primaryStage) {  
  
    Rectangle r1 = new Rectangle(200, 50);  
    r1.setFill(Color.RED);  
    r1.setStroke(Color.web("0x00FF00"));  
    r1.setStrokeWidth(10);  
  
    Circle c1 = new Circle(50);  
    c1.setFill(Color.rgb(0, 0, 255));  
    c1.setCenterX(50);  
    c1.setCenterY(100);  
  
    Group g = new Group();  
  
    g.getChildren().add(r1);  
    g.getChildren().add(c1);  
  
    ...  
}
```

Testi

- La classe `Text` definisce un nodo che contiene appunto testo
- Come per le forme (e per tutti i nodi) per aggiungere un testo alla GUI:
 - Istanziare un oggetto della classe, settando eventualmente anche la posizione
 - Settare le proprietà del testo
 - Colore
 - Font
 - Effetti
 - ...
 - Aggiunge il nodo ad un gruppo/layout

Forme 2D

- Scrivere un'applicazione `HelloWorldText` per ottenere la seguente visualizzazione
- Posizione, colori e stili esatti non sono rilevanti



Testi

```
Text t1 = new Text(50, 50, "Orange");
t1.setFont(Font.font("verdana", FontWeight.BOLD, 60));
t1.setFill(Color.YELLOW);
t1.setStroke(Color.RED);
t1.setStrokeWidth(4);

Text t2 = new Text(100,100, "Apple");
t2.setFont(Font.font("arial", FontPosture.ITALIC, 40));

Text t3 = new Text(20,150, "Lemon");
t3.setFont(Font.font("courier", 30));
t3.setStrikethrough(true);

Group g = new Group();
g.getChildren().addAll(Arrays.asList(t1,t2,t3));
```


Immagini

- Le immagini possono essere aggiunte alla scena istanziando opportuni nodi (classe `Image`) e:
 - caricando l'immagine tramite un `InputStream`, che a sua volta può leggere il file sul computer locale o da remoto
 - aggiungendo una `ImageView` dell'immagine

```
Image picture1 = new
    Image(getClass().getResourceAsStream("dama.jpg"));
ImageView picture1View = new ImageView(picture1);

FileInputStream streamPicture2 =
    new FileInputStream("src/gui/fx/scacchi.jpg");
ImageView picture2View = new ImageView(new
    Image(streamPicture2));

