

JavaFX

JavaFX si configura come un insieme di librerie (o *frameworks*) per la creazione di **interfacce grafiche in Java**. Questo ambiente di lavoro si sviluppa molto bene secondo le regole base di Java che abbiamo visto, riprendendo spesso i concetti di polimorfismo ed ereditarietà.

Le Basi della UI

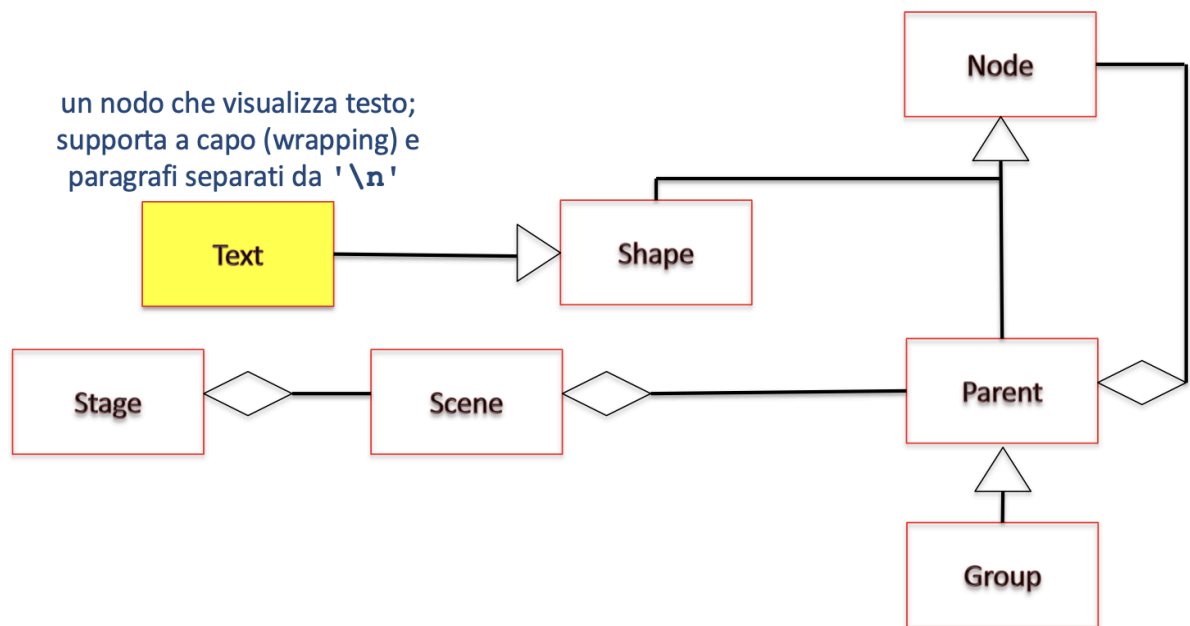
Una finestra di dialogo con l'utente, sviluppata con il *framework* di JavaFX, è formata da molti livelli sovrapposti di **nodi** ossia elementi grafici. In particolare partendo dalla base fino all'elemento più in alto si trovano:

- Stage
- Scene (contenuta nello Stage)
- Parent (possono essere più di uno, tutti contenuti nella Scene)
- Node (ha spesso un Parent ma vale che il Parent è un Node)

Possiamo vedere questa gerarchia come una catena di: Stage → Scene → Parent → Node che può essere vista a sua volta come: Finestra → Contenitore → Contenuto.

Considerando chiari i concetti di Stage e Scene, quindi di Finestra e Contenitore, è utile soffermarsi sul comprendere il vero significato di Parent e Node. Possiamo vedere un **Parent** come una **disposizione di Nodes** secondo uno specifico criterio. Allo stesso tempo possiamo vedere un **Node come un oggetto grafico**, ultima rappresentazione grafica possibile, che viene posizionato all'interno della Scena secondo delle disposizioni stabilite da un Parent.

Si noti che anche il Parent è effettivamente un Node perché può essere all'interno di un Parent più "generico" insieme ad altri Parents.



E' inoltre possibile disporre di più finestre (Stage) aggiornate simultaneamente in tempo reale. E' sufficiente passare due Scene diverse nei due Stage separati.

Gestione degli eventi

Le interfacce grafiche sfrutta un paradigma architetturale strutturato **ad eventi**, questo perché l'esecuzione del programma è inevitabilmente determinata da **azioni compiute dall'utente**, anziché una sequenza prefissata come nell'architettura del C++.

