Grafica e non solo: Java FX

Marco Patrignani

mailto: marco.patrignani@unitn.it

(basato sulle slides di Picco, Ronchetti, Marchese)

JavaFX

- Insieme di librerie (***framework***)per la creazione di interfacce grafiche in Java
  - Utilizzabile sia per applicazioni desktop che Web
    - Esistono port commerciali per iOS e Android
- A partire da Java8, parte della distribuzione ufficiale del linguaggio
  - Rimpiazza Swing e prima ancora Abstract Window Toolkit (AWT)
- Molto più ricco e potente di quanto vedremo nel corso: curiosate nella documentazione!
  - **`https://docs.oracle.com/javase/8/javase-clienttechnologies.htm`**

Java FX

```
public class JavaFXApplicationTEST extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Per usare JavaFX in IntelliJ dobbiamo creare un progetto usando il nostro template (vedi istruzioni)

Otteniamo un'applicazione base "Hello World" da modificare

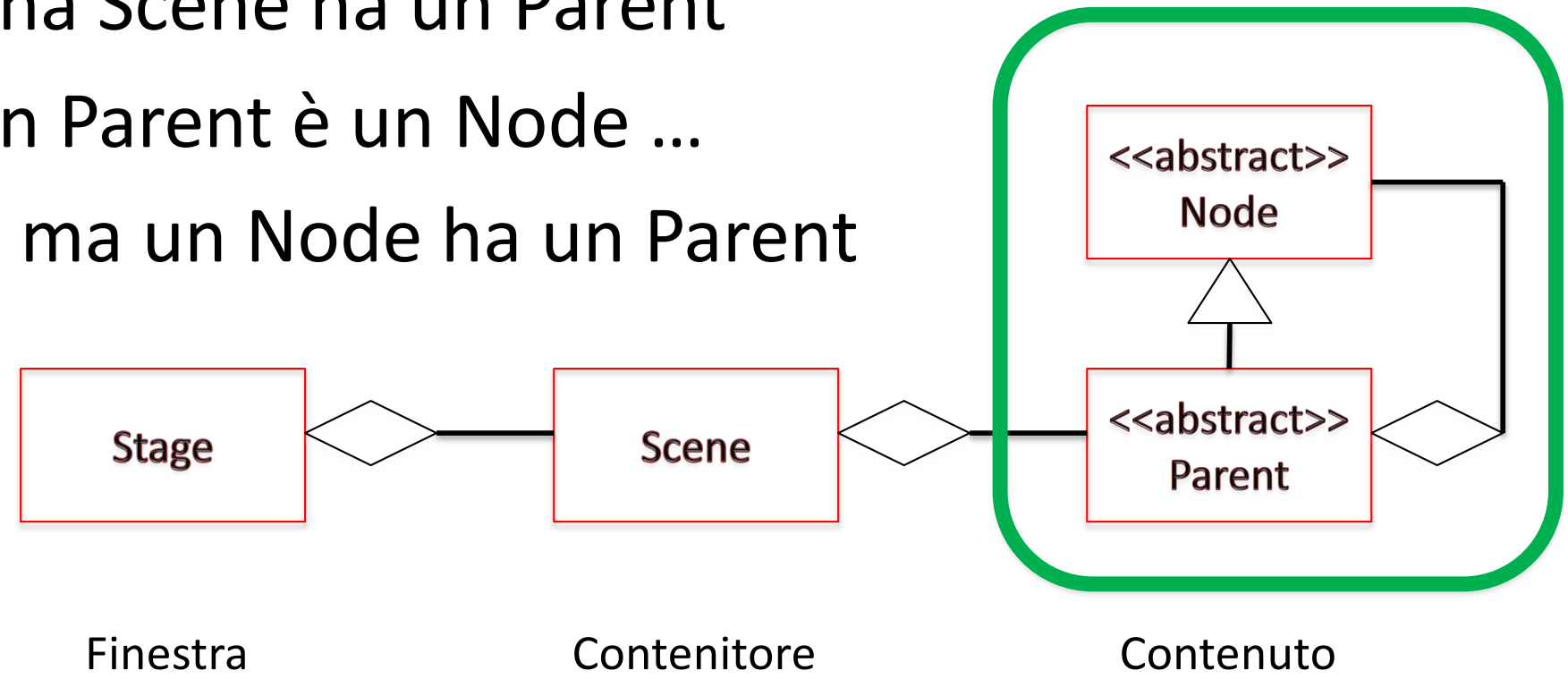
```
public class JavaFXApplicationTEST extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });
        StackPane root = new StackPane();
        root.getChildren().add(btn);
        Scene scene = new Scene(root, 300, 250);
        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

Java FX

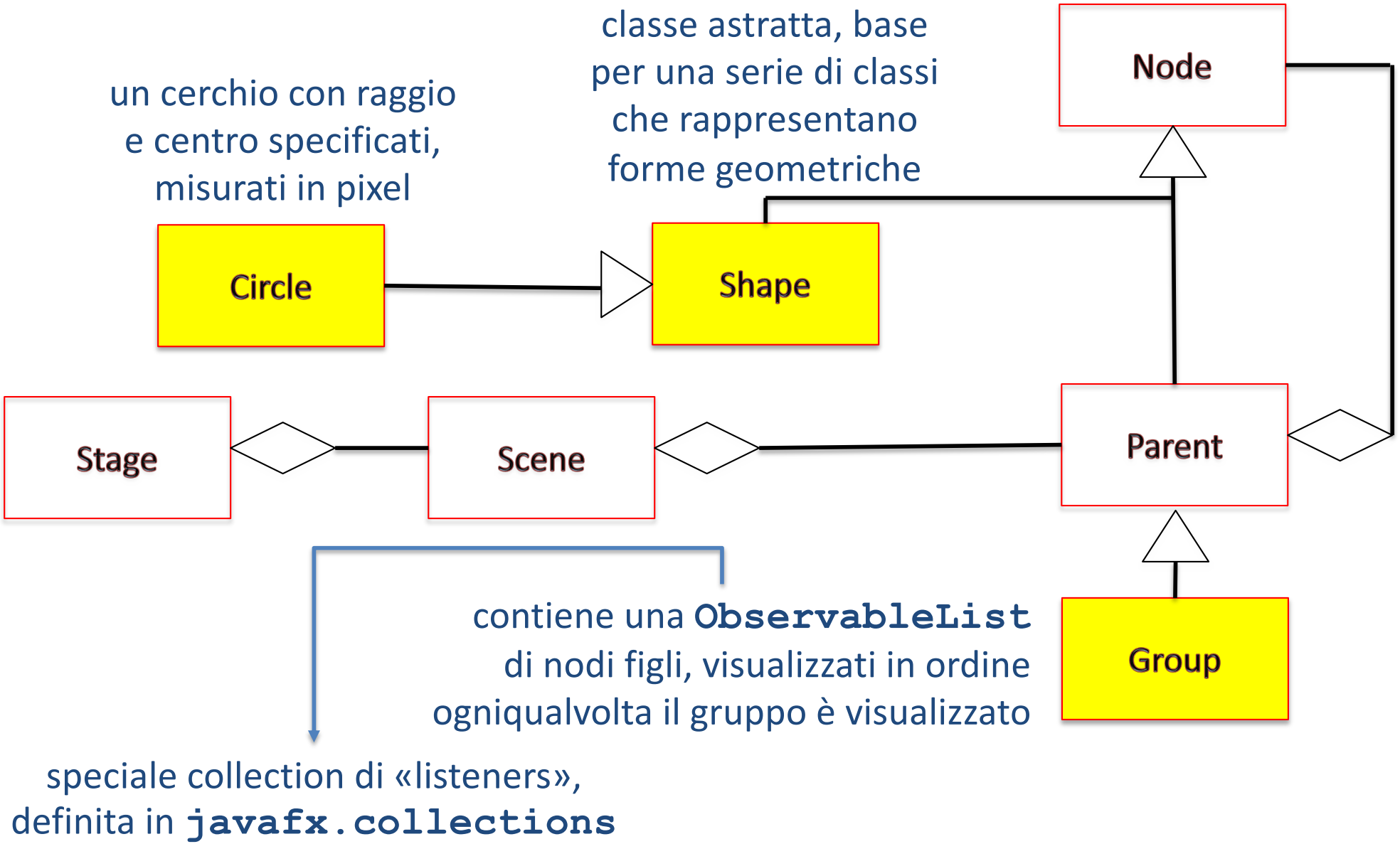
Stage/Scene/Parent/Node

Finestra == Stage

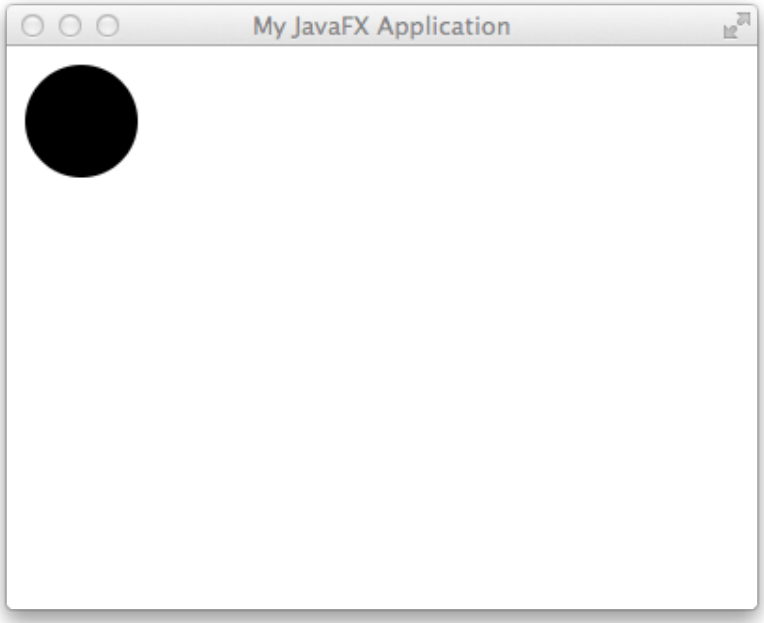
- Uno Stage contiene una Scene
- Una Scene ha un Parent
- Un Parent è un Node …
- … ma un Node ha un Parent



Group – Shape - Circle



Applicazione minima



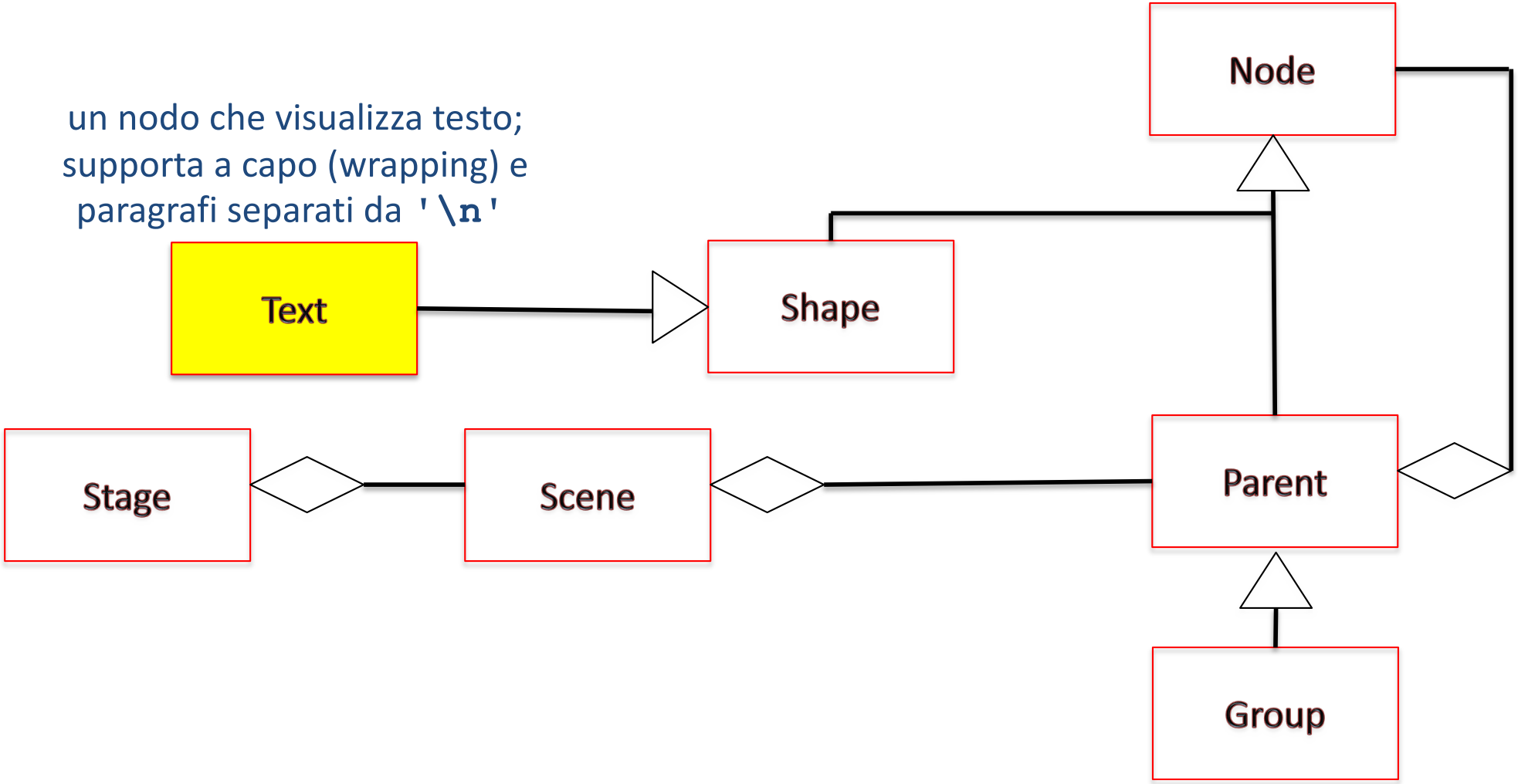

```
import javafx.application.*;
import javafx.scene.*;
import javafx.scene.shape.*;
import javafx.stage.*;
public class MinimalApp extends Application {
  public void start(Stage stage) {
    Node circ = new Circle(40, 40, 30);
    Parent root = new Group(circ);
    Scene scene = new Scene(root, 400, 300);
    stage.setTitle("My JavaFX Application");
    stage.setScene(scene);
    stage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

x,y del centro e raggio;
l'origine è in alto a sinistra

larghezza e altezza
dell'area associata

Shape & Text

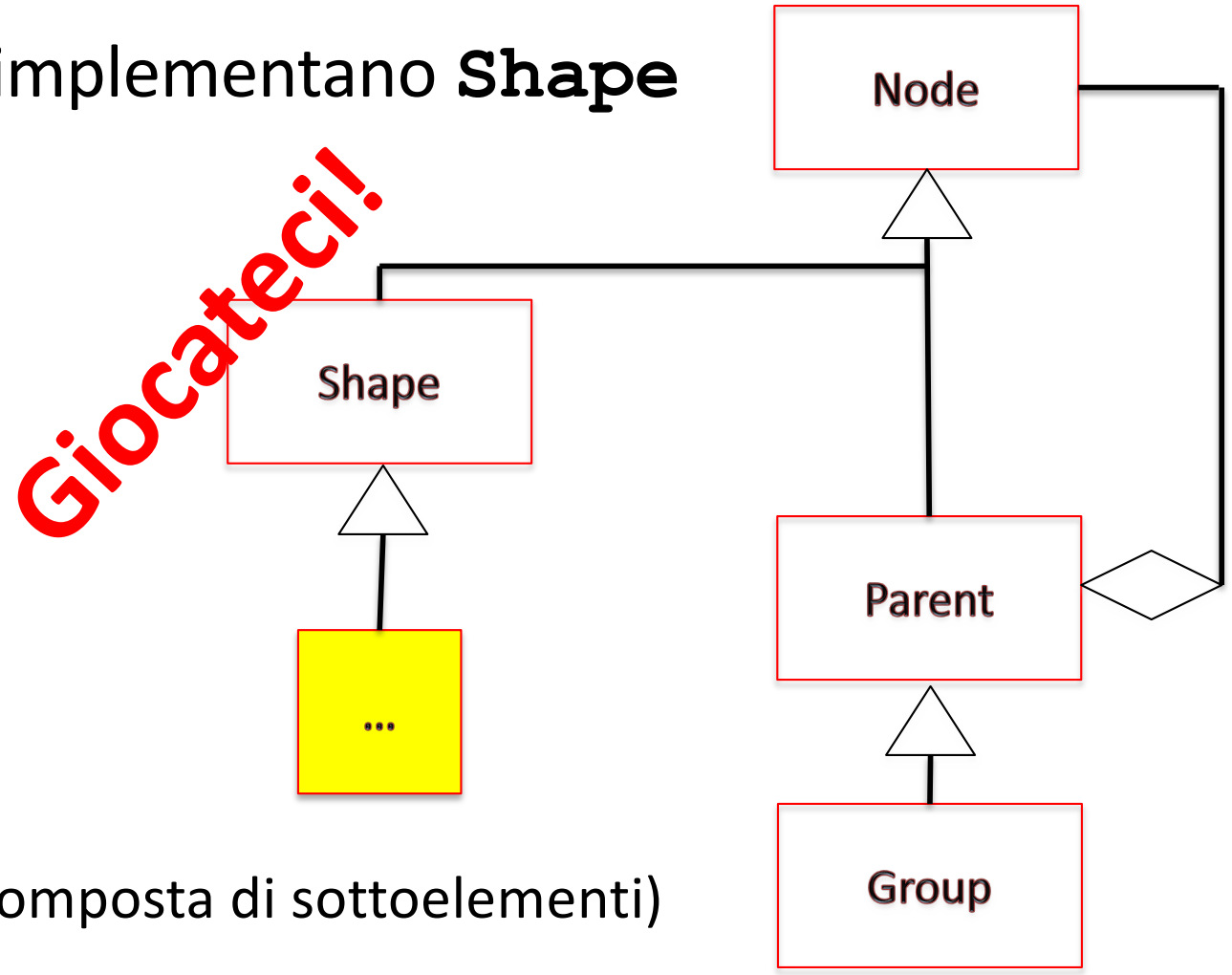
un nodo che visualizza testo;
supporta a capo (wrapping) e
paragrafi separati da `'\n'`



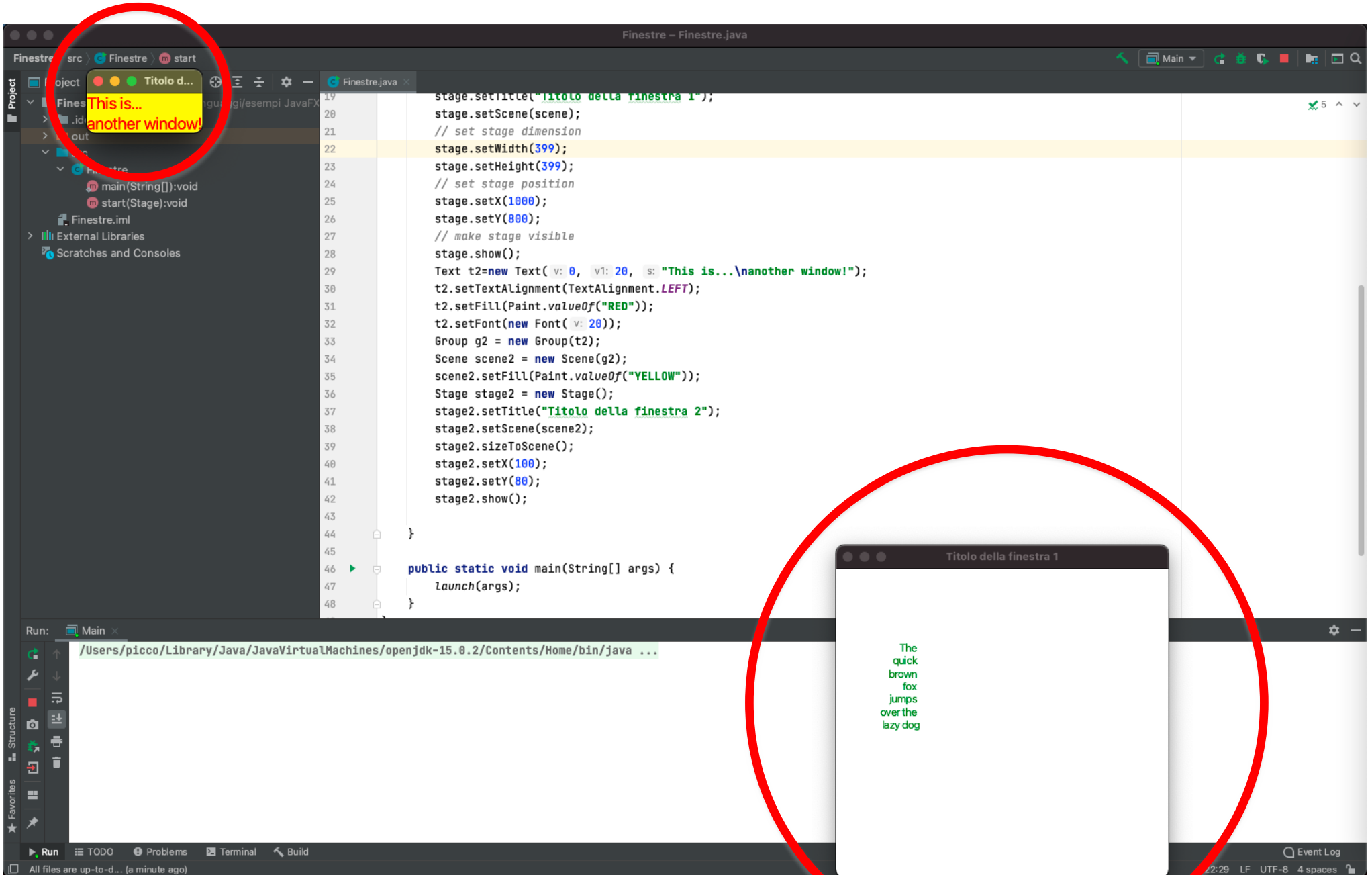
Gerarchia di Shape

Le seguenti classi implementano **`Shape`**

- **`Line`**
- **`Polyline`**
- **`Polygon`**
- **`Rectangle`**
- **`Arc`**
- **`Circle`**
- **`Ellipse`**
- **`QuadCurve`**
- **`CubicCurve`**
- **`Text`**
- **`SVGPath`** (linea composta di sottoelementi)



Esempio: Finestre multiple



Finestre multiple: prima finestra