// omessi add() a gruppo o layout
```

Layouts

- Dopo aver costruito i nodi è necessario disporli nello spazio. Java FX fornisce diversi *layout predefiniti* per organizzare in modo flessibile i nodi all'interno della scena
- https://docs.oracle.com/javafx/2/layout/builtin_layouts.htm
- Ogni layout è rappresentato da una diversa classe e corrisponde ad una struttura nello spazio
- Per creare un layout quindi è necessario:
 - creare i nodi che lo comporranno
 - istanziare un oggetto della classe corrispondente al layout
 - decidere le proprietà del layout
 - aggiungere i nodi al layout
- I layout possono essere annidati per permettere effetti sofisticati

Layouts

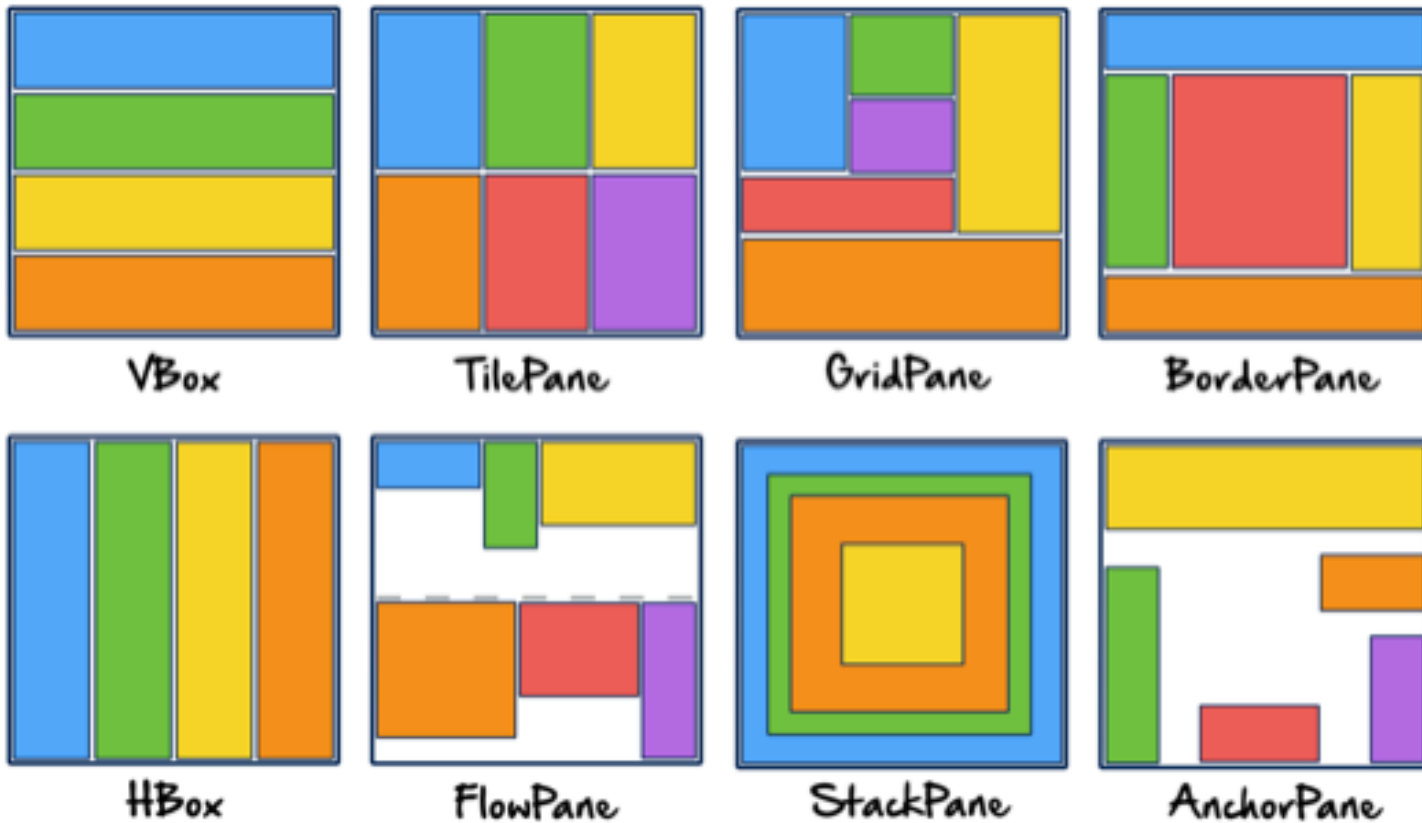


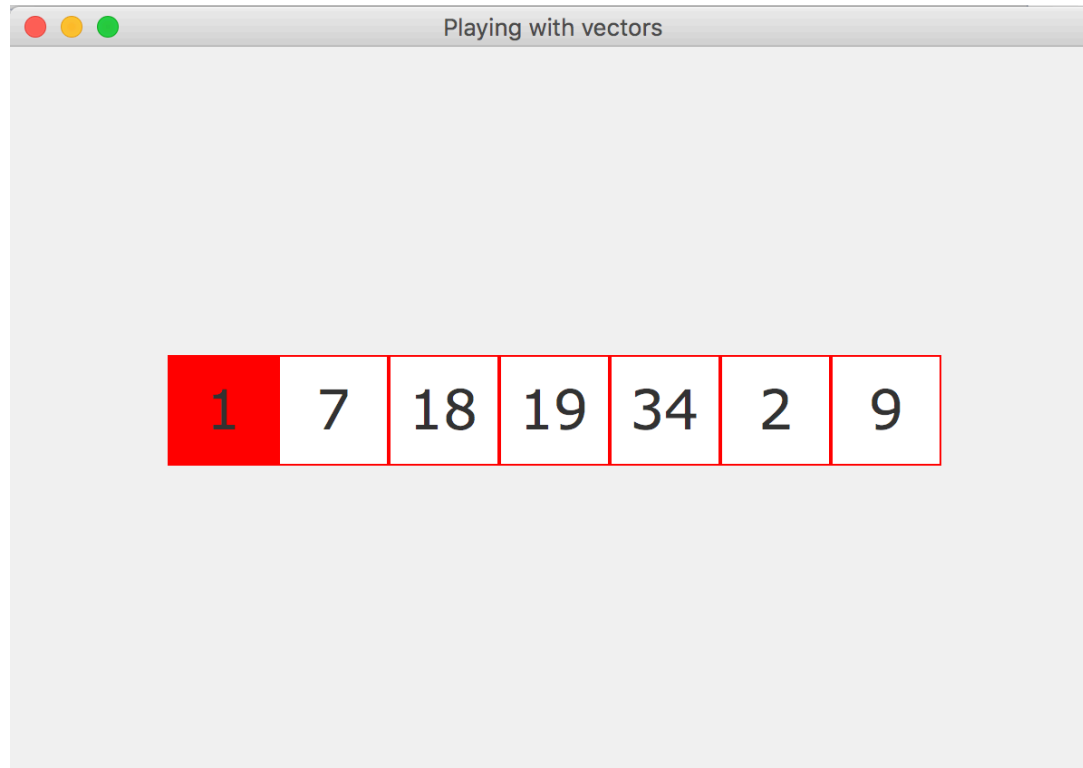
Immagine da: <https://dzone.com/refcardz/javafx-8-1?chapter=9>

Alcuni layout

- **BorderPane**: cinque regioni (top, bottom, left, center) in cui disporre i nodi
- **Hbox**: nodi disposti orizzontalmente su una sola riga
- **Vbox**: nodi disposti verticalmente su una riga
- **StackPane**: nodi sovrapposti (la posizione dei nodi figlio corrisponde al layer)
- **GridPane**: griglia in cui disporre gli oggetti decidendo quante celle occupano (sia sulla riga che sulla colonna) e lo spazio tra celle
- **TilePane**: simile ad una griglia ma impone che tutti gli oggetti abbiano la stessa dimensione
- ...

Esercizio

- Scrivere un'applicazione JavaFX che mostra un vettore di interi come in figura
- Il primo elemento del vettore ha sfondo rosso



