Programmazione III

Prof.ssa Liliana Ardissono
Dipartimento di Informatica
Università di Torino

Interfacce Utente Grafiche (GUI)

Overview di JavaFX

JavaFX - I

JavaFX è una libreria grafica di Java per lo sviluppo di GUI in applicazioni stand-alone.

- **Per usare JavaFX serve aver compreso SWING:** SWING fornisce i concetti fondamentali di componente, listener, etc. in modo «didattico».
- Nella programmazione delle GUI, JavaFX separa il contenuto dalla sua visualizzazione tramite **fogli stile CSS** (simile a HTML).
- JavaFX permette il **binding di property dei Model** con elementi dell'interfaccia utente per aggiornare automaticamente le viste.
- JavaFX offre le **classi/interface che implementano Observer Observable** e non sono deprecate.
- JavaFX(ML) permette anche di **scrivere le GUI con XML**.

JavaFX - II

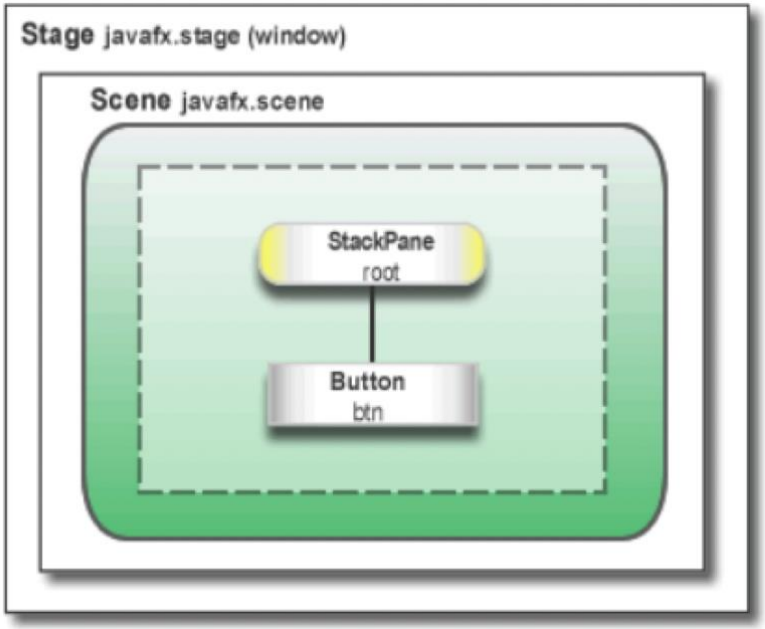
- Tutorials: https://docs.oracle.com/javafx/2/get_started/jfxpub-get_started.htm
- Documentazione: http://docs.oracle.com/javafx/2/
- API: https://openjfx.io/javadoc/13/

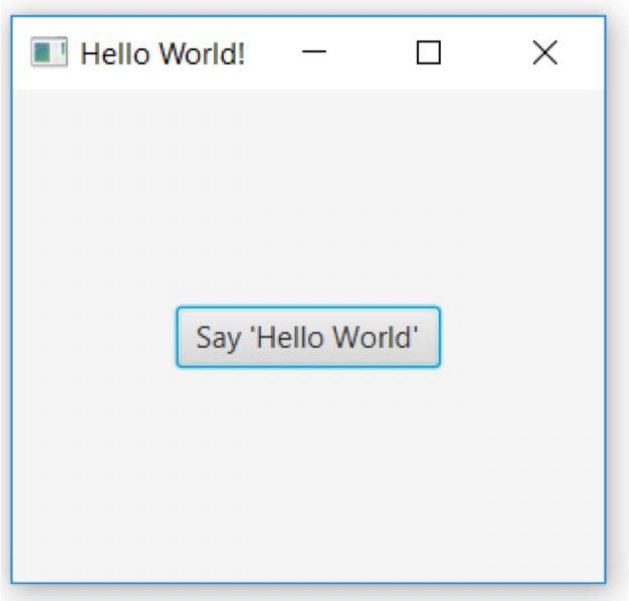
JavaFX – struttura delle applicazioni

- Ogni applicazione JavaFX deve estendere **javafx.application.Application**
- **L'entrypoint di un'applicazione JavaFX** (che viene eseguito da JVM quando la si lancia) è il metodo **start()**. Questo metodo prende uno **Stage (finestra dell'applicazione)** come parametro e viene invocato dal launcher dell'applicazione

```
public class JavaFXApplication1 extends Application {
    @Override
    public void start(Stage primaryStage) {
                        ...
    }
    public static void main(String[] args) {
      launch(args); //esegue il launcher dell'applicazione
    }
```