```
public class Finestre extends Application {
public void start(Stage stage) {
    Text t=new Text(50, 100, "The quick brown fox jumps over
       the lazy dog");
    t.setTextAlignment(TextAlignment.RIGHT);
    t.setWrappingWidth(50);
    t.setFill(Paint.valueOf("GREEN"));
    Group g = new Group(t);
    Scene scene = new Scene(g);
    stage.setTitle("Titolo
                della finestra 1");
    stage.setScene(scene);
    // set stage dimension
    stage.setWidth(399);
    stage.setHeight(399);
   // set stage position
    stage.setX(1000);
    stage.setY(800);
    // make stage visible
    stage.show();
```

Finestre multiple: seconda finestra

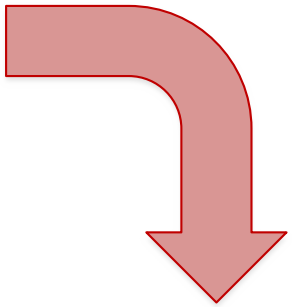
```
        Text t2=new Text(0, 20, "This is...\nanother window!");
        t2.setTextAlignment(TextAlignment.LEFT);
        t2.setFill(Paint.valueOf("RED"));
        t2.setFont(new Font(20));
        Group g2 = new Group(t2);
        Scene scene2 = new Scene(g2);
        scene2.setFill(Paint.valueOf("YELLOW"));
        Stage stage2 = new Stage();
        stage2.setTitle("Titolo della finestra 2");
        stage2.setScene(scene2);
        stage2.sizeToScene();
        stage2.setX(100);
        stage2.setY(80);
        stage2.show();
    }
    public static void main(String[] args) {
            launch(args);
    }
}
```

Nota pratica: attenzione agli **`import`**!

```
Text t2=new Text( v: 0,  v1: 20,  s: "This is...\nanother window!");
t2.setTextAlignment(TextAlignment.LEFT);
t2.setFill(Paint.valueOf("RED"));
t2.setFont(new Font(20));
Group g2 = new G
Scene scene2 = n
scene2.setFill(P
Stage stage2 = new Stage();
```

Cannot resolve symbol 'Font'

Import class ⌥⇧↵ More actions... ⌥↵



```
Text t2=new Text( v: 0,  v1: 20,  s: "This is...\nanother window!");
t2.setTextAlignment(TextAlignment.LEFT);
t2.setFill(Paint.valueOf("RED"));
t2.setFont(new Font(20));
Group g2 = new G
Scene scene2 = n
scene2.setFill(P
Stage stage2 = new Stage();
```

Class to Import

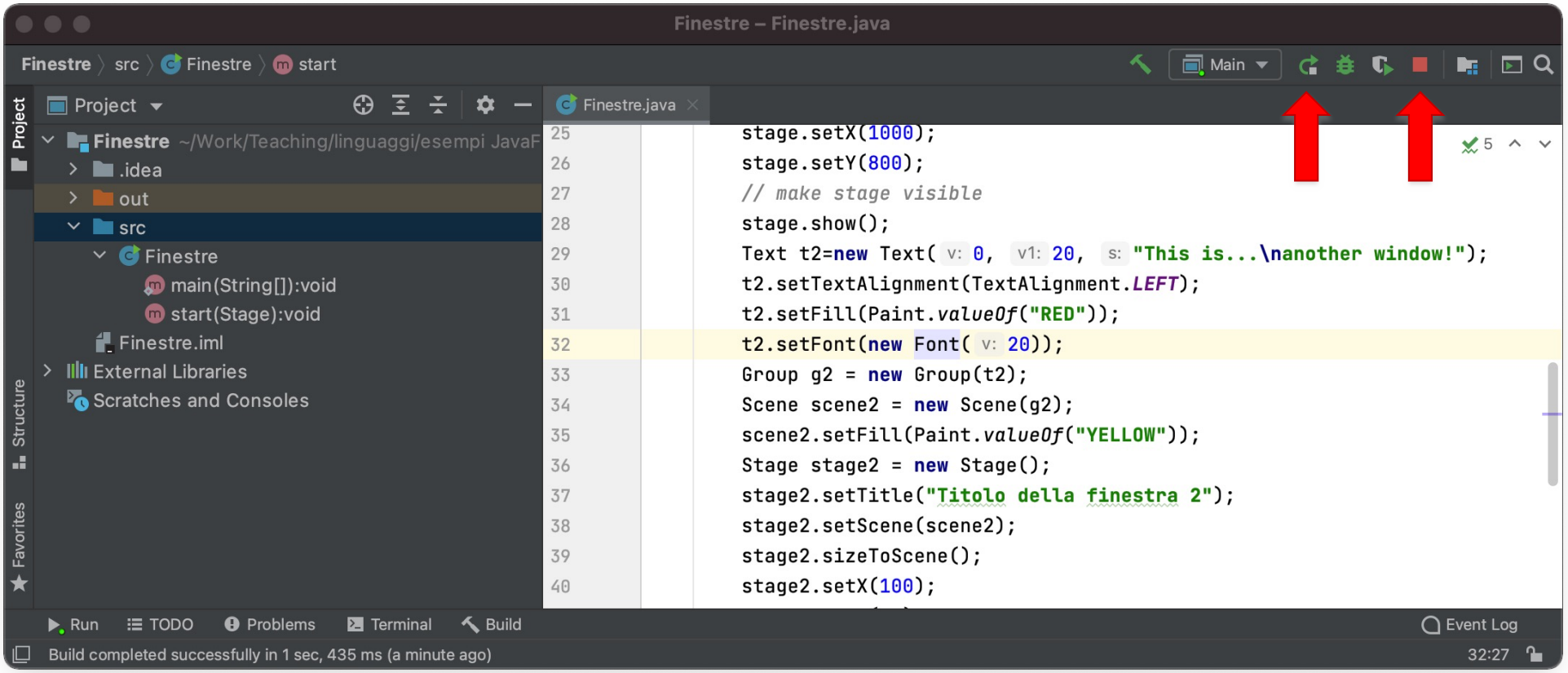
Font (javafx.scene.text) lib (javafx.graphics.jar)

Font (java.awt) < 15 >

- Molte delle classi di JavaFX hanno lo stesso nome in altre librerie di Java (es., AWT) … **fate attenzione a quale selezionate!**

Quando termina l'esecuzione?

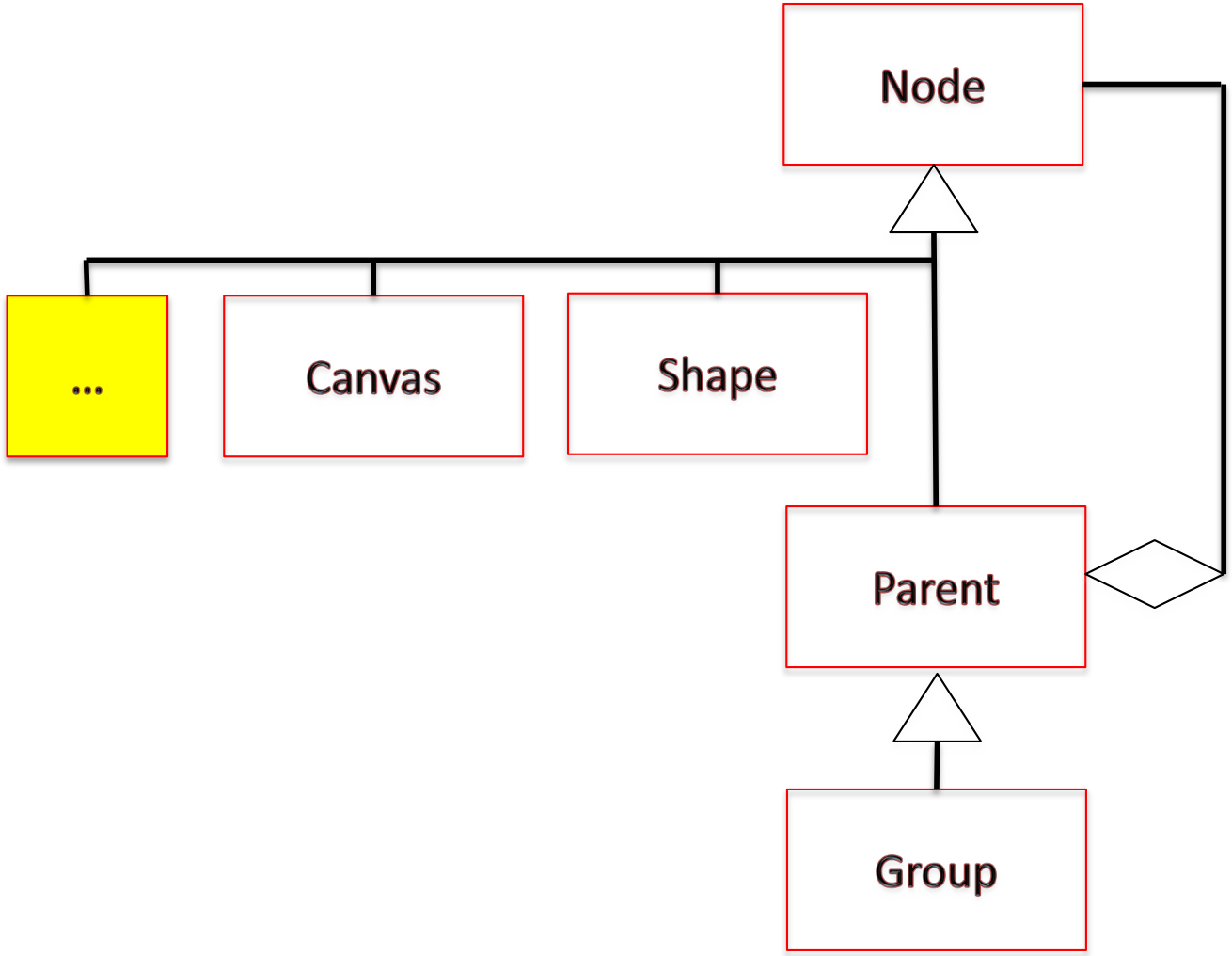
- Il *processo* associato a un'applicazione JavaFX rimane attivo anche dopo la fine di **`start()`**
  - l'applicazione va esplicitamente terminata
  - un *processo* è un *programma* in esecuzione, con il suo stato



Node hierarchy

Node

- Parent
- Shape
- Canvas



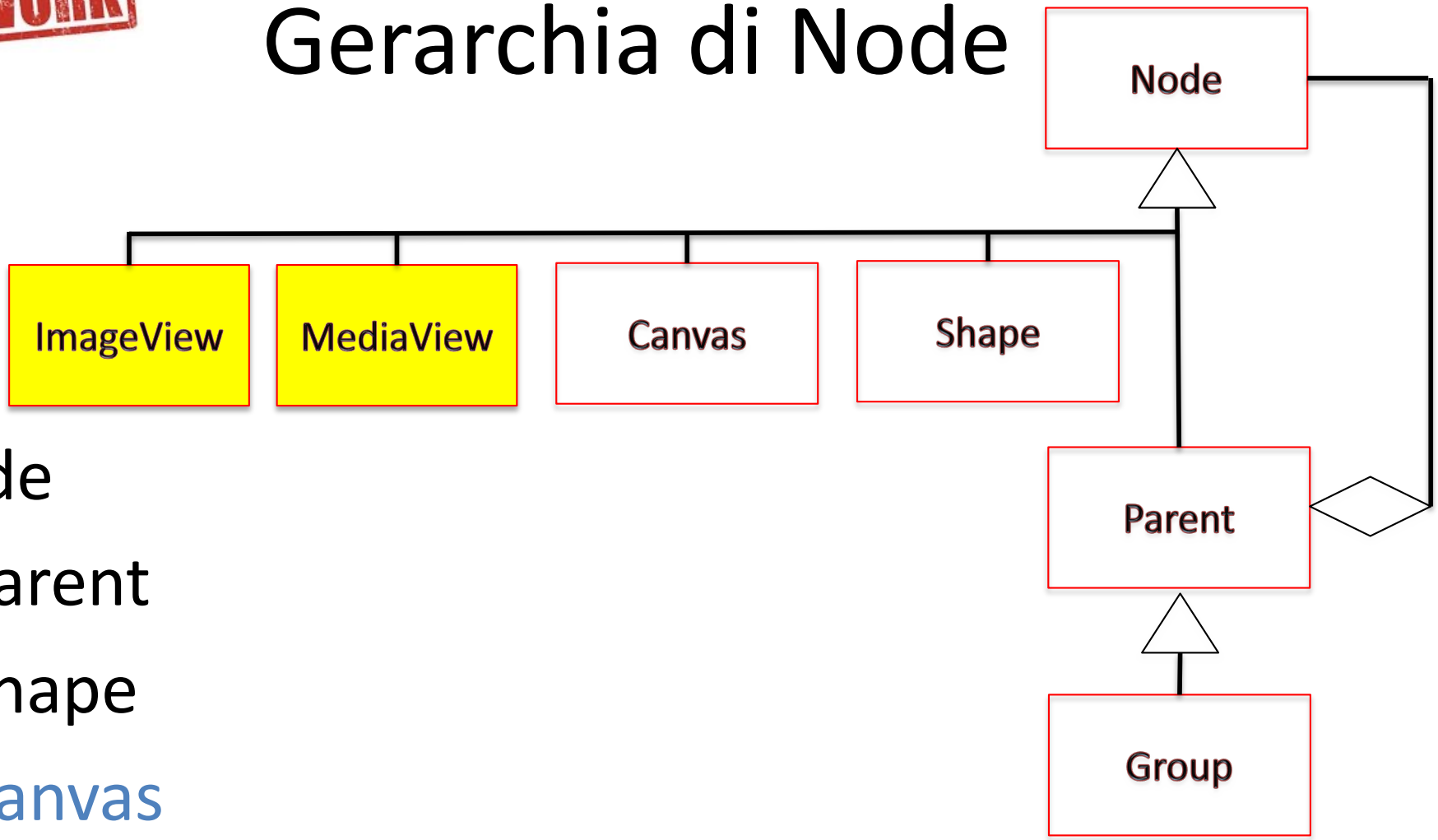
Canvas

Una "tela del pittore" con un metodo per ottenere il suo `GraphicContext` con varie primitive per disegnarci sopra:

- `fillArc()`
- `fillRect()`
- `drawImage()`
- …

`http://docs.oracle.com/javafx/2/canvas/jfxpub-canvas.htm`

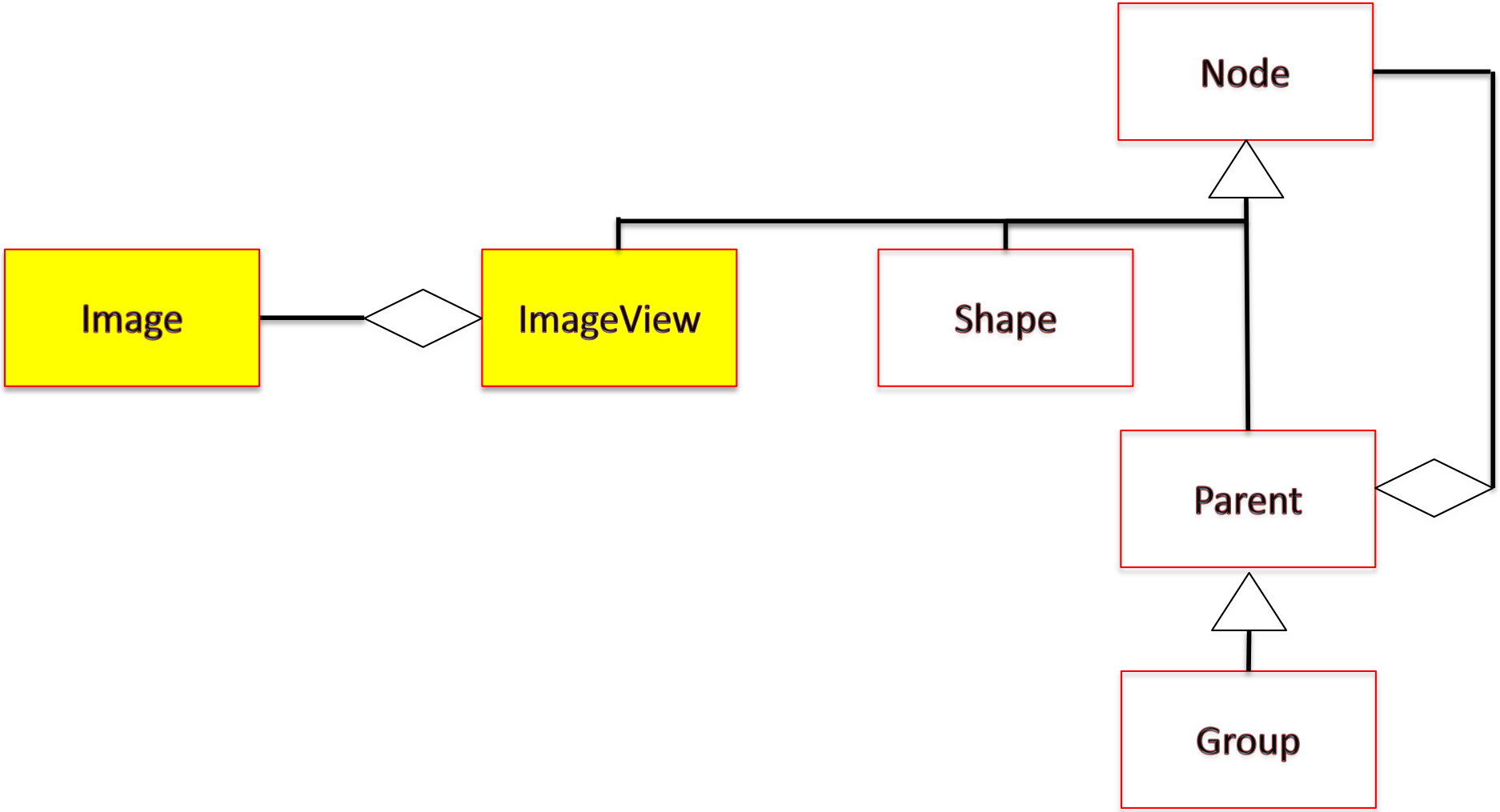
Gerarchia di Node



Node

- Parent
- Shape
- Canvas
- ImageView
- MediaView

ImageView & Image



Gerarchia di Parent (parziale)

Parent

- Control
  - superclasse di vari widget, tra cui FileChooser
- ...

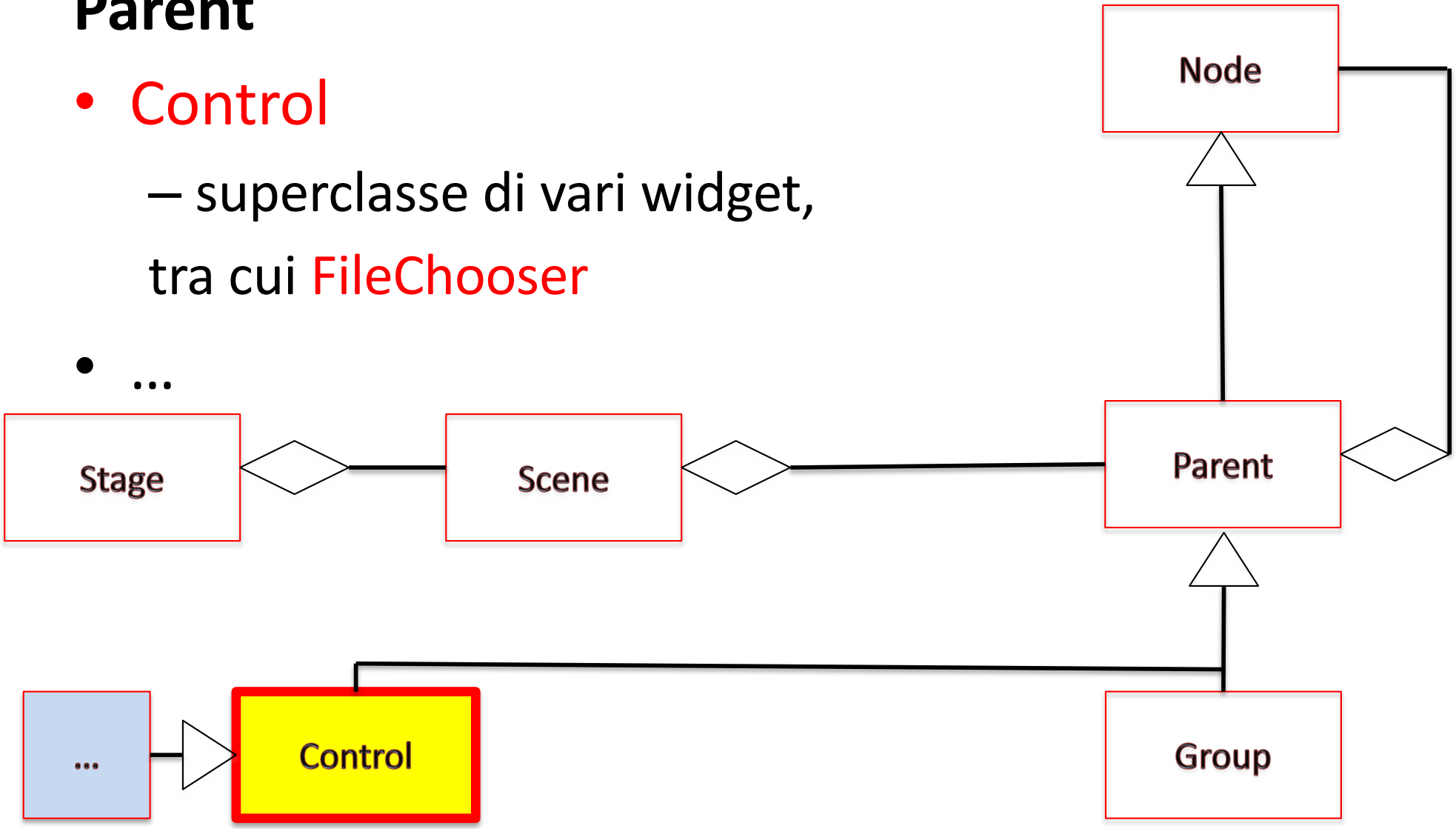
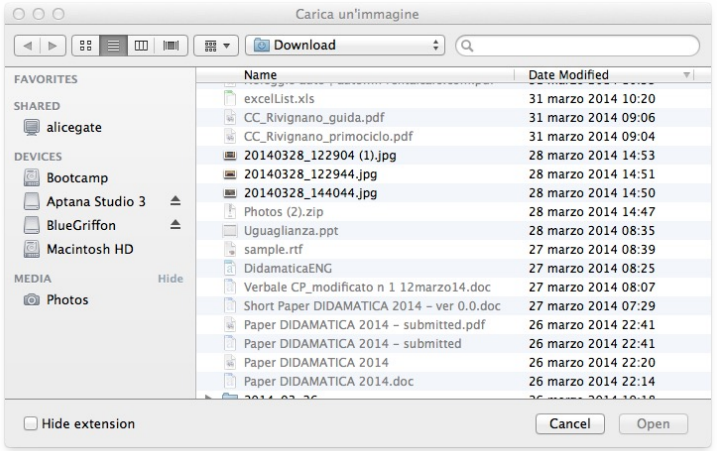


Image & File

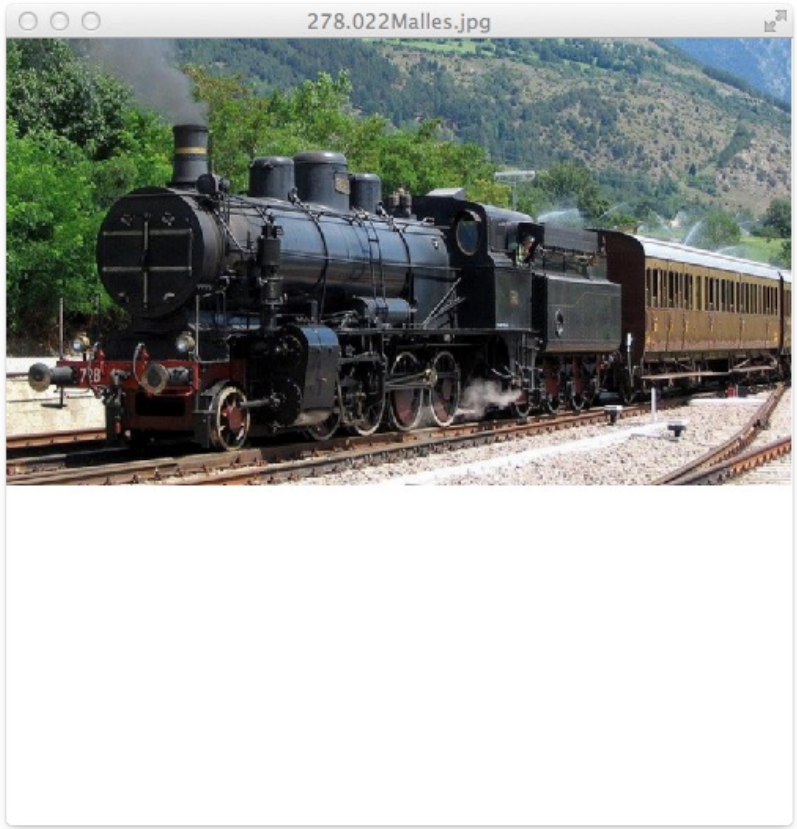



