

Metode avansate de programare

Informatică Română, Curs 6

GUI - JavaFX



Cuprins

- Ce este JavaFX
- Graful de scene
- Lucrul cu componentele grafice
- Gestionarea pozitionării
- Tratarea evenimentelor

Ce este JavaFX ?

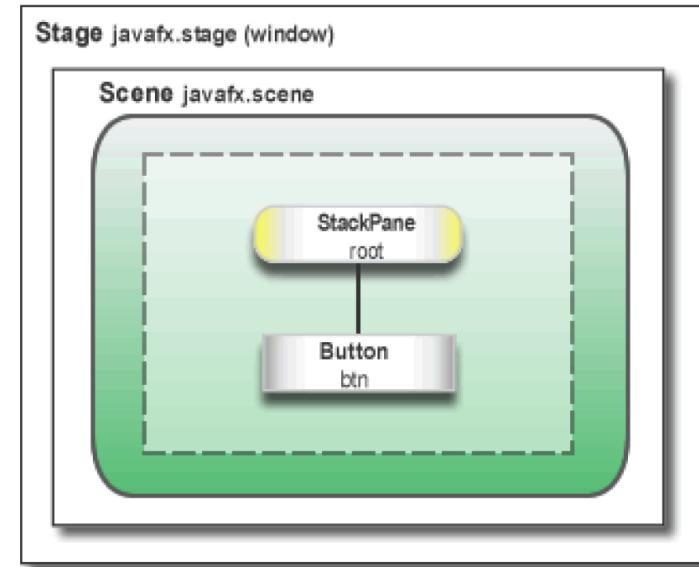
- Clase si interfete care asigura suport pentru crearea de aplicatii Java care se pot proiecta, implementa, testa pe diferite platforme.
- Asigura suport pentru utilizarea de componente Web cum ar fi apeluri de scripturi JavaScript sau cod HTML5
- Contine componente grafice UI pentru crearea de interfete grafice si gestionarea aspectului lor prin fisiere CSS
- Asigura suport pentru grafica interactiva 3D
- Asigur suport pentru manipulare de continut multimedia
- Portabilitate: desktop, browser, dispozitive mobile, TV, console jocuri, Blu-ray, etc.
- Asigura interoperabilitate Swing

JavaFX APIs -Scene Graph

scene-graph-based programming model

O aplicatie JavaFX conține:

- un obiect Stage (fereastra)
- unul sau mai multe obiecte Scene



Graful de scene(Scene Graph) este o structură arborescentă de componente grafice ale interfetei utilizator.

Un element din **graful de scene** este un **Node**.

- Fiecare nod are un **id**, un **stil grafic** asociat și o **suprafață ocupată** (*ID, style class, bounding volume, etc.*)
- Cu exceptia nodului rădăcină, fiecare nod are un singur părinte și 0 sau mai mulți fii.
- Un nod mai poate avea asociate diverse proprietăți (efecte (blur, shadow), opacitate, transformari) și evenimente (*event handlers* (mouse, tastatură))
- Nodurile pot fi interne (Parent) sau frunza

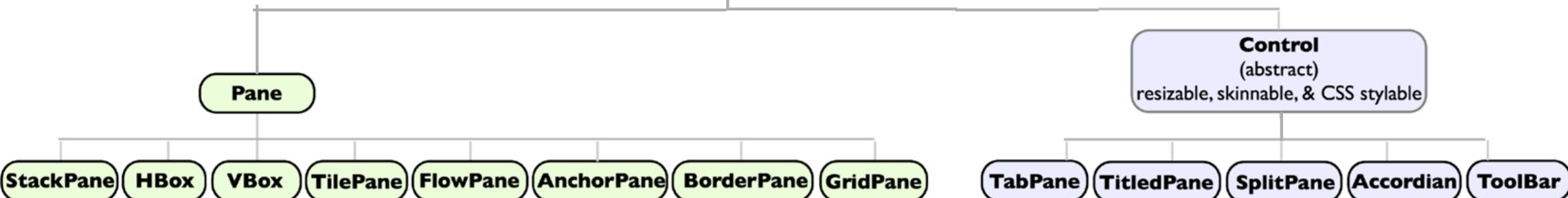
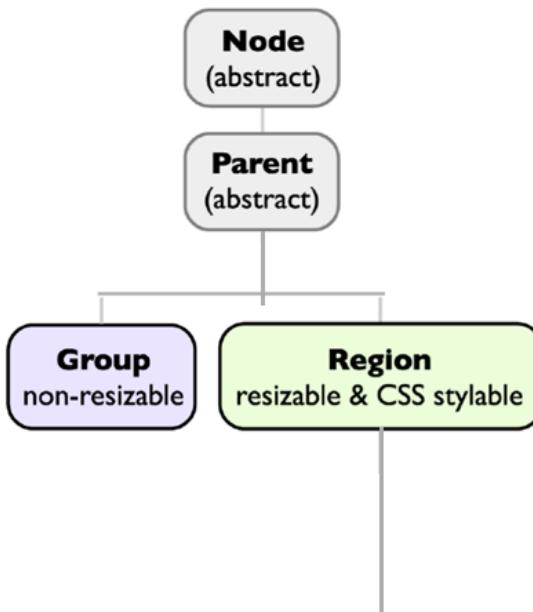
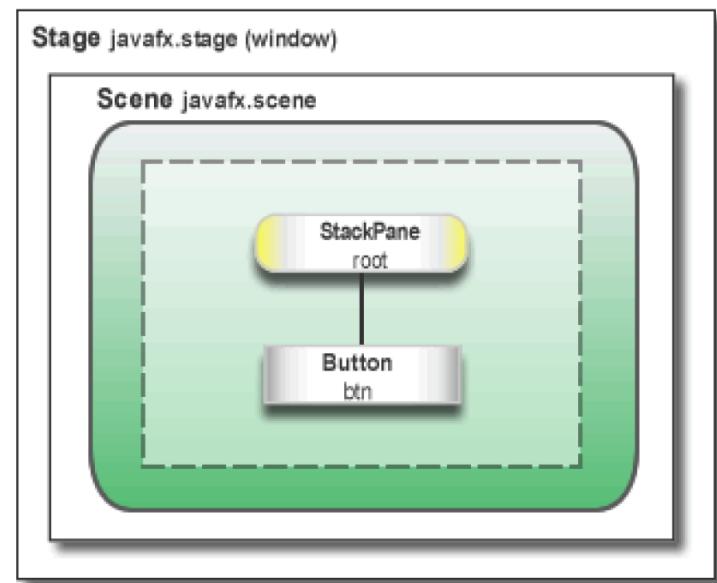
Arhitectura JavaFX

■ controale

- definite in pachetul `javafx.scene.control`
- pot fi grupate in containere / panouri

■ stilizarea continutului folosind foi de stil:

- pot fi aplicate oricarui obiect de tip Node
- proprietățile JavaFX prefixate de-fx-



Aplicații java FX

- O aplicatie JavaFX este o instanta a clasei **Application**
`public abstract class Application extends Object;`
- Instantierea unui obiect Application se face prin executarea metodei statice **launch()**
`public static void launch(String... args);`
args **parametrii aplicatiei**(parametrii metodei *main*).
- JavaFX runtime execută urmatoarele operațiuni:
 1. Crează un obiect **Application**
 2. Apelează metoda **init** a obiectului **Application**
 3. Apelează metoda **start** a obiectului **Application**
 4. Așteaptă sfârșitul aplicatiei
- Parametrii aplicatiei sunt obținuti prin metoda **getParameters()**

Scheletul unei aplicatii JavaFX

```
public class Main extends Application {  
    @Override  
    public void start(Stage stage) {  
        Parent root= initRoot();  
        Scene scene = new Scene(root, 550, 500);  
        stage.setTitle("Welcome to JavaFX!!");  
        stage.setScene(scene);  
        stage.show();  
    }  
  
    public static void main(String[] args) {  
        Launch(args);  
    }  
}
```

Exemplu 1 Group

```
public class Main extends Application {  
  
    @Override  
    public void start(Stage stage) {  
        Group root = new Group();  
        Scene scene = new Scene(root, 500, 500, Color.PINK);  
        stage.setTitle("Welcome to JavaFX!");  
  
        stage.setScene(scene);  
        stage.show();  
    }  
  
    public static void main(String[] args) {  
        Launch(args); //se creaza un obiect de tip Application  
    }  
}
```

Adăugarea nodurilor

```
// Cream un nod de tip Group
```

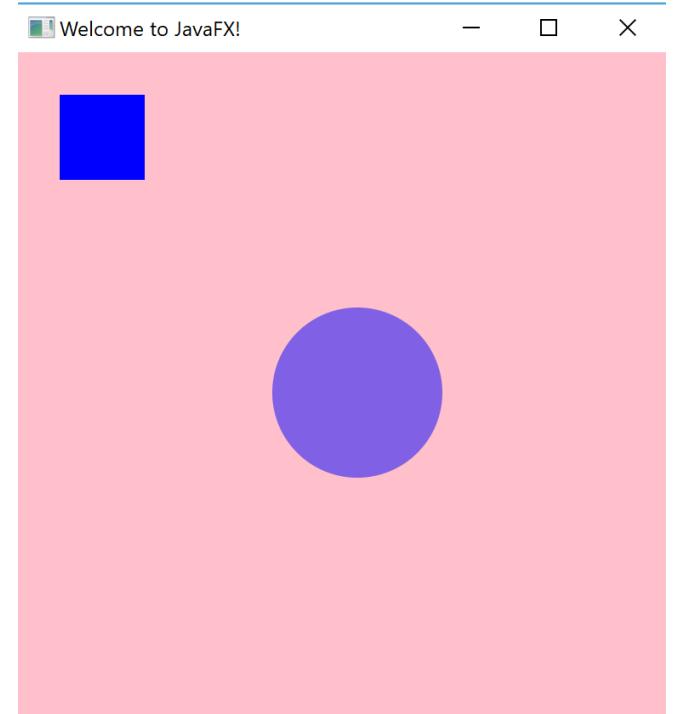
```
Group group = new Group();
```

```
// Cream un nod de tip Rectangle
```

```
Rectangle r = new Rectangle(25,25,50,50);  
r.setFill(Color.BLUE);  
group.getChildren().add(r);
```

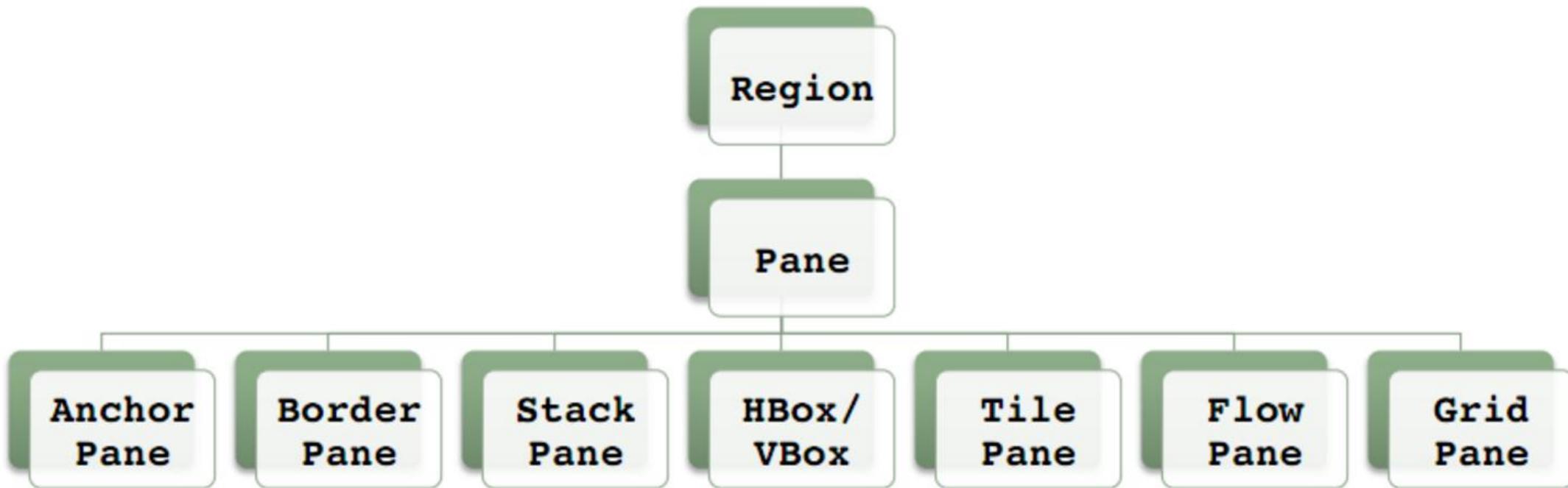
```
// Cream un nod de tip Circle
```

```
Circle c = new Circle(200,200,50, Color.web("blue", 0.5f));  
group.getChildren().add(c);
```