JavaFX – Componenti principali - I

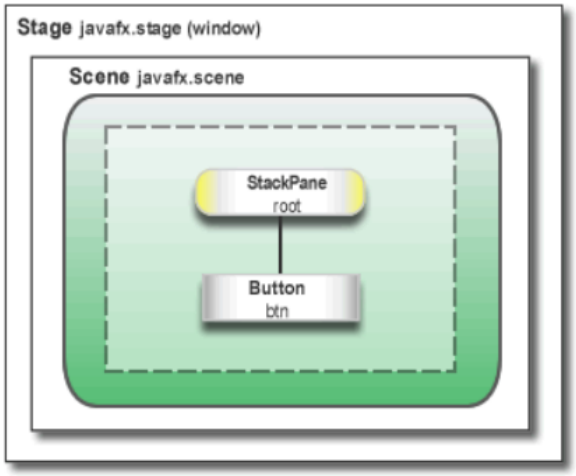




Componenti principali di JavaFX:

- Lo **Stage** rappresenta la finestra (simile a JFrame di SWING)
- La **Scene** è il contenitore principale (e unico) da aggiungere a una finestra. Una Scene va associata a un pannello per definire il layout (per es. **StackPane**, posiziona i componenti "figli" inseriti al suo interno in una pila, uno sopra l'altro)

JavaFX – Componenti principali - II



- **La struttura della GUI è gerarchica**: nel **pannello** della Scene noi possiamo inserire i componenti figli. I componenti figli possono a loro volta avere componenti figli.
  Non si possono avere 2 Scene in un solo Stage: bisogna dare alla Scene un pannello con il layout desiderato,
  es., StackPane, o GridPane, e aggiungere al pannello
  i sotto-componenti grafici come suoi figli.

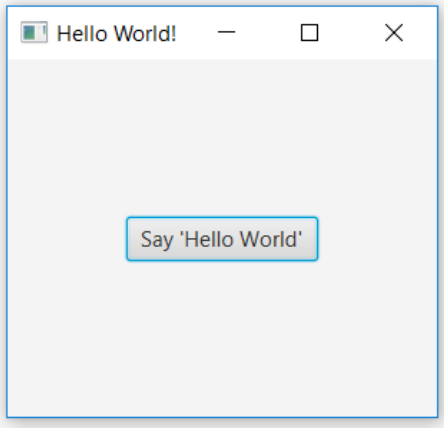
Attenzione: usare JavaFXML

- Nel seguito vedremo una bozza di applicazione JavaFX per capire quali tipi di oggetti formano le GUI.
- Questo NON sarà il modo di sviluppare le applicazioni. A tale scopo useremo JavaFXML, che permette di progettare la GUI graficamente.

JavaFX – Componenti principali - III

Estratto di codice per inserire in uno Stage una Scene (con pannello StackPane) e un bottone:

```
public void start(Stage primaryStage) {
   Button btn = new Button();       // crea il bottone
   btn.setText("Say 'Hello World'");
   StackPane root = new StackPane(); // componente ROOT
   root.getChildren().add(btn); //aggiunge btn a pannello
   Scene scene = new Scene(root, 300, 250); //dimensioni
   primaryStage.setTitle("Hello World!");
   primaryStage.setScene(scene);
   primaryStage.show();       // visualizza la finestra
}
```