```
public class FilesAndImages extends Application {
    public void start(Stage stage) {
        FileChooser fileChooser = new FileChooser();
         fileChooser.setTitle("Carica un'immagine");
        fileChooser.getExtensionFilters().addAll(
                new FileChooser.ExtensionFilter("JPG", "*.jpg"),
                new FileChooser.ExtensionFilter("PNG", "*.png")
         );
        String url = System.getProperty("user.home");
         File f=new File(url);
        fileChooser.setInitialDirectory(f); // bugged on MacOsX
        File file = fileChooser.showOpenDialog(stage);
         if (file == null) {
             System.out.println("No file chosen");
             System.exit(1);
         }
```

Image & File

```
        Image image = new Image("file://" +
                    file.getAbsolutePath(), 500, 500, true, true);
        ImageView iw = new ImageView(image);
        Group root = new Group(iw);
        Scene scene = new Scene(root, 500,500);
        stage.setTitle(file.getName());
        stage.setScene(scene);
        stage.sizeToScene();
        stage.show();
    }

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



Parent hierarchy

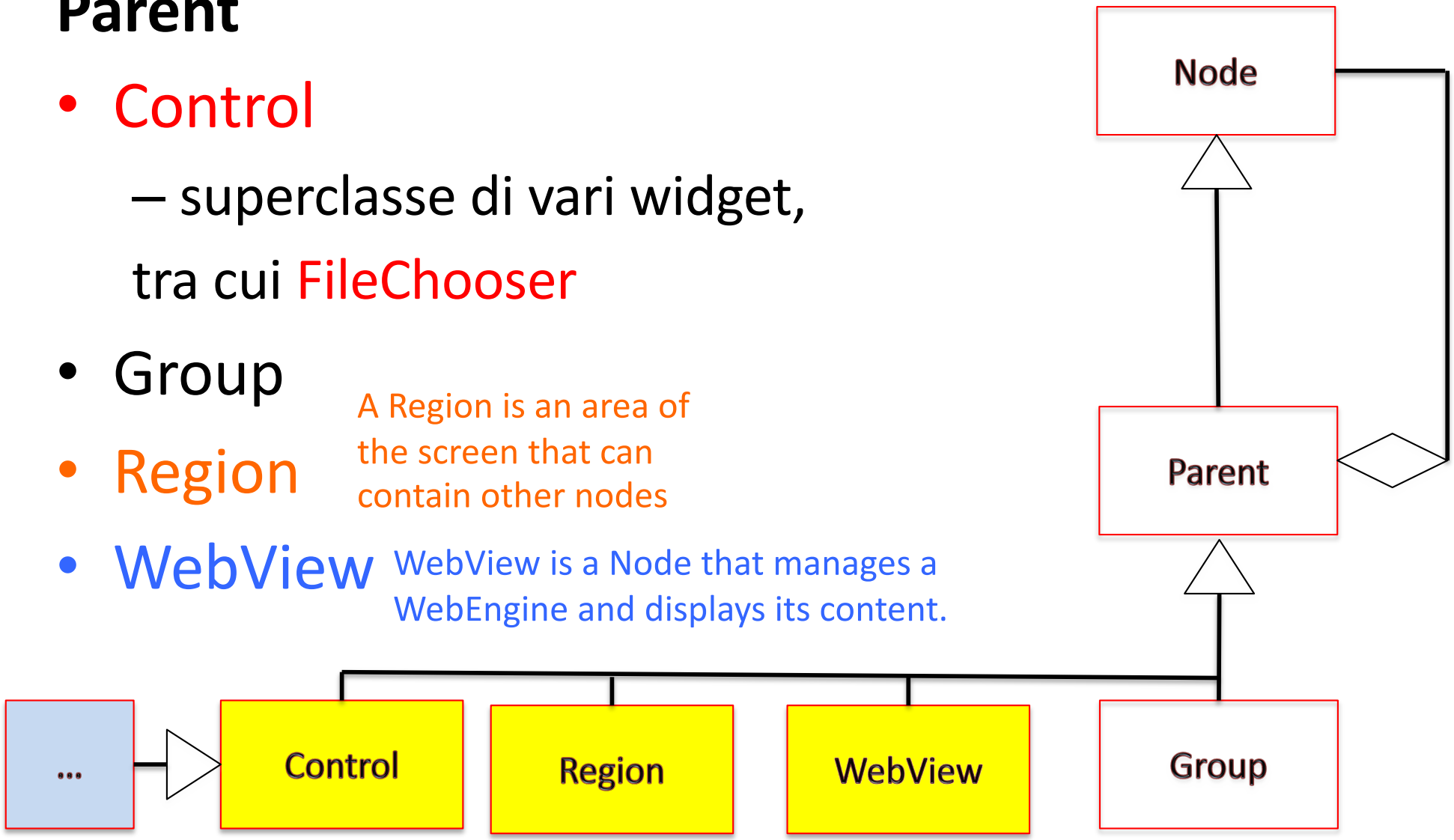
Parent

- Control
  - superclasse di vari widget,

    tra cui FileChooser
- Group
- Region
- WebView

A Region is an area of the screen that can contain other nodes

WebView is a Node that manages a WebEngine and displays its content.



JavaFX UI Controls

Controls


Button:
Cancel
Boxy
Cancel
Toggle Button:
First
Second
Third
Long ..... really Long
Hyperlink:
Hello I am a hyperlink
I can have an icon too
CheckBox:
Normal/Undefined/Selected
Hit me dude
Are you sure
There you go
Radio Button:
Bye
Hello
Menu Buttons:
Menu Button
Menu Button
ScrollView:
Re...
ProgressBar:
ProgressIndicator
Indeterminate:
ListView:
Item0
Item1
Item2
Item3
Item4
7
8
9
10
11


HOMEWORK

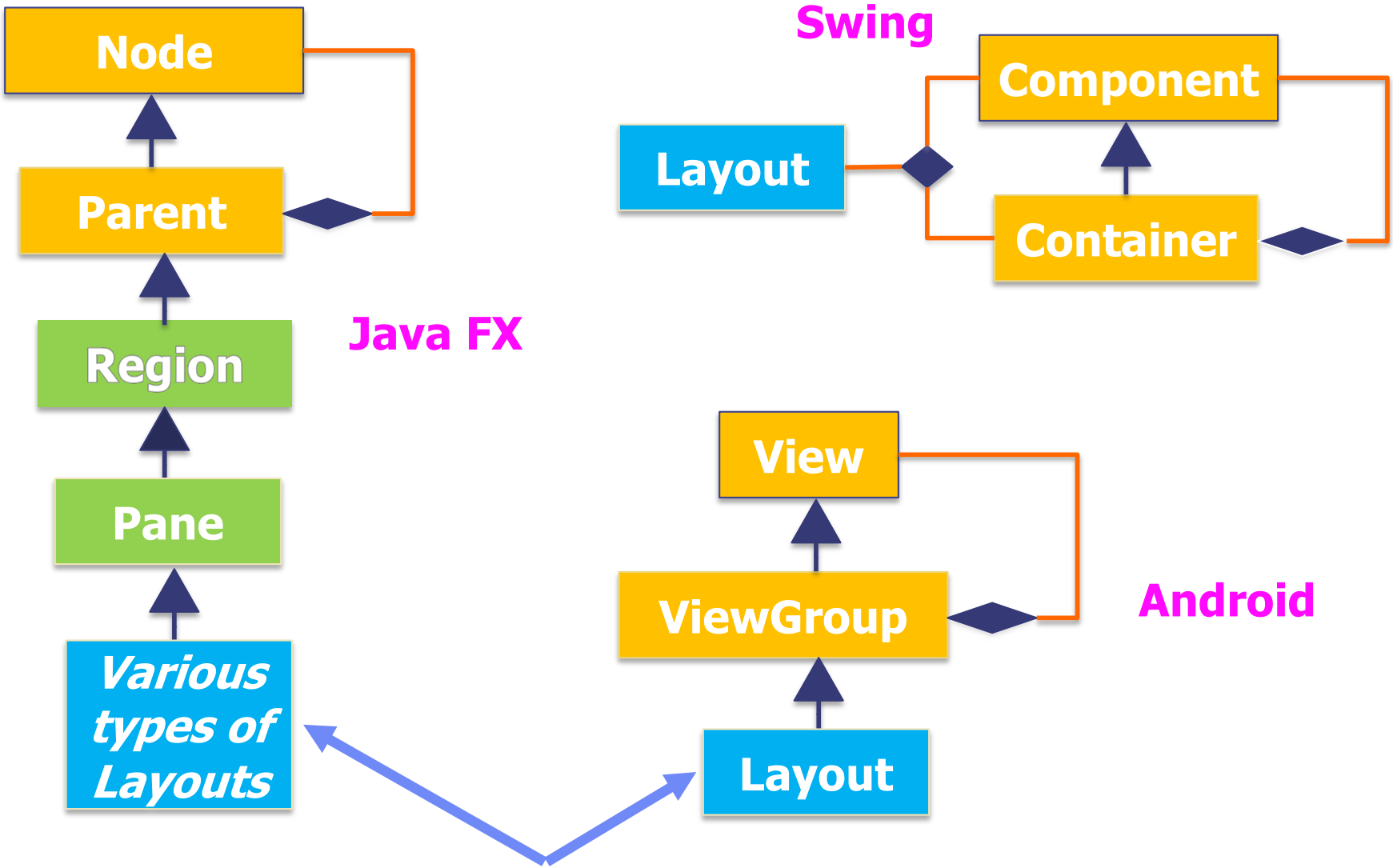
Posizionamento di un **`Node`**

- Non vengono forniti metodi per il posizionamento assoluto …
  - es., **`setX`**, **`setY`** come per **`Stage`**
  - sono tuttavia presenti in alcune sottoclassi
- … in compenso sono definiti i metodi **`setLayoutX`**, **`setTranslateX`**
- A cosa servono?

