

GUI E INTRODUZIONE A Java FX - parte II

Angelo Di Iorio

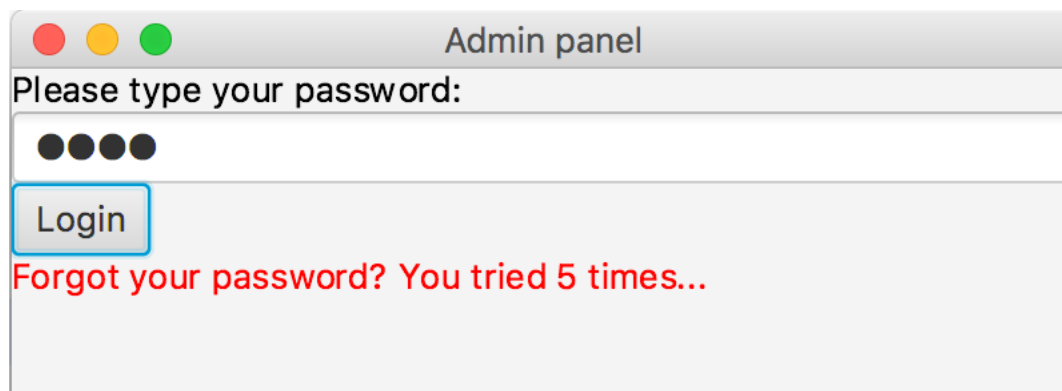
Università di Bologna

Java FX e stato applicazione

- Negli esempi visti finora abbiamo scritto il codice di semplici *handler* che verificano i dati inseriti dagli utenti e mostrano messaggi sull'interfaccia
- Molto spesso è utile mantenere informazioni sullo stato dell'applicazione a run-time
- E' opportuno quindi definire *variabili istanza* che mantengono questi dati e li rendono disponibili all'applicazione
- Se si vogliono mantenere informazioni anche dopo la chiusura dell'applicazione sarà necessario memorizzare i dati in modo persistente

Esempio

- Realizziamo un'applicazione che chiede all'utente una password, verifica che sia corretta e mostra un messaggio:
 - Se corretta: *"Login successful"*
 - Se sbagliata: numero di tentativi dal lancio dell'applicazione



The image shows a web application window with a title bar containing three colored buttons (red, yellow, green) and the text 'Admin panel'. Below the title bar, the text 'Please type your password:' is displayed. Underneath is a password input field containing four black dots. A 'Login' button is positioned below the input field. At the bottom of the window, a red error message reads: 'Forgot your password? You tried 5 times...'.

- Per semplicità: memorizziamo la password corretta in una variabile istanza (e in chiaro)

```
// import omessi
public class AdminDemo extends Application {

    @Override
    public void start(Stage stage) throws Exception {

        VBox root = new VBox();
        Text t = new Text("Please type your password:");

        final PasswordField pf = new PasswordField();
        Button b = new Button("Login");
        final Text loginMsg = new Text();

        // EventHandler<ActionEvent> bh : vd. prossima slide
        b.setOnAction(bh);

        root.getChildren().addAll(t, pf, b, loginMsg);

        Scene s1 = new Scene(root, 400, 300);
        stage.setTitle("Admin panel");
        stage.setScene(s1);
        stage.show();
    }
}
```