```
// import, dichiarazione classe e invocazione start omessi
HBox root = new HBox();

Integer[] integers = {1, 7, 18, 19, 34, 2, 9};

for (int i = 0; i < integers.length; i++) {

    StackPane sp = new StackPane();

    Rectangle background = new Rectangle(60, 60);
    background.setStroke(Color.RED);
    background.setFill(Color.WHITE);

    Label l = new Label();
    l.setText(integers[i].toString());
    l.setFont(new Font("Verdana", 30));

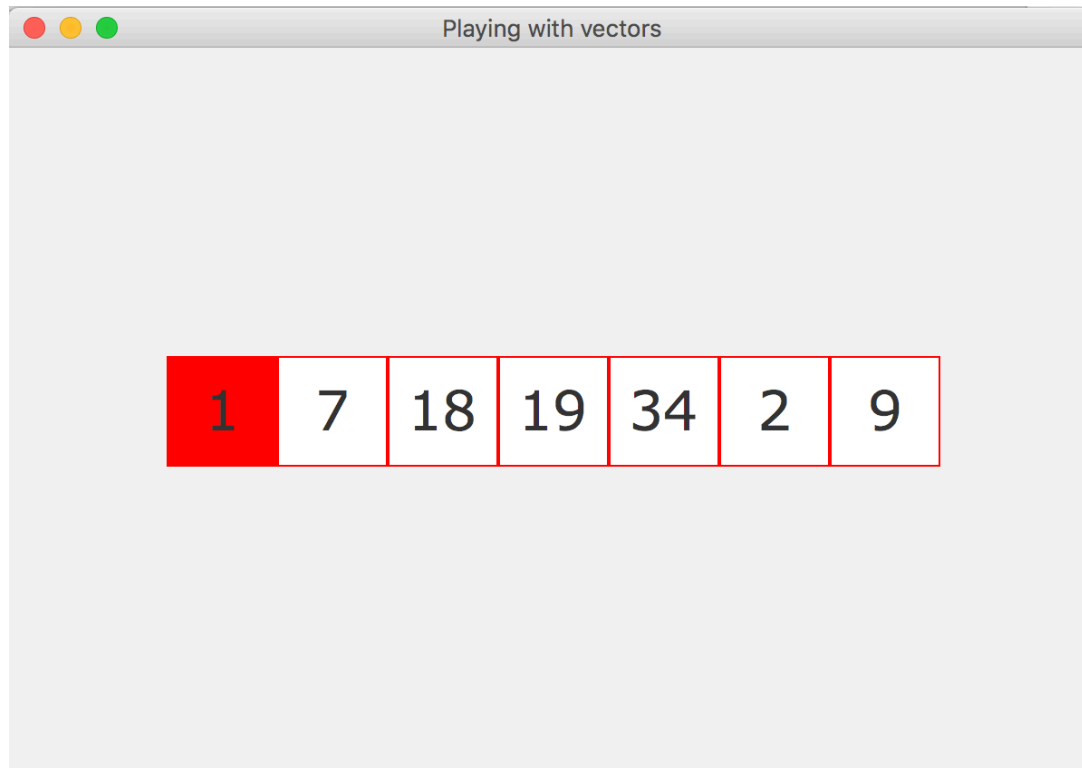
    sp.getChildren().addAll(background, l);