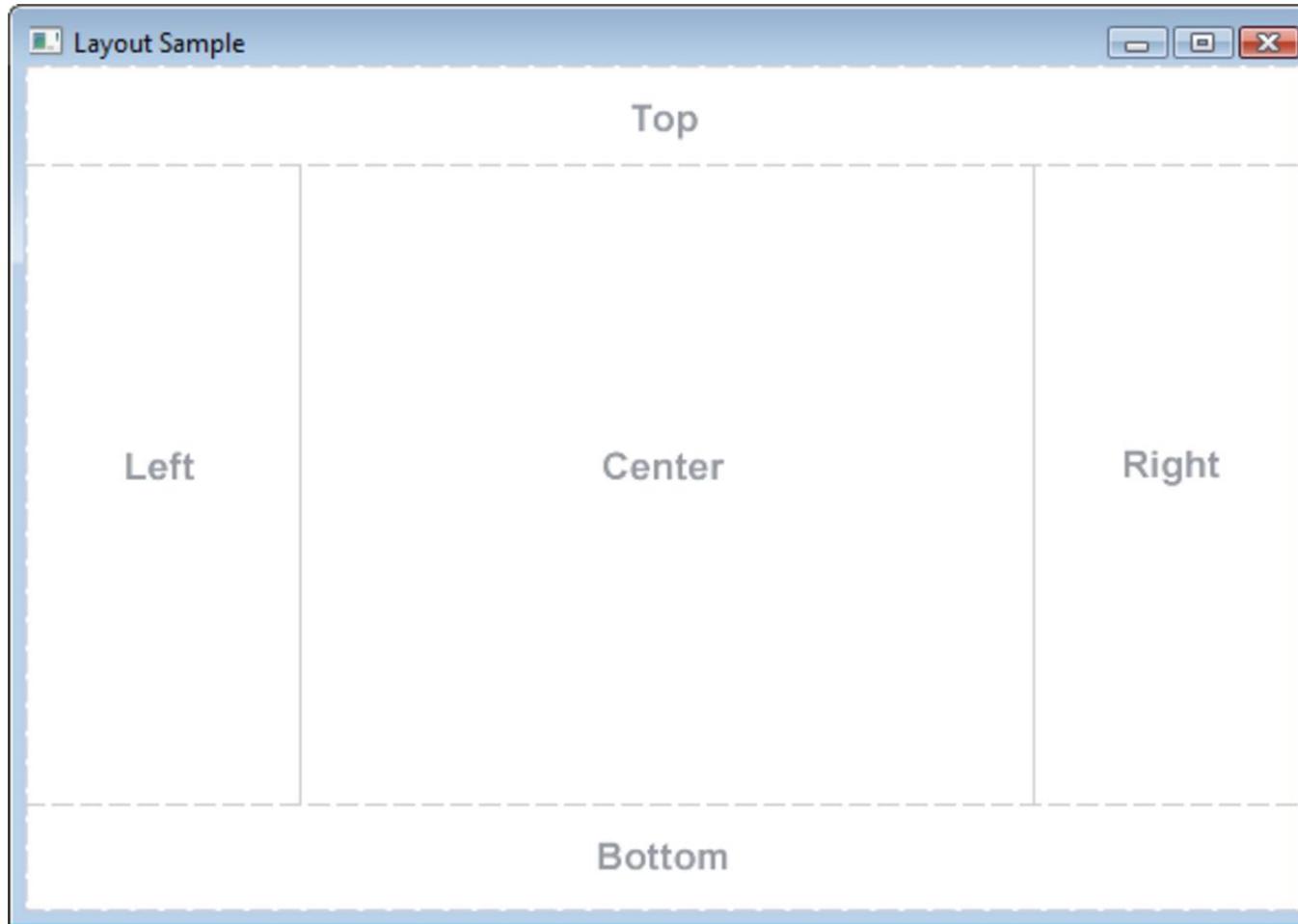
Gestionarea pozitionării componentelor UI

Componete de pozitionare – containere de tip Panou (Pane)



Gestionarea poziționării componentelor UI

- BorderPane



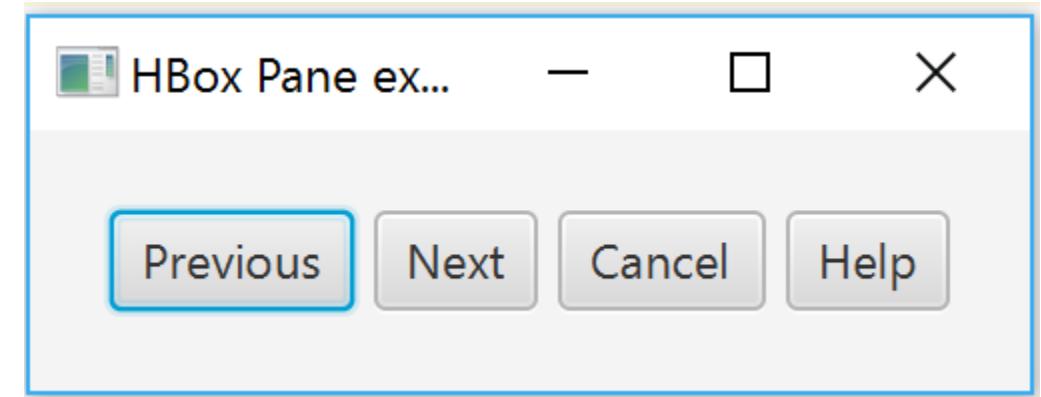
Gestionarea poziționării componentelor UI

■ HBOX

```
HBox root = new HBox(5);  
root.setPadding(new Insets(100));  
root.setAlignment(Pos.BASELINE_RIGHT);
```

```
Button prevBtn = new Button("Previous");  
Button nextBtn = new Button("Next");  
Button cancBtn = new Button("Cancel");  
Button helpBtn = new Button("Help");
```

```
root.getChildren().addAll(prevBtn, nextBtn,  
cancBtn, helpBtn);
```



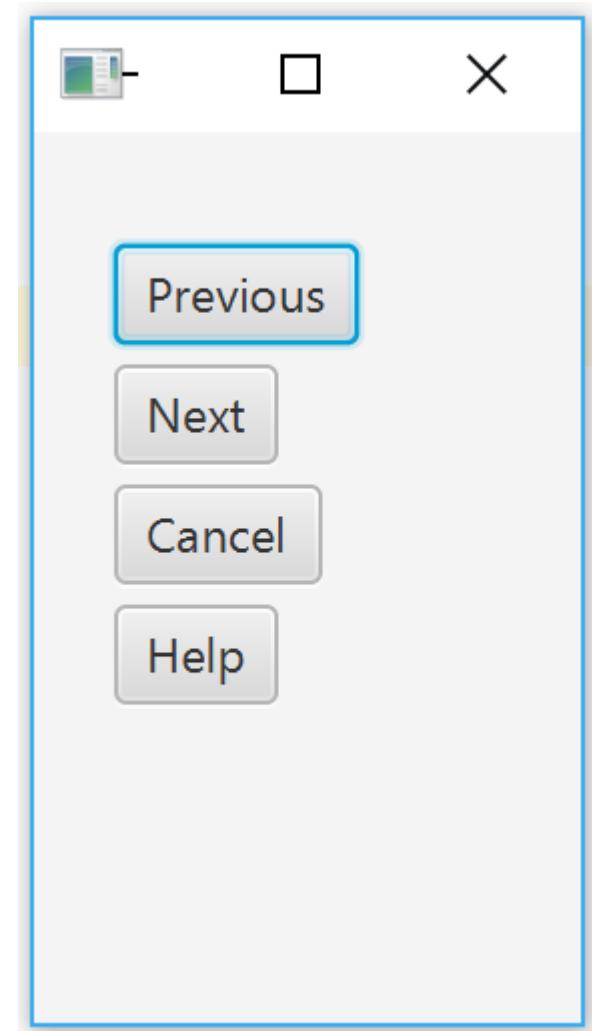
Gestionarea poziționării componentelor UI

- VBOX

```
VBox root = new VBox(5);
root.setPadding(new Insets(20));
root.setAlignment(Pos.BASELINE_LEFT);

Button prevBtn = new Button("Previous");
Button nextBtn = new Button("Next");
Button cancBtn = new Button("Cancel");
Button helpBtn = new Button("Help");

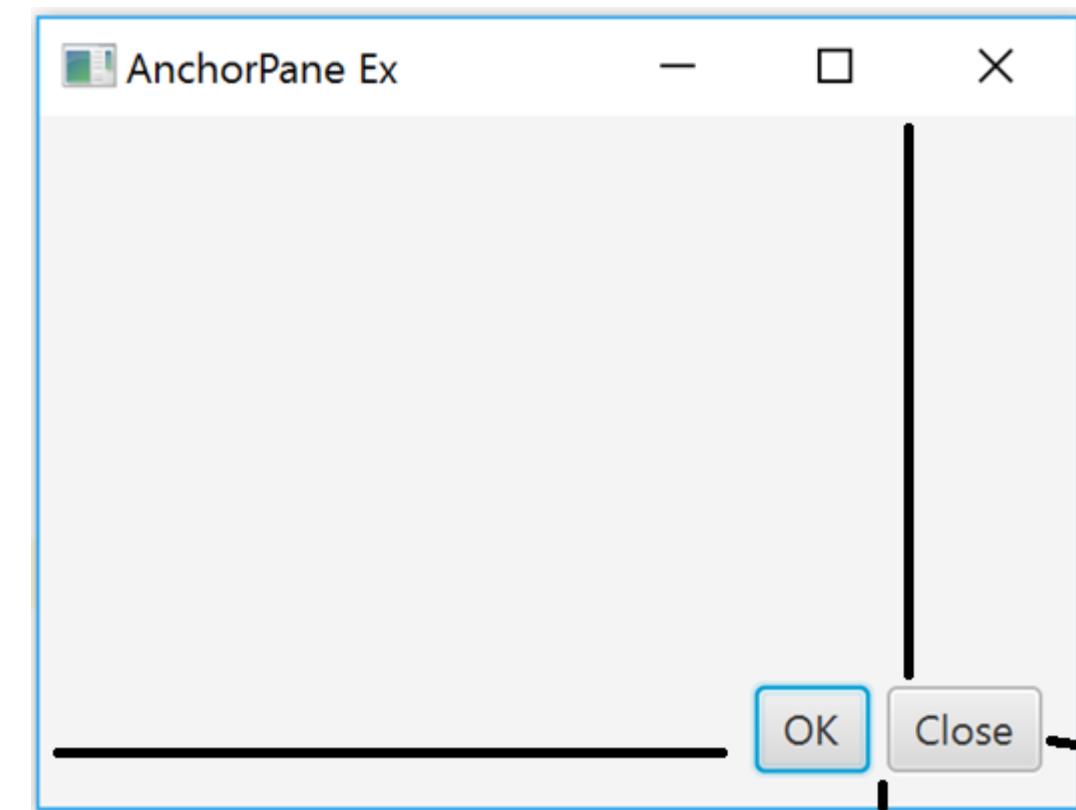
root.getChildren().addAll(prevBtn, nextBtn,
cancBtn, helpBtn);
Scene scene = new Scene(root, 150, 200);
```