Posizionamento ***automatico*** mediante ***layout***

- Semplificano il compito del programmatore definendo ***contenitori*** di oggetti che vengono posizionati secondo regole prestabilite
- Fondamentale nel progetto di interfacce
  - Es., sviluppo di applicazioni su Android
- JavaFX fornisce numerose alternative
- Per i curiosi:
  `https://docs.oracle.com/javase/8/javafx/layout-tutorial/index.html`

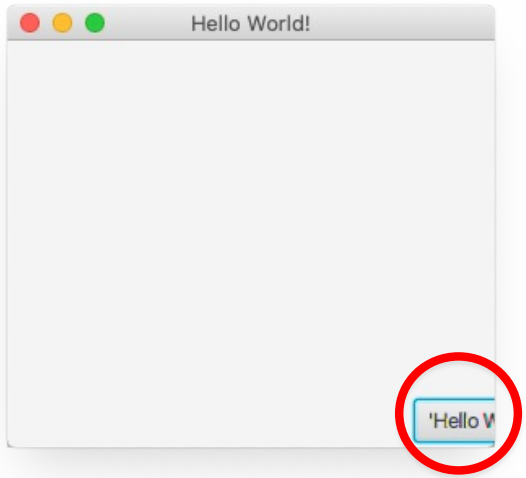
FX vs. Swing and Android architectures



contenitori con una loro regola di posizionamento delle componenti

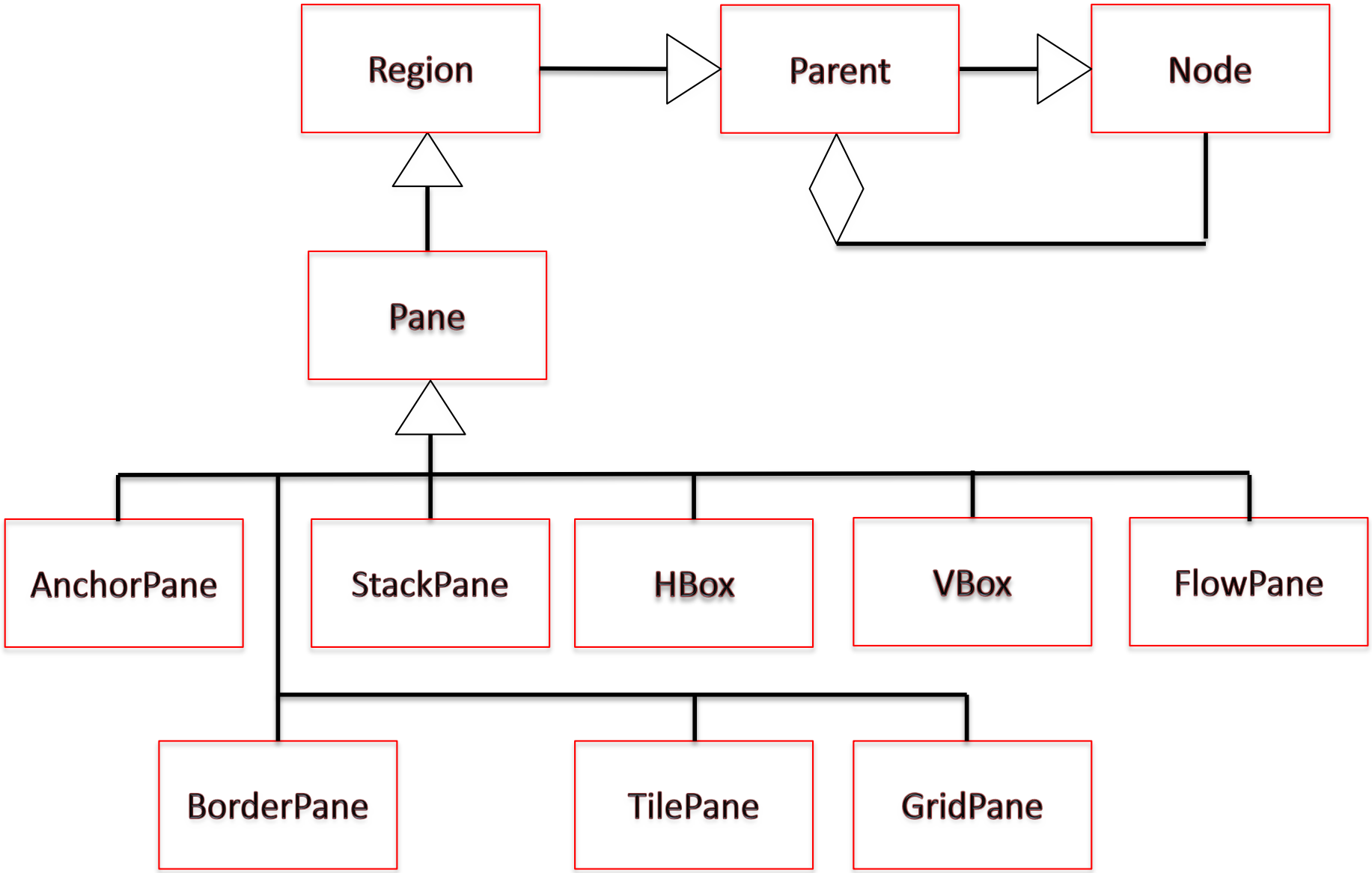
Posizionamento assoluto: **Pane**

```
public void start(Stage primaryStage) {
    primaryStage.setTitle("Hello World!");
    Button btn = new Button();
    btn.setText("'Hello World'");
    Pane root = new Pane();
    btn.setLayoutX(250);
    btn.setLayoutY(220);
    root.getChildren().add(btn);
    primaryStage.setScene(new Scene(root, 300, 250));
    primaryStage.show();
}
```



In generale, ***da evitare***!

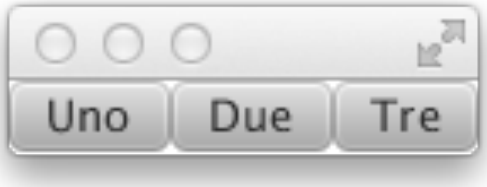
Layout predefiniti



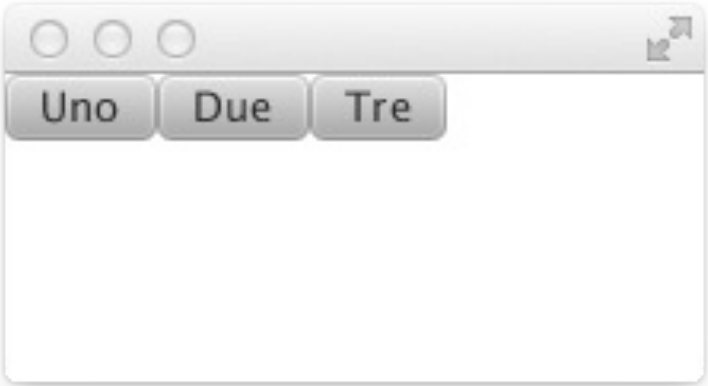
HBox

```
public class Layout1 extends Application {
    public void start(Stage stage) {
        Pane layout = new HBox();
        layout.getChildren().add(new Button("Uno"));
        layout.getChildren().add(new Button("Due"));
        layout.getChildren().add(new Button("Tre"));
        Group root = new Group(layout);
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }
    …
}
```

alla creazione



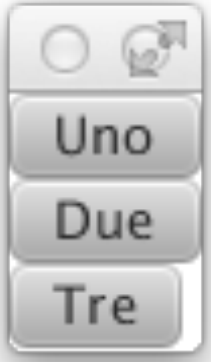
allargando la finestra



Allinea in orizzontale

VBox

```
public class Layout1 extends Application {
    public void start(Stage stage) {
        Pane layout = new VBox();
        layout.getChildren().add(new Button("Uno"));
        layout.getChildren().add(new Button("Due"));
        layout.getChildren().add(new Button("Tre"));
        Group root = new Group(layout);
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }
…
}
```



Allinea in verticale

StackPane

```
public class Layout1 extends Application {
    public void start(Stage stage) {
        StackPane layout = new StackPane();
        layout.getChildren().add(new Button("Uno"));
        layout.getChildren().add(new Button("Due"));
        layout.getChildren().add(new Button("Tre"));
        Group root = new Group(layout);
        Scene scene = new Scene(root);
        stage.setScene(scene);
        stage.show();
    }
…
}
```

Impila gli elementi

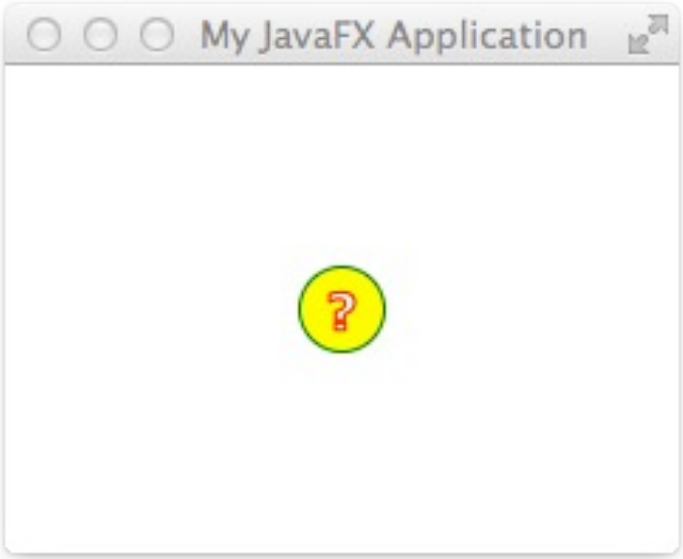
StackPane

```
public class Layout1 extends Application {
    public void start(Stage stage) {
        StackPane layout = new StackPane();
        layout.getChildren().add(new Button("Uno"));
        layout.getChildren().add(new Button("Due"));
        layout.getChildren().add(new Button("Tre"));
        //Group root = new Group(layout);
        //Scene scene = new Scene(root);
        Scene scene = new Scene(layout);
        stage.setScene(scene);
        stage.show();
    }
…
}
```

In generale, è possibile creare una **Scene** direttamente da un **Layout**, ma attenzione: ognuno ha un suo allineamento predefinito

StackPane

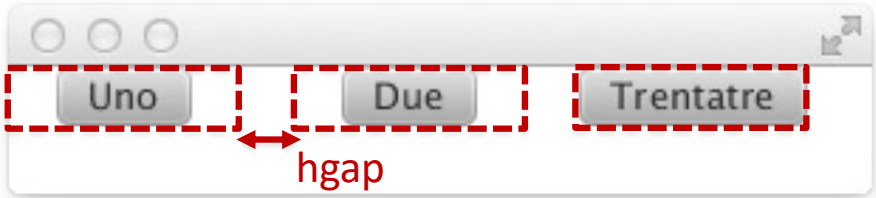
```
public class Layout1 extends Application {
    public void start(Stage stage) {
        StackPane stack = new StackPane();
        Circle helpIcon = new Circle(15, 15, 15);
        helpIcon.setFill(Color.YELLOW);
        helpIcon.setStroke(Color.GREEN);
        Text helpText = new Text("?");
        helpText.setFont(Font.font("Verdana", FontWeight.BOLD, 18));
        helpText.setFill(Color.WHITE);
        helpText.setStroke(Color.RED);
        stack.getChildren().addAll(helpIcon, helpText);
        stack.setAlignment(Pos.CENTER);
        Scene scene = new Scene(stack);
        stage.setTitle("My JavaFX Application");
        stage.setScene(scene);
        stage.show();
    }
…
}
```

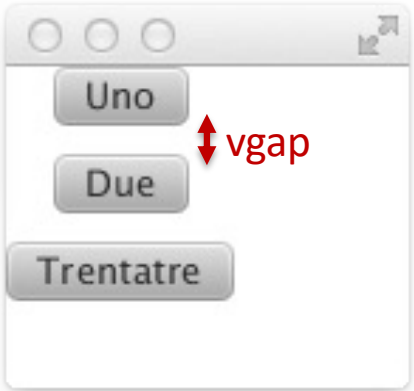


StackPane è utile per creare «overlay» di elementi

TilePane

```
public class Layout1 extends Application {
    public void start(Stage stage) {
        TilePane layout = new TilePane();
        layout.setVgap(10);
        layout.setHgap(20);
        layout.setPrefColumns(2);
        layout.getChildren().add(new Button("Uno"));
        layout.getChildren().add(new Button("Due"));
        layout.getChildren().add(new Button("Trentatre"));
        Scene scene = new Scene(layout);
        stage.setScene(scene);
        stage.show();
    }
…}
```

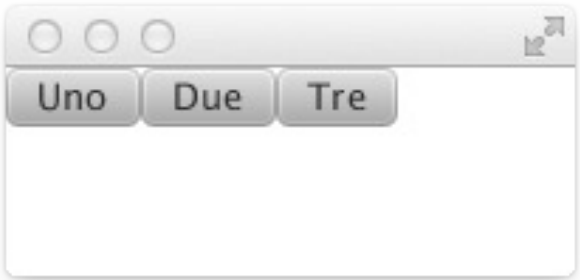




Organizza gli elementi in una griglia di celle di **eguale** dimensione

FlowPane

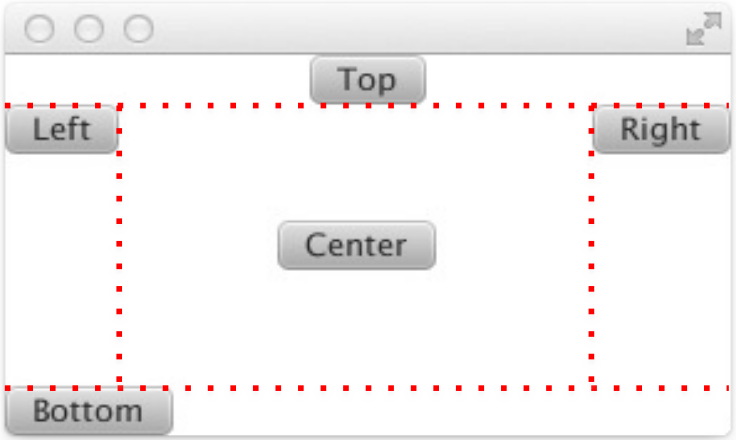
```
public class Layout1 extends Application {
    public void start(Stage stage) {
        FlowPane layout = new FlowPane();
        layout.setPrefWrapLength(100);
        layout.getChildren().add(new Button("Uno"));
        layout.getChildren().add(new Button("Due"));
        layout.getChildren().add(new Button("Tre"));
        Scene scene = new Scene(layout);
        stage.setScene(scene);
        stage.show();
    }…
}
```



Organizza gli elementi in una sequenza continua, che va «a capo» a una distanza configurabile

BorderPane

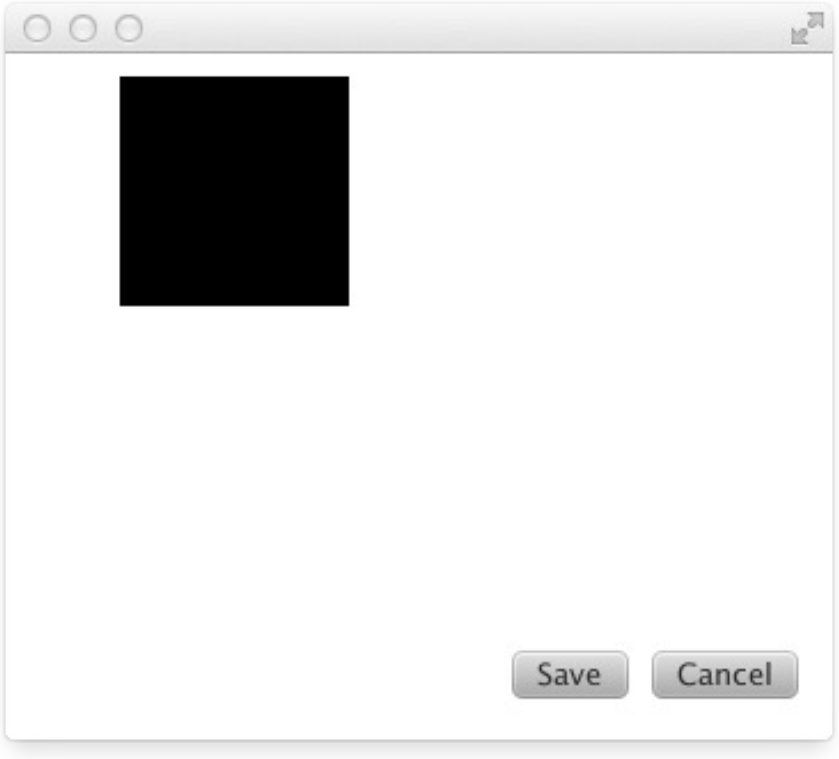
```
public class Layout1 extends Application {
  public void start(Stage stage) {
    BorderPane layout = new BorderPane();
    Button top = new Button("Top");
    BorderPane.setAlignment(top, Pos.TOP_CENTER);
    layout.setTop(top);
    layout.setBottom(new Button("Bottom"));
    layout.setLeft(new Button("Left"));
    layout.setRight(new Button("Right"));
    layout.setCenter(new Button("Center"));
    Scene scene = new Scene(layout);
    stage.setScene(scene);
    stage.show();
  }
…
}
```



Organizza gli elementi in «zone»

AnchorPane

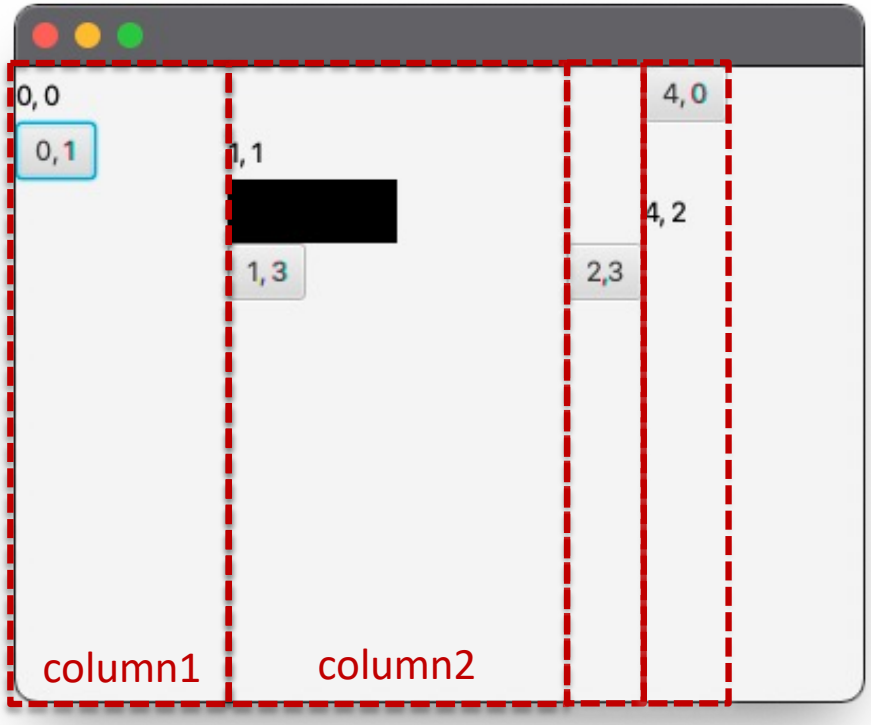
```
public void start(Stage stage) {
    AnchorPane layout = new AnchorPane();
    Button buttonSave = new Button("Save");
    Button buttonCancel = new Button("Cancel");
    HBox hb = new HBox();
    hb.setPadding(new Insets(0, 10, 10, 10));
    hb.setSpacing(10);
    hb.getChildren().addAll(buttonSave, buttonCancel);
    Rectangle r=new Rectangle(100,100);
    layout.getChildren().addAll(r,hb);
    layout.setBottomAnchor(hb, 8.0);
    layout.setRightAnchor(hb, 5.0);
    layout.setTopAnchor(r, 10.0);
    layout.setLeftAnchor(r, 50.0);
    Scene scene = new Scene(layout);
    stage.setScene(scene);
    stage.show();
}
```



Permette di «ancorare» elementi a una zona

GridPane

```
public void start(Stage stage) {
   GridPane layout = new GridPane();
   layout.add(new Text("0, 0"), 0, 0);
   layout.add(new Button("0, 1"), 0, 1);
   layout.add(new Text("1, 1"), 1, 1);
   Rectangle r = new Rectangle(80,30);
   layout.add(r, 1, 2);
   layout.add(new Button("1, 3"), 1, 3);
   layout.add(new Button("2,3"), 2, 3);
   layout.add(new Button("4, 0"), 4, 0);
   layout.add(new Text("4, 2"), 4, 2);
   ColumnConstraints column1 = new ColumnConstraints(100);
   ColumnConstraints column2 = new ColumnConstraints();
   column2.setPercentWidth(40);
   column2.setHgrow(Priority.ALWAYS);
   layout.getColumnConstraints().addAll(column1, column2);
   stage.setScene(scene);
   stage.show();
}
```



vedi documentazione!