Esiste già una gerarchia di eventi predefinita in JavaFX, associata agli elementi disponibili per l'interfaccia grafica. Ogni evento “ **Event** ” raggruppa uno o più sotto eventi definiti da costanti.

Normalmente per associare una reazione ad un'azione compiuta dall'utente è necessario **creare un oggetto listener**, derivante direttamente dalla classe **Listener** e successivamente chiamare sul bottone il metodo “ addEventHandler ” passando come parametri il tipo di evento e il listener associato.

Questo procedimento va fatto perché di base “ **EventHandler** ” è un'interfaccia **implementata dalla classe Listener**. Inoltre l'interfaccia “ **EventHandler** ” è studiata sui Generics quindi va specificato il tipo di evento. La descrizione di ciò

che va fatto al momento dell'azione è tutta presente nella classe Listener che implementa l'interfaccia EventHandler.

Alcuni eventi base

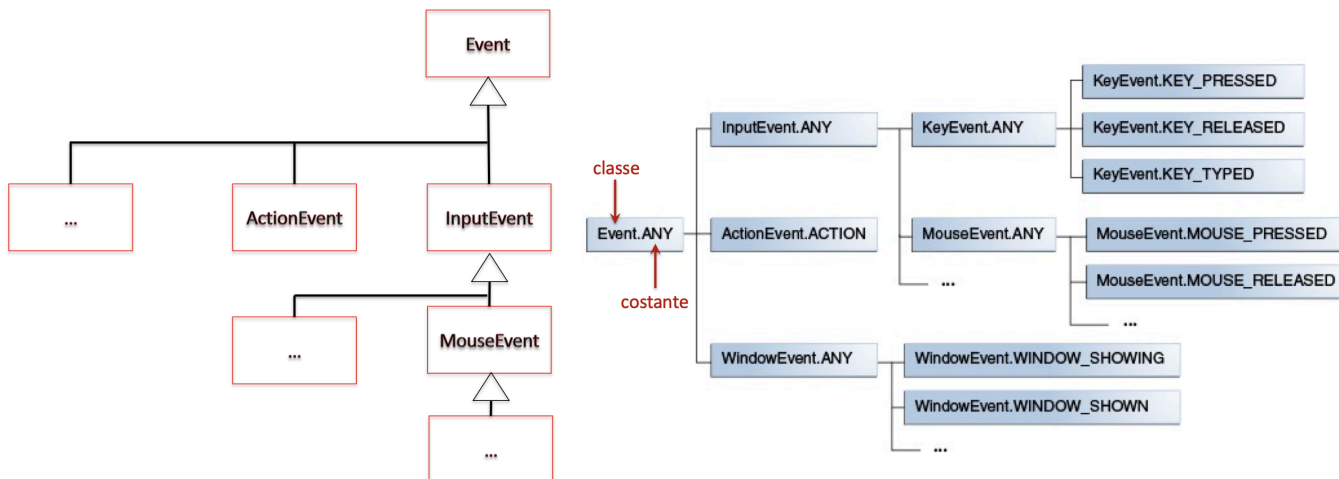
```
public class Test extends Application {  
    public void start(Stage stage) {  
        Button btn = new Button();  
        btn.setText("Click me");  
        Listener a = new Listener();  
        btn.addEventHandler(Event.ANY, a);  
        Group root = new Group(btn);  
        Scene scene = new Scene(root, 300, 250);  
        stage.setScene(scene);  
        stage.show();  
    }  
    // main...  
}  
  
class Listener implements EventHandler<Event> {  
    int counter=0;  
    public void handle(Event t) {  
        System.out.println(++counter + " Ricevuto un evento di tipo "  
            + t.getEventType());  
    }  
}
```

1 Ricevuto un evento di tipo MOUSE_ENTERED
2 Ricevuto un evento di tipo
 MOUSE_ENTERED_TARGET
3 Ricevuto un evento di tipo MOUSE_MOVED
...
12 Ricevuto un evento di tipo MOUSE_MOVED
13 Ricevuto un evento di tipo MOUSE_PRESSED
14 Ricevuto un evento di tipo ACTION
15 Ricevuto un evento di tipo MOUSE_RELEASED
16 Ricevuto un evento di tipo MOUSE_CLICKED
17 Ricevuto un evento di tipo MOUSE_MOVED



```
interface EventHandler<T extends Event>  
    extends EventListener
```

Si noti che il primo parametro passato al metodo EventHandler rappresenta un'interazione dell'utente con l'ambiente. A tal proposito esiste una **gerarchia di eventi** predefinita in JavaFX associata agli elementi disponibili per l'interfaccia grafica. In particolare ogni oggetto **Event** raggruppa uno o più sottoeventi definiti da costanti ed ogni sottoclasse definisce attributi e metodi specifici per tipologia di eventi. E quindi come accennato prima **il codice può reagire a questi eventi** specificando uno o più **listener** che



implementino l'interfaccia EventHandler.

