Java FX: il modello degli eventi

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) {
   …
```

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

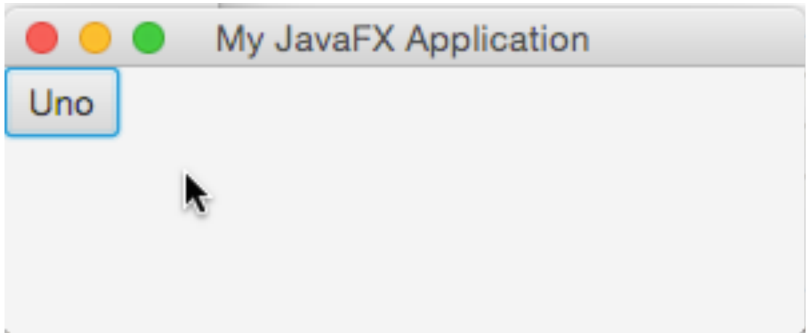
```
if (e.getCharacter().equals("u")) …
if (e.getCode() == KeyCode.U) …
```

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

Use getCharacter with KEYTYPED and getCode with KEYPRESSED

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){Application.launch(args);}
}
```



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

… che si può premere anche via tastiera

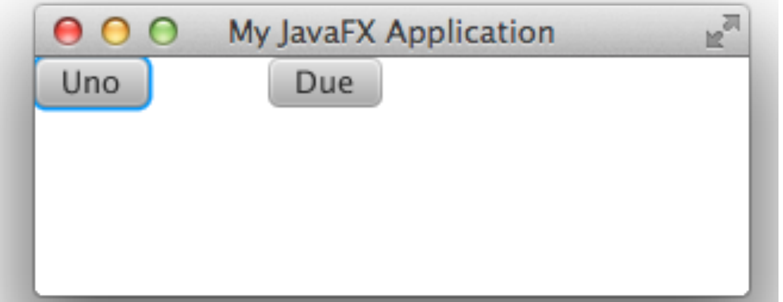
```
// dentro start()…
EventHandler<KeyEvent> keyEventHandler =
  new EventHandler<KeyEvent>(){
  public void handle(KeyEvent keyEvent) {
    if (keyEvent.getCharacter().equals("u”)) {
      b1.fireEvent(new ActionEvent());
      System.out.println(keyEvent.getSource() +
                                 " =>"+keyEvent.getTarget());
    }
  }
};
b1.addEventHandler(KeyEvent.KEY_TYPED,keyEventHandler);
```

siamo noi a generare un `ActionEvent`!

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

Un app con due bottoni…

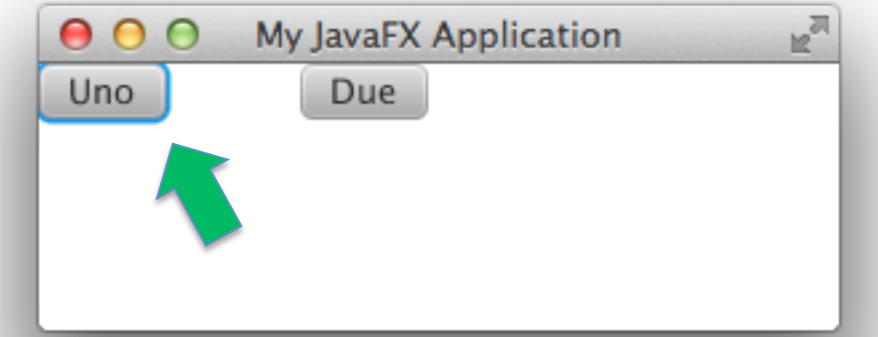
```
public class Keyboard1 extends Application {
  int counter=0;
  public void start(Stage stage) {
    TilePane box = new TilePane();
    box.setHgap(50);
    Button b1 = new Button("Uno");
    Button b2 = new Button("Due");
    box.getChildren().addAll(b1,b2);
    EventHandler<ActionEvent> actionHandler =
      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);
    Scene scene = new Scene(box, 400, 300);
    stage.setTitle("My JavaFX Application");
    stage.setScene(scene); stage.show();
  }
…}
```



Riuso lo stesso listener!