Organizza gli elementi in una griglia di cui non è necessario dare dimensione prefissata

Combinazioni di layout

Container classes that automate common layout models

The HBox class arranges its content nodes horizontally in a single row.

The VBox class arranges its content nodes vertically in a single column.

The StackPane class places its content nodes in a back-to-front single stack.

The TilePane class places its content nodes in uniformly sized layout cells or tiles

The FlowPane class arranges its content nodes in either a horizontal or vertical "flow", wrapping at the specified width (for horizontal) or height (for vertical) boundaries.

The BorderPane class lays out its content nodes in the top, bottom, right, left, or center region.

The AnchorPane class enables developers to create anchor nodes to the top, bottom, left side, or center of the layout.

The GridPane class enables the developer to create a flexible grid of rows and columns in which to lay out content nodes.

To achieve a desired layout structure, different containers can be nested within a JavaFX application.

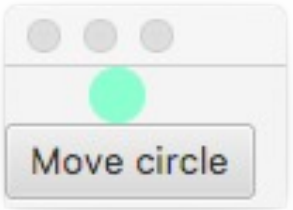
Posizionamento, dimensionamento, allineamento

- È possibile modificare il posizionamento automatico degli elementi
  - **`setLayoutX`** e **`setTranslateX`** (e analoghi per la Y)
- È possible dimensionare direttamente gli elementi
  - Es., **`btn.setPrefWidth(200);`**
- … oppure specificare vincoli sulle dimensioni
  - Es., **`btn.setMinWidth(100);`**
- Per ulteriori tecniche ed esempi di dimensionamento e allineamento:
  `https://docs.oracle.com/javafx/2/layout/size_align.htm`

Esempio di posizio-namento

```
class Controller implements EventHandler<ActionEvent>{
      int inc = 0; Circle c; Controller(Circle c)
                        @Override
          public void handle(ActionEvent event) {
                          inc += 10;
                    c.setLayoutX(inc % 100);
                   //c.setCenterX(inc % 100);
                  //c.setTranslateX(inc % 100);
                System.out.println(c.getLayoutX()
    + " " + c.getTranslateX()+" "+c.getCenterX());}}
```

```
public class MoveBall extends Application{
  @Override
  public void start(Stage primaryStage) {
    Circle c = new Circle(200, 200, 10);
    c.setFill(Color.AQUAMARINE);
    Button btn = new Button();
    btn.setText("Move circle");
    btn.setOnAction(new Controller(c));
    VBox root = new Vbox();
    root.getChildren().addAll(c, btn);
    Scene scene = new Scene(root, 100, 50);
    primaryStage.setScene(scene);
    primaryStage.show();
}
```



Settando la LayoutX…

```
class Controller implements EventHandler<ActionEvent>{
      int inc = 0; Circle c; Controller(Circle c)
                        @Override
        public void handle(ActionEvent event) {
                          inc += 10;
                    c.setLayoutX(inc % 100);
                   //c.setCenterX(inc % 100);
                  //c.setTranslateX(inc % 100);
                 System.out.println(c.getLayoutX()
     + " " + c.getTranslateX()+" "+c.getCenterX());}}
```


```
10.0 0.0 200.0
20.0 0.0 200.0
30.0 0.0 200.0
40.0 0.0 200.0
50.0 0.0 200.0
```

I dati cambiano, ma non funziona:
il cerchio non si muove!

Provando con CenterX…

```
class Controller implements EventHandler<ActionEvent>{
      int inc = 0; Circle c; Controller(Circle c)
                        @Override
        public void handle(ActionEvent event) {
                          inc += 10;
                    //c.setLayoutX(inc % 100);
                     c.setCenterX(inc % 100);
                   //c.setTranslateX(inc % 100);
                 System.out.println(c.getLayoutX()
     + " " + c.getTranslateX()+" "+c.getCenterX());}}
```


```
-190.0 0.0 10.0
0.0 0.0 20.0
-10.0 0.0 30.0
-20.0 0.0 40.0
-30.0 0.0 50.0
```

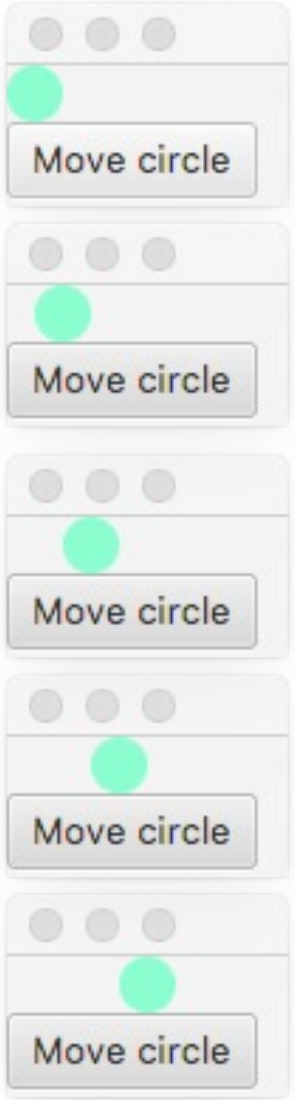
Non funziona neanche così…

e finalmente con setTranslateX…

```
class Controller implements EventHandler<ActionEvent>{
      int inc = 0; Circle c; Controller(Circle c)
                        @Override
        public void handle(ActionEvent event) {
                          inc += 10;
                    //c.setLayoutX(inc % 100);
                    //c.setCenterX(inc % 100);
                    c.setTranslateX(inc % 100);
                System.out.println(c.getLayoutX()
     + " " + c.getTranslateX()+" "+c.getCenterX());}}
```

Funziona!

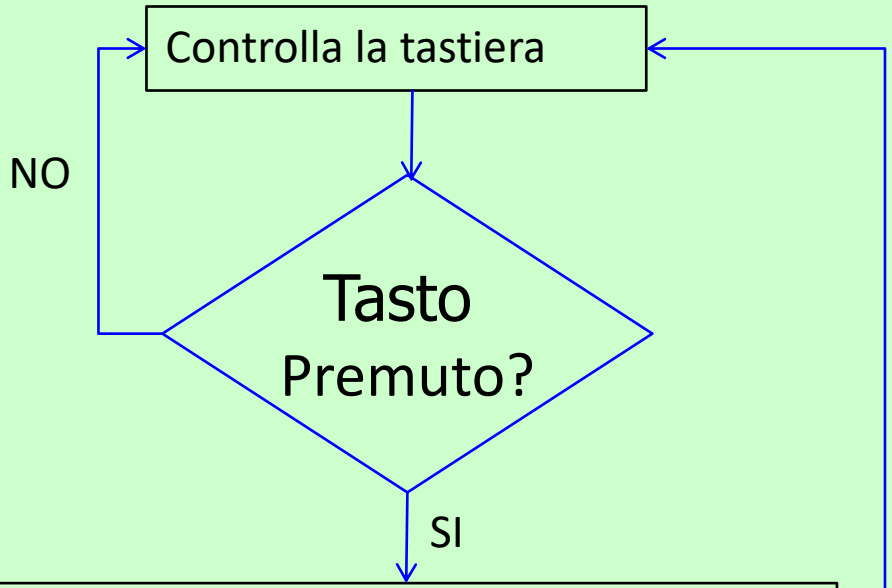
```
-190.0 10.0 200.0
-190.0 20.0 200.0
-190.0 30.0 200.0
-190.0 40.0 200.0
```



Architettura di un framework

FRAMEWORK

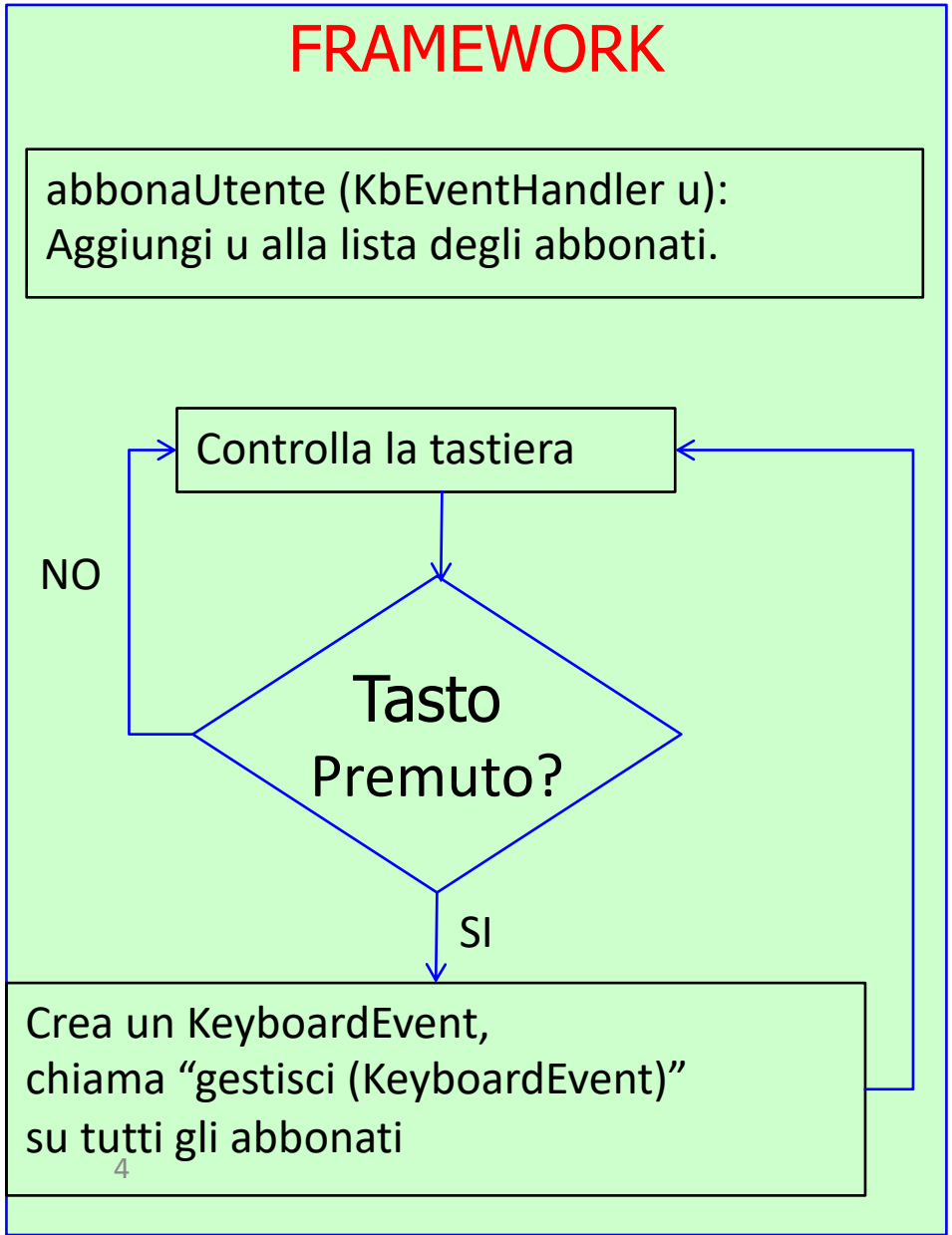
abbonaUtente (KbEventHandler u):
Aggiungi u alla lista degli abbonati.




```
interface KbEventHandler {
  gestisci (KeyboardEvent k)
}
```

```
class MyProgram extends Framework
  implements  KbEventHandler  {
    MyProgram (){
        startFramework();
        abbonaUtente(this);
    }
    void gestisci (KeyboardEvent k) {
          …;
    }
}
```

Architettura di un framework




```
interface KbEventHandler {
  gestisci (KeyboardEvent k)
}
```

```
class MyProgram extends Framework
  implements  KbEventHandler  {
    MyProgram (){
        startFramework();
        Listener c=new Listener();
        abbonaUtente(c);
    }

}
```

```
class Listener
  implements  KbEventHandler  {
     void gestisci (KeyboardEvent k) {
         …;
     }
}
```

ascoltatore degli eventi (controller)

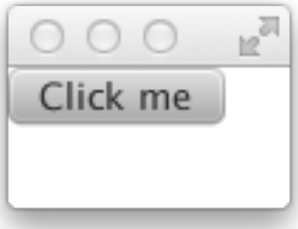
Gestione degli eventi

- Le interfacce grafiche sfruttano un paradigma architetturale «ad eventi»
- Risulta naturale, poiché l'esecuzione del programma è intrinsecamente determinata dagli eventi esterni generati dall'utente
  - anziché da una sequenza di azioni prefissata
- Paradigma generale, con applicazioni al di là di JavaFX e interfacce utente…
- Necessita di concetti e costrutti linguistici dedicati

Alcuni eventi base

```
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
```

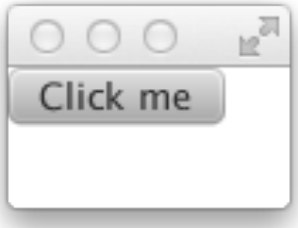
```
public class Test extends Application {
  public void start(Stage stage) {
    Button btn = new Button();
    btn.setText("Click me");
    Listener a = new Listener();
    btn.addEventHandler(Event.ANY, a);
    Group root = new Group(btn);
    Scene scene = new Scene(root, 300, 250);
    stage.setScene(scene);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



Alcuni eventi base

1 Ricevuto un evento di tipo ACTION

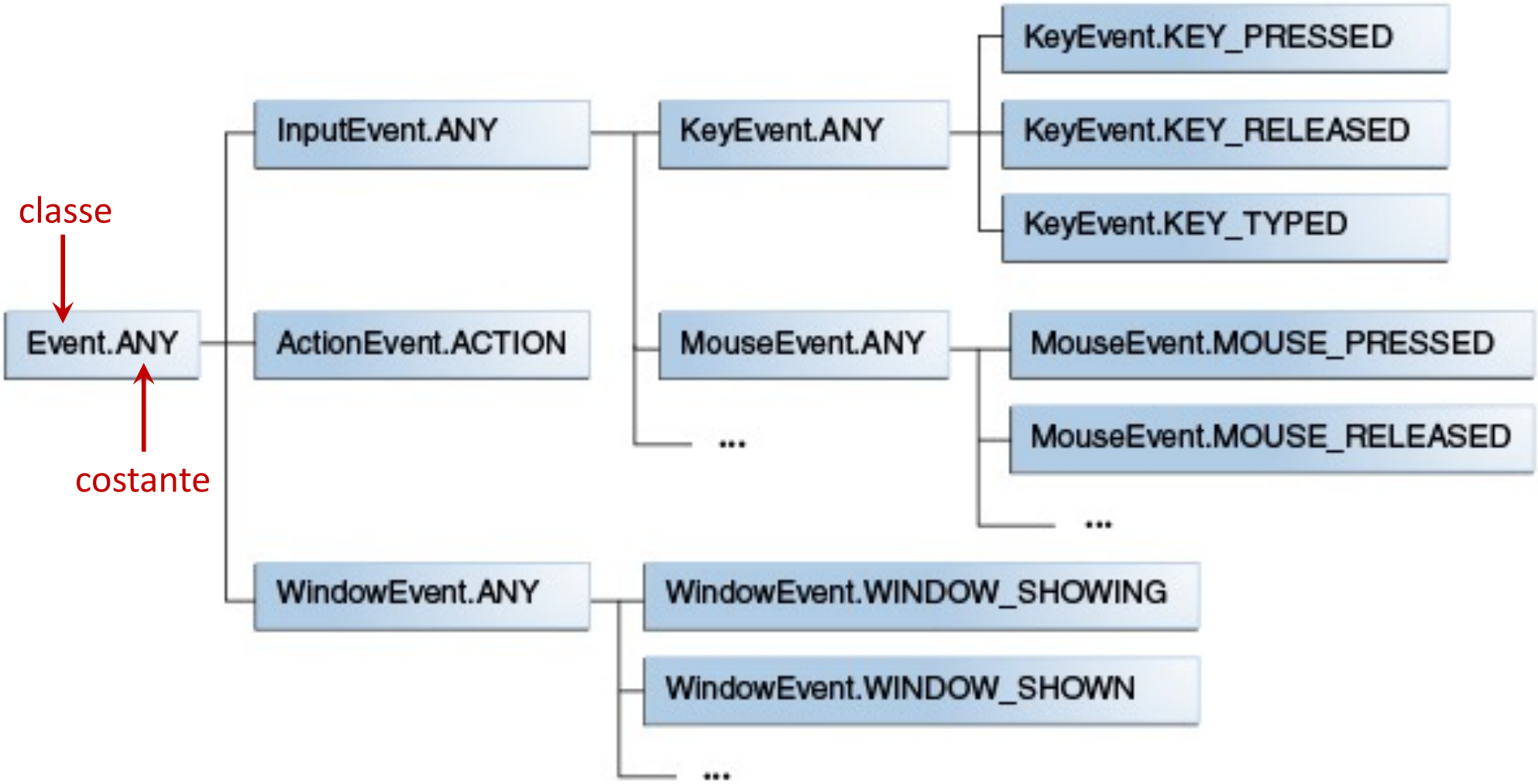
```
public class Test extends Application {
      public void start(Stage stage) {
        Button btn = new Button();
        btn.setText("Click me");
        Listener a=new Listener();
        btn.addEventHandler(ActionEvent.ACTION, a);
        Group root = new Group(btn);
        Scene scene = new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