JavaFX – Event Handlers

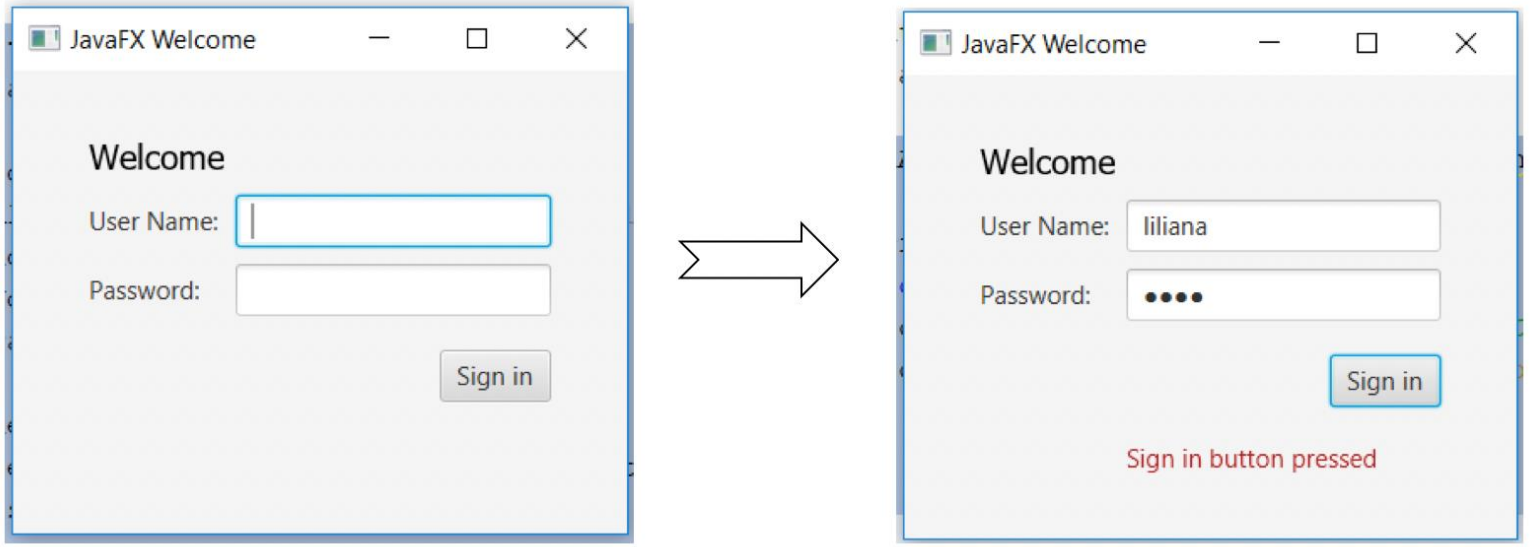
Alle componenti grafiche possono essere associati i Listener, per reagire alle azioni dell'utente. Es. per i bottoni:

```
Button btn = new Button(); // creo un bottone
btn.setText("Say 'Hello World'");
btn.setOnAction(new EventHandler<ActionEvent>() {
        public void handle(ActionEvent event) {
           System.out.println("Hello World!");
       }
    });
  // il listener è stato definito come classe
        // anonima. Il metodo handle() viene ridefinito
        // per scrivere su output standard un
        // messaggio ad ogni click sul bottone
```

JavaFX – Codice completo dell'applicazione

```
public class JavaFXApplication1 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);
    }
}
```

JavaFX – Forms (moduli) - I



Si basano su:

- Pannello **GridPane** per inserire i componenti grafici in una griglia
- **Label** per scrivere i titoli dei campi delle form
- **TextField** per definire i campi di input delle form
- **Bottoni** con listener per sottomettere le form
- **Text** per scrivere messaggi di output sulla finestra

JavaFX – Forms (moduli) - II

```
public void start(Stage primaryStage) {
   primaryStage.setTitle("JavaFX Welcome");
   GridPane grid = new GridPane();// pannello a griglia

   // … omissis …

   Text scenetitle = new Text("Welcome");
   scenetitle.setFont(Font.font("Tahoma",FontWeight.NORMAL,20));
                        // columnindex, rowindex, column span, row span
   grid.add(scenetitle, 0, 0, 2, 1);

   Label userName = new Label("User Name:");
   grid.add(userName, 0, 1);

   TextField userTextField = new TextField();
   grid.add(userTextField, 1, 1);

   // continua ....
```

sceneTitle	sceneTitle
userName	userTextField

JavaFX – Forms (moduli) - III

```
// continua ....

   Label pw = new Label("Password:");
   grid.add(pw, 0, 2);

   PasswordField pwBox = new PasswordField();
   grid.add(pwBox, 1, 2);

   // creo il bottone di sottomissione dei dati
   Button btn = new Button("Sign in");
   // aggiungo il bottone alla griglia… omesso

// continua ....
```

sceneTitle	sceneTitle
userName	userTextField
pw	pwBox
	hbBtn

JavaFX – Forms (moduli) - IV

```
// continua ....
   final Text actiontarget = new Text();
   grid.add(actiontarget, 1, 6);

   btn.setOnAction(new EventHandler<ActionEvent>() {
           @Override
           public void handle(ActionEvent e) {
               actiontarget.setFill(Color.FIREBRICK);
               actiontarget.setText("Sign in button pressed");
           }
       });
   // qui creo la Scene e la inserisco nella finestra
   Scene scene = new Scene(grid, 300, 275);
   primaryStage.setScene(scene);
   primaryStage.show();
}
```