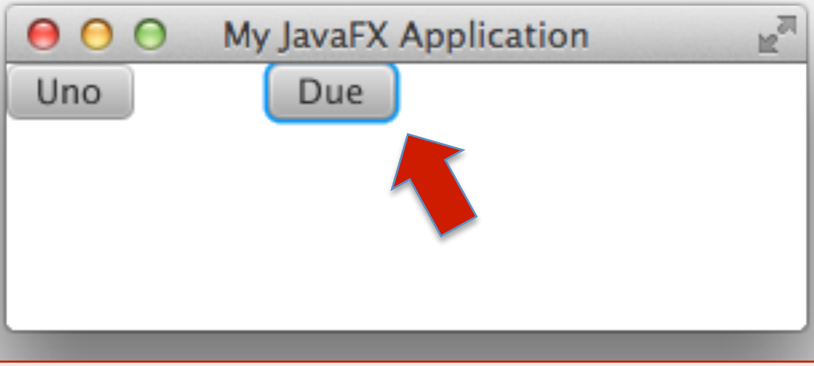
```
0Uno
1Uno
2Due
3Uno
4Due
5Due
```

La selezione da tastiera funziona?



SI!

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

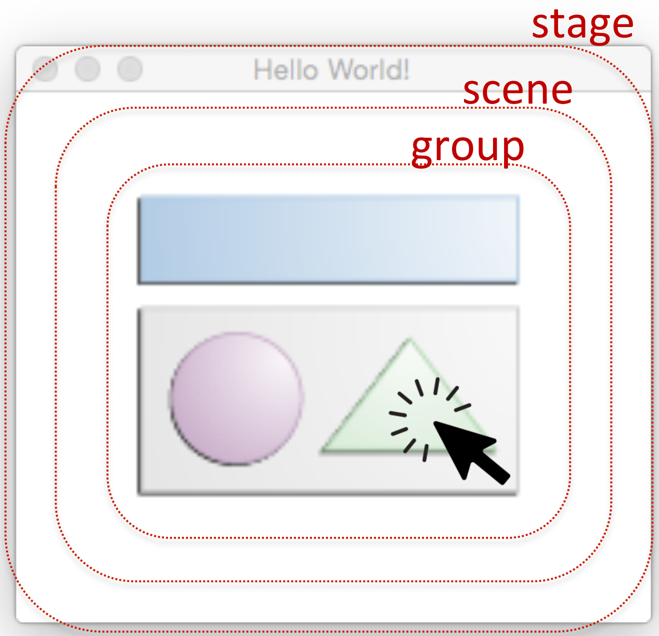


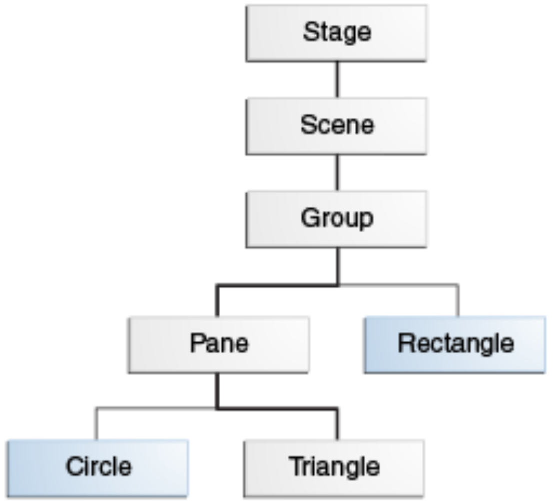
NO!

Perché?!?