Gestionarea poziționării componentelor UI

- AnchorPane

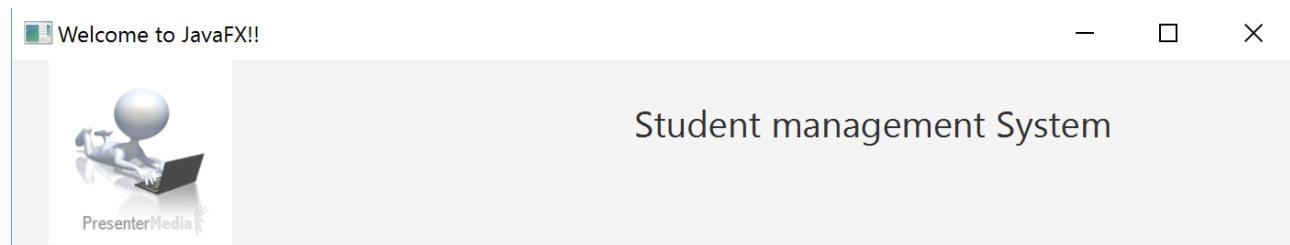
```
AnchorPane root = new AnchorPane();  
  
Button okBtn = new Button("OK");  
Button closeBtn = new Button("Close");  
HBox hbox = new HBox(5, okBtn, closeBtn);  
  
root.getChildren().addAll(hbox);  
  
AnchorPane.setRightAnchor(hbox, 10d);  
AnchorPane.setBottomAnchor(hbox, 10d);  
  
Scene scene = new Scene(root, 300, 200);  
  
stage.setTitle("AnchorPane Ex");  
stage.setScene(scene);  
stage.show();
```



Gestionarea pozitionarii componentelor UI

■ AnchorPane StudentView1.java Example

```
private Node initTop() {  
    AnchorPane anchorPane=new AnchorPane();  
  
    Label l=new Label("Student management System");  
    l.setFont(new Font(20));  
  
    AnchorPane.setTopAnchor(l,20d);  
    AnchorPane.setRightAnchor(l,100d);  
    anchorPane.getChildren().add(l);  
  
    Image img = new Image("logo.gif");  
    ImageView imgView = new ImageView(img);  
    imgView.setFitHeight(100);  
    imgView.setFitWidth(100);  
    imgView.setPreserveRatio(true);  
  
    AnchorPane.setLeftAnchor(imgView,20d);  
    AnchorPane.setRightAnchor(imgView,10d);  
    anchorPane.getChildren().add(imgView);  
  
    return anchorPane;  
}
```



Gestionarea pozitionarii componentelor UI

■ GridPane

```
GridPane gr=new GridPane();
gr.setPadding(new Insets(20));
gr.setAlignment(Pos.CENTER);

gr.add(createLabel("Username:"),0,0);
gr.add(createLabel("Password:"),0,1);

gr.add(new TextField(),1,0);
gr.add(new PasswordField(),1,1);

Scene scene = new Scene(gr, 300, 200);
stage.setTitle("Welcome to JavaFX!!!");
stage.setScene(scene);
stage.show();
```



Componente grafice de control - CGC

- Componentele grafice de control – elemente de bază ale unei aplicații cu interfață grafică utilizator.
- O componentă grafică de control este un nod în graful scenă
- CGC-urile pot fi manipulate de către un utilizator.
- Java FX Controls: https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm#JFXUI336

[Label](#)

[Button](#)

[Radio Button](#)

[Toggle Button](#)

[Checkbox](#)

[Choice Box](#)

[Text Field](#)

[Password Field](#)

[Scroll Bar](#)

[Scroll Pane](#)

[List View](#)

[Table View](#)

[Tree View](#)

[Combo Box](#)

[Separator](#)

[Slider](#)

[Progress Bar and Progress Indicator](#)

[Hyperlink](#)

[Tooltip](#)

[HTML Editor](#)

[Titled Pane and](#)

[Accordion](#)

[Menu](#)

[Color Picker](#)

[Pagination Control](#)

[File Chooser](#)

[Customization of UI](#)

[Controls](#)

Event Driven Programming

- **Eveniment:** Orice acțiune efectuată de utilizator generează un eveniment
 - apasarea sau eliberarea unei taste - de la tastatură,
 - deplasarea mouse-ului,
 - apăsarea sau eliberarea unui buton de mouse,
 - deschiderea sau închiderea unei ferestre,
 - efectuarea unui clic de mouse pe o componentă din interfață,
 - intrarea/părăsirea cursorului de mouse în zona unei componente, etc.).
- Există și evenimente care nu sunt generate de utilizatorul aplicației.
- **Un eveniment poate să fie tratat prin execuția unui modul de program.**

Tratarea evenimentelor - Delegation Event Model.

- Distingem trei categorii de obiecte utilizate la tratarea evenimentelor:
 - **surse de evenimente (Event Source)** - acele obiecte care generează evenimente;
 - **evenimentele propriu-zise (Event)**, care sunt tot obiecte (generate de surse și recepționate de consumatori).
 - **consumatori sau ascultători de evenimente** - acele obiecte care recepționează și tratează evenimentele.

Tratarea evenimentelor



- Fiecare consumator trebuie să fie înregistrat la sursa de eveniment.
- **Modelul "delegării"** presupune că sursa (un obiect) transmite evenimentele generate de ea către obiectele consumatori, care s-au înregistrat la sursa respectiva a evenimentului.
- Un obiect consumator recepționează evenimente numai de la obiectele sursă la care s-a înregistrat!!!

Tratarea evenimentelor - event handler

```
@FunctionalInterface  
Interface EventHandler<T extends Event> extends EventListener{  
    void handle(T event);  
}
```

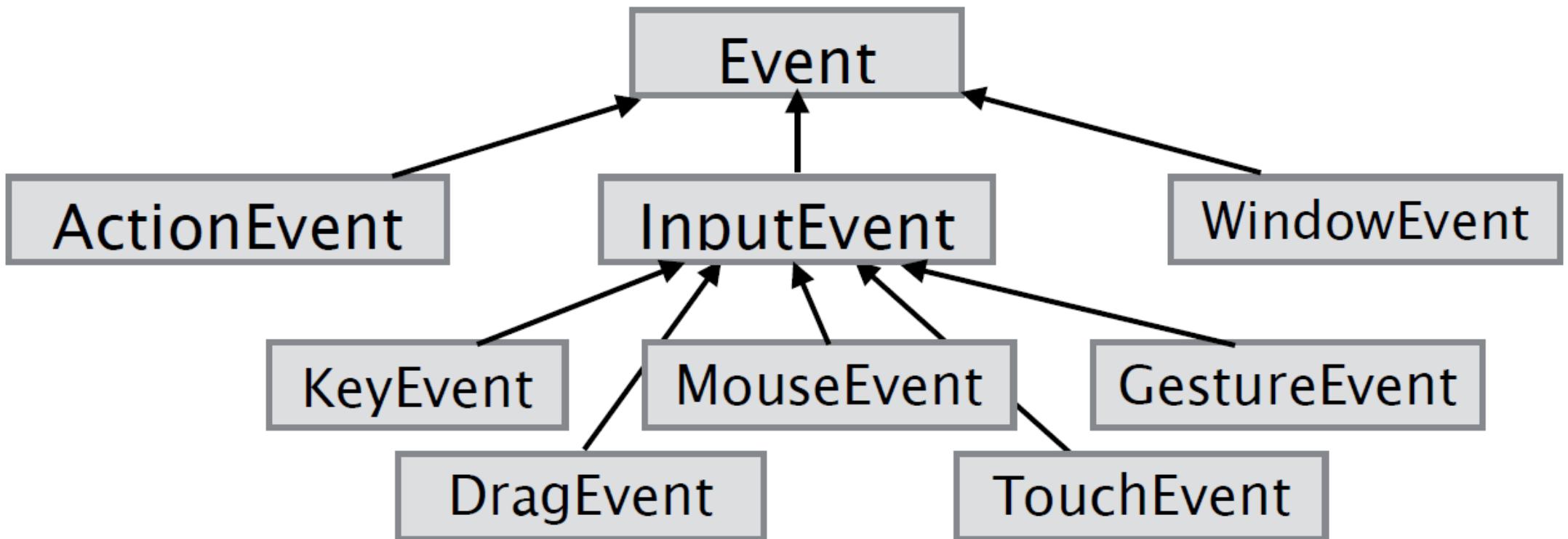
Tratarea evenimentului click pe buton

```
Button btn = new Button("Ding!");  
  
// handle the button clicked event  
btn.setOnAction(new EventHandler<ActionEvent>() {  
    public void handle(ActionEvent e) {  
        Toolkit.getDefaultToolkit().beep();  
    }  
});  
  
btn.setOnAction(e->Toolkit.getDefaultToolkit().beep());
```

Ce sablon de proiectare este folosit?

Se poate asocia o singura metoda handler evenimentului click pe buton!!!

Tipuri de evenimente



Comparatie cu JButton - Java Swing

- Ascultători de evenimente (listener)

```
JButton b = new JButton("Click me!"); //Java Swing
b.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent e) {
        System.out.println("listener1");
    }
});

b.addActionListener(e-> System.out.println("listener2"));
```

ObservableValue<T>

- Interfata generica ObservableValue<T> este utilizata pentru a încapsula diverse tipuri de valori și a asigura un mecanism de schimbare a acestora prin notificari.

```
public interface ObservableValue<T> extends Observable;
```

- *Metode:*

```
T getValue(); //furnizeaza valoarea acoperita
```

```
void addListener(ChangeListener<? super T> listener);
```

```
void removeListener(ChangeListener<? super T> listener); // furnizare mecanism  
de inregistrare/stergere ascultatori
```

- *Exemple de implementari:*

```
public class SimpleStringProperty extends StringPropertyBase;
```

```
public class SimpleObjectProperty<T> extends ObjectPropertyBase<T>;
```

```
public class SimpleDoubleProperty extends DoublePropertyBase;
```

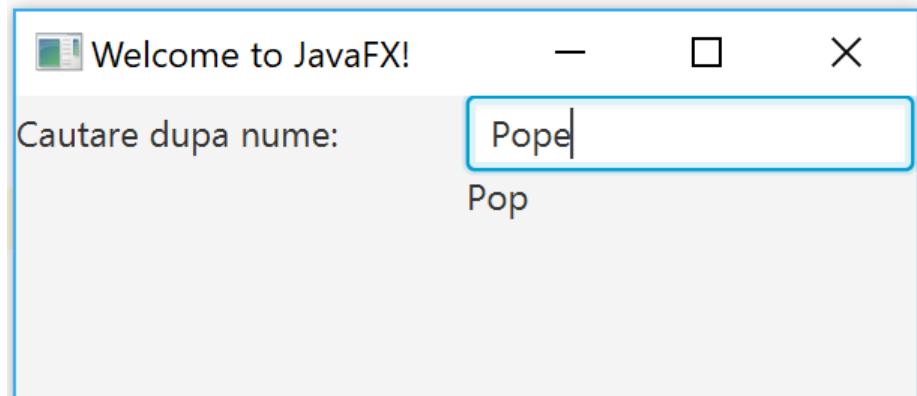
Property - Observable - Listener

```
BooleanProperty booleanProperty = new SimpleBooleanProperty(true);
// Add change listener
booleanProperty.addListener(new ChangeListener<Boolean>() {
    @Override
    public void changed(ObservableValue<? extends Boolean> observable,
                        Boolean oldValue, Boolean newValue) {
        System.out.println("changed " + oldValue + "->" + newValue);
        //myFunc();
    }
});
Button btn = new Button();
btn.setText("Switch boolean flag");
btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        booleanProperty.set(!booleanProperty.get()); //switch
        System.out.println("Switch to " + booleanProperty.get());
    }
});
// Bind to another property variable
btn.underlineProperty().bind(booleanProperty);
```