sceneTitle	sceneTitle
userName	userTextField
pw	pwBox
	hbBtn
	actionTarget

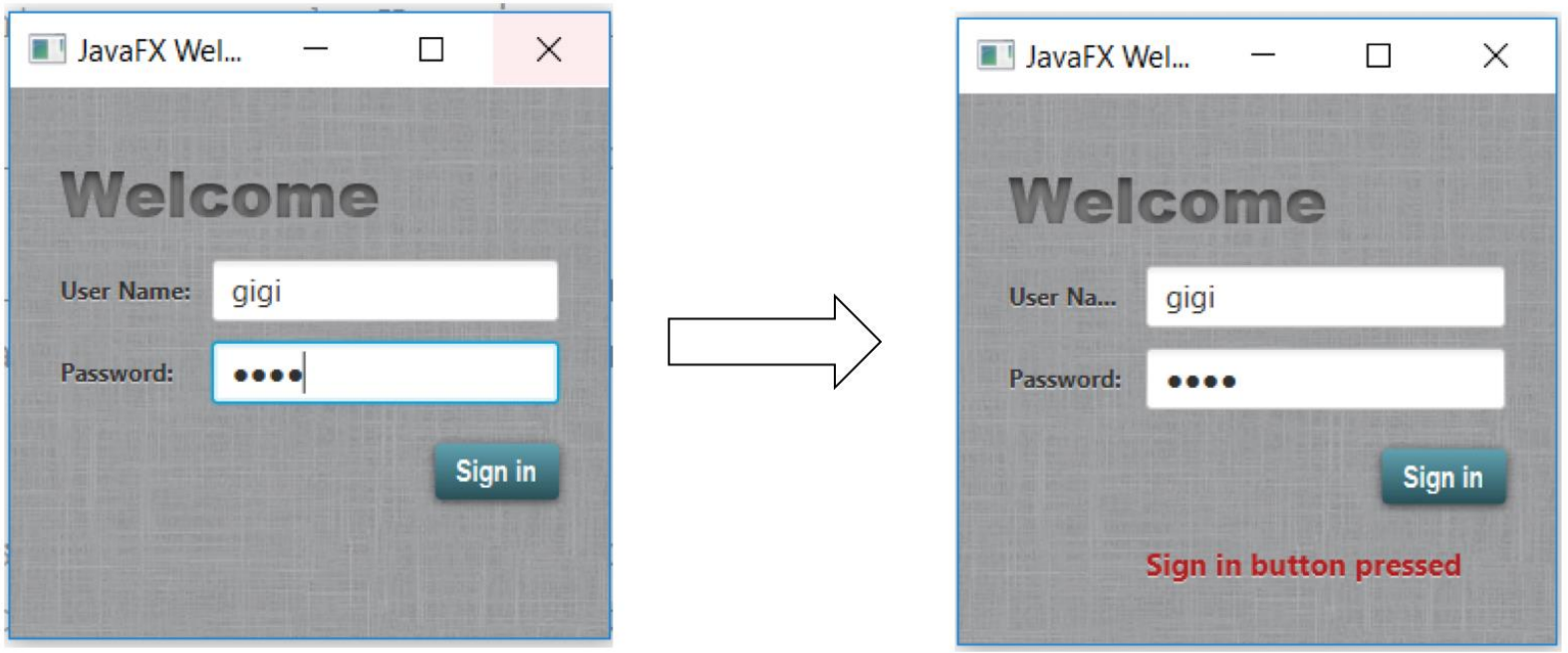
JavaFX – uso di CSS - I

JavaFX permette di separare il contenuto di una GUI dal suo layout. A tale scopo, JavaFX offre i seguenti elementi:

- **Un foglio stile CSS** (es. **stile.css**), che posizioniamo nella cartella del codice dell'applicazione.
- **Un metodo di Scene per caricare il foglio stile:**
  scene.getStylesheets().add(
  MyApp.class.getResource("stile.css").toExternalForm()**);**
- **Un modo di dare gli ID ai componenti grafici,** ove si voglia
  applicare una regola CSS ad uno specifico elemento.
  Es: **scenetitle.setId("welcome-text");**