    root.getChildren().add(sp);
}

root.setAlignment(Pos.CENTER);
... // add() di root alla scena omesso (vedi precedenti)
```

Comportamenti dinamici

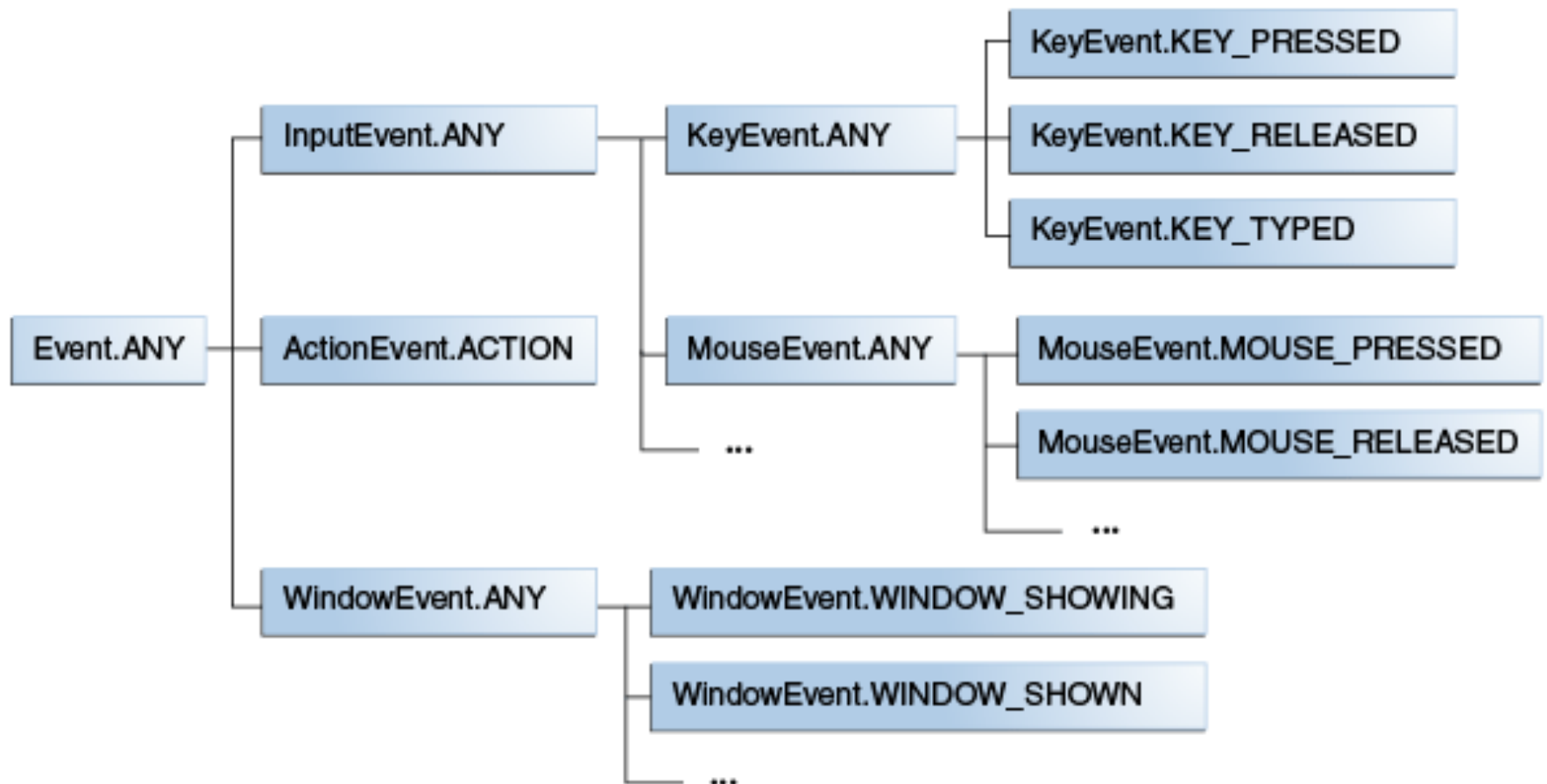
- Aggiungere questo comportamento dinamico all'esempio precedente:
 - puntando su un elemento lo sfondo diventa rosso
 - spostando il puntatore fuori dall'elemento lo sfondo torna bianco



Eventi in Java FX

- In JavaFX un evento è un istanza della classe `javafx.event.Event` o qualunque sottoclasse di `Event`.
- Ogni evento è caratterizzato da tre proprietà:
 - **Type**: il tipo di evento, secondo una gerarchia pre-definita ed estensibile
 - **Source**: origine dell'evento, può cambiare mentre l'evento viene "processato"
 - **Target**: nodo su cui l'azione è avvenuta; se ci sono più nodi annidati si considera il nodo più in profondità della gerachia (e/o nel layer più alto)

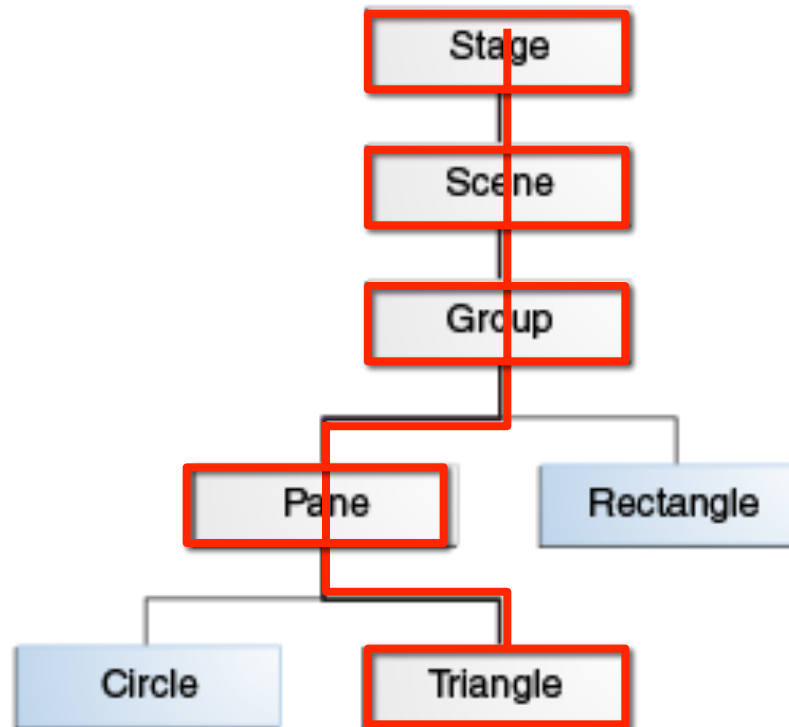
Tipi di eventi



Processare un evento

- La gestione di un evento prevede 4 fasi:
 - **Target selection:** identificazione del nodo su cui è avvenuto l'evento in base al tipo di evento
 - **Route construction:** costruzione del percorso dalla radice al nodo (dispatch chain)
 - **Event Capturing:** propagazione dell'evento dalla radice al target, eseguendo eventuali *filtri* registrati per gestire quell'evento
 - **Event Bubbling:** propagazione dell'evento dalla radice al target, eseguendo eventuali *handler* registrati per gestire quell'evento

Processare un evento

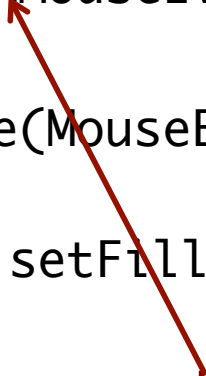


Event Handler

- Per processare un evento un node deve registrare un `EventHandler`, che implementa l'interfaccia `EventHandler` ed è associato ad un dato evento
- Il metodo principale dell'interfaccia è `handle()` che contiene il codice che sarà eseguito quando il nodo che ha registrato l'handler riceve l'evento
- Due modi per registrare un handler:
 - metodo `addEventHandler()`
 - metodi `setOn<EVENT>`