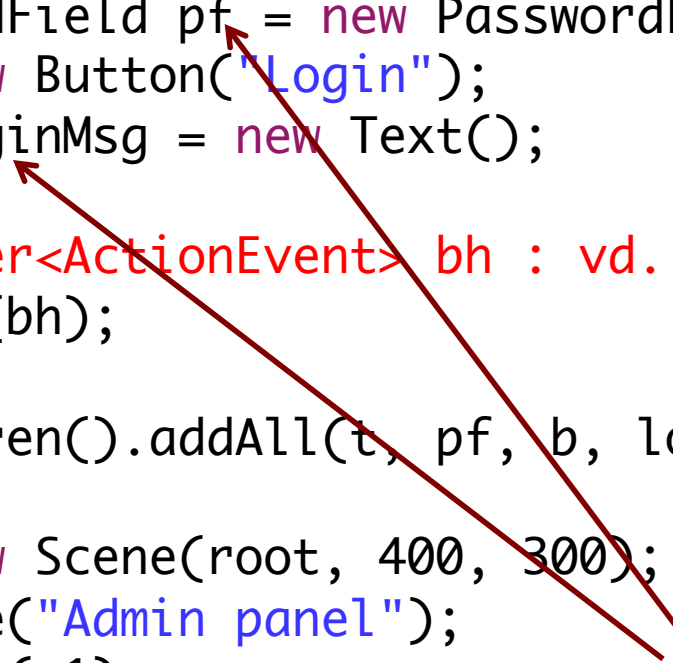


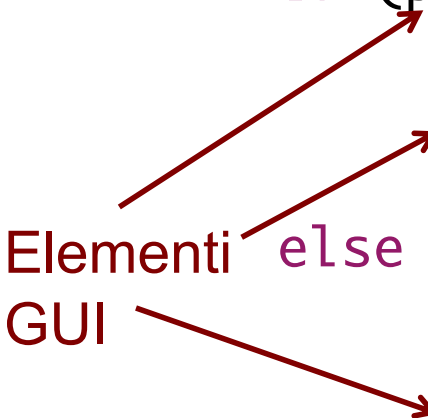
Diagram illustrating the GUI elements (PasswordField, Button, Text) being passed to the EventHandler (bh) for the button's onAction event.

Elementi GUI
visibili nell'handler

```
public class AdminDemo extends Application {  
    // hard-coded (and stupid) password...  
    private static String password = "admin";  
    private static Integer trials = 0;  
    ...  
}
```

```
EventHandler<ActionEvent> bh = new  
    EventHandler<ActionEvent>() {  
        public void handle(ActionEvent event) {  
            trials++;  
  
            if (pf.getText().equals(password)) {  
                loginMsg.setText("Login successfull!");  
                loginMsg.setFill(Color.GREEN);  
            }  
            else {  
                loginMsg.setText("Forgot your password? You  
                    tried " + trials + " times...");  
                loginMsg.setFill(Color.RED);  
            }  
        }  
    }  
};
```

Elementi GUI



Separare logica e presentazione

- Nell'esempio visto finora l'applicazione gestisce sia la logica che l'interfaccia grafica
- E' buona regola separare il più possibile queste due parti per garantire modularità, flessibilità e una più facile manutenzione
- Dal punto di vista logico il gestore delle password non è parte della GUI
- A maggior ragione questo vale per dati più complessi su cui si prevedono più operazioni (e più complesse)
- Si possono usare interfacce diverse per accedere alla stessa applicazione; cambiare l'interfaccia non dovrebbe richiedere cambiamenti alla logica dell'applicazione stessa

```
public class PasswordManager {  
    protected boolean checkPassword(String pwd) {...}  
    ...  
}
```

```
...  
final PasswordManager pwdManager = new PasswordManager();
```

```
EventHandler<ActionEvent> bh = new  
    EventHandler<ActionEvent>() {  
        public void handle(ActionEvent event) {  
            trials++;  
  
            if (pwdManager.checkPassword(pf.getText())) {  
                loginMsg.setText("Login successfull!");  
                loginMsg.setFill(Color.GREEN);  
            }  
            else {  
                ...  
            }  
        }  
    }  
}
```

