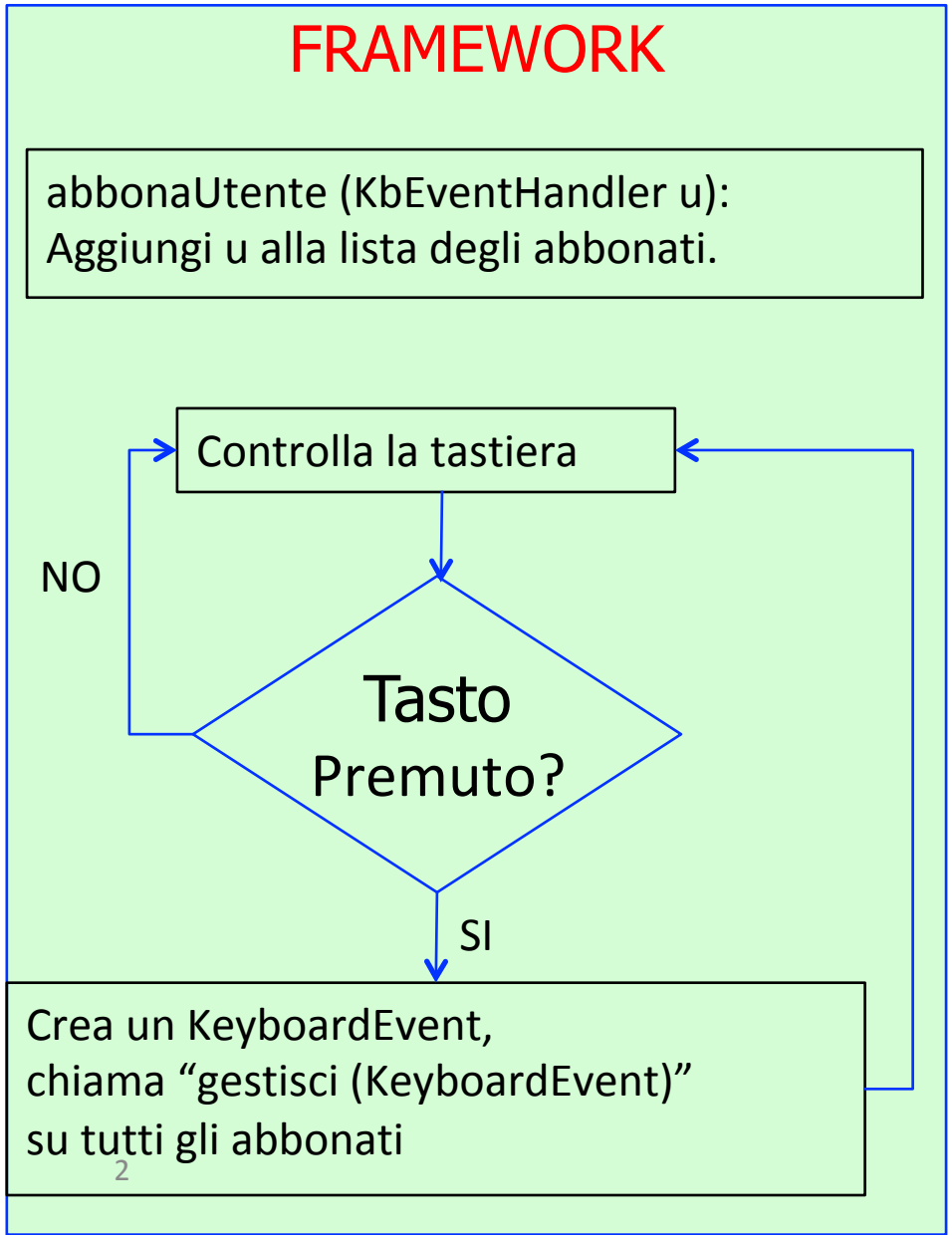
Eventi in Java FX

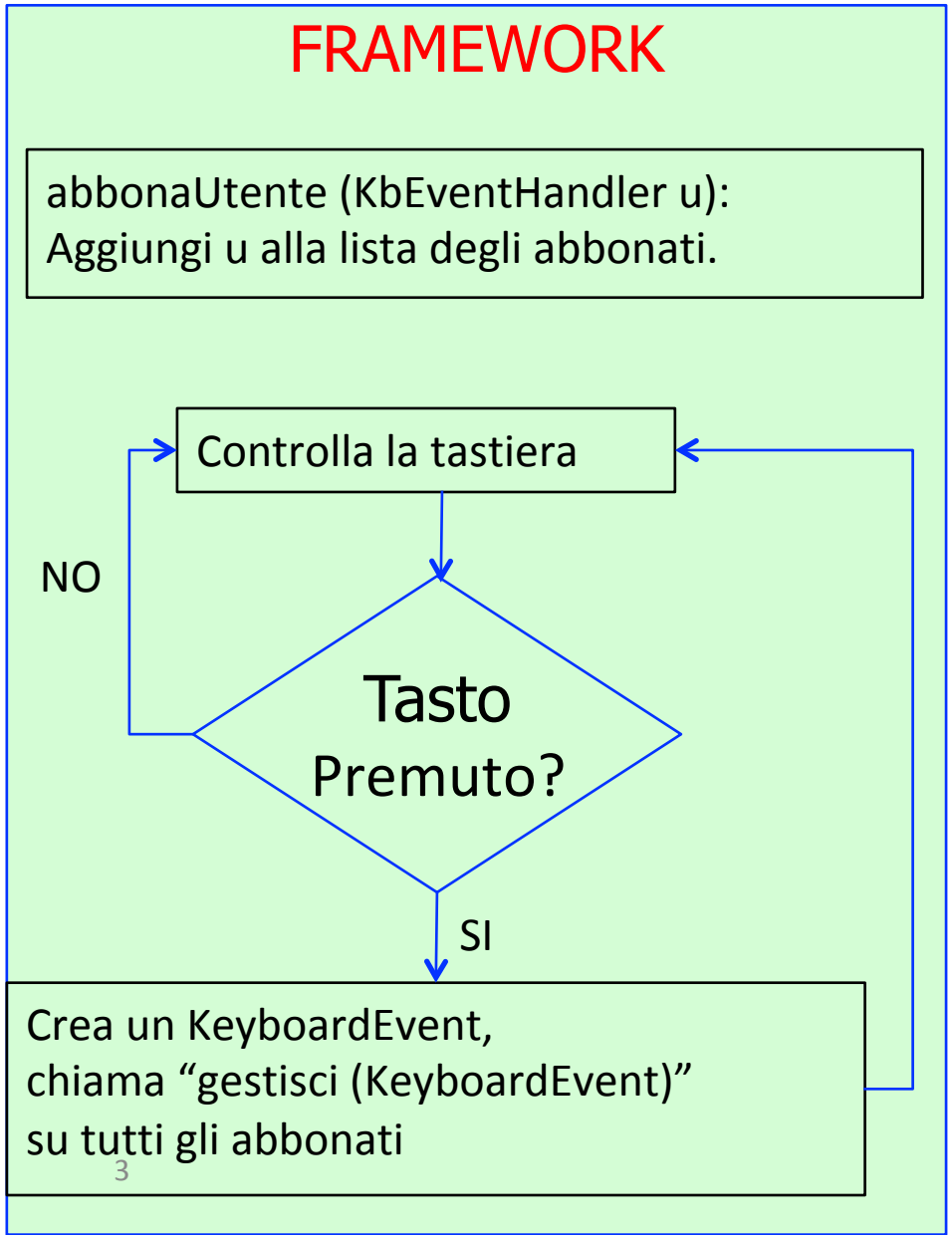
Architettura di un framework




```
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)
}
```

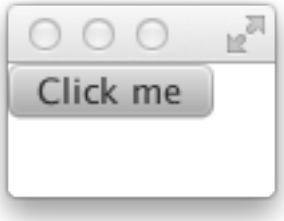
```
class MyProgram extends Framework
  implements  KbEventHandler  {
    MyProgram (){
        startFramework();
        Controller c=new Controller();
        abbonaUtente(c);
    }

}
```

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

Alcuni eventi base

```
public class Event0 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()); } }
```

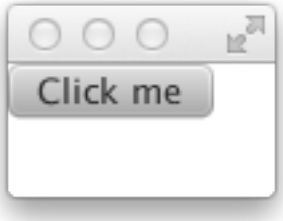
```
1 Ricevuto un evento di tipo
                INPUT_METHOD_TEXT_CHANGED
2 Ricevuto un evento di tipo MOUSE_ENTERED
3 Ricevuto un evento di tipo
                MOUSE_ENTERED_TARGET
4 Ricevuto un evento di tipo MOUSE_MOVED
…
12 Ricevuto un evento di tipo MOUSE_MOVED
13 Ricevuto un evento di tipo MOUSE_PRESSED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



Alcuni eventi base

1 Ricevuto un evento di tipo ACTION