Generazione e propagazione degli eventi

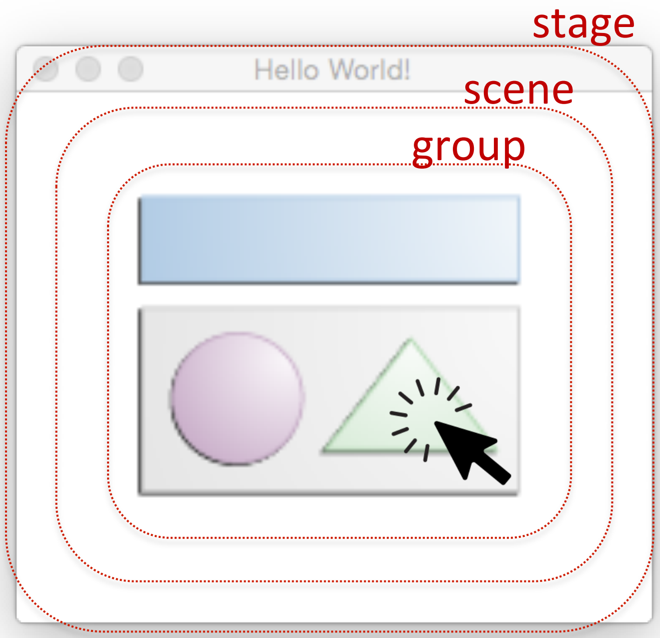
- Primo problema: un evento può essere generato in un’area di interesse per più di un oggetto… chi lo riceve?
- Regole per assegnare il «target»:
  - Key events: il nodo che ha il ***focus***
  - Mouse events: il nodo nella posizione del mouse. Se ce n’è più di uno, viene scelto quello «in suoerficie»,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

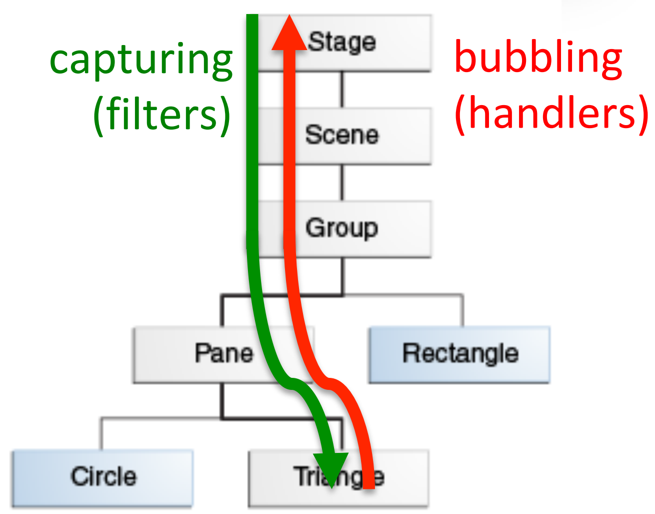




Generazione e propagazione degli eventi

- 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);
  }
};
```

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);
```

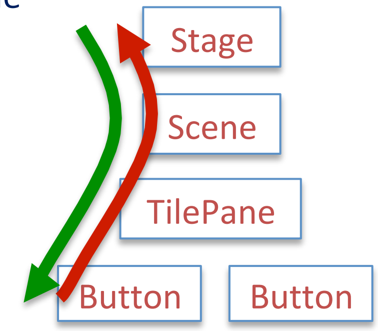
Vediamo se è vero...

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!!! (mentre il target rimane identico)

È possibile interrompere la catena?