Gestore dati
separato dalla GUI

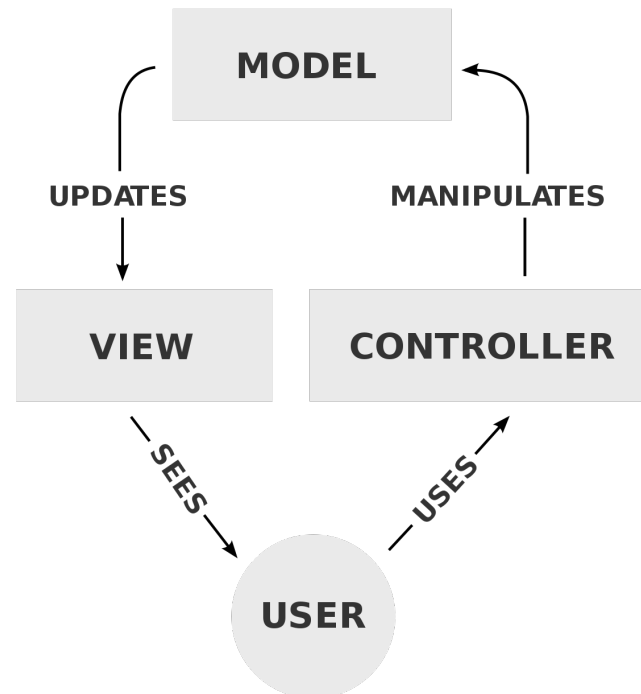
...

Model View Controller (MVC)

- Il pattern *Model-View-Controller* permette di effettuare una netta divisione tra presentazione e logica dividendo l'applicazione in tre componenti separati
 - **Model**: descrizione e modellazione dei dati, fornendoli o modificandoli quando il controller lo richiede
 - **View**: interfaccia e interazione con l'utente, visualizzando i dati in modo appropriato
 - **Controller**: la logica dell'applicazione. Riceve le richieste, recupera i dati (dal model), li elabora e li passa alla view per essere visualizzati

MVC

- Lo schema concettuale MVC è condiviso ma i dettagli dipendono dal tipo di applicazione (es. Web o stand-alone) e dalle tecnologie i (es. linguaggi di programmazione o framework che supportano il pattern)

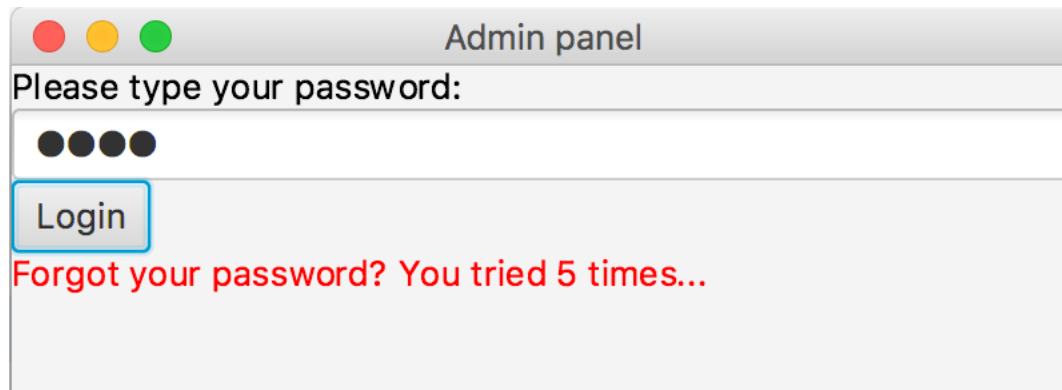


FXML

- FXML è un linguaggio XML per esprimere **viste** in modo **dichiarativo** in applicazioni *MVC*
- L'interfaccia è descritta in un file esterno, letto all'avvio per disegnare l'interfaccia
- Il linguaggio fornisce un insieme di elementi annidati (tag di apertura/chiusura) corrispondenti ai nodi di Java FX
 - le proprietà di ogni nodo sono specificate tramite attributi di questi elementi
- Anche gli import sono definiti tramite appositi elementi
- Documentazione:
https://docs.oracle.com/javafx/2/fxml_get_started/jfxpub-fxml_get_started.htm

Esempio Login in FXML

- Realizziamo un'applicazione che chiede all'utente una password, verifica che sia corretta e mostra un messaggio:
 - Se corretta: *"Login successful"*
 - Se sbagliata: numero di tentativi dal lancio dell'applicazione



The screenshot shows a window titled "Admin panel" with a standard macOS-style title bar (red, yellow, green buttons). Below the title bar, the text "Please type your password:" is displayed. Underneath this text is a password input field containing four black dots. A blue rectangular button labeled "Login" is positioned below the input field. At the bottom of the window, a red error message reads: "Forgot your password? You tried 5 times..."