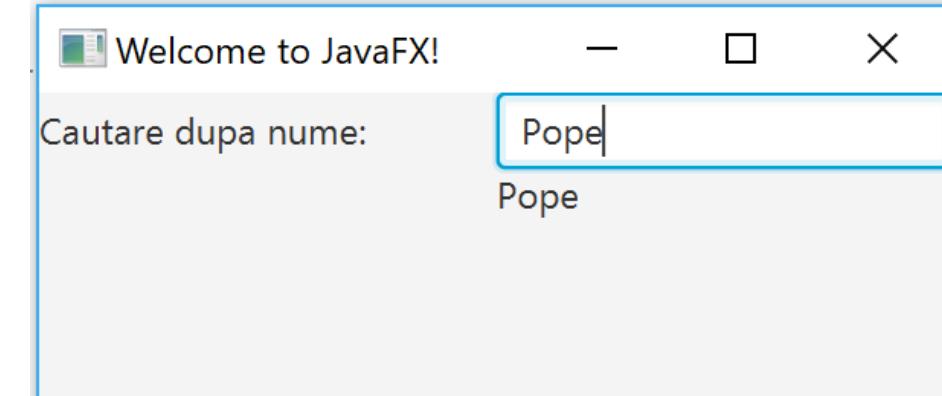
Se pot adauga oricati ascultatori!
(Design pattern: ???)

TextField- events

```
TextField txt=new TextField();  
  
txt.setOnKeyPressed(new  
EventHandler<KeyEvent>() {  
    @Override  
    public void handle(KeyEvent event) {  
        l.setText(txt.getText());  
    }  
});
```

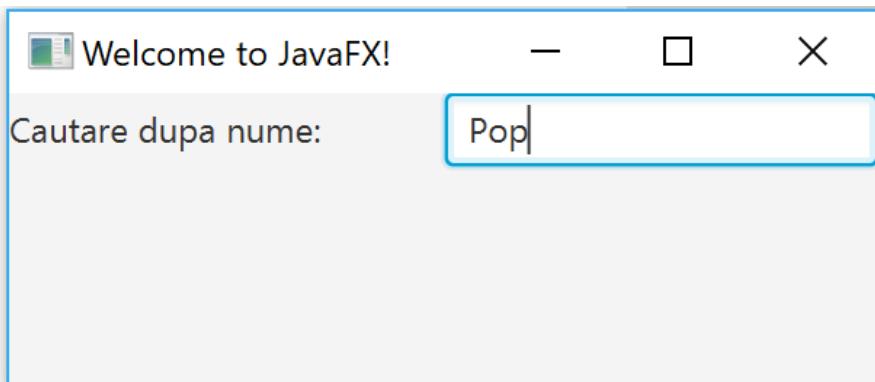


```
txt.textProperty().addListener(new  
ChangeListener<String>() {  
    @Override  
    public void changed(ObservableValue<?  
extends String> observable, String oldValue,  
String newValue) {  
    l.setText(newValue);  
}  
});
```

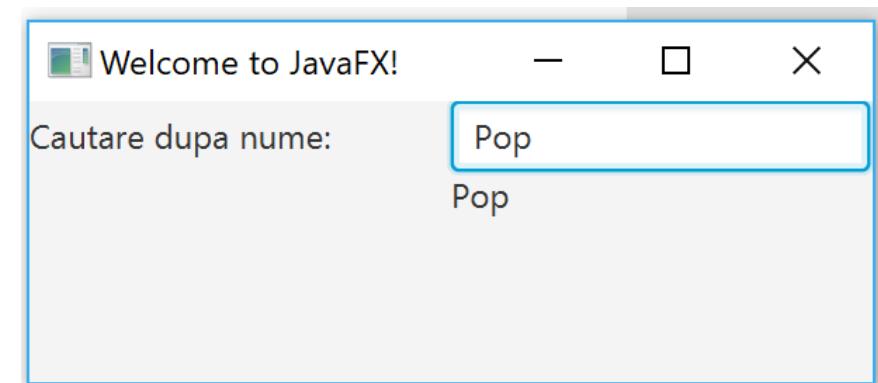


TextField- events

```
//Handle TextField enter key event.  
txt.setOnAction(new EventHandler<ActionEvent>() {  
    @Override  
    public void handle(ActionEvent event) {  
        l.setText(txt.getText());  
    }  
});
```



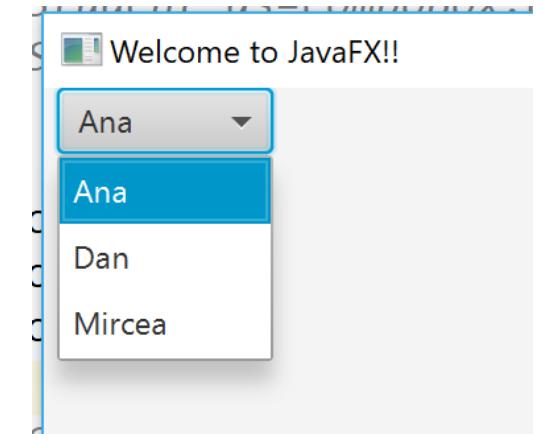
Enter ->



Combobox

```
ComboBox<String> comboBox2=new ComboBox<>();  
comboBox2.getItems().addAll("Ana", "Dan", "Mircea");  
comboBox2.getSelectionModel().selectFirst();
```

```
//Listen to selectedItemProperty changes  
comboBox2.getSelectionModel().selectedItemProperty().addListener(  
new ChangeListener<String>() {  
    @Override  
    public void changed(ObservableValue<? extends String>  
observable, String oldValue, String newValue)  
    {  
        System.out.println(oldValue);  
    }  
});
```



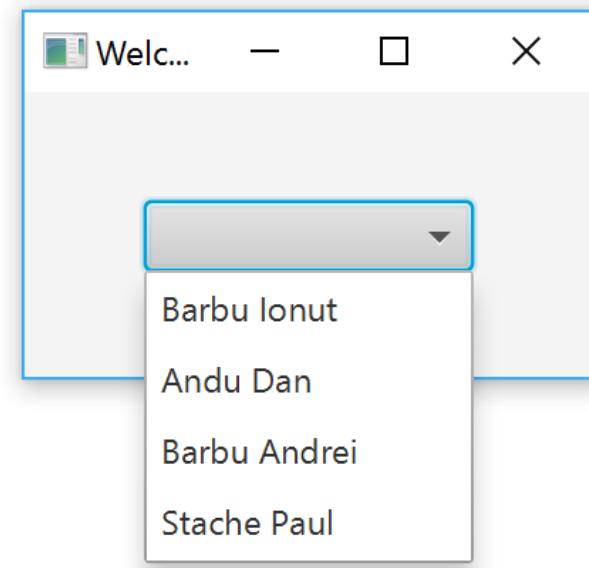
ComboBox with data object – handle events

Initializing the ComboBox

```
ComboBox<Student> comboBox=new ComboBox<Student>();  
ObservableList<Student>  
s=FXCollections.observableArrayList(getStudList());  
comboBox.setItems(s);
```

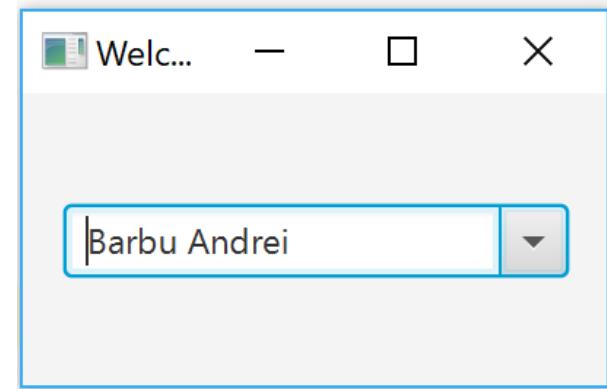
ComboBox Rendering

```
// Define rendering of the list of values in ComboBox drop down.  
comboBox.setCellFactory(new Callback<ListView<Student>, ListCell<Student>>() {  
    @Override  
    public ListCell<Student> call(ListView<Student> param) {  
        return new ListCell<Student>(){  
            @Override  
            protected void updateItem(Student item, boolean empty) {  
                super.updateItem(item, empty);  
  
                if (item == null || empty) {  
                    setText(null);  
                } else {  
                    setText(item.getFirstName() + " " + item.getLastName())  
                }  
            }  
        };  
    } );
```



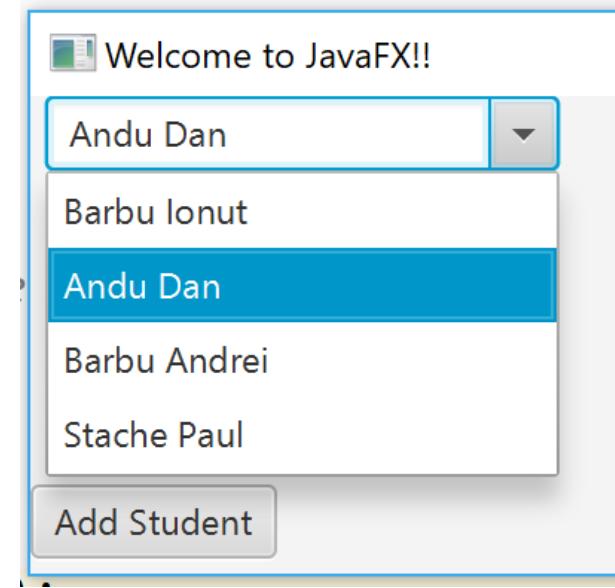
ComboBox handle events

```
// Define rendering of selected value shown in ComboBox.  
comboBox.setConverter(new StringConverter<Student>() {  
    @Override  
    public String toString(Student s) {  
        if (s == null) {  
            return null;  
        } else {  
            return s.getFirstName() + " " + s.getLastName();  
        }  
    }  
  
    @Override  
    public Student fromString(String studentString) {  
        return null; // No conversion fromString needed.  
    }  
});
```



ComboBox handle events

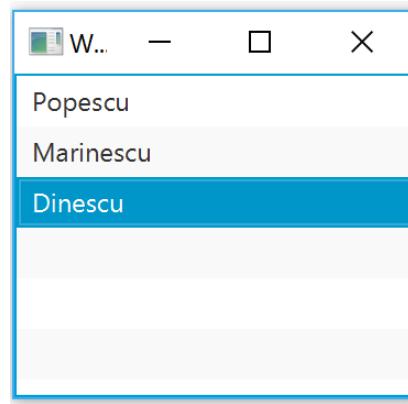
```
//handle selection event
comboBox.setOnAction(ev->{
    Student as=comboBox.getSelectionModel().getSelectedItem();
    System.out.println(as.toString());
});
```



```
//listen to selectedItemProperty changes
comboBox.getSelectionModel().selectedItemProperty().addListener(new
ChangeListener<Student>() {
    @Override
    public void changed(ObservableValue<? extends Student>
observable, Student oldValue, Student newValue) {
        System.out.println(newValue.toString());
    }
});
```