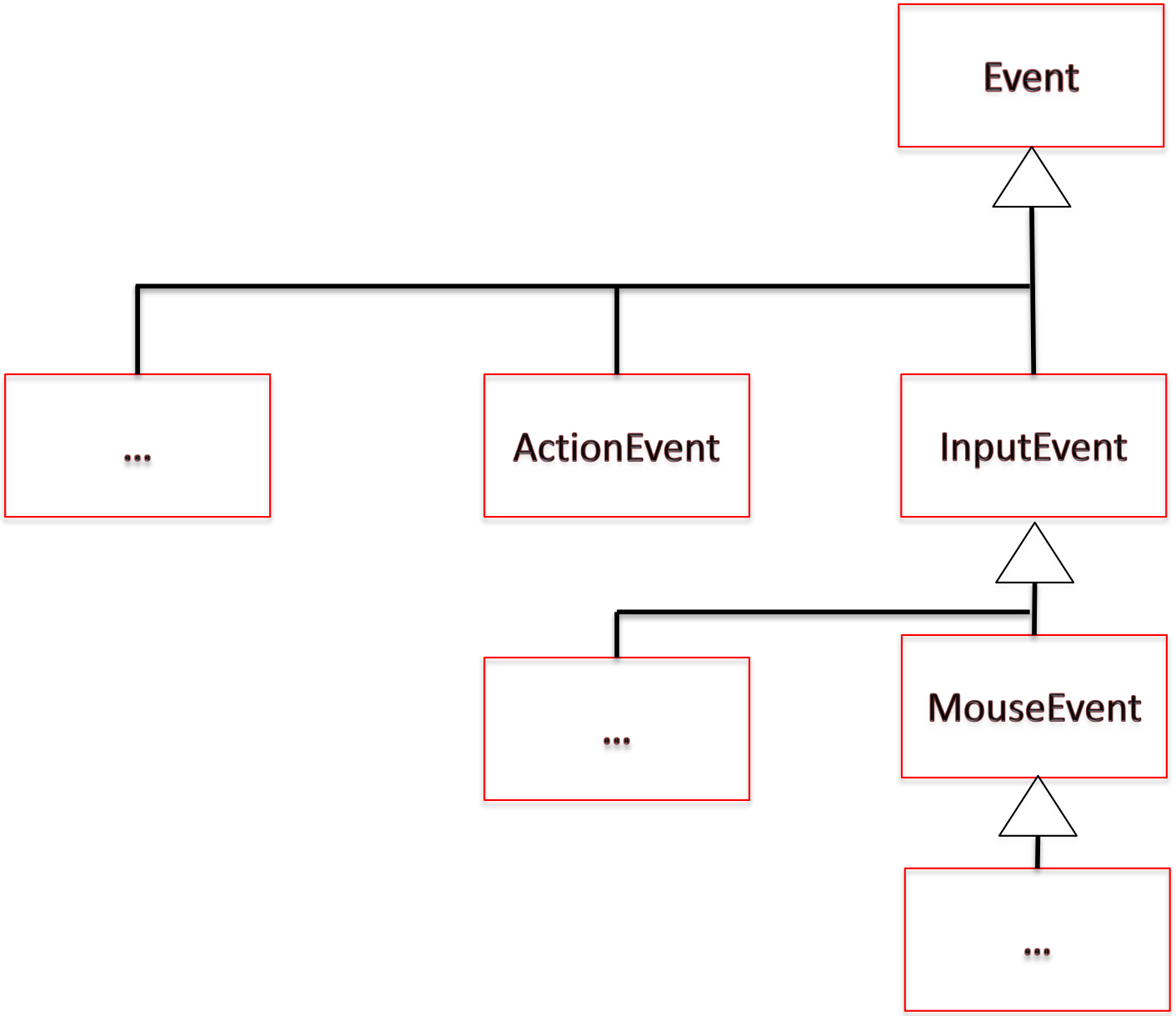
Gestire gli eventi

- Esiste una gerarchia di eventi predefinita in JavaFX, associata agli elementi disponibili per l'interfaccia grafica
  - Ogni oggetto **`Event`** raggruppa uno o più sottoeventi, definiti da costanti
  - Ogni sottoclasse definisce attributi e metodi specifici per tipologia di eventi
- Il codice applicativo può «reagire» a questi eventi specificando uno o più ***listener*** («ascoltatore»)
  - deve implementare l'interfaccia **`EventHandler`**
  - tipicamente è un oggetto di classe apposita, ma può essere l'applicazione stessa

Gerarchia di **`Event`** (codici)

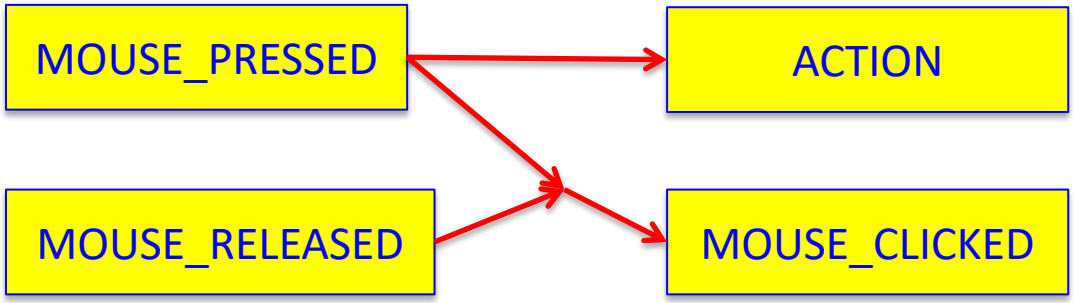


Gerarchia di **`Event`** (classi)



Eventi del Button

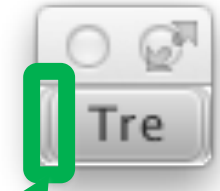
1 Ricevuto un evento di tipo
INPUT_METHOD_TEXT_CHANGED
2 Ricevuto un evento di tipo MOUSE_ENTERED
3 Ricevuto un evento di tipo
MOUSE_ENTERED_TARGET
4 Ricevuto un evento di tipo MOUSE_MOVED
…
12 Ricevuto un evento di tipo MOUSE_MOVED
13 Ricevuto un evento di tipo MOUSE_PRESSED
14 Ricevuto un evento di 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



(pressed and released on the same node)

`StackPane`: chi riceve gli eventi?

```
public class Layout1 extends Application {
  public void start(Stage stage) {
    StackPane layout = new StackPane();
    EventHandler<Event> eh = new EventHandler<Event>() {
      public void handle(Event event) {
        Button source = (Button) event.getSource();
        System.out.println("Ricevuto da " + source.getText() + ": "
                           + event.getEventType());
    }};
    Button b1 = new Button("Uno");
    Button b2 = new Button("Due");
    Button b3 = new Button("Tre");
    // b3.setMouseTransparent(true);
    b1.addEventHandler(Event.ANY, eh);
    b2.addEventHandler(Event.ANY, eh);
    b3.addEventHandler(Event.ANY, eh);
    layout.getChildren().addAll(b1,b2,b3);
    Scene scene = new Scene(layout);
    stage.setScene(scene);
    stage.show();
  }
…
}
```



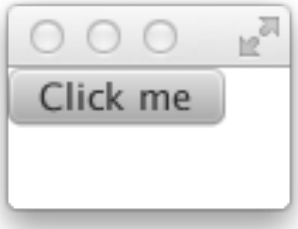
Ricevuto da Due: MOUSE_ENTERED
Ricevuto da Due: MOUSE_MOVED
Ricevuto da Due: MOUSE_MOVED
Ricevuto da Due: MOUSE_EXITED
Ricevuto da Tre: MOUSE_ENTERED
Ricevuto da Tre: MOUSE_MOVED
Ricevuto da Tre: MOUSE_MOVED
Ricevuto da Tre: MOUSE_PRESSED
Ricevuto da Tre: ACTION
Ricevuto da Tre: MOUSE_RELEASED
Ricevuto da Tre: MOUSE_CLICKED

Listener multipli

```
class OListener
      implements EventHandler{
  public void handle(Event t) {
       System.out.println(t); }
}
```

```
class EListener
      implements EventHandler{
  public void handle(Event t) {
       System.err.println(t); }
}
```

```
public class Test extends Application {
      public void start(Stage stage) {
        Button btn = new Button();
        btn.setText("Click me");
        OListener o = new OListener();
        EListener e = new EListener();
        btn.addEventHandler(ActionEvent.ACTION, o);
        btn.addEventHandler(ActionEvent.ACTION, e);
        Group root = new Group(btn);
        Scene scene = new Scene(root, 300, 250);
        stage.setScene(scene);
        stage.show();
    }
    public static void main(String[] args){
      launch(args);
    }
}
```



… l'ordine di esecuzione di listener associati allo stesso elemento non è specificato da JavaFX

Listener esterno


```
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");
      Listener a = new Listener(this);
      btn.addEventHandler(ActionEvent.ACTION, a);
      Group root = new Group(btn);
      root.getChildren().add(text);
      Scene scene = new Scene(root);
      stage.setScene(scene);
      stage.show();
    }
    public void updateText(int n){
      text.setText("Hai cliccato "
                   + n +" volte");
    }
    public static void main(String[] args) {
      launch(args);
    }}
```

```
class Listener
      implements EventHandler {
  AppWithEvents awe = null;
  int counter = 0;
  Listener(AppWithEvents a){
    awe = a;
  }
  public void handle(Event t) {
    awe.updateText(++counter);
  }
}
```

Dichiarato fuori dalla classe principale

Listener interno


```
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");
      Listener a = new Listener(this);
      btn.addEventHandler(ActionEvent.ACTION, a);
      Group root = new Group(btn);
      root.getChildren().add(text);
      Scene scene = new Scene(root);
      stage.setScene(scene);
      stage.show();
    }
    public void updateText(int n){
      text.setText("Hai cliccato "
                   + n +" volte");
    }
    public static void main(String[] args) {
      launch(args);
    }}
```

Dichiarato dentro la classe principale

```
class Listener
      implements EventHandler {
  AppWithEvents awe = null;
  int counter = 0;
  Listener(AppWithEvents a){
    awe = a;
  }
  public void handle(Event t) {
    awe.updateText(++counter);
  }
}
```

Listener interno


```
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");
      Listener a = new Listener();
      btn.addEventHandler(ActionEvent.ACTION, a);
      Group root = new Group(btn);
      root.getChildren().add(text);
      Scene scene = new Scene(root);
      stage.setScene(scene);
      stage.show();
    }
    class Listener
      implements EventHandler{
      int counter =  0;
      public void handle(Event t) {
        updateText(++counter);
      }
    } // end of Listener
```

```
  public void updateText(int n){
    text.setText("Hai cliccato "
                 +n+" volte");
  }
  public static void main(
       String[] args) {
    launch(args);
  }
} // end of AppWithEvents
```

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(){
        int counter=0;
        public void handle(Event 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 è «anonima», implementa l'interfaccia EventHandler …

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

```
    public void updateText(int n){
      text.setText("Hai cliccato "
                   +n+" volte");
    }
    public static void main(
          String[] args) {
      launch(args);
    }
} // end of AppWithEvents
```

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

Listener “integrato”


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

La classe principale funge altresì da gestore degli eventi

```
    public void updateText(int n){
      text.setText("Hai cliccato "
                   +n+" volte");
    }
    public static void main(
          String[] args) {
      launch(args);
    }
} // end of AppWithEvents
```

Listener e generics


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

Usate le versioni generiche!

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

```
  public void updateText(int n){
    text.setText("Hai cliccato "
                 +n+" volte");
  }
  public static void main(
        String[] args) {
    launch(args);
  }
} // end of AppWithEvents
```

Convenience methods


```
public class AppWithEvents extends Application
             implements EventHandler<ActionEvent> {
    Text text = null;
    int counter = 0;
    public void start(Stage stage) {
      text = new Text(10,50,"Non hai mai cliccato ");
      Button btn = new Button();
      btn.setText("Click me");
      btn.setOnAction(this);
      //btn.addEventHandler(ActionEvent.ACTION, this);
      Group root = new Group(btn);
      root.getChildren().add(text);
      Scene scene = new Scene(root);
      stage.setScene(scene);
      stage.show();
    }
    public void handle(ActionEvent t) {
      updateText(++counter);
    }
    public void updateText(int n){
      text.setText("Hai cliccato "
                   +n+" volte");
    }
    public static void main(
         String[] args) {
      launch(args);
    }
} // end of AppWithEvents
```

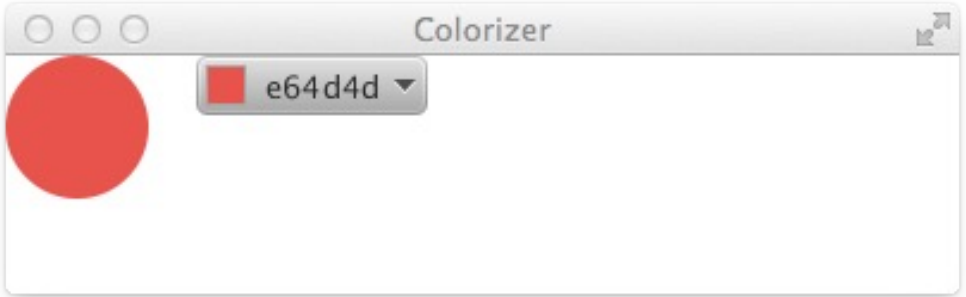
Convenience methods

- Consentono di scrivere codice più conciso
- Sono definiti per gli eventi più comuni…
- … per la maggior parte nella classe **`Node`**…
- … ma non solo: ad esempio, in **`Button`**
  `setOnAction(EventHandler<ActionEvent> value)`
- … tipicamente usati insieme a listener anonimi, poiché ne riducono la verbosità
- Lista completa:
  `https://docs.oracle.com/javafx/2/events/convenience_methods.htm`

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

parte del template nell'IDE…

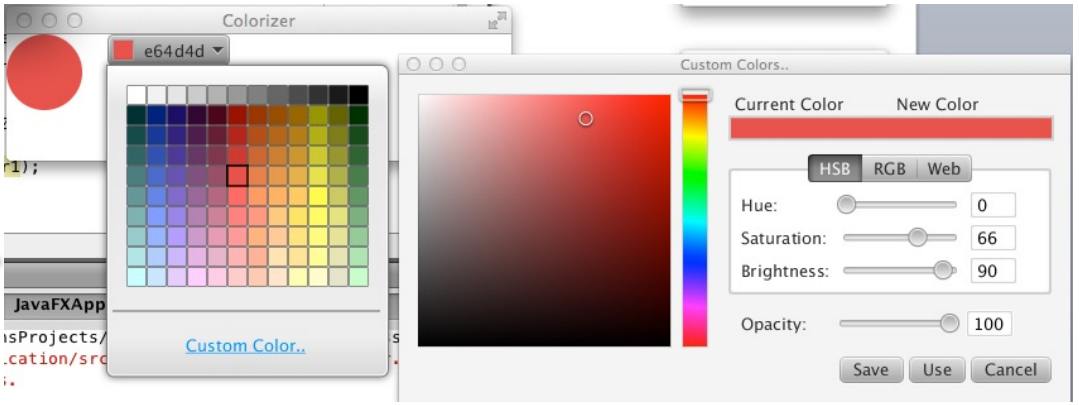
Esempio



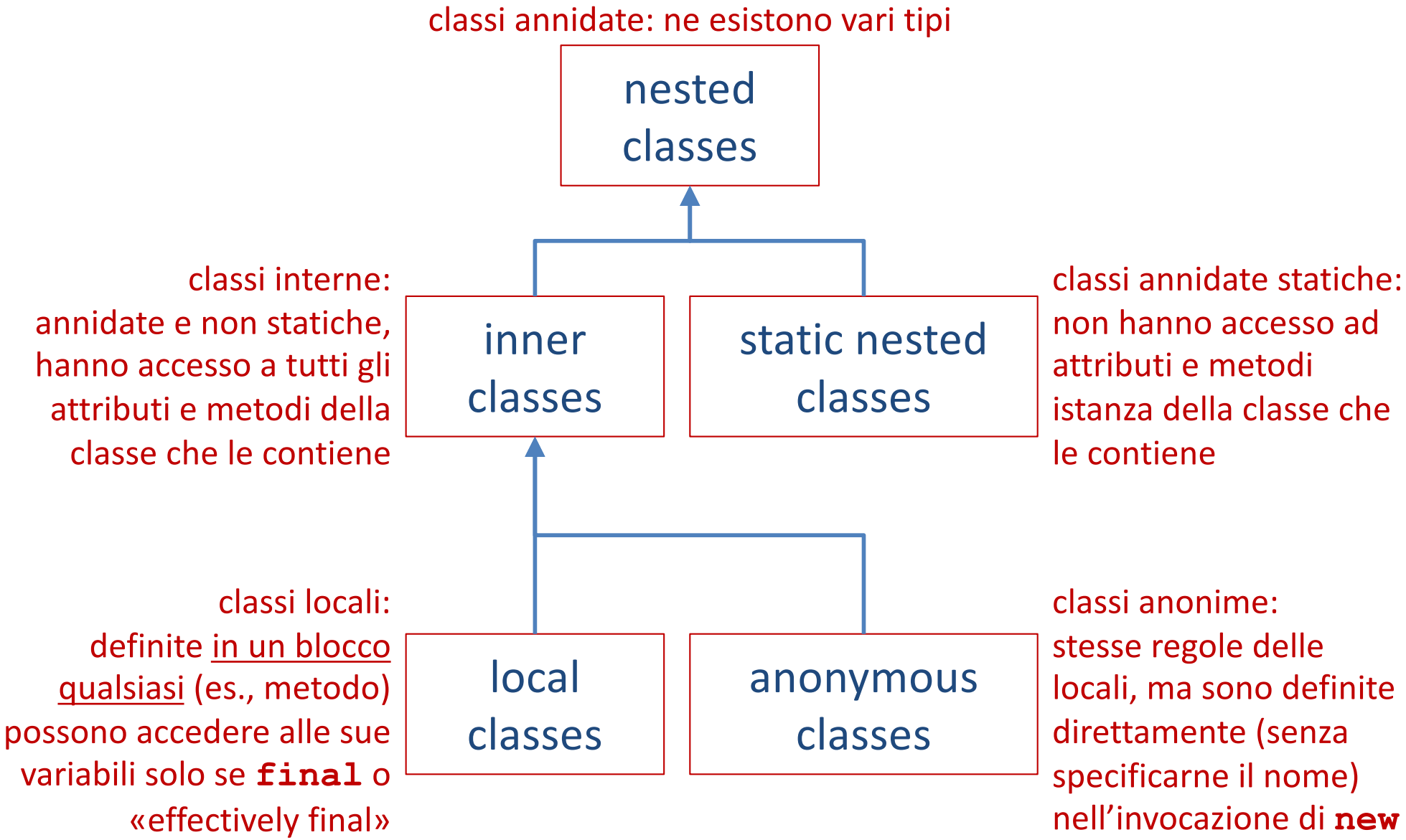

```
public class Colorizer extends Application {
  public void start(final Stage stage) {
    final Circle circ = new Circle(40, 40, 30);
    final ColorPicker colorPicker1 = new ColorPicker(Color.BLACK);
    colorPicker1.setOnAction(new EventHandler(){
    // colorPicker1.addEventHandler(ActionEvent.ACTION, new EventHandler() {
      @Override
      public void handle(Event t) {
        System.out.println(t.getEventType());
        circ.setFill(colorPicker1.getValue());
    }});
    Scene scene = new Scene(new HBox(20), 400, 100);
    HBox box = (HBox) scene.getRoot();
    box.getChildren().addAll(circ,colorPicker1);
    stage.setScene(scene);
    stage.show();
  }
}
```

È un layout (vedi dopo). Viceversa, con Group gli elementi vengono posizionati tutti in (0,0)

Le azioni possono essere molto complesse!