```
public class Event0 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()); } }
```



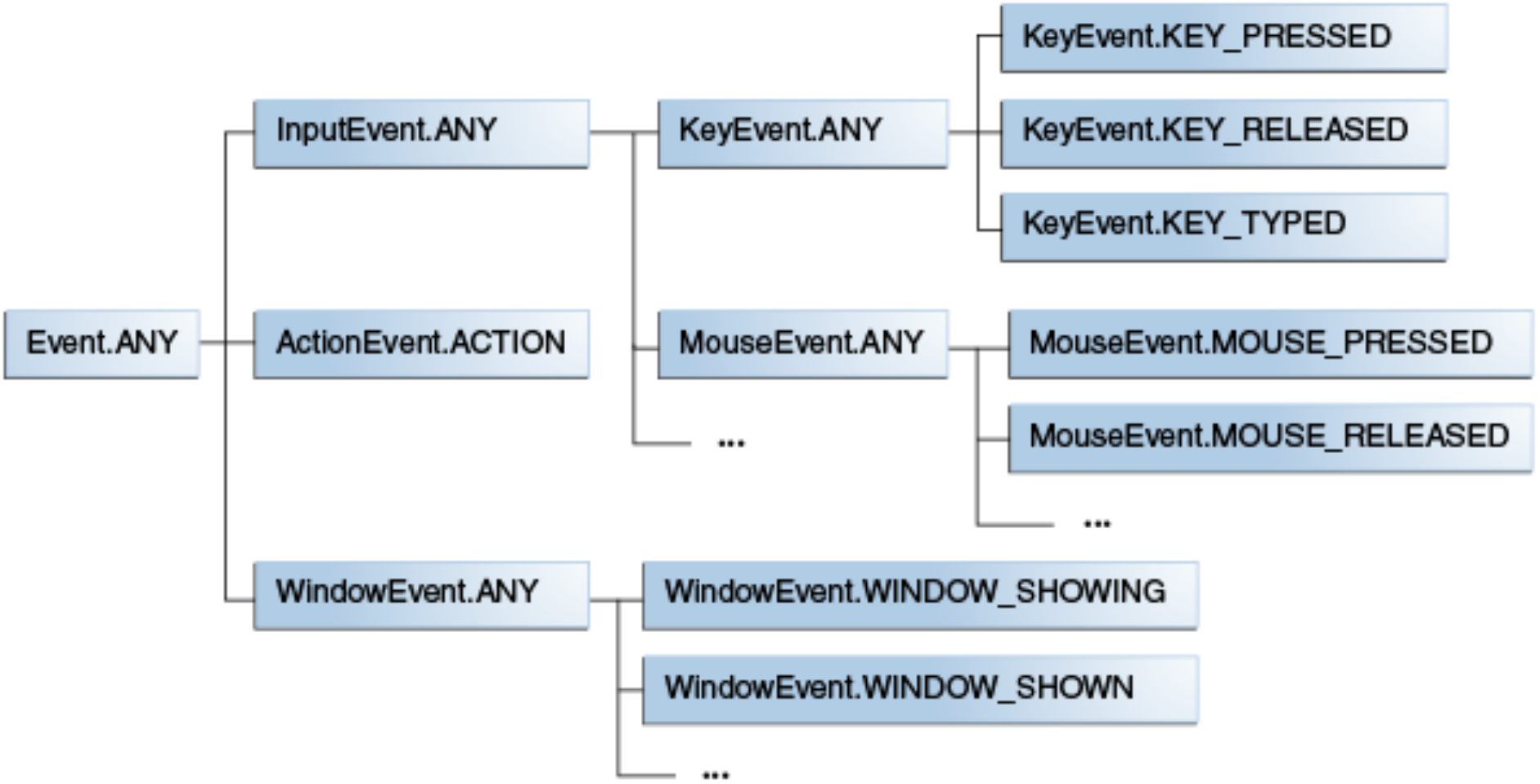
Gestire gli eventi

- 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

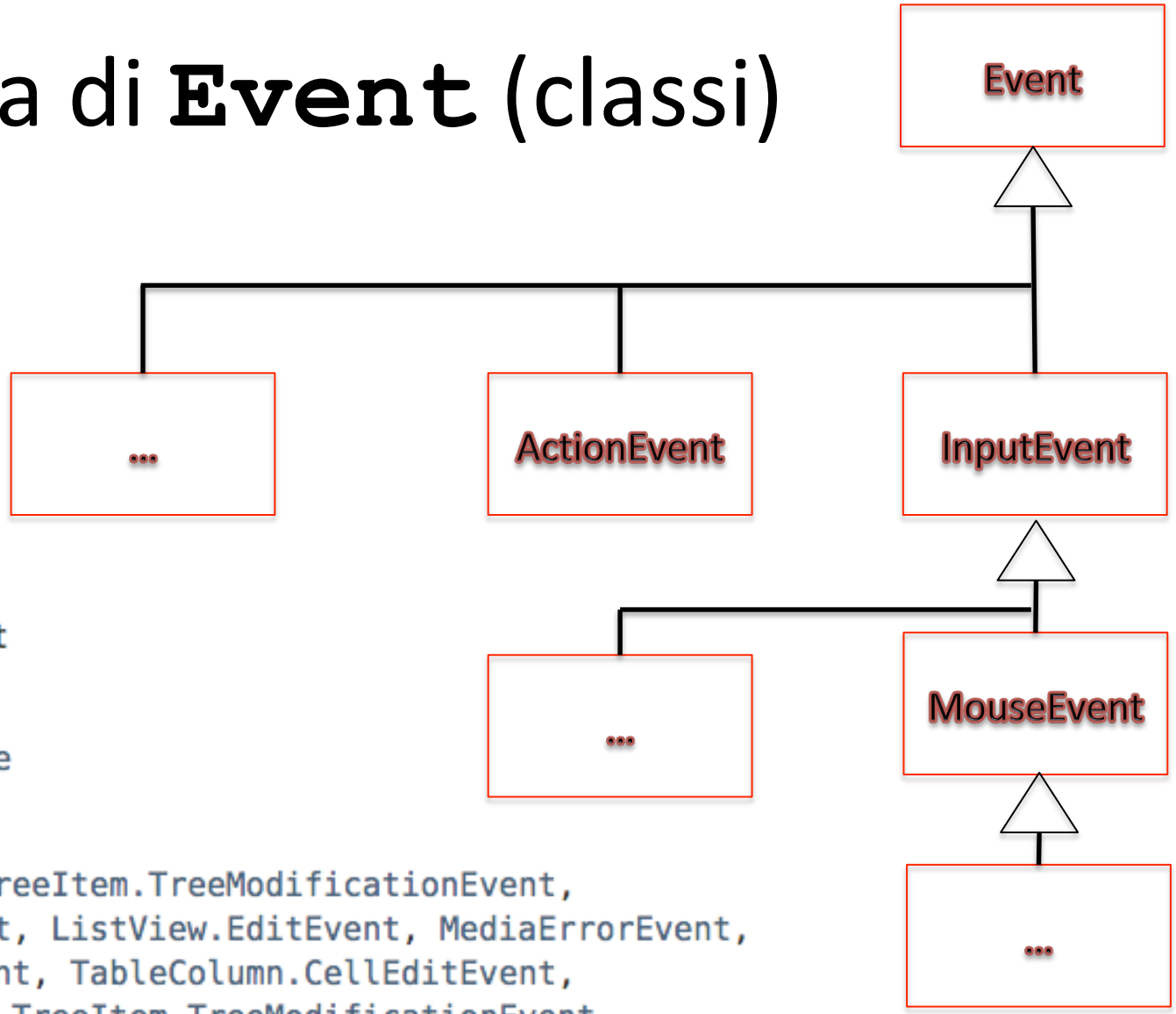
Aggiunta – rimozione di un ascoltatore