Stage

Scene

TilePane

Button

Button

«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'
```

Come risolve il nostro problema?

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

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

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

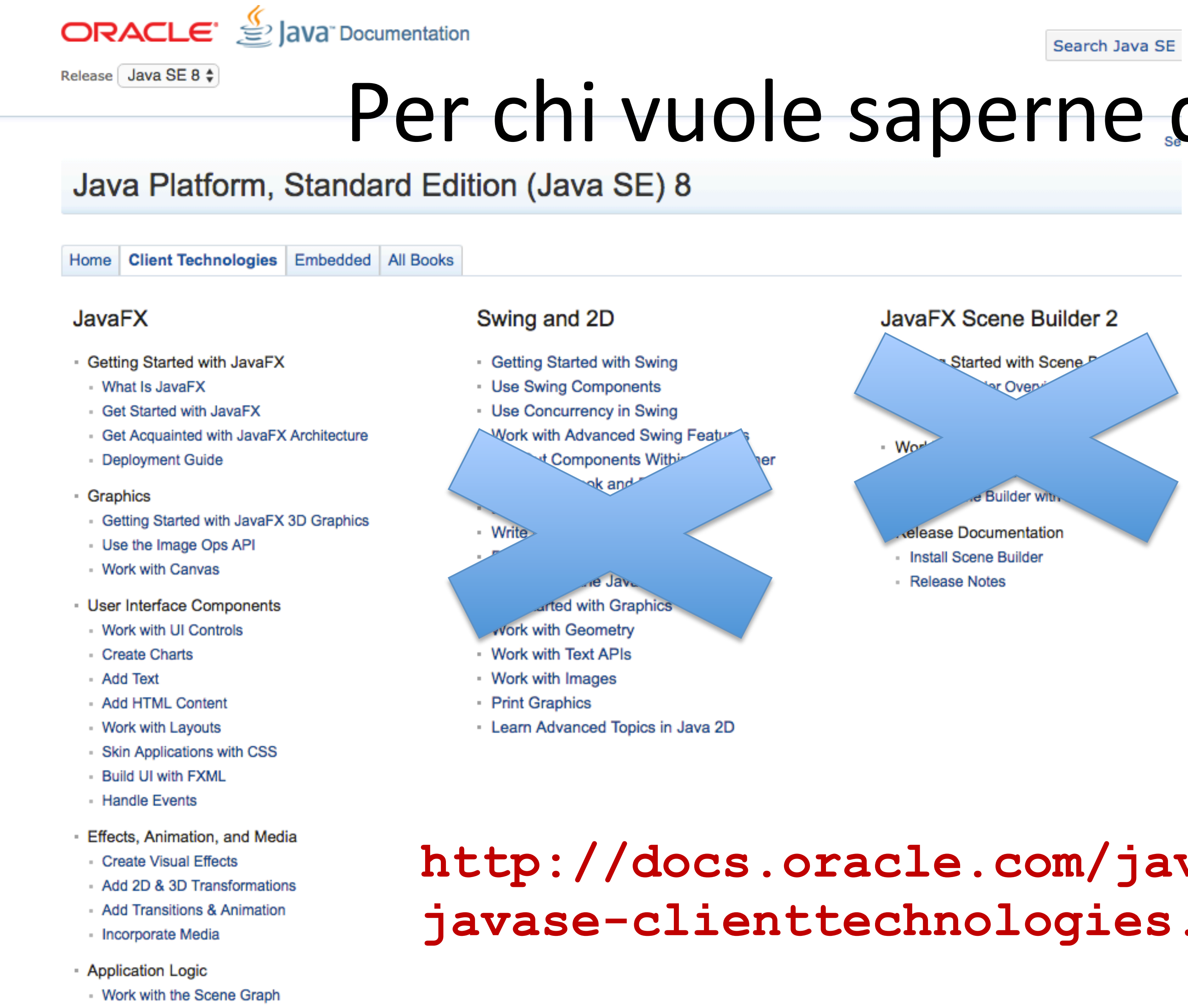
Gestire ambedue i bottoni...

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

… spostando anche il focus

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

Per chi vuole saperne di più…



Java Platform, Standard Edition (Java SE) 8

JavaFX

- Getting Started with JavaFX
  - What Is JavaFX
  - Get Started with JavaFX
  - Get Acquainted with JavaFX Architecture
  - Deployment Guide
- Graphics
  - Getting Started with JavaFX 3D Graphics
  - Use the Image Ops API
  - Work with Canvas
- User Interface Components
  - Work with UI Controls
  - Create Charts
  - Add Text
  - Add HTML Content
  - Work with Layouts