Struttura base applicazione

```
// dichiarazione package e altri import omessi
import javafx.fxml.FXMLLoader;

public class FXMLAdminDemo extends Application {

    @Override
    public void start(Stage stage) throws Exception {
        Parent root =
            FXMLLoader.load(getClass().getResource("admin.fxml"));

        stage.setTitle("Admin panel");
        stage.setScene(new Scene(root, 400, 300));
        stage.show();
    }
}
```



sorgente FXML
(vd. prossima slide)

Esempio FXML

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

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

```
<?xml version="1.0" encoding="UTF-8"?>
<?import java.net.URL?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.control.PasswordField?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.text.Text?>
<?import javafx.scene.layout.VBox?>
```

Import in sintassi
(F)XML

```
<VBox styleClass="root"
xmlns="http://javafx.com/javafx/8.0.121"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="gui.fx.fxml.FXMLAdminController">
  <children>
```

```
    <Label text="Please type your password:"/>
```

```
    <PasswordField fx:id="pf"/>
```

```
    <Button onAction="#checkLogin" text="Login" />
```

```
    <Text fx:id="loginMsg"/>
```

```
  </children>
```

```
</VBox>
```

Elementi GUI
visibili al Controller

Controller

- FXML facilita la separazione tra la View e il Controller
- E' possibile indicare nel file `.fxm1` il controller che gestire l'interfaccia (o le sue parti)
`fx:controller="gui.fx.fxm1.FXMLAdminController"`
- ...e i metodi del Controller che saranno invocati quando si scatena un evento
`<Button onAction="#checkLogin" text="Login" />`
- *Nota: attenzione al nome della classe Controller e il package in cui è contenuta*

```
// dichiarazione package e import omessi
```

```
public class FXMLAdminController {
```

```
    @FXML private Text loginMsg;
```

```
    @FXML private PasswordField pf;
```

Elementi GUI



```
    private String password = "admin19"; // hard-coded...
```

```
    private Integer trials = 0;
```

```
    @FXML protected void checkLogin(ActionEvent event) {
```

```
        this.trials++;
```

```
        if (pf.getText().equals(this.password)) {
```

```
            loginMsg.setText("Login successfull!");
```

```
            loginMsg.setFill(Color.GREEN);
```

```
        }
```

```
        else {
```

```
            loginMsg.setText("Forgot ... ..");
```

```
            loginMsg.setFill(Color.RED);
```

```
        }
```

```
    }
```

```
}
```


Accedere ai controller e trasferire informazioni

- Quando si cambia vista (*view*) può essere utile trasferire informazioni al controller associato alla nuova vista
- Per farlo si può usare la classe `FXMLLoader` che carica la nuova vista ed espone il metodo `getController()` per accedere al controller associato
- Questo controller espone i metodi per ricevere i dati

```
FXMLLoader loader = new  
FXMLLoader(getClass().getResource("nuovascena.fxml"));  
  
Parent root = loader.load();  
  
NextStepController nextController = loader.getController();  
  
nextController.transferData(inputField.getText());
```

Esempio

- Da un Controller *Admin* passa il controllo a un nuovo controller *Help* a cui è associata la view *help.fxml* e che espone un metodo *copyTrials(trials)* per ricevere un valore intero

```
@FXML protected void showHelp(ActionEvent event) throws IOException
{
    FXMLLoader loader = new
    FXMLLoader(getClass().getResource("help.fxml"));

    Parent help = loader.load();

    FXMLLoaderHelpController helpController = loader.getController();

    Scene helpScene = new Scene(help, 300, 300);

    helpController.copyTrials(trials);

    Stage stage = (Stage) loginMsg.getScene().getWindow();

    stage.setScene(helpScene);

    stage.show();
}
```

helpController

```
public class FXMLHelpController {  
    @FXML private Label trialsMsg;  
    protected void copyTrials(Integer trials){  
        trialsMsg.setText("You tried " + trials + " times in the  
            login window...");  
    }  
  
    @FXML protected void tornaIndietro(ActionEvent event) throws  
        IOException {  
        Parent root =  
            FXMLLoader.load(getClass().getResource("admin.fxml"));  
        Stage stage = (Stage) trialsMsg.getScene().getWindow();  
        stage.setTitle("Admin panel");  
        stage.setScene(new Scene(root, 500, 300));  
    }  
}
```