```
EventHandler l=new Listener()

btn.addEventHandler(
    ActionEvent.ACTION,listener);

btn.removeEventHandler(
    ActionEvent.ACTION,listener);
```

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



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



javafx.event

Class Event

java.lang.Object
    java.util.EventObject
        javafx.event.Event

All Implemented Interfaces:

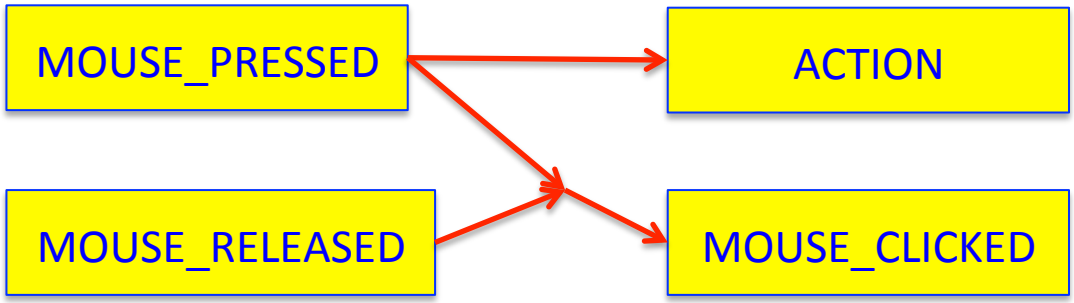
`Serializable, Cloneable`

Direct Known Subclasses:

`ActionEvent, CheckBoxTreeItem.TreeModificationEvent, DialogEvent, InputEvent, ListView.EditEvent, MediaErrorEvent, ScrollToEvent, SortEvent, TableColumn.CellEditEvent, TransformChangedEvent, TreeItem.TreeModificationEvent, TreeTableColumn.CellEditEvent, TreeTableView.EditEvent, TreeView.EditEvent, WebErrorEvent, WebEvent, WindowEvent, WorkerStateEvent`

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
```