Esempio: layout desiderato

Data l'applicazione che genera la form precedente, aggiungiamo il layout come foglio stile



Foglio stile CSS – stile.css - I

```
.root { /* selettore di classe → per componenti tipati */
    -fx-background-image: url("background.jpg"); }
.label {
   -fx-font-size: 12px;
   -fx-font-weight: bold;
   -fx-text-fill: #333333; /* grigio */
   -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 ); /*
testo shaded */
}
#welcome-text { /* selettore per ID */
  -fx-font-size: 32px;
  -fx-font-family: "Arial Black";
  -fx-fill: #818181;
  -fx-effect: innershadow( three-pass-box , rgba(0,0,0,0.7) , 6, 0.0 , 0 , 2 );
}
```

Foglio stile CSS – stile.css - I

```
#actiontarget {
  -fx-fill: FIREBRICK;
  -fx-font-weight: bold;
  -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );}
.button {
   -fx-text-fill: white;
   -fx-font-family: "Arial Narrow";
   -fx-font-weight: bold;
   -fx-background-color: linear-gradient(#61a2b1,#2A5058);
   -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 5, 0.0 , 0 , 1 ); }
.button:hover {
    -fx-background-color: linear-gradient(#2A5058, #61a2b1);
}
```

Pannello con ID etc.

```
public MyPanel() {
        super();// pannello a griglia
        this.setAlignment(Pos.CENTER);
        ...
        Text scenetitle = new Text("Welcome");
        //scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
                                                    // NON SERVE PIÙ
        this.add(scenetitle, 0, 0, 2, 1);
        scenetitle.setId("welcome-text"); // per applicare CSS
        Label userName = new Label("User Name:");
        this.add(userName, 0, 1);
        ...
        final Text actiontarget = new Text();
        actiontarget.setId("actiontarget"); // per applicare CSS
        this.add(actiontarget, 1, 6);
        ...
}
```

JavaFX – tipi di pannelli (Pane)

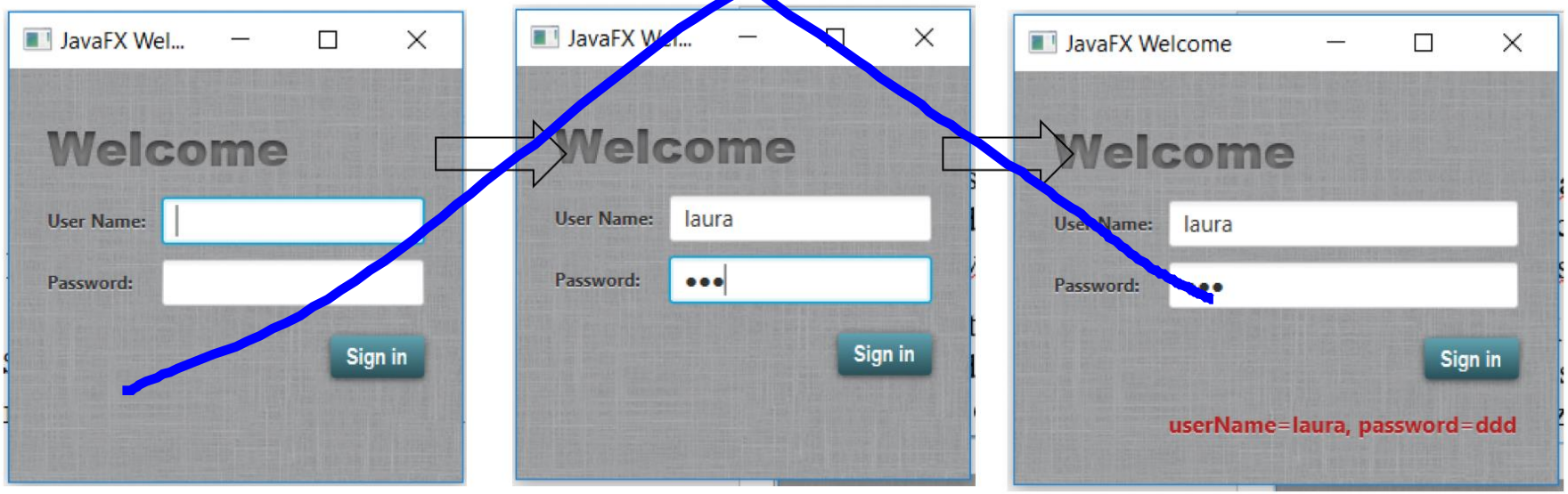
JavaFX offre i seguenti tipi di pannello per organizzare il layout grafico dell'interfaccia utente:

- **GridPane** fa inserire i componenti figli in una griglia
- **StackPane** fa sovrapporre i figli in uno stack
- **BorderPane** fa inserire i figli a nord, est, ovest, sud, centro come per il BorderLayout di SWING.