ELEMENTI GRAFICI E STILI CSS

Styling degli elementi: CSS

- Java FX permette di usare CSS per decidere la formattazione degli elementi
- CSS è uno standard Web che permette la definizione di regole di presentazione, applicate a cascata
- CSS è uno standard molto ricco e una parte è supportata in Java FX
- Due concetti principali:
 - **Selettori**: permettono di identificare gli elementi (o l'elemento singolo) su cui si applica la formattazione
 - **Proprietà**: strutturate nella forma `<chiave>: <valore>` permettono di decidere la formattazione

Esempio CSS

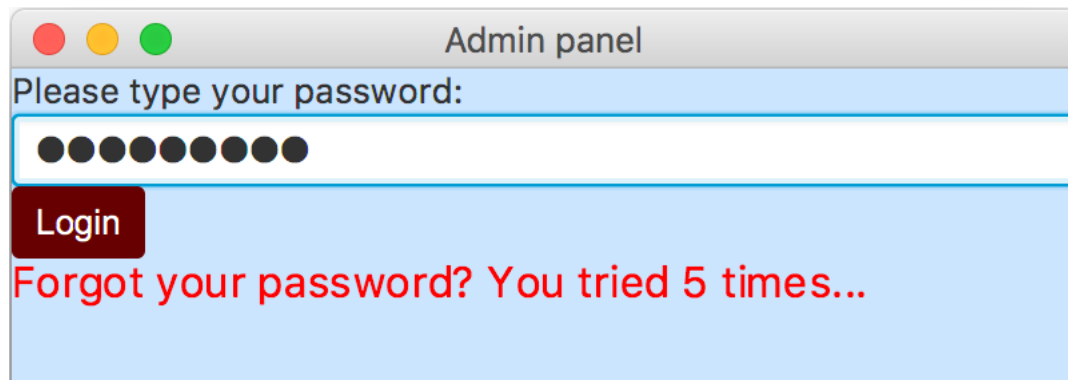
```
.root {  
    -fx-background-color: #CCE5FF;  
}  
  
#loginMsg {  
    -fx-fill: #660000;  
    -fx-font-weight: bold;  
    -fx-font-size: 16px;  
}  
  
.button {  
    -fx-font-family: "Arial";  
    -fx-text-fill: white;  
    -fx-background-color: #660000;  
}
```

Java FX e CSS

- Applichiamo questo CSS all'esempio visto finora
 - Nota: CSS si può usare anche senza FXML
 - Nota: si può usare anche stream e metodo `getResource()`
 - Attenzione al path dei file!

```
Scene s1 = new Scene(root, 400, 300);
```

```
File css = new File("src/gui/jfx/fxml/Login.css");  
s1.getStylesheets().add("file://" + css.getAbsolutePath());
```



Proprietà CSS

- CSS fornisce diverse proprietà (con prefisso `-fx`) per i diversi elementi dell'interfaccia: font, colori, dimensioni, spaziature, etc.
- Selettori di classe (iniziano con `.`) o di elemento (iniziano con `#`)

```
.important {  
    -fx-background: #AA4411;  
    -fx-fill: green;  
  
#t3 {  
    -fx-fill: red;  
    -fx-font: 60px Verdana; }  
  
.root{  
    -fx-font-size: 16pt;  
    -fx-font-family: "Courier New";}
```



```
Text t1 = new Text(50, 50, "Orange");  
t1.getStyleClass().add("important");
```

```
Text t2 = new Text(100,100, "Apple");
```

```
Text t3 = new Text(20,150, "Lemon");  
t1.getStyleClass().add("important");  
t3.setId("t3");
```

```
Group g = new Group();
```

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

```
Scene s1 = new Scene(g);
```

```
File css = new File("src/gui/fx/style.css");
```

```
s1.getStylesheets().add("file:/// " + css.getAbsolutePath());
```