```
sp.setOnMouseEntered(new EventHandler<MouseEvent>() {  
    @Override  
    public void handle(MouseEvent arg0)  
    {  
        background.setFill(Color.RED);  
    }  
});
```

Registrazione
EventHandler



```
sp.addEventHandler(MouseEvent.MOUSE_EXITED, new  
EventHandler<MouseEvent>() {  
    @Override  
    public void handle(MouseEvent arg0) {  
        background.setFill(Color.WHITE);  
    }  
});
```

Un punto importante

- I nodi da modificare negli handler devono essere variabili **final**

```
HBox root = new HBox();  
  
Integer[] integers = {1, 7, 18, 19, 34, 2, 9};  
  
for (int i = 0; i < integers.length; i++) {  
    final StackPane sp = new StackPane();  
    final Rectangle background = new Rectangle(60, 60);  
    ...  
}
```

UI Controls

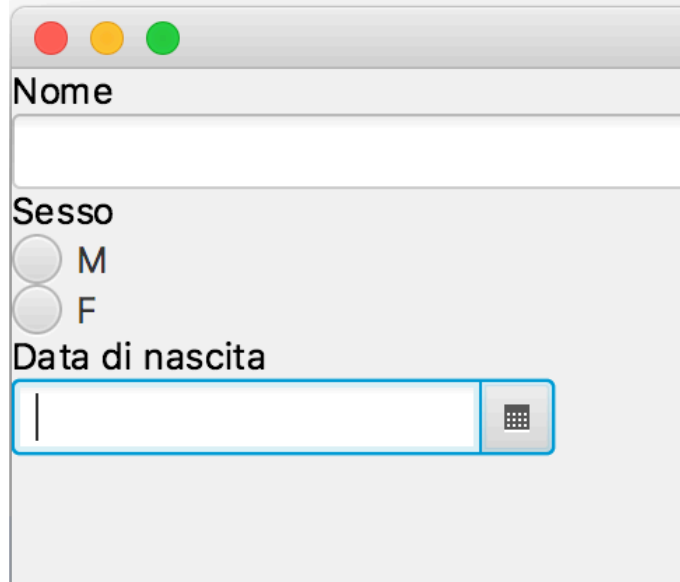
- Java FX fornisce inoltre un vasto insieme di elementi visuali con cui l'utente interagisce (bottoni, menù, aree di testo, picker, etc.)
- Ogni elemento è rappresentato da una classe che deve essere quindi istanziata in un oggetto-nodo
- Il nodo va poi aggiunto ad un layout e decide le sue proprietà, molte specifiche per quel controllo
- L'interazione è definita tramite *event handler* (con i meccanismi descritti nelle slide precedenti, associati ad eventi specifici)

UI Controls



UI Controls

- Scrivere un'applicazione `HelloWorldControls` per ottenere la seguente visualizzazione
- Posizione, colori e stili esatti non sono rilevanti



A screenshot of a simple GUI window with three sections:

- Nome**: A text input field.
- Sesso**: Two radio buttons labeled **M** and **F**.
- Data di nascita**: A date picker control with a calendar icon.