ListView

```
ListView<String> lview=new ListView<>(FXCollections.observableArrayList());
lview.getItems().addAll("Popescu", "Marinescu", "Dinescu");
```



```
List<String> l=Arrays.asList("Popescu", "Marinescu", "Dinescu"));
ObservableList<String>
observableList=FXCollections.observableArrayList(l);
lview.setItems(observableList);
```

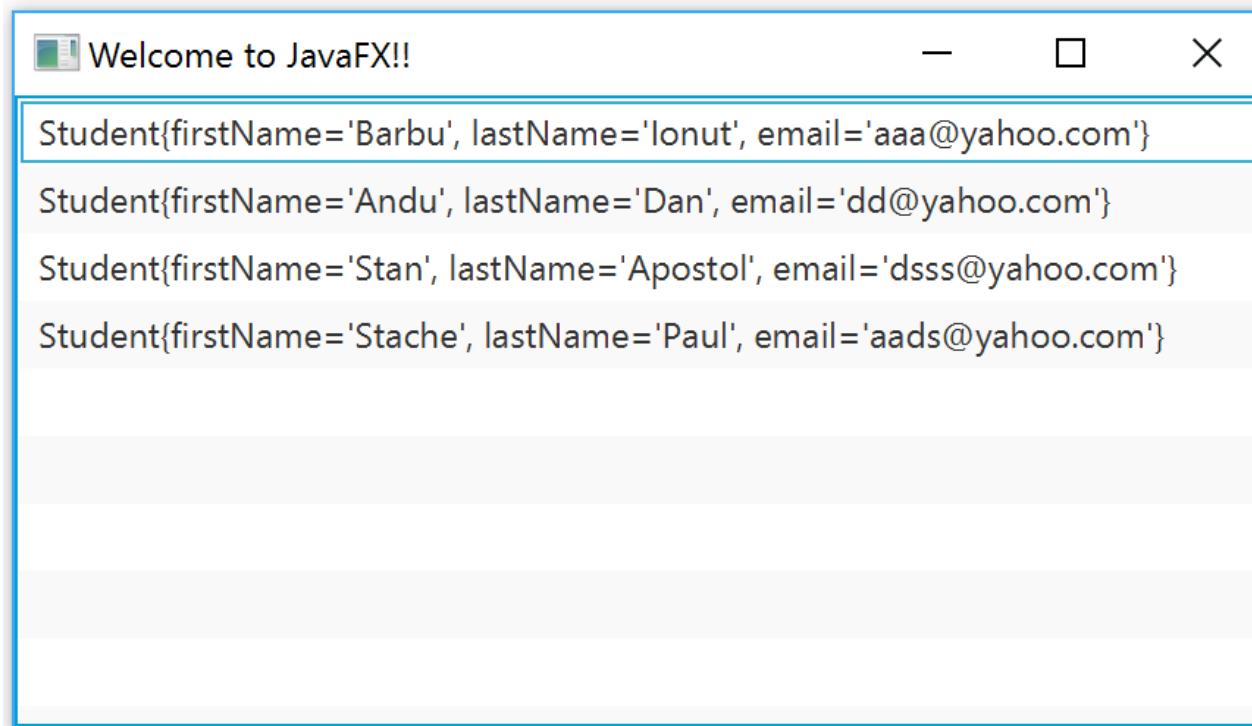
ListView

- Asemantor cu Combobox, dar ListView nu are ActionEvent, in schimb are selectedItemProperty

```
myListView.getSelectionModel().selectedItemProperty().addListener((observable, oldValue,  
newValue) -> {  
    System.out.println("ListView Selection Changed (selected: " + newValue.toString() + ")");  
});
```

ListView for custom object

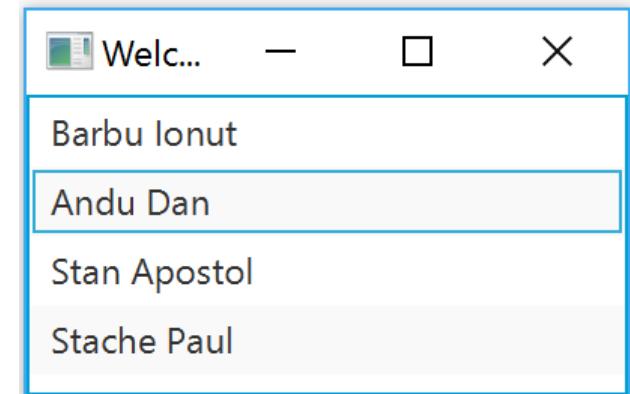
```
ListView<Student> listView=new ListView<>(students);
```



ListView for custom object

cellFactory method

```
ListView<Student> listView=new ListView<>(students);
```



```
//cellFactory method
listView.setCellFactory(new Callback<ListView<Student>, ListCell<Student>>()
{
    @Override
    public ListCell<Student> call(ListView<Student> param) {
        ListCell<Student> listCell=new ListCell<Student>(){
            // how to update text in this cell?
        };
        return listCell;
    }
});
```

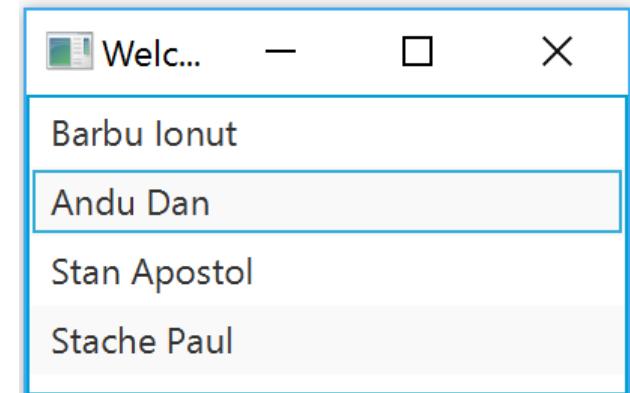
ListView for custom object

cellFactory method

```
ListView<Student> listView=new ListView<>(students);
```

Override updateItem() method from ListCell

```
//rendering data
listView.setCellFactory(list -> new ListCell<Student>(){
    @Override
    protected void updateItem(Student item, boolean empty) {
        super.updateItem(item, empty);
        if (item == null || empty) {
            setText(null);
        } else {
            setText(item.getFirstName() + " " + item.getLastName());
        }
    }
});
```



ListView add new value

```
private ObservableList<Student> studs= FXCollections.observableArrayList();
■ ListView<Student> list=new ListView<>();
list.setItems(studs);

■ studs.add(new Student("45","andrei","nistor","gdhgh"));
```

ListView selection

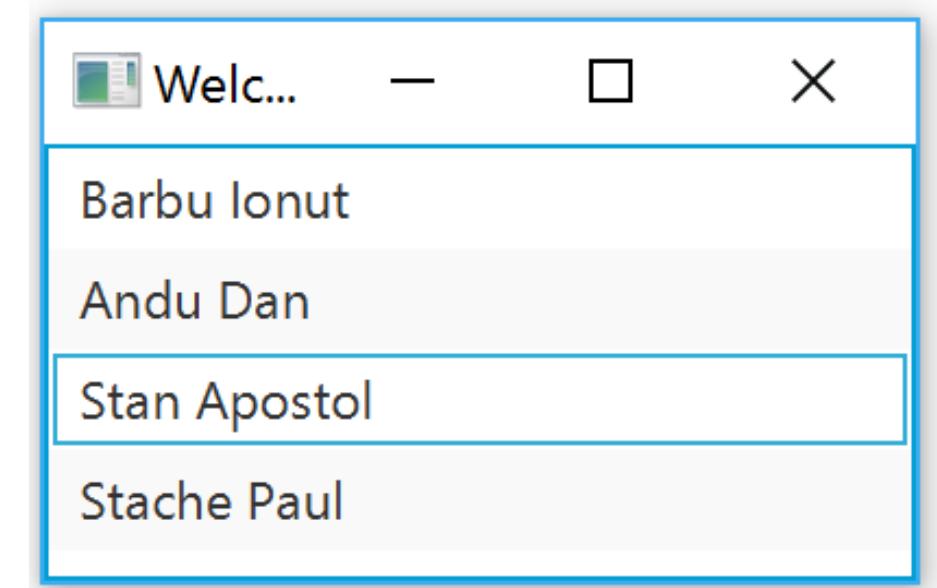
```
//itemul selectat
Student s=listView.getSelectionModel().getSelectedItem();

listView.getSelectionModel().selectedItemProperty().addListener(new ChangeListener<Student>()
{
    @Override
    public void changed(ObservableValue<? extends Student> observable, Student oldValue,
    Student newValue) {
        System.out.println(newValue.toString());
    }
});
```

ListView set focus

```
ListView<Student> listView=new ListView<>(students);
```

```
listView.getFocusModel().focus(2);
```



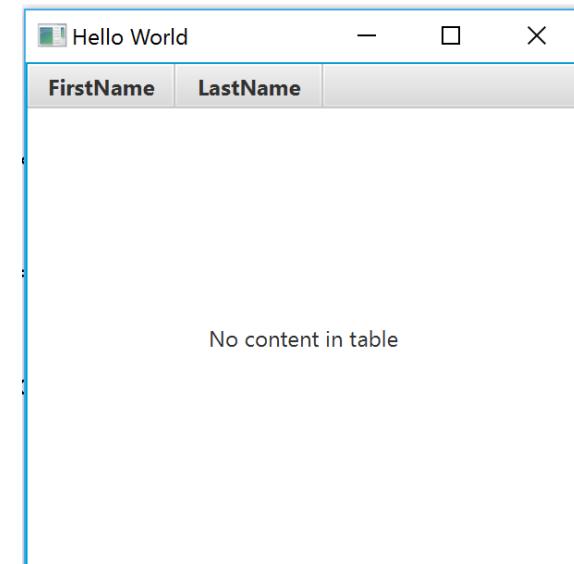
TableView

- Creare

```
TableView<Student> tableView=new TableView<Student>();
```

```
TableColumn<Student, String> columnName=new TableColumn<>("FirstName");  
TableColumn<Student, String> columnLastName=new TableColumn<>("LastName");
```

```
tableView.getColumns().addAll(columnName, columnLastName);
```



TableView

- Binding data

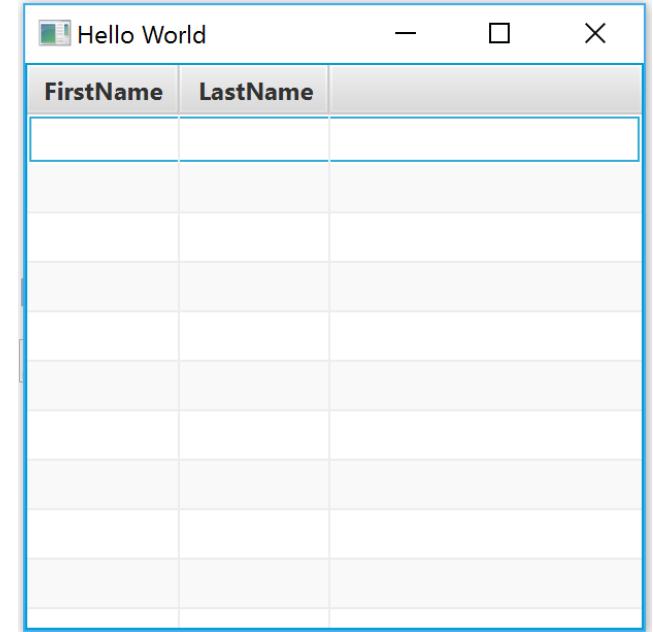
```
List<Student> l=new ArrayList<Student>();  
l.add(new Student("Barbu","Ionut","aaa@yahoo.com"));  
l.add(new Student("Andu","Dan","dd@yahoo.com"));  
l.add(new Student("Stan","Apostol","dsss@yahoo.com"));  
l.add(new Student("Stache","Paul","aads@yahoo.com"));
```

```
ObservableList<Student> students = FXCollections.observableArrayList(l);
```

```
TableView<Student> tableView=new TableView<Student>();
```

```
TableColumn<Student, String> columnName=new TableColumn<>("FirstName");  
TableColumn<Student, String> columnLastName=new TableColumn<>("LastName");  
tableView.getColumns().addAll(columnName,columnLastName);
```

```
tableView.setItems(students);
```



The screenshot shows a JavaFX application window titled "Hello World". The window contains a TableView with two columns: "FirstName" and "LastName". The table has one row, which is currently empty. The entire application window is enclosed in a blue border.