  - Skin Applications with CSS
  - Build UI with FXML
  - Handle Events
- Effects, Animation, and Media
  - Create Visual Effects
  - Add 2D & 3D Transformations
  - Add Transitions & Animation
  - Incorporate Media
- Application Logic
  - Work with the Scene Graph

Swing and 2D

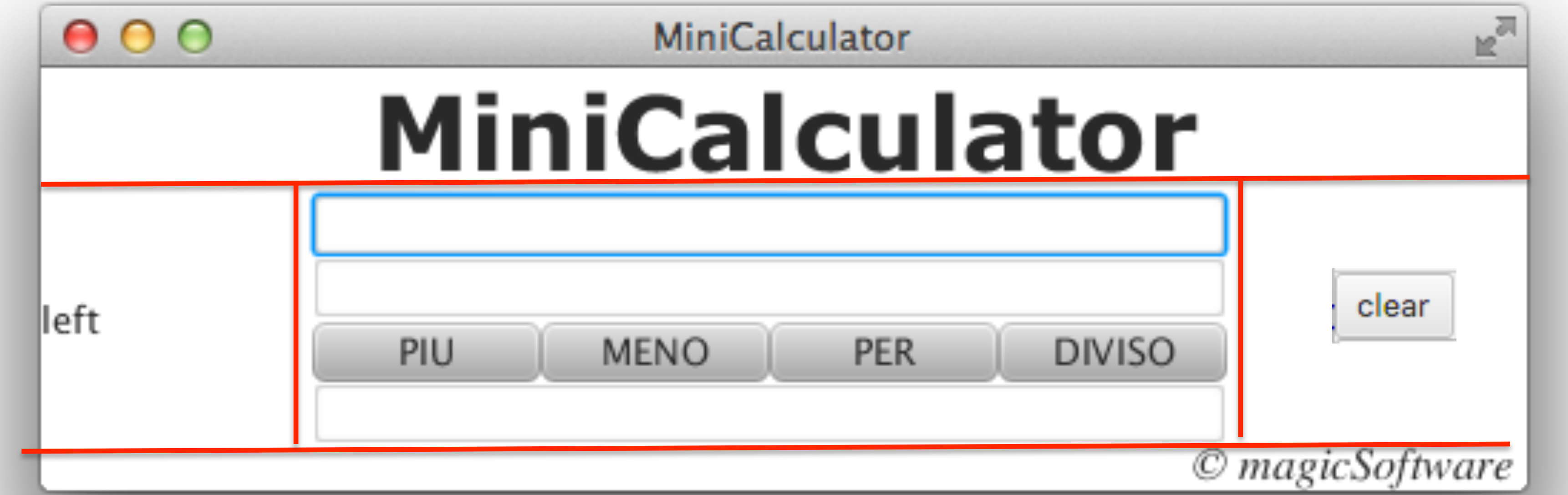
- Getting Started with Swing
- Use Swing Components
- Use Concurrency in Swing
- ~~Work with Advanced Swing Featu s~~
- ~~t Components Withi er~~
- ~~ok and~~
- ~~Write~~
- ~~e Jav~~
- ~~rted with Graphics~~
- ~~Work with Geometry~~
- Work with Text APIs
- Work with Images
- Print Graphics
- Learn Advanced Topics in Java 2D

JavaFX Scene Builder 2

- ~~Started with Scene~~
  - ~~r Overv~~
- ~~Wor~~
  - ~~e Builder with~~
- ~~elease Documentation~~
  - Install Scene Builder
  - Release Notes

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

Soluzione Esercizio 4



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