Digressione: classi annidate



Esempio

```
public class Outer {
  int attr = 1;
  Inner i = new Inner();
  public String m() {
    attr++;
    int local = 5;
    Object o = new Object() {
      public String toString() {
✓       attr++;
✗       local = 6;
✓       return Integer.toString(local);
      }
    };
    return o.toString();
  }
  public String m2(){
    String s = null;
    if (true) {
      int local = 1;
      class Local {
        public String toString() {
✓         attr++;
✗         return Integer.toString(local);
        }
      }
      local++;
      s = new Local().toString();
    }
    return s;
  }
  class Inner {
    int y = attr + 1;
    void m(int z) {
✓     attr += z;
✓     System.out.println("In inner class: x=" + attr + ", y=" + y);
    }
  }
}
```

una inner class (anonima o meno) non può **modificare** variabili locali

senza l'istruzione precedente è ok perché `local` è «effectively final»

```
public static void main(String[] args) {
    Outer c = new Outer();
    c.i.m(5);
    System.out.println("in m()=" + c.m());
    System.out.println("in m2()=" + c.m2());
}
```

una inner class (anonima o meno) può **leggere** variabili locali solo se final o «effectively final»

togliendo questa istruzione, `local` diventa «effectively final»

ogni tipo di nested class può leggere/modificare attributi di quella esterna

Classi annidate … a cosa servono?

- Come abbiamo già visto, servono a tenere il codice «vicino» a dove viene usato
- Inoltre, aumentano incapsulamento e information hiding: le classi annidate possono essere rese invisibili all'esterno
  - A differenza di quelle «normali», possono essere dichiarate **`protected`** o **`private`**
- In definitiva, servono a rendere il codice più leggibile, comprensibile, e quindi a migliorarne la scrittura e manutenzione

Scope delle variabili in classi interne anonime

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        Button b=new Button("click me");
        f(b);
        Group root=new Group(b);
        Scene scene = new Scene(root, 320, 240);
        stage.setTitle("Test!");
        stage.setScene(scene);
        stage.show();
    }
    private void f(Button b) {
        int counter=0;
        b.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                    counter++;
            }
        });
     }
```

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

Scope delle variabili in classi interne anonime

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        Button b=new Button("click me");
        f(b);
        Group root=new Group(b);
        Scene scene = new Scene(root, 320, 240);
        stage.setTitle("Test!");
        stage.setScene(scene);
        stage.show();
    }
    private void f(Button b) {
        int counter=0;
        b.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                    counter++;
            }
        });
     }
```

Errore in compilazione!

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

Scope delle variabili in classi interne anonime

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage stage) throws IOException {
        Button b=new Button("click me");
        f(b);
        Group root=new Group(b);
        Scene scene = new Scene(root, 320, 240);
        stage.setTitle("Test!");
        stage.setScene(scene);
        stage.show();
    }
    private void f(Button b) {
        int counter=0;
        b.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                println (counter);
            }
        });
     }
```

Questo é ok! ("effectively final")

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

Scope delle variabili in classi interne anonime

```
public class HelloApplication extends Application {
    int counter=0;
    @Override
    public void start(Stage stage) throws IOException {
        Button b=new Button("click me");
        f(b);
        Group root=new Group(b);
        Scene scene = new Scene(root, 320, 240);
        stage.setTitle("Test!");
        stage.setScene(scene);
        stage.show();
    }
    private void f(Button b) {
            b.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent actionEvent) {
                println (++counter);
            }
        });
     }
```

Questo é ok! (instance variable)

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

Gestire la pressione di tasti

```
Button b = new Button("PLUS");
EventHandler<KeyEvent> keyEventHandler = new
                                        EventHandler<KeyEvent>() {
    @Override
    public void handle(KeyEvent e) {
        if (e. getCharacter().equals("+")) {
            System.out.println("Buttom + pressed");
        }
    }
};
b.addEventHandler(KeyEvent.KEY_TYPED,keyEventHandler);
```

Gestire la pressione di tasti

```
public void handle(KeyEvent e) {
   ...
if (e.getCharacter().equals("u")) ...
if (e.getCode() == KeyCode.U) ...
```

Il carattere frutto della pressione di una combinazione di tasti (inclusi shift, alt, control…)

Il codice ottenuto da un singolo tasto (inclusi tutti I tasti speciali: frecce, control ecc.

Use getCharacter with KEYTYPED and getCode with KEYPRESSED and KEYRELEASED

Gestire la pressione di tasti

"*"Key pressed" and "key released" events* are lower-level and depend on the platform and keyboard layout. They are generated whenever a key is pressed or released, and are the only way to find out about keys that don't generate character input (e.g., action keys (Fn), modifier keys, etc.). The key being pressed or released is indicated by the code variable, which contains a virtual key code."

Una app con un bottone…

```
public class Keyboard1 extends Application {
  int counter=0;
  public void start(Stage stage) {
    TilePane box = new TilePane();
    box.setHgap(50);
    Button b1 = new Button("Uno");
    box.getChildren().add(b1);
    EventHandler<ActionEvent> actionHandler =
      new EventHandler<ActionEvent>(){
      public void handle(ActionEvent t) {
        System.out.println((counter++) +
                           ((Button)(t.getTarget())).getText());
    }};
    b1.addEventHandler(ActionEvent.ACTION, actionHandler);
    Scene scene = new Scene(box, 400, 300);
    stage.setTitle("My JavaFX Application");
    stage.setScene(scene); stage.show();
  }
  public static void main(String[] args){ launch(args);}
}
```


```
0Uno
1Uno
2Uno
3Uno
```

… che si può premere anche via tastiera

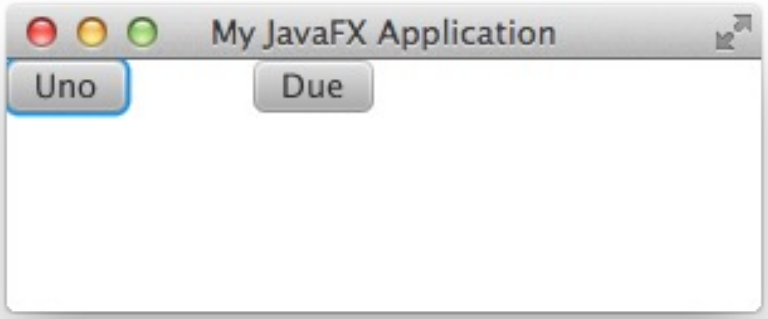
```
// dentro start()…
EventHandler<KeyEvent> keyEventHandler =
  new EventHandler<KeyEvent>(){
  public void handle(KeyEvent keyEvent) {
    if (keyEvent.getCode() == KeyCode.U) {
      b1.fireEvent(new ActionEvent());
      System.out.println(keyEvent.getSource() +
                         " =>"+keyEvent.getTarget());
    }
  }
};
b1.addEventHandler(KeyEvent.KEY_PRESSED,keyEventHandler);
```

siamo noi a generare un **ActionEvent**!

```
Button@4e0cf854[styleClass=button]'Uno'=>
Button@4e0cf854[styleClass=button]'Uno'
```

Un app con due bottoni...

```
public class Keyboard1 extends Application {
  int counter=0;
  public void start(Stage stage) {
    TilePane box = new TilePane();
    box.setHgap(50);
    Button b1 = new Button("Uno");
    Button b2 = new Button("Due");
    box.getChildren().addAll(b1,b2);
    EventHandler<ActionEvent> actionHandler =
      new EventHandler<ActionEvent>(){
      public void handle(ActionEvent t) {
        System.out.println((counter++) +
                           ((Button)(t.getTarget())).getText());
    }};
    b1.addEventHandler(ActionEvent.ACTION, actionHandler);
    b2.addEventHandler(ActionEvent.ACTION, actionHandler);
    …
```



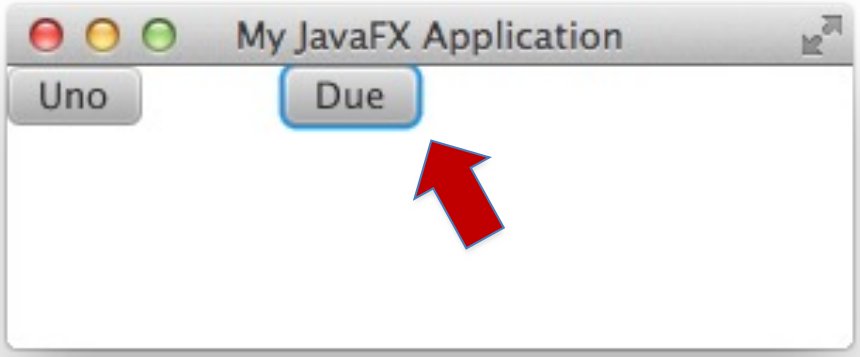
Riuso lo stesso listener!

```
0Uno
1Uno
2Due
3Uno
4Due
5Due
```

La selezione da tastiera funziona?

SI!

```
Button@4e0cf854[styleClass=button]'Uno'=>
Button@4e0cf854[styleClass=button]'Uno'
```

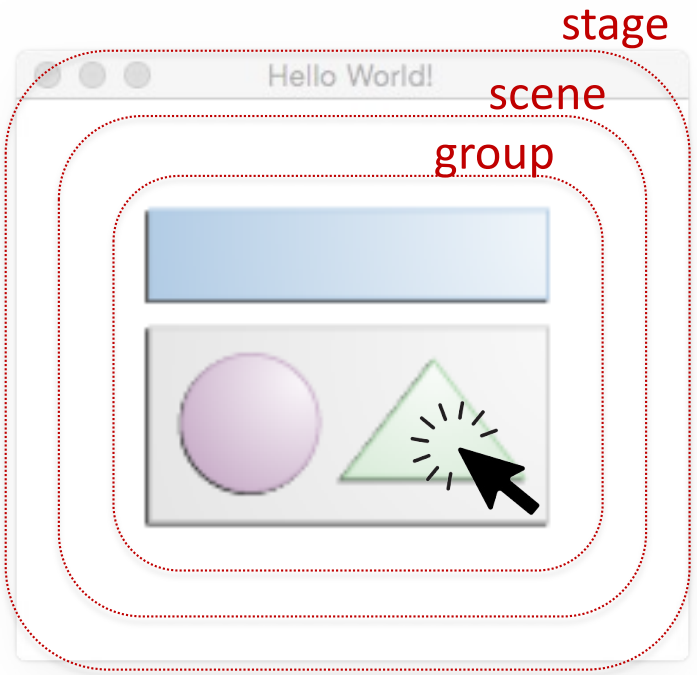


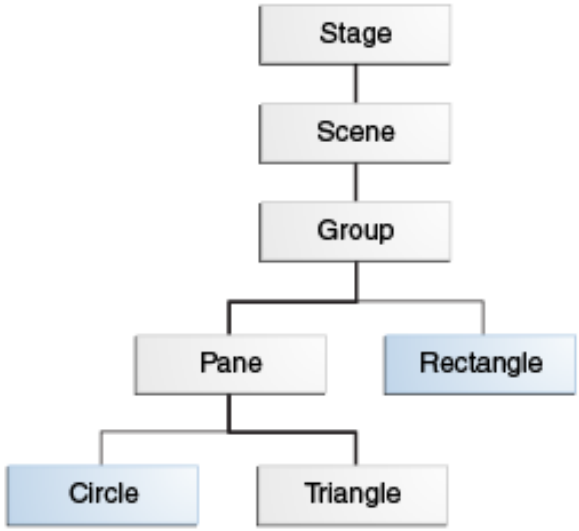
NO!

Perché?!?

Generazione e propagazione degli eventi

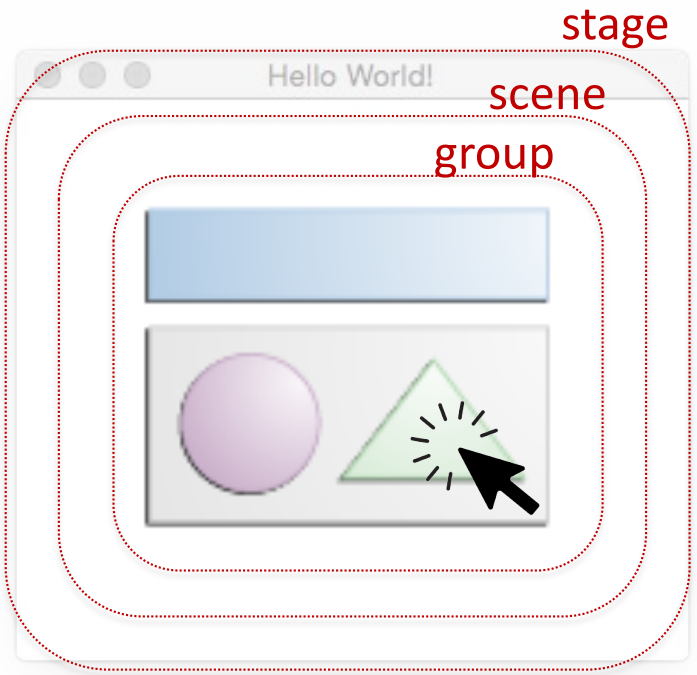
- Primo problema: un evento può essere generato in un'area di interesse per più di un oggetto… chi lo riceve?
- Regole per assegnare il «target»:
  - Key event: il nodo che ha il ***focus***
  - Mouse event: il nodo nella posizione del mouse. Se ce n'è più di uno, viene scelto quello «in cima»
  - Sono definite regole per altri tipi di eventi (touch screen)

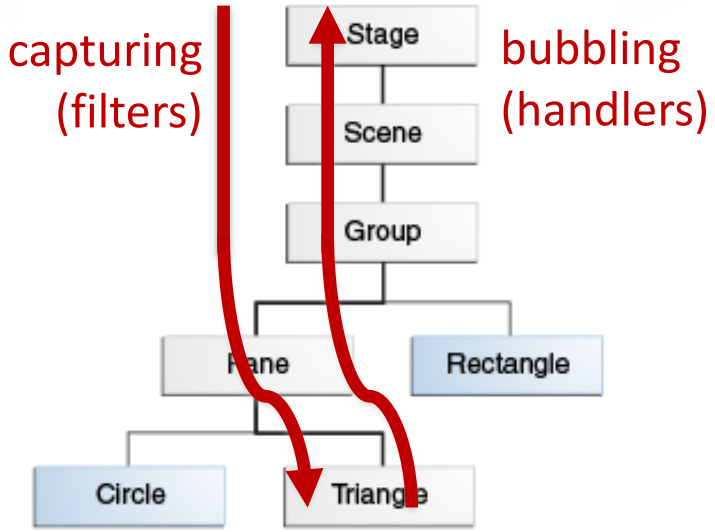




Generazione e propagazione degli eventi

- Secondo problema: a volte può essere utile far gestire un evento al contenitore e non al contenuto…
- Regola base: tutti gli eventi partono dallo stage, arrivano al target, e tornano allo stage
  - ***event capturing***: stage ➔ target
    - eventi intercettati mediante ***filter***
  - ***event bubbling***: target ➔ stage
    - eventi intercettati mediante ***handler***
- La sequenza di componenti stage ↔ target si chiama ***event dispatch chain***





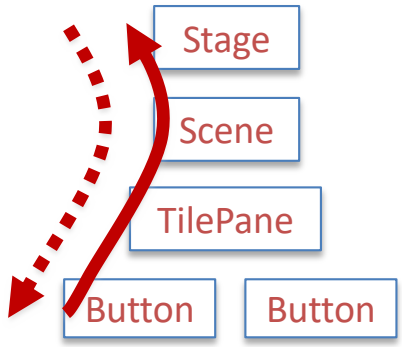
Vediamo se è vero…

```
EventHandler handler = new EventHandler<ActionEvent>() {
  public void handle(ActionEvent t) {
    EventTarget target = t.getTarget();
    Object source = t.getSource();
    String id=null;
    if (source instanceof Node) {
      id = ((Node) source).getId();
    } else if (source instanceof Stage) {
      id="STAGE";
    } else if (source instanceof Scene) {
      id="SCENE";
    } else
      System.out.println("Unrecognized object: "+source);
    System.out.println("HANDLER: "+id+" "+source+" =>"+target);
  }
};
```

interfaccia implementata da tutte le classi JavaFX che possono essere target di un evento

Il ***target*** è il componente dove l'evento si verifica (secondo le regole viste)

Vediamo se è vero…




```
box.setId("TILEPANE");
b1.setId("BUTTON1");
b2.setId("BUTTON2");
stage.addEventHandler(ActionEvent.ACTION, handler);
scene.addEventHandler(ActionEvent.ACTION, handler);
box.addEventHandler(ActionEvent.ACTION, handler);
b1.addEventHandler(ActionEvent.ACTION, handler);
b2.addEventHandler(ActionEvent.ACTION, handler);
```

```
HANDLER: BUTTON1 Button[id=BUTTON1, styleClass=button]'Uno' =>Button[id=BUTTON1, styleClass=button]'Uno'
HANDLER: TILEPANE TilePane[id=TILEPANE, styleClass=root] =>Button[id=BUTTON1, styleClass=button]'Uno'
HANDLER: SCENE javafx.scene.Scene@40410aad =>Button[id=BUTTON1, styleClass=button]'Uno'
HANDLER: STAGE javafx.stage.Stage@4a2e6207 =>Button[id=BUTTON1, styleClass=button]'Uno'
HANDLER: BUTTON2 Button[id=BUTTON2, styleClass=button]'Due' =>Button[id=BUTTON2, styleClass=button]'Due'
HANDLER: TILEPANE TilePane[id=TILEPANE, styleClass=root] =>Button[id=BUTTON2, styleClass=button]'Due'
HANDLER: SCENE javafx.scene.Scene@40410aad =>Button[id=BUTTON2, styleClass=button]'Due'
HANDLER: STAGE javafx.stage.Stage@4a2e6207 =>Button[id=BUTTON2, styleClass=button]'Due'
```

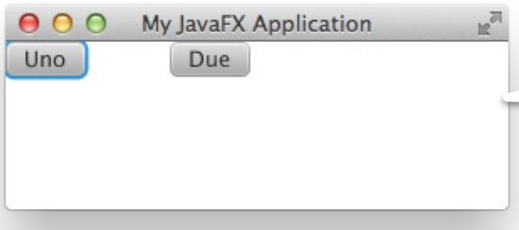
la sorgente dell'evento cambia a ogni passo!!!

il target rimane identico: è il punto «vero» in cui si è generato l'evento)

Come risolve il nostro problema?