FirstName	LastName

TableView

- `setCellValueFactory` method

```
columnName.setCellValueFactory(new PropertyValueFactory<Student, String>("firstName"));  
columnLastName.setCellValueFactory(new PropertyValueFactory<Student, String>("lastName"));
```

FirstName	LastName
Barbu	Ionut
Andu	Dan
Barbu	Andrei
Stache	Paul

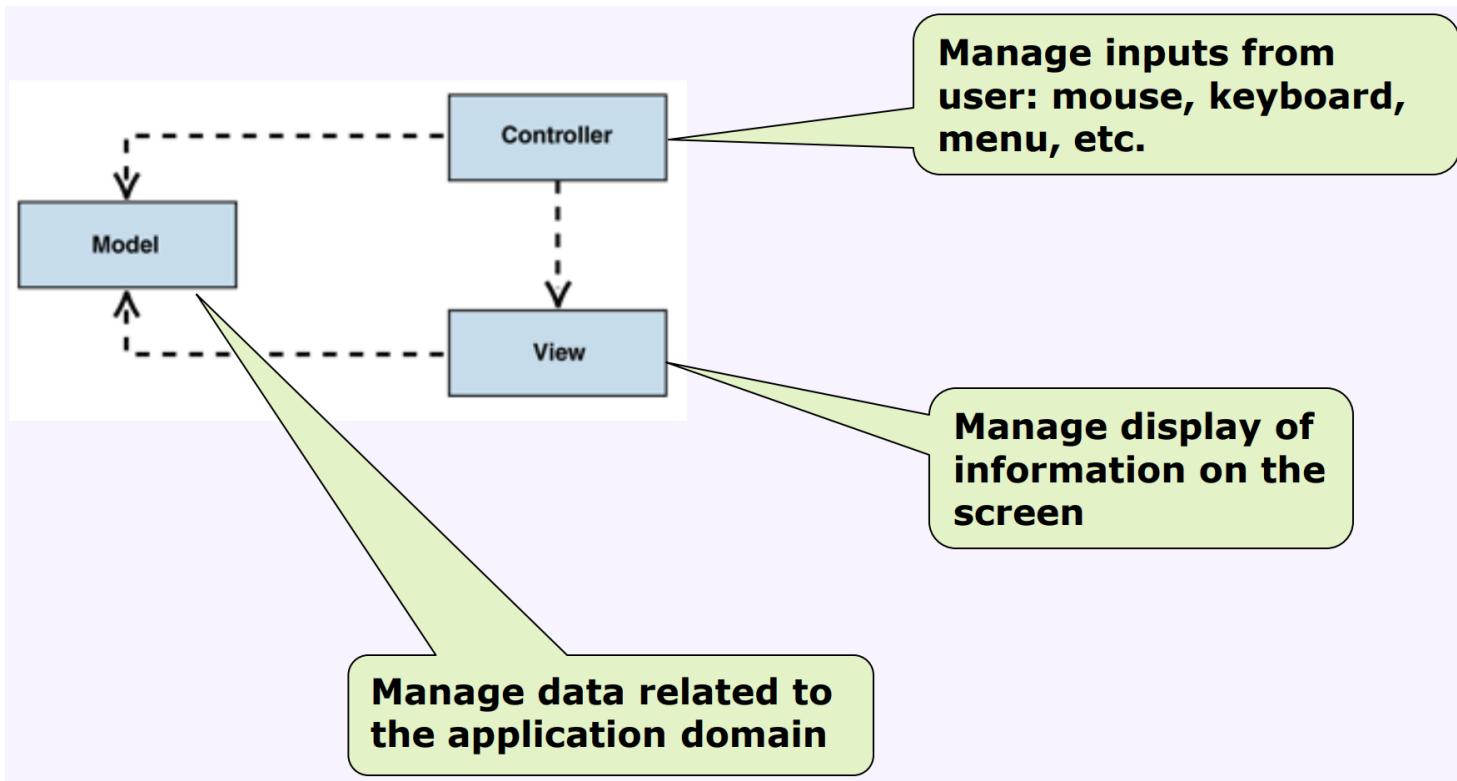
TableView

- Listen for table selection changes

```
tableView.getSelectionModel().selectedItemProperty().addListener(new  
ChangeListener<Student>() {  
    @Override  
    public void changed(ObservableValue<? extends Student> observable, Student  
oldValue, Student newValue) {  
        System.out.println("A fost selectat"+ newValue.toString());  
    }  
});
```

Model View Controller (MVC)

- JavaFX este dezvoltata dupa filozofia **Model View Controller** (MVC) separand partea de logica de partea de vizualizare si manipulare.



FXML

- FXML este un limbaj declarativ de adnotare bazat pe XML prin intermediul căruia pot fi dezvoltate interfețe grafice cu utilizatorul, fără a fi necesar ca aplicația să fie recompilată de fiecare dată când sunt modificate elemente din cadrul acesteia.
- În acest mod se realizează o separare între nivelul de prezentare și nivelul de logică a aplicației.
- SceneBuilder permite construirea interfeței în mod vizual, generând automat și documentul FXML asociat, acesta putând fi integrat apoi în orice mediu de dezvoltare.
- Astfel, nu mai este necesară decât implementarea mecanismelor de tratare a evenimentelor corespunzătoare diferitelor controale (elemente din cadrul interfeței grafice);

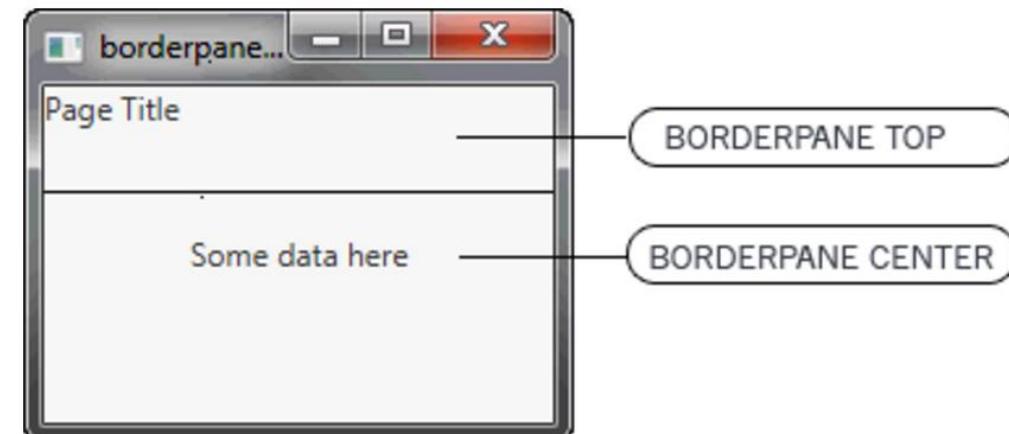
Programmatic vs. Declarativ

- *Programmatic*

```
BorderPane border = new BorderPane();
Label top = new Label("Page Title");
border.setTop(top);
Label center = new Label ("Some data here");
border.setCenter(center);
```

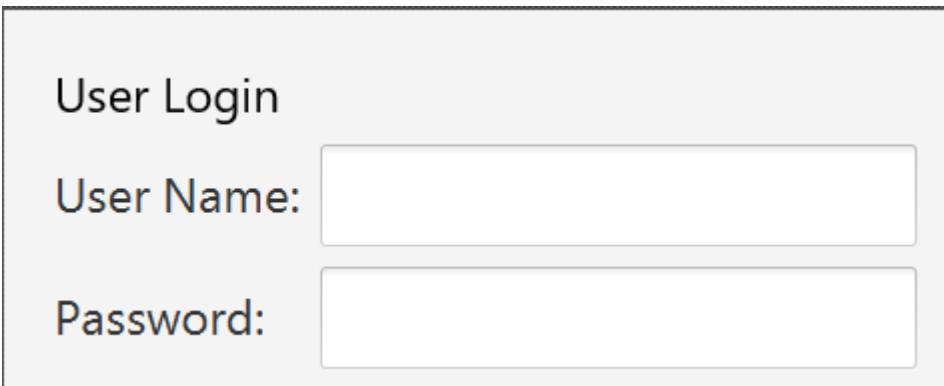
- *Declarativ*

```
<BorderPane>
    <top>
        <Label text="Page Title"/>
    </top>
    <center>
        <Label text="Some data here"/>
    </center>
</BorderPane>
```