Dichiarazione del Listener

La dichiarazione del Listener può avvenire in diversi modi che implicano però accorgimenti differenti.

Listener esterno

Dichiarando il Listener esternamente vuol dire che la classe Listener viene creata **al di fuori della classe principale**, con la diretta conseguenza che per accedere ai metodi e attributi di quest'ultima è necessario passare un oggetto del tipo della classe principale.

Listener interno

Alternativamente possiamo dichiarare la **classe Listener internamente alla classe principale** così da sfruttare direttamente i metodi e attributi già dichiarati, senza necessità di passarli.

Listener anonimo

Il metodo più comodo di dichiarazione della classe Listener è la dichiarazione **anonima**. In particolare viene creato un oggetto di tipo Interfaccia (ammissibile solo se il tipo interfaccia costituisce il tipo dinamico) e l'implementazione dell'azione da compiere viene fornita direttamente "inline".

Listener interno anonimo


```
public class AppWithEvents extends Application {
    Text text = null;
    public void start(Stage stage)
    {
        text = new Text(10,50,"Non hai mai cliccato ");
        Button btn = new Button();
        btn.setText("Click me");
        EventHandler a = new EventHandler<ActionEvent>() {
            int counter=0;
            public void handle(ActionEvent t) {
                updateText(++counter);
            }
        };
        btn.addEventHandler(ActionEvent.ACTION, a);
        Group root = new Group(btn);
        root.getChildren().add(text);
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }
}
```

è un'interfaccia!!!

crea un oggetto la cui classe «anonima», implementa l'interfaccia EventHandler ...

... e la cui implementazione viene data «inline» contestualmente alla creazione dell'oggetto

Migliora la leggibilità del codice: il listener è vicino al suo (unico) uso



```
public void updateText(int n){
    text.setText("Hai cliccato "
        +n+" volte");
}

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

// end of AppWithEvents
```

Convenience method

Si può ulteriormente rendere più conciso il codice utilizzando i cosiddetti **convenience methods**. In poche parole questi metodi risparmiano il dover poi chiamare il metodo “addEventHandler” sul bottone, direttamente chiamando il metodo “**setOnAction**” sul bottone e come parametro passargli la descrizione dell’EventHandler.

```
btn.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent event) {  
        System.out.println("Hello World!");  
    }  
});
```

parte del template nell’IDE...

Classi annidate

L’utilità delle classi annidate si ritrova principalmente nel **tenere il codice vicino** rispetto **a dove viene usato**. Esistono varie tipologie di classi annidate ma per semplicità si vedrà solamente il caso delle **inner class** ovvero classi interne, annidate ma non statiche. Le inner class hanno il vantaggio di avere **accesso a tutti gli attributi e metodi** della **classe che le contiene**.