```
Text nameLabel = new Text("Nome");
TextField nameText = new TextField();

Text dataLabel = new Text("Data di nascita");
DatePicker datePicker = new DatePicker();

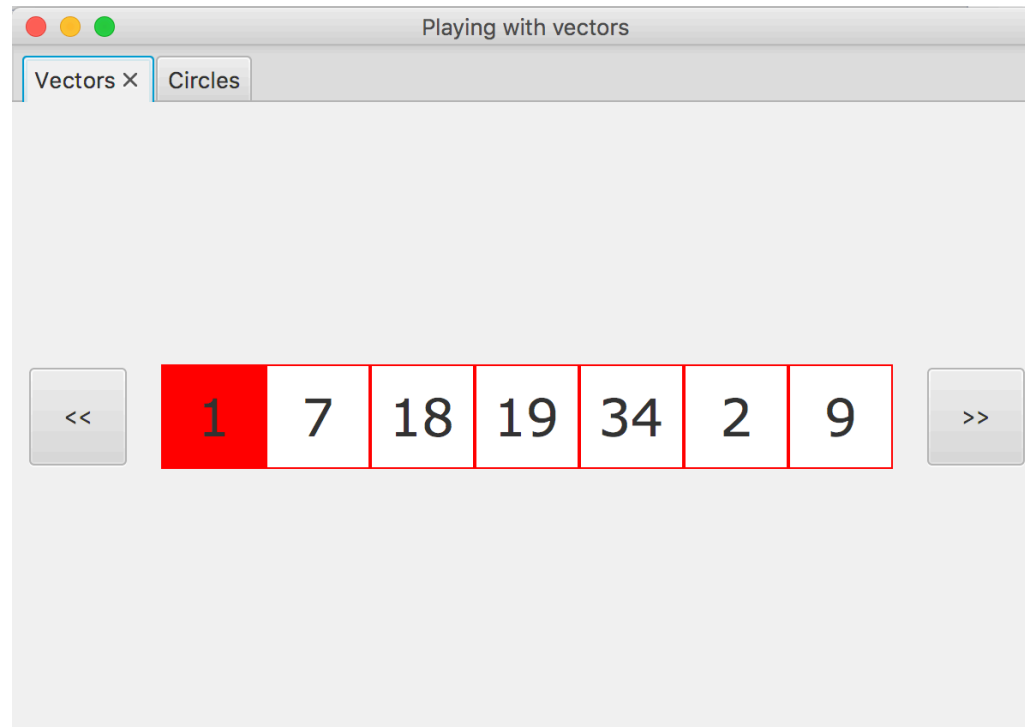
Text genderLabel = new Text("Sesso");
ToggleGroup groupGender = new ToggleGroup();
RadioButton maleRadio =
    new RadioButton("M");
maleRadio.setToggleGroup(groupGender);
RadioButton femaleRadio =
    new RadioButton("F");
femaleRadio.setToggleGroup(groupGender);

VBox box = new VBox();
```

...

Esercizio

- Aggiungere all'applicazione `HelloWorldControls` due bottoni ('<<' e '>>') per muoversi di una posizione avanti o indietro sul vettore, come mostrato in figura
- Organizzare l'applicazione in due tab



Altri aspetti di Java FX

- Oggi abbiamo visto le componenti principali di Java FX e alcune delle proprietà dei nodi
- Il framework fornisce molti altri oggetti e comportamenti che non vedremo:
 - Diagrammi
 - Animazioni
 - Grafica 3D
 - ...
- Due aspetti interessanti: supporto per CSS e FXML

FXML

- FXML è un linguaggio XML per esprimere viste in modo dichiarativo in applicazioni *MVC*
- Non ne parleremo ma si può usare per il progetto

```
<Button
    layoutX="172.0"
    layoutY="45.0"
    prefWidth="200.0"
    text="Say hello!"
    styleClass="btn, btn-primary"
    fx:id="sayHelloActionTarget"/>