View definit ca fișier FXML

- *Exemplu fereastra de autentificare (login)*



Definim un GridPane pe care
il vom adăuga unui
AnchorPane

Exemplu Login FXML

- Exemplu fereastra de autentificare (login)

```
<GridPane xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10">
    <padding><Insets top="25" right="25" bottom="10" left="25"/></padding>

    <Text text="User Login "
        GridPane.columnIndex="0" GridPane.rowIndex="0"
        GridPane.columnSpan="2"/>

    <Label text="User Name:"
        GridPane.columnIndex="0" GridPane.rowIndex="1"/>

    <TextField
        GridPane.columnIndex="1" GridPane.rowIndex="1"/>

    <Label text="Password:"
        GridPane.columnIndex="0" GridPane.rowIndex="2"/>

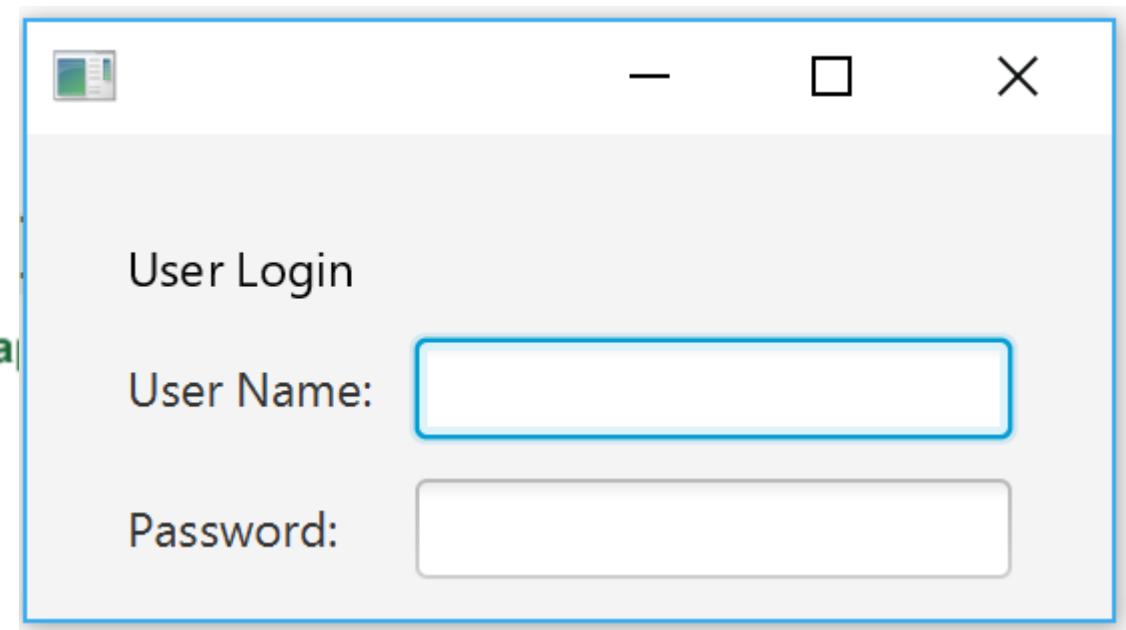
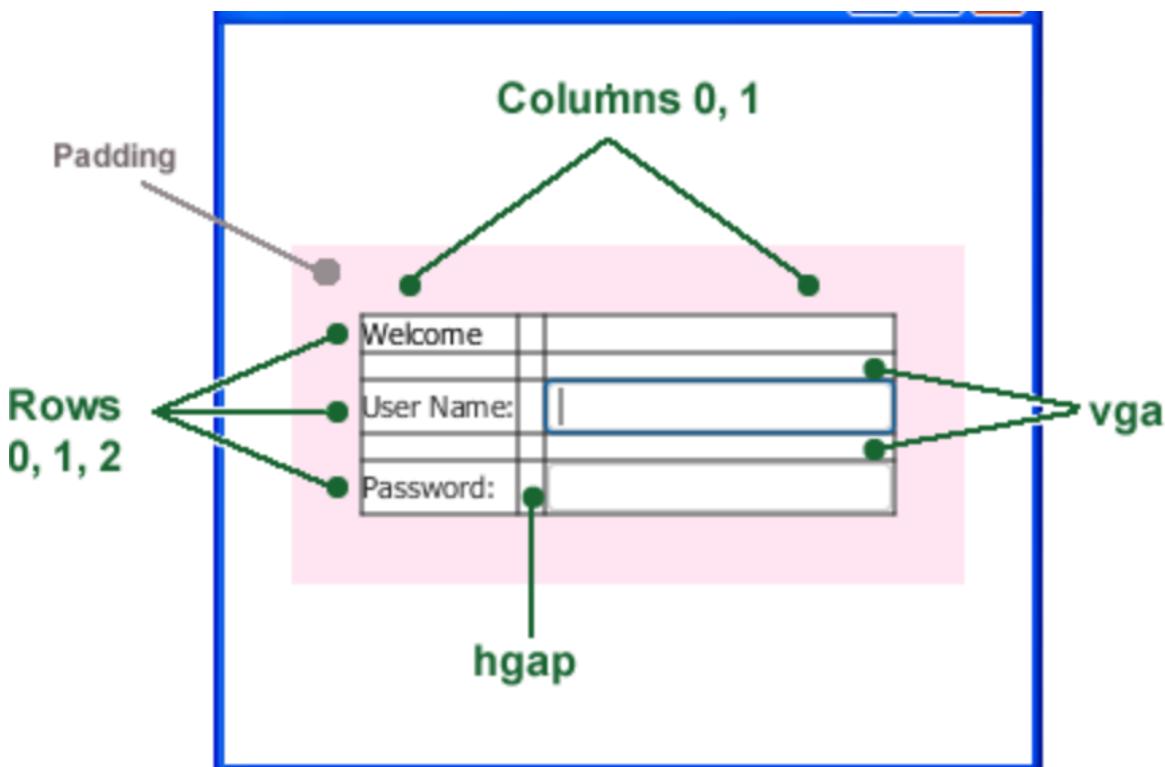
    <PasswordField GridPane.columnIndex="1" GridPane.rowIndex="2"/>
</GridPane>
```

```
//cod JavaFX
GridPane gr=new GridPane();
//alignment="center" hgap="10" vgap="10"
gr.setAlignment(Pos.CENTER);
gr.setHgap(10);
gr.setVgap(10);
Text t=new Text("User Login ");
gr.add(t,0,0);

Label l=new Label("User Login ");
gr.add(l,0,1);
```

Exemplu Login FXML

```
<padding><Insets top="25" right="25" bottom="10" left="25"/></padding>
```



Exemplu Login FXML

- *Adaugare Buton*

```
<HBox spacing="10" alignment="bottom_right"
       GridPane.columnIndex="1" GridPane.rowIndex="4">
    <Button text="Sign In"/>
</HBox>
```

FXML Loader

```
public class Main extends Application {  
    public static void main(String[] args) {  
        Launch(args);  
    }  
    @Override  
    public void start(Stage primaryStage) {  
        try {  
            //Load root layout from fxml file.  
            FXMLLoader loader=new FXMLLoader();  
            loader.setLocation(getClass().getResource("LoginExample.fxml")); //URL  
            GridPane rootLayout= (GridPane) loader.load();  
  
            // Show the scene containing the root layout.  
            Scene scene = new Scene(rootLayout);  
            primaryStage.setScene(scene);  
            primaryStage.show();  
  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

FXML - Controller

```
<GridPane fx:controller="Exemplu.LoginExampleController"  
xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10">
```

- În fisierul XXX.fxml ne definim view-l
- Actiunile utilizator (evenimentele) le tratam într-un fisier Controller
- Cum?
 - Definim un fisier java, de exemplu cu numele XXXController.java
 - Specificăm legătura cu fisierul XXX.fxml:


```
<GridPane fx:controller="Exemplu.LoginExampleController">
```
 - Definim metode handle pentru tratarea evenimentelor

Obtinerea unui obiect de tip controller

```
public class Main1 extends Application {  
    public static void main(String[] args) {  
        Launch(args);  
    }  
    @Override  
    public void start(Stage primaryStage) {  
        try {  
            //Load root layout from fxml file.  
            FXMLLoader loader=new FXMLLoader();  
            loader.setLocation(getClass().getResource("LoginExample.fxml")); //URL  
            GridPane rootLayout= (GridPane) loader.load();  
            LoginExampleController controller=loader.getController();  
  
            // Show the scene containing the root layout.  
            Scene scene = new Scene(rootLayout);  
            primaryStage.setScene(scene);  
            primaryStage.show();  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Tratarea evenimentelor

- *Handle Event via Controller class*

```
<HBox spacing="10" alignment="bottom_right" GridPane.columnIndex="1" GridPane.rowIndex="4">
    <Button text="Sign In" onAction="#handleSubmitButtonAction"/>
</HBox>
<Text GridPane.columnIndex="1" GridPane.rowIndex="6"/>
```

```
public class LoginExampleController {

    @FXML
    public void handleSubmitButtonAction(ActionEvent actionEvent) {
        System.out.println("Login button was pressed!");
    }
}
```

FXML – Controller initialize

```
public class LoginExampleController {  
    /**  
     * Initializes the controller class. This method is automatically called  
     * after the fxml file has been loaded.  
     */  
    @FXML  
    public void initialize() {  
    }  
}
```

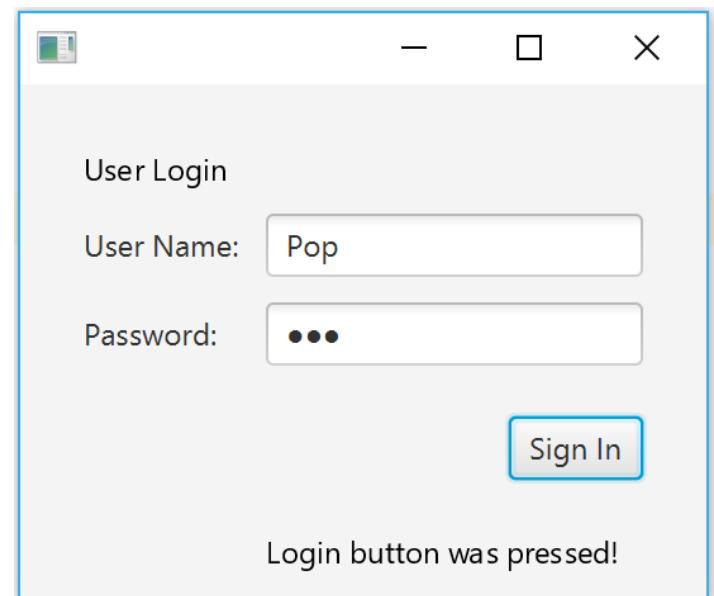
Adnotarea FXML a elementelor din view

```
<TextField fx:id="usernameField"
           GridPane.columnIndex="1" GridPane.rowIndex="1"/>

<PasswordField fx:id="passwordField" GridPane.columnIndex="1" GridPane.rowIndex="2"/>

<HBox spacing="10" alignment="bottom_right" GridPane.columnIndex="1" GridPane.rowIndex="4">
    <Button text="Sign In" onAction="#handleSubmitButtonAction"/>
</HBox>
<Text fx:id="textResponse" GridPane.columnIndex="1" GridPane.rowIndex="6"/>
```

```
public class LoginExampleController {
    @FXML
    private Text textResponse;
    @FXML
    private TextField usernameField;
    @FXML
    private PasswordField passwordField;
    @FXML
    public void handleSubmitButtonAction(ActionEvent actionEvent) {
        textResponse.setText("Login button was pressed!");
        User u=new User(usernameField.getText(),passwordField.getText());
    }
}
```



CSS

```
<GridPane stylesheets="@login.css" fx:controller="Exemplu.LoginExampleController"
          xmlns:fx="http://javafx.com/fxml" alignment="center" hgap="10" vgap="10">

    .root {
        -fx-background-image: url("logo.gif");
    }
    .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 );
    }
    .label {
        -fx-font-size: 12px;
        -fx-font-weight: bold;
        -fx-text-fill: #2A5058;
        -fx-effect: dropshadow( gaussian , rgba(214, 66, 20, 0.5), 0,0,0,1 );
    }
    #logintext{
        -fx-font-size: 32px;
        -fx-font-family: "Arial Black";
        -fx-fill: #2A5058;
    }
    #textResponse {
        -fx-fill: FIREBRICK;
        -fx-font-weight: bold;
        -fx-effect: dropshadow( gaussian , rgba(255,255,255,0.5) , 0,0,0,1 );
    }
}
```