Orange

Apple

Lemon

Note finali su CSS

- CSS si può usare indipendentemente da FXML
- Sintassi FXML per includere un foglio di stile CSS:

```
<stylesheets>
```

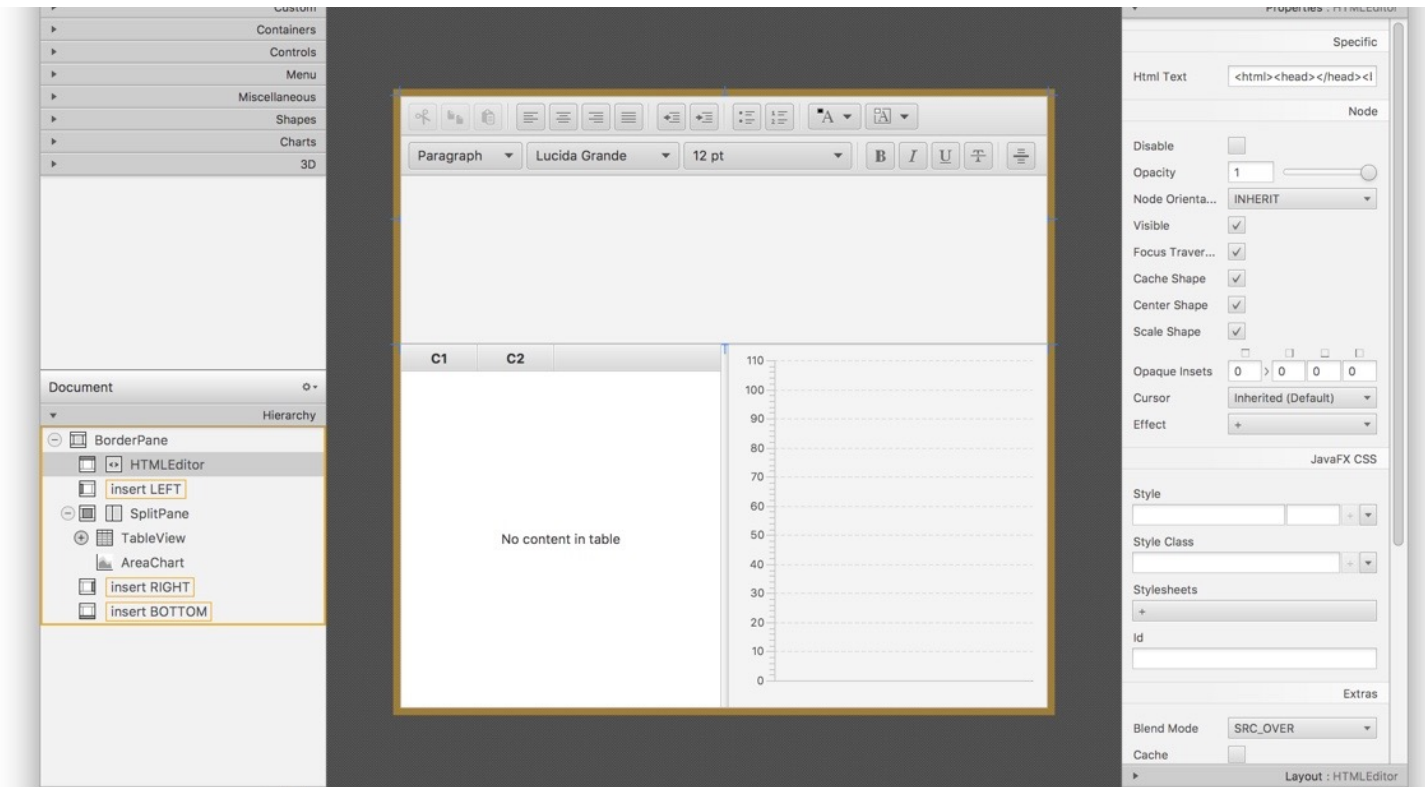
```
    <URL value="@Login.css" />
```

```
</stylesheets>
```

- E' opportuno usare i layout predefiniti di JavaFX per l'organizzazione spaziale dell'interfaccia

TOOLS

Java FX Scene Builder



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