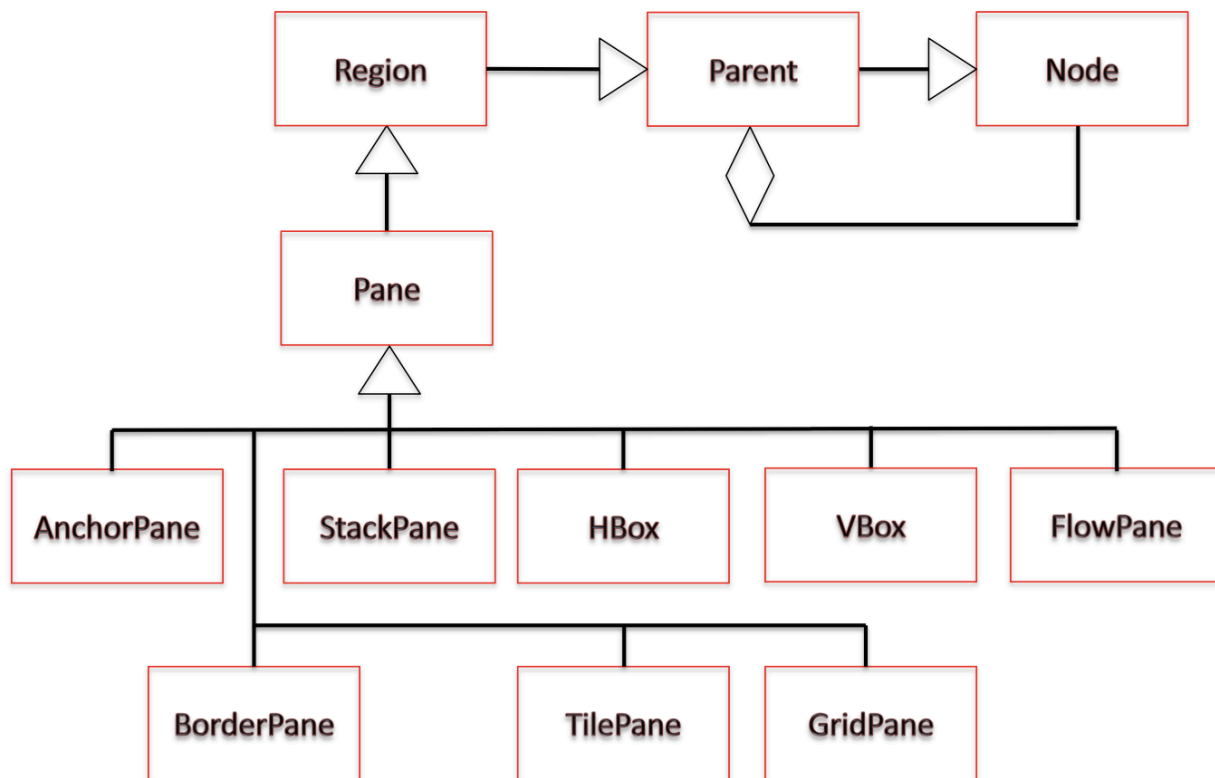
Un ulteriore vantaggio delle classi annidate è **aumentare l’incapsulamento delle informazioni** e i processi di **information hiding** in quanto a differenza di classi normali, la *nested class* possono essere dichiarate **protected o private** in modo da rendere ancora più esclusiva la protezione dei dati.

Posizionamento di un Node

Per quanto riguarda il posizionamento di un Node, non sono forniti metodi assoluti (come per esempio per gli Stage che godono di metodi come SetX e SetY)ma soltanto soluzioni che ne determinano la distanza rispetto al Parent. E’ quindi fortemente **sconsigliato gestire manualmente** la posizione dei diversi Node.

A tal proposito si sfrutta la proprietà di **posizionamento automatico** rispetto al layout. Il compito del programmatore viene notevolmente semplificato definendo dei contenitori di oggetti che godono dei propri metodi di posizionamento e hanno le proprie regole predefinite per quanto riguarda la posizione.

Layout predefiniti



Dei Layout predefiniti si discuteranno solo i casi più particolari in quanto i casi “banali” sono di immediata comprensione.

StackPane

Lo **StackPane** serve ad **impilare gli elementi** con una **gerarchia** che **dipende dall'ordine** con il quale gli **elementi** vengono **aggiunti**. **StackPane** può risultare utile anche per creare rappresentazioni grafiche impilando elementi grafici come forme geometriche ecc.

TilePane

Il Tilepane organizza gli elementi in una griglia di celle di **uguale dimensione**. E' possibile impostare la distanza verticale e orizzontale che queste celle mantengono tra loro e in aggiunta è necessario dire che il Tilepane si adatta alla dimensione della finestra. E' possibile impostare un **numero predefinito di colonne o righe** in cui arrangiare le celle al momento della prima inizializzazione.

FlowPane

Come concetto si avvicina molto al concetto del Tilepane con la differenza che non vi sono celle in cui sono inseriti i nodi ma di base vengono inizializzati tutti attaccati. La caratteristica principale del FlowPane è quella di **poter ridimensionare i blocchi all'interno di esso, rispetto alla dimensione della finestra definita dinamicamente dall'utente**. E' possibile inoltre fissare una larghezza sotto la quale il Flowpane si riadatterà.

BorderPane

Il BorderPane risulta utile nel momento in cui si vogliono **disporre più oggetti in una finestra**, arrangiandoli in **diverse regioni della finestra**. E' possibile specificare dove posizionare i vari elementi specificando la costante relativa alla regione.

AnchorPane

Permette di **ancorare un oggetto ad una determinata zona** della scena, permettendo agli oggetti ancorati di seguire i vari adattamenti della finestra.

GridPane

Permette di **disporre gli oggetti in una griglia**, specificando la loro posizione espressa in indici che vanno da 0 a un numero di righe/colonne.