Ci interessa sapere che questi tipi di pannello esistono, anche se non li usiamo direttamente qui, perché ci serviranno per la grafica di JavaFXML. Svilupperemo tale grafica *in combinazione con un tool grafico (SceneBuilder) per il design dell'interfaccia utente che genera il codice XML della GUI.*

JavaFX e MVC

- Le classi di JavaFX possono essere integrate con Observer Observable per sviluppare applicazioni MVC
- Es: ristrutturiamo la precedente applicazione in modo che salvi in un data model i dati inseriti dall'utente e utilizzi il data model per visualizzarli sulla GUI:



JavaFX e MVC – versione base (usa le classi/interface Observer e Observable - deprecate)

Per iniziare, sviluppiamo il data model etc. utilizzando le librerie di base del pattern Observer Observable in java

NB: In JavaFX esistono librerie che offrono le funzionalità degli Observer e degli Observable, le useremo:

- **ObservableList** implementa una lista di oggetti osservabili. ObservableList invia in automatico le notifiche agli osservatori quando cambia lo stato degli oggetti.
- **ListView** è un visualizzatore di liste. Una ListView può essere agganciata come osservatore ad una ObservableList per far sì che si aggiorni la visualizzazione della lista ad ogni cambiamento dell'osservato.
- …
- **https://docs.oracle.com/javafx/2/collections/jfxpub-collections.htm**

Vecchio

Data Model

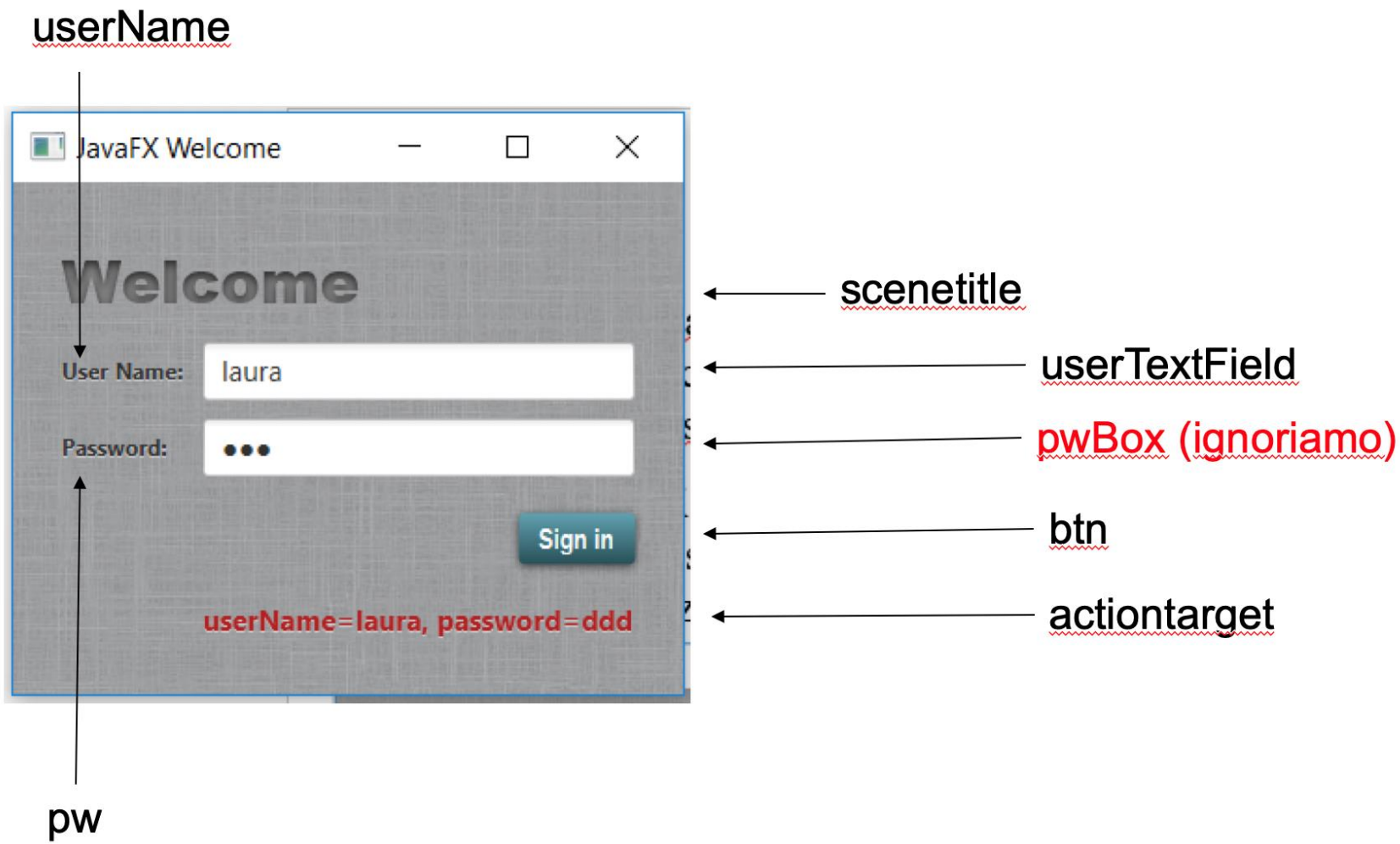
```
public class Utente extends Observable { // DEPRECATO!!!!!
    private String userName;
    private String password;
    public Utente() { userName = ""; password = ""; }
    public String getUserName() { return userName; }
    public String getPassword() { return password; }
    public void setData(String u, String p) {
        userName = u; password = p;
        setChanged(); notifyObservers();
    }
    @Override
    public String toString() {
        return "userName=" + userName +  ", password=" + password;
    }
}
```