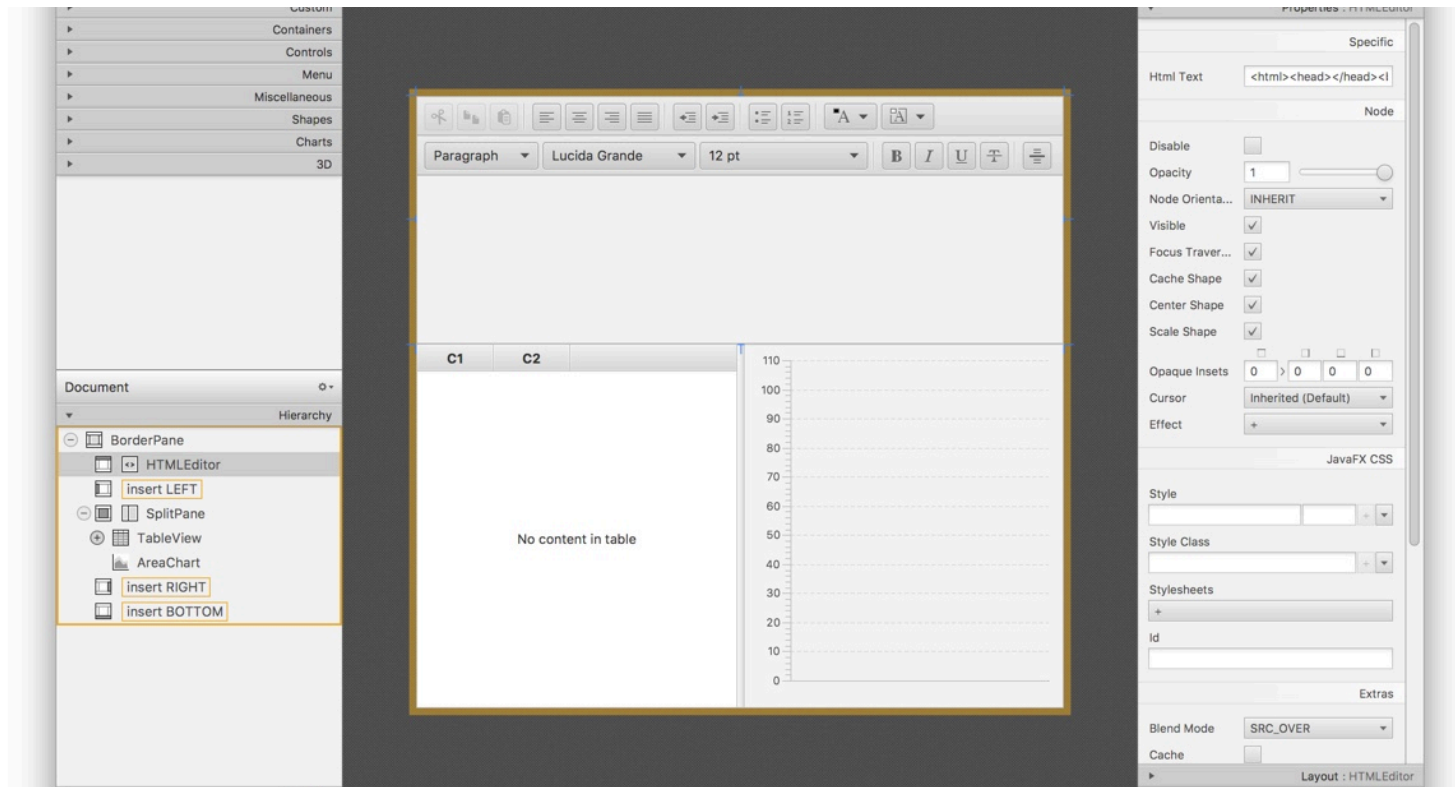
<Label layoutX="14.0" layoutY="80.0"
    prefWidth="360.0" fx:id="output"/>
```

TOOLS

IDE Integration

- NetBeans IDE: JavaFX Application wizard
- E(fx)clipse: plug-in sviluppato dalla community di Eclipse
<https://www.eclipse.org/efxclipse/>

Java FX Scene Builder



<http://gluonhq.com/products/scene-builder/>