```
// dentro start()…
EventHandler<KeyEvent> keyEventHandler =
  new EventHandler<KeyEvent>(){
  public void handle(KeyEvent keyEvent) {
    if (keyEvent.getCode() == KeyCode.U) {
      b1.fireEvent(new ActionEvent());
      System.out.println(keyEvent.getSource() +
                         " =>"+keyEvent.getTarget());
    }
  }
};
//b1.addEventHandler(KeyEvent.KEY_PRESSED,keyEventHandler);
stage.addEventHandler(KeyEvent.KEY_PRESSED,keyEventHandler);
```

E l'altro bottone?!?



gestisco la pressione del tasto nel contenitore (stage) anziché nel contenuto (bottoni)

```
javafx.stage.Stage@63e71ca8 =>
Button@4e0cf854[styleClass=button]'Uno'
```

```
javafx.stage.Stage@63e71ca8 =>
Button@73f19cb2[styleClass=button]'Due'
```

Gestire ambedue i bottoni…

```
// dentro start()…
EventHandler<KeyEvent> keyEventHandler =
  new EventHandler<KeyEvent>(){
  public void handle(KeyEvent keyEvent) {
     System.out.println(keyEvent.getSource() +
                          " =>"+keyEvent.getTarget());
     switch (keyEvent.getCode()){
       case U:
       case DIGIT1:
         b1.fireEvent(new ActionEvent());
         break;
       case D:
       case DIGIT2:
         b2.fireEvent(new ActionEvent());
         break;
  }
};
stage.addEventHandler(KeyEvent.KEY_PRESSED,keyEventHandler);
```

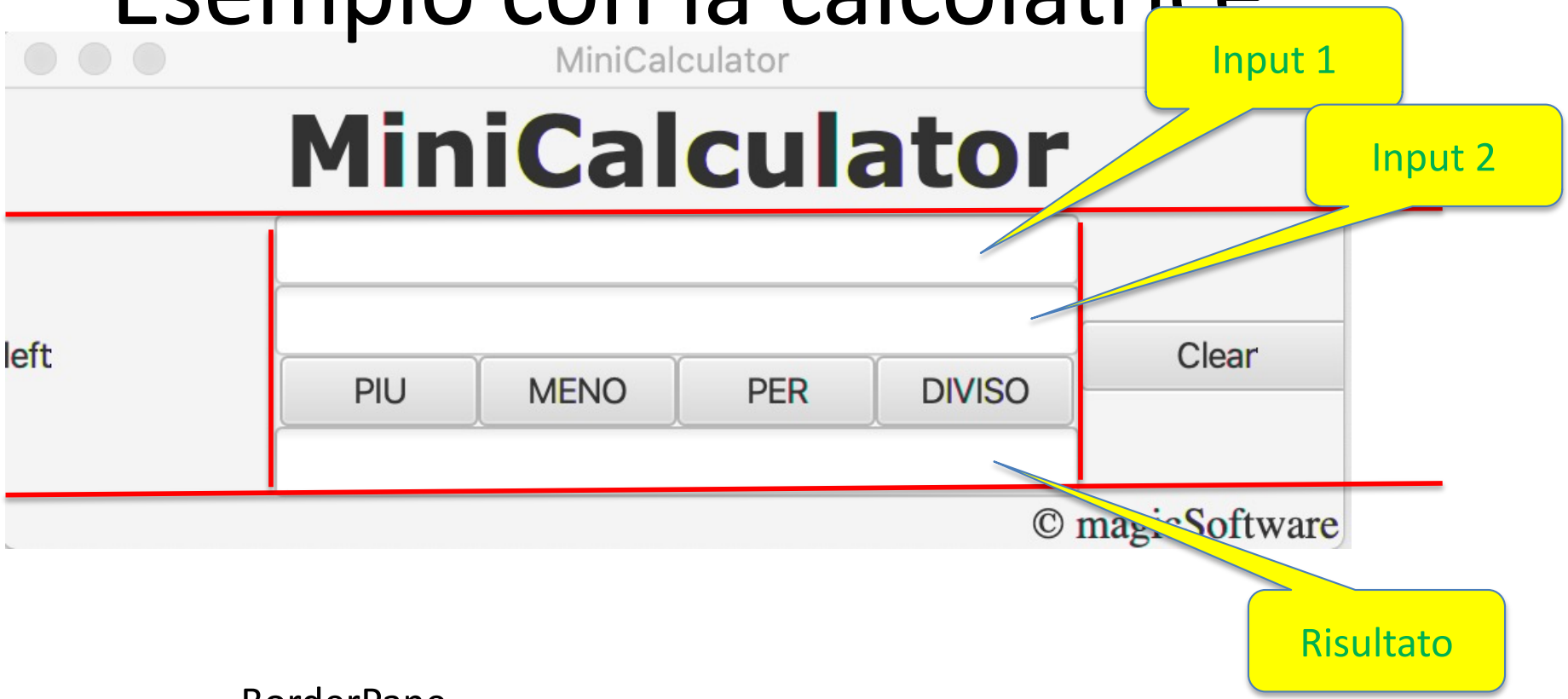
… spostando il focus

```
// dentro start()…
EventHandler<KeyEvent> keyEventHandler =
  new EventHandler<KeyEvent>(){
  public void handle(KeyEvent keyEvent) {
     System.out.println(keyEvent.getSource() +
                          " =>"+keyEvent.getTarget());
     switch (keyEvent.getCode()){
       case U:
       case DIGIT1:
         b1.fireEvent(new ActionEvent()); b1.requestFocus();
         break;
       case D:
       case DIGIT2:
         b2.fireEvent(new ActionEvent()); b2.requestFocus();
         break;
  }
};
stage.addEventHandler(KeyEvent.KEY_PRESSED,keyEventHandler);
```

Per chi vuole saperne di più…

http://docs.oracle.com/javase/8/javase-clienttechnologies.htm

Esempio con la calcolatrice



BorderPane,
al centro un TilePane di una colonna,
in terza riga un TilePane di quattro colonne

Bottone customizzato

```
class OperationButton extends Button implements
    EventHandler<ActionEvent> {

        MiniCalculator2 mc = null;

        public OperationButton(MiniCalculator2 mc, String
               label, String id) {
            super(label);
            this.mc = mc;
            setId(id);
            addEventFilter(ActionEvent.ACTION, this);
        }
        void setOBwidth(double w) {
            this.setMaxWidth(w);
            this.setMinWidth(w);
        }
        @Override
        public void handle(ActionEvent t) {
            mc.compute(this.getId());
        }
    }
```

TextField customizzato

```
class NonEditableTextField extends TextField {
        NonEditableTextField(String s) {
            super(s);
            this.setEditable(false);
        }
    }
```

Application

```
public class MiniCalculator2 extends Application {
    final TextField input1 = new TextField("");
    final TextField input2 = new TextField("");
    final NonEditableTextField output = new NonEditableTextField("");

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("MiniCalculator");
        BorderPane borderP = new BorderPane();
        // ======== Top
        Label lt = new Label("MiniCalculator");
        lt.setFont(Font.font("Verdana", FontWeight.BOLD, 36));
        borderP.setTop(lt);
        BorderPane.setAlignment(lt, Pos.CENTER);
        // ======== Right
        Button clear = new Button("Clear");
        clear.setMinWidth(100.0);
        borderP.setRight(clear);
        BorderPane.setAlignment(clear, Pos.CENTER);
        clear.setOnAction(new EventHandler<ActionEvent>(){
            @Override
            public void handle(ActionEvent event) {
                input1.clear();
                input2.clear();
                output.clear();
            }
    });
```

Application

```
    // ======== Left
    Label lableft = new Label("left");
    lableft.setMinWidth(100.0);
    borderP.setLeft(lableft);
    BorderPane.setAlignment(lableft, Pos.CENTER_LEFT);
    // ======== Bottom
    Label lb = new Label("© magicSoftware ");
    lb.setFont(Font.font("Times", FontPosture.ITALIC, 16));
    borderP.setBottom(lb);
    BorderPane.setAlignment(lb, Pos.BOTTOM_RIGHT);
    // ======== Center
    final TilePane box = new TilePane();
    box.setPrefColumns(1);
    final TilePane hb = new TilePane();
    hb.setAlignment(Pos.CENTER);
    final OperationButton sum = new OperationButton(this,"PIU", "+");
    final OperationButton divide = new OperationButton(this,"DIVISO",
"/");
    final OperationButton multiply = new OperationButton(this,"PER",
"*");
    final OperationButton subtract = new OperationButton(this,"MENO",
"-");
    // ----
    hb.getChildren().addAll(sum, subtract, multiply, divide);
    box.getChildren().addAll(input1, input2, hb, output);
```

Application

```
    // ======== Behaviour
    borderP.setCenter(box);
    Scene scene = new Scene(borderP);
    scene.addEventFilter(KeyEvent.KEY_TYPED, new KBFilter(this));
    primaryStage.setScene(scene);
    primaryStage.sizeToScene();
    primaryStage.widthProperty().addListener(new
                                        ChangeListener<Number>() {
        @Override
        public void changed(ObservableValue<? extends Number> ov,
                    Number oldValue, Number newValue) {
            double w = newValue.doubleValue() * 3 / 5;
            box.setMaxWidth(w);
            box.setMinWidth(w);
            hb.setMaxWidth(w);
            hb.setMinWidth(w);
            double iw = Math.floor(w/4);
            sum.setOBwidth(iw);
            subtract.setOBwidth(iw);
            divide.setOBwidth(iw);
            multiply.setOBwidth(iw);
        }
    });
    primaryStage.show();
}
```

Gestione della tastiera:
La vediamo dopo.

Solo per I più curiosi e temerari: questa sezione(righe rosse) effettua un resizing dei TilePane e del loro Contenuto quando la finestra viene ridimensionata

Application

```
public void compute(String operator) {
    double o1, o2;
    try {
        o1 = Double.parseDouble(input1.getText());
        o2 = Double.parseDouble(input2.getText());
    } catch (NumberFormatException e) {
        Label msg = new Label("Errore - Not A Number!");
        StackPane g = new StackPane();
        g.getChildren().add(msg);
        Scene stageScene = new Scene(g, 300, 200);
        Stage errorStage = new Stage();
        errorStage.setScene(stageScene);
        errorStage.show();
        return;
    }
    switch (operator) {
        case "+":
            output.setText("" + (o1 + o2)); break;
        case "*":
            output.setText("" + (o1 * o2)); break;
        case "-":
            output.setText("" + (o1 - o2)); break;
        case "/":
            output.setText("" + (o1 / o2)); break;
    }
}
public static void main(String[] args) {Application.launch(args);}
```

Application

```
public class KBFilter implements EventHandler<KeyEvent> {
        MiniCalculator2 mc = null;
        KBFilter(MiniCalculator2 mc) {
            this.mc = mc;
        }
        @Override
        public void handle(KeyEvent e) {
            String t = e.getCharacter();
            if ("1234567890".contains(t)) {
               return;
            } else if (t.equals(".")) {
                if (e.getTarget() instanceof TextField) {
                    TextField tf = (TextField) (e.getTarget());
                    System.out.println(tf.getText());
                    if (tf.getText().contains(".")) {
                        e.consume();
                    }
                    return;
                }
            } else if ("+-/*".contains(t)) {
                mc.compute(t);
            }
            e.consume();
            return;
        }
    }
```

Gestione della tastiera:
Filtriamo gli eventi a livello di Scene

- Lasciamo arrivare al TextField
  numeri e **punto**,
- Interpretiamo I tasti operazione,
- Buttiamo tutto il resto

Per i curiosi…

- Quanto segue sono slide aggiuntive che mostrano come in JavaFX sia possibile:
  - definire filter in maniera analoga agli handler
  - “consumare” eventi lungo il loro percorso
- Negli esami non useremo queste caratteristiche

Vediamo se è vero…

```
EventHandler filter = new EventHandler<ActionEvent>() {
  public void handle(ActionEvent t) {
    EventTarget target = t.getTarget();
    Object source = t.getSource();
    String id=null;
    if (source instanceof Node {
      id = ((Node) source).getId();
    } else if (source instanceof Stage) {
      id="STAGE";
    } else if (source instanceof Scene) {
      id="SCENE";
    } else
      System.out.println("Unrecognized object: "+source);
    System.out.println("FILTER: "+id+" "+source+" =>"+target);
  }
};
```

Filter e handler sono definiti con le ***stesse*** modalità; ambedue devono implementare **`EventHandler`**

Cambia solo il modo con cui sono associati ai nodi

Vediamo se è vero…

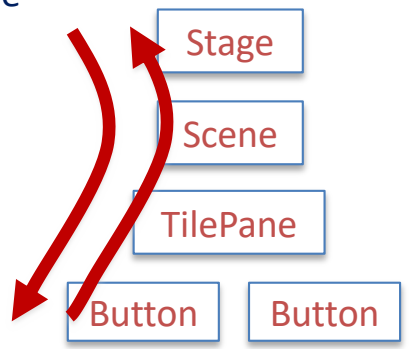
```
box.setId("TILEPANE");
b1.setId("BUTTON1");
b2.setId("BUTTON2");
stage.addEventFilter(ActionEvent.ACTION, filter);
stage.addEventHandler(ActionEvent.ACTION, handler);
scene.addEventFilter(ActionEvent.ACTION, filter);
scene.addEventHandler(ActionEvent.ACTION, handler);
box.addEventFilter(ActionEvent.ACTION, filter);
box.addEventHandler(ActionEvent.ACTION, handler);
b1.addEventFilter(ActionEvent.ACTION, filter);
b1.addEventHandler(ActionEvent.ACTION, handler);
b2.addEventFilter(ActionEvent.ACTION, filter);
b2.addEventHandler(ActionEvent.ACTION, handler);
```

Vediamo se è vero…

OUTPUT

```
FILTER: STAGE javafx.stage.Stage@4a2e6207 =>Button[id=BUTTON1, styleClass=button]'Uno'
FILTER: SCENE javafx.scene.Scene@40410aad =>Button[id=BUTTON1, styleClass=button]'Uno'
FILTER: TILEPANE TilePane[id=TILEPANE, styleClass=root] =>Button[id=BUTTON1, styleClass=button]'Uno'
FILTER: BUTTON1 Button[id=BUTTON1, styleClass=button]'Uno' =>Button[id=BUTTON1, styleClass=button]'Uno'
HANDLER: BUTTON1 Button[id=BUTTON1, styleClass=button]'Uno' =>Button[id=BUTTON1, styleClass=button]'Uno'
HANDLER: TILEPANE TilePane[id=TILEPANE, styleClass=root] =>Button[id=BUTTON1, styleClass=button]'Uno'
HANDLER: SCENE javafx.scene.Scene@40410aad =>Button[id=BUTTON1, styleClass=button]'Uno'
HANDLER: STAGE javafx.stage.Stage@4a2e6207 =>Button[id=BUTTON1, styleClass=button]'Uno'
FILTER: STAGE javafx.stage.Stage@4a2e6207 =>Button[id=BUTTON2, styleClass=button]'Due'
FILTER: SCENE javafx.scene.Scene@40410aad =>Button[id=BUTTON2, styleClass=button]'Due'
FILTER: TILEPANE TilePane[id=TILEPANE, styleClass=root] =>Button[id=BUTTON2, styleClass=button]'Due'
FILTER: BUTTON2 Button[id=BUTTON2, styleClass=button]'Due' =>Button[id=BUTTON2, styleClass=button]'Due'
HANDLER: BUTTON2 Button[id=BUTTON2, styleClass=button]'Due' =>Button[id=BUTTON2, styleClass=button]'Due'
HANDLER: TILEPANE TilePane[id=TILEPANE, styleClass=root] =>Button[id=BUTTON2, styleClass=button]'Due'
HANDLER: SCENE javafx.scene.Scene@40410aad =>Button[id=BUTTON2, styleClass=button]'Due'
HANDLER: STAGE javafx.stage.Stage@4a2e6207 =>Button[id=BUTTON2, styleClass=button]'Due'
```

La sorgente dell'evento cambia a ogni passo!!!
(il target rimane identico: è il punto «vero» in cui si è generato l'evento)



«Consumare» eventi

```
class SuperHandler implements EventHandler<ActionEvent>{
  protected EventTarget target;
  protected Object source;
  protected String id;
  @Override
  public void handle(ActionEvent t) {
  target = t.getTarget();
  source = t.getSource();
  id = null;
  if (source instanceof Node {
    id = ((Node) source).getId();
  } else if (source instanceof Stage) {
    id="STAGE";
  } else if (source instanceof Scene) {
    id="SCENE";
  } else
    System.out.println("Unrecognized object: "+source);
}
```

Stesso codice di prima, ma ora possiamo specializzarlo

oppure

```
id = source.getClass().getSimpleName().toUpperCase();
```

«Consumare» eventi

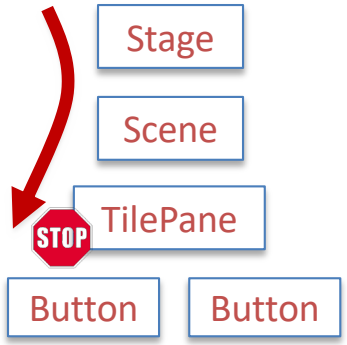
```
SuperHandler filter = new SuperHandler () {
  public void handle(ActionEvent t) {
    super.handle(t);
    System.out.println("FILTER:"+id+" "+source+" ==> "+target);
  }
};
SuperHandler handler = new SuperHandler() {
  public void handle(ActionEvent t) {
    super.handle(t);
    System.out.println("HANDLER:"+id+" "+source+" ==> "+target);
  }
};
SuperHandler cutter = new SuperHandler() {
  public void handle(ActionEvent t) {
    super.handle(t);
    System.out.println("CUTTER:"+id+" "+source+" ==> "+target);
    t.consume();
  }
};
```

Dichiarano una sottoclasse anonima di **SuperHandler**

Interrompe la propagazione dell'evento

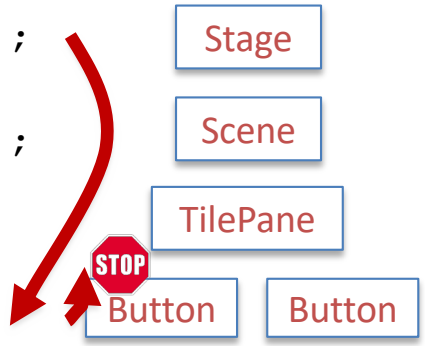
Vediamo se è vero…

```
stage.addEventFilter(ActionEvent.ACTION, filter);
stage.addEventHandler(ActionEvent.ACTION, handler);
scene.addEventFilter(ActionEvent.ACTION, filter);
scene.addEventHandler(ActionEvent.ACTION, handler);
box.addEventFilter(ActionEvent.ACTION, cutter);
box.addEventHandler(ActionEvent.ACTION, handler);
b1.addEventFilter(ActionEvent.ACTION, cutter);
b1.addEventHandler(ActionEvent.ACTION, handler);
```




```
FILTER:STAGE javafx.stage.Stage@6418ebbb ==> Button@3b019254[styleClass=button]'Uno'
FILTER:SCENE javafx.scene.Scene@640f1a9d ==> Button@3b019254[styleClass=button]'Uno'
CUTTER:TILEPANE TilePane@69638f42[styleClass=root] ==> Button@3b019254[styleClass=button]'Uno'
```

```
stage.addEventFilter(ActionEvent.ACTION, filter);
stage.addEventHandler(ActionEvent.ACTION, handler);
scene.addEventFilter(ActionEvent.ACTION, filter);
scene.addEventHandler(ActionEvent.ACTION, handler);
box.addEventFilter(ActionEvent.ACTION, filter);
box.addEventHandler(ActionEvent.ACTION, cutter);
b1.addEventFilter(ActionEvent.ACTION, filter);
b1.addEventHandler(ActionEvent.ACTION, cutter);
```




```
FILTER:STAGE javafx.stage.Stage@45d7a782 ==> Button@327e1a10[styleClass=button]'Uno'
FILTER:SCENE javafx.scene.Scene@15be4106 ==> Button@327e1a10[styleClass=button]'Uno'
FILTER:TILEPANE TilePane@7818d4fc[styleClass=root] ==> Button@327e1a10[styleClass=button]'Uno'
FILTER:BUTTON Button@327e1a10[styleClass=button]'Uno' ==> Button@327e1a10[styleClass=button]'Uno'
CUTTER:BUTTON Button@327e1a10[styleClass=button]'Uno' ==> Button@327e1a10[styleClass=button]'Uno'
```