Controller

```
public class Controller {

    public Controller(MyPanel grid, Utente person) {
       Button btn = grid.getButton(); // prendo il
    // riferimento al bottone per agganciare il listener (EventHandler)
             btn.setOnAction(new EventHandler<ActionEvent>() {
           @Override
           public void handle(ActionEvent e) {
                    person.setData(grid.getUserName(),
                                   grid.getPassword());
           }
       });
    }
}
```

View – componenti



View- MyPanel – I

```
public class MyPanel extends GridPane
                                    implements Observer { //DEPRECATO
    private final Text actiontarget;
    private final Button btn;
    private final TextField userTextField;
    …

    public MyPanel() {
        super();
        ...
        Text scenetitle = new Text("Welcome");
        this.add(scenetitle, 0, 0, 2, 1);
        scenetitle.setId("welcome-text");
          // continua…
```

View – MyPanel - II

```
Label userName = new Label("User Name:");
// 0,1: posizionamento della label in griglia
this.add(userName, 0, 1);
userTextField = new TextField();
this.add(userTextField, 1, 1);

//… field text per la password – ignoriamo

// continua ...
```

View – MyPanel - III

```
        btn = new Button("Sign in");
          …
        hbBtn.getChildren().add(btn);
        this.add(hbBtn, 1, 4);

        actiontarget = new Text();
        actiontarget.setId("actiontarget");
        this.add(actiontarget, 1, 6);
}

Button getButton() {
    return btn;
}
```

View – MyPanel - IV

```
String getUserName() {
    return userTextField.getText();
}

String getPassword() {
    return pwBox.getText();
}

public void update(Observable obs, Object extra_arg) {
    System.out.println("Refresh GUI");
    actiontarget.setFill(Color.FIREBRICK);
    if (obs instanceof Utente) {
        actiontarget.setText(((Utente) obs).toString()); }
}
} // end MyPanel
```

Applicazione principale - I

```
public class JavaFXApplication5MVCsimple extends Application {
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("JavaFX Welcome");
        Utente person = new Utente(); // creo il model
        MyPanel grid = new MyPanel(); // creo la view
                        // aggancio la view come observer del model
        person.addObserver(grid);
        Controller control = new Controller(grid, person);
            Scene scene = new Scene(grid, 300, 275);
        primaryStage.setScene(scene);
    ...
```

Applicazione principale – versione base - II

```
            scene.getStylesheets().add(      // foglio stile
          JavaFXApplication5MVCsimple.class.getResource("stile.cs
s").toExternalForm());
        primaryStage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```