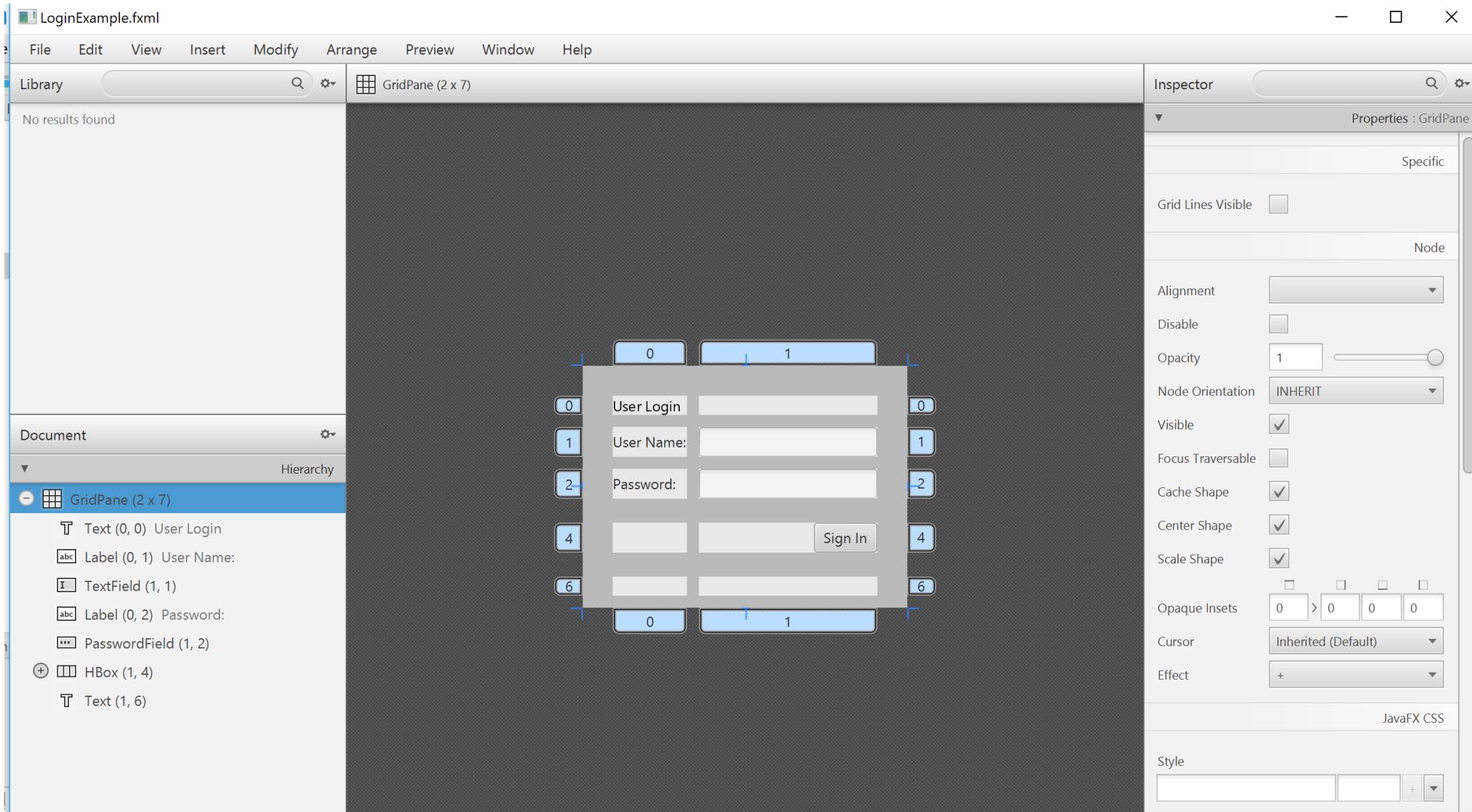
Login.css file

<http://www.w3schools.com/css/>

FXML and Scene Builder

<http://www.oracle.com/technetwork/java/javase/downloads/sb2download-2177776.html>

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

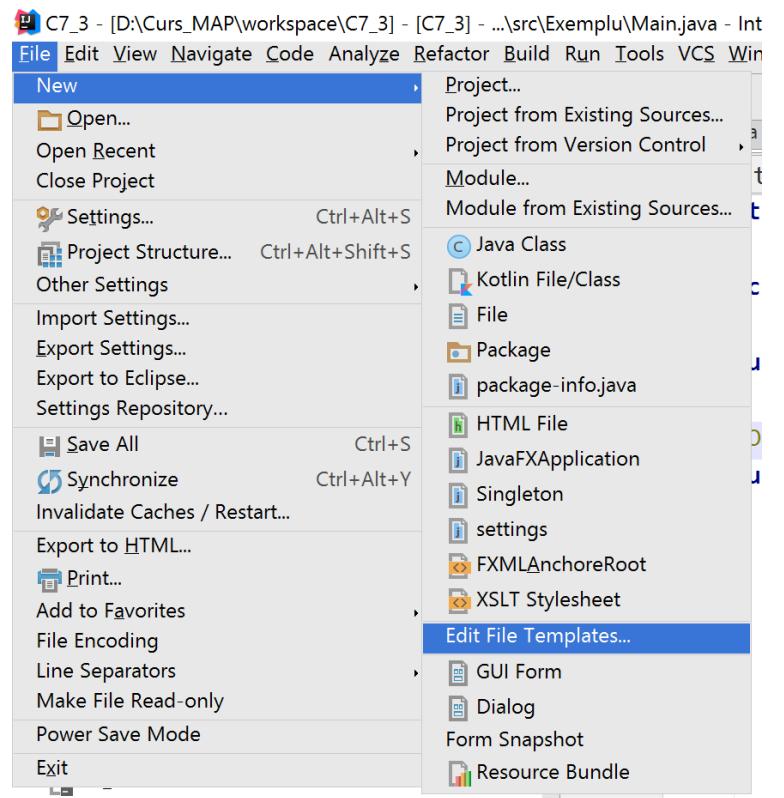
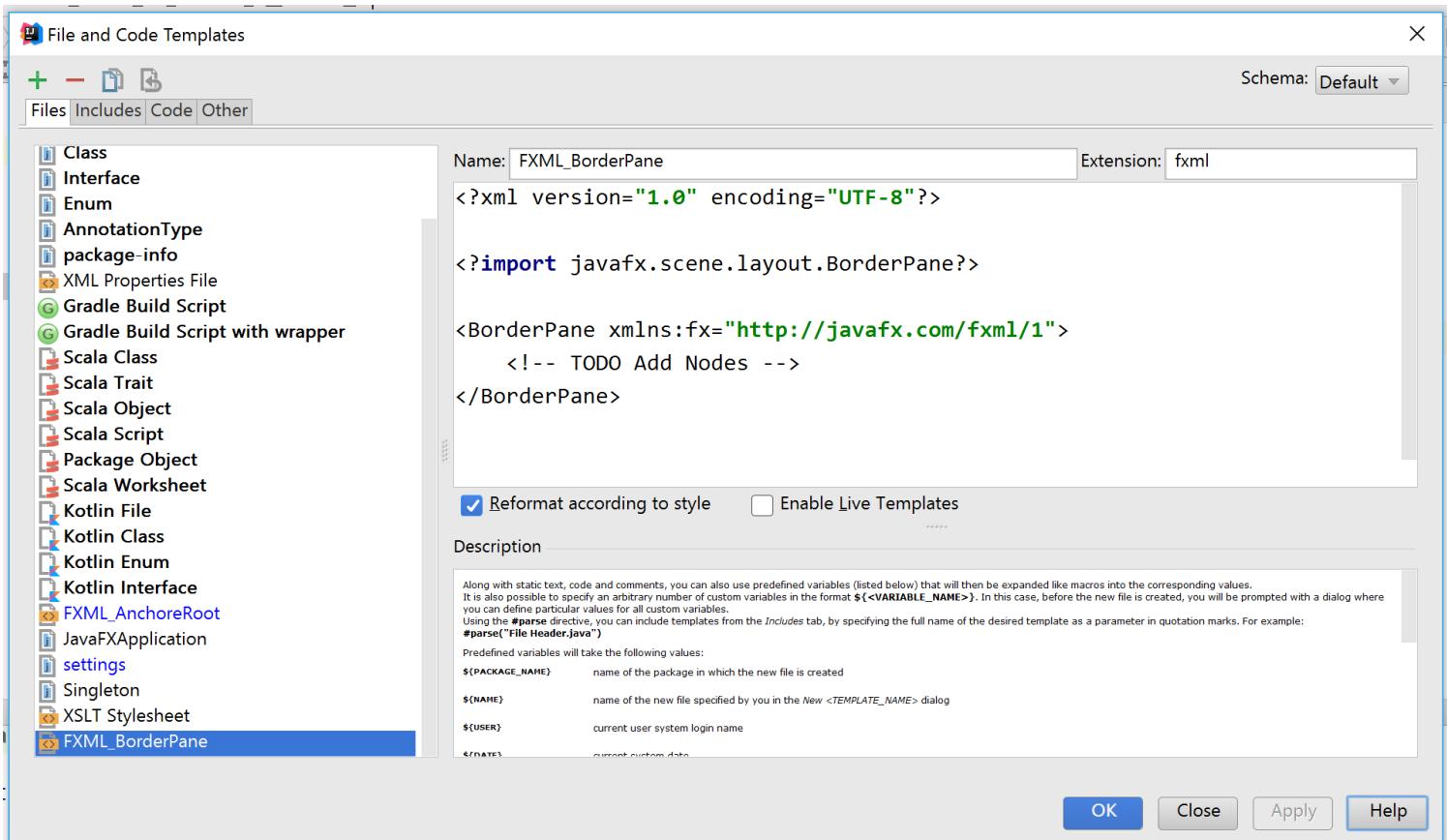


Scene Builder

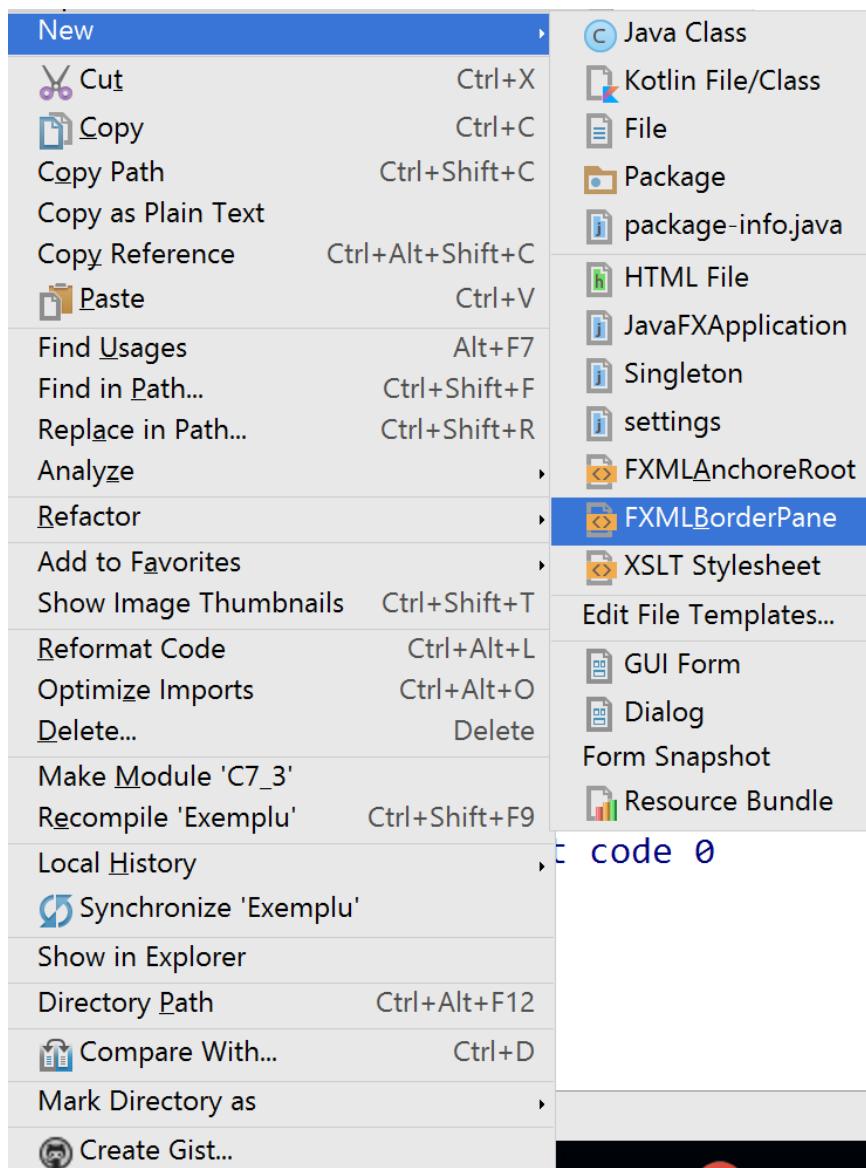
- Specifying the path to the JavaFX Scene Builder executable:
- In Eclipse:
 - Window -> Preferences ->Scene Builder
- In IntelliJ:
 - File->Settings-Languages and Frameworks->Java FX
- Scene Builder download:
 - http://docs.oracle.com/javafx/scenebuilder/1/use_java_ides/sb-with-eclipse.htm
 - <http://gluonhq.com/labs/scene-builder/#download>

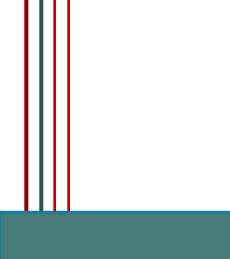
FXML File templates

- In Eclipse există predefinite
- In IntelliJ definim noi



FXML File templates





Sabloanele folosite:

Observer, Command

Şablonul Observer