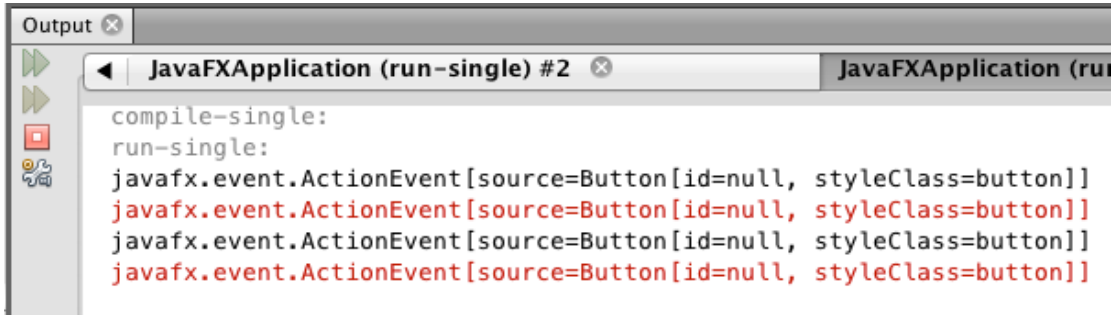
(pressed and released on the same node)

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 Event0 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);
    }
}
```



Intermezzo: classi interne

```
public class ClasseEsterna {
  public static void main(String a[]) {
    class Punto {
        int x,y;
        Punto(int x,int y) {
            this.x=x;
            this.y=y;
        }
        double dist(Punto p) {
            return Math.sqrt((x-p.x)^2+(y-p.y)^2);
        }
    }
    Punto p1=new Punto(2,3);
    Punto p2=new Punto(4,6);
    System.out.println("Distanza tra p1 e p2 = "+p1.dist(p2));
  }
}
```

Classe visibile solo nel main

```
public class ClasseEsterna {
  class Punto {
        int x,y;
        Punto(int x,int y) {
            this.x=x;
            this.y=y;
        }
        double dist(Punto p) {
            return Math.sqrt((x-p.x)^2+(y-p.y)^2);
        }
  }
  public static void main(String a[]) {
    Punto p1=new Punto(2,3);
    Punto p2=new Punto(4,6);
    System.out.println("Distanza tra p1 e p2 = "+p1.dist(p2));
  }
}
```

Classe visibile solo dentro ClasseEsterna

Interazione tra l’originatore e il gestore degli eventi

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);
    }
```

```
(Compiler) warning:
uses unchecked or unsafe operations
```

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

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();
    }
    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);
    }
```

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

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

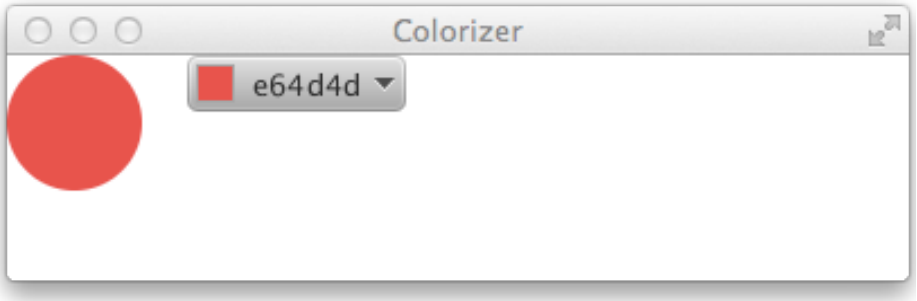
Convenience methods


```
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`

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 .
Viceversa, con Group gli elementi vengono posizionati tutti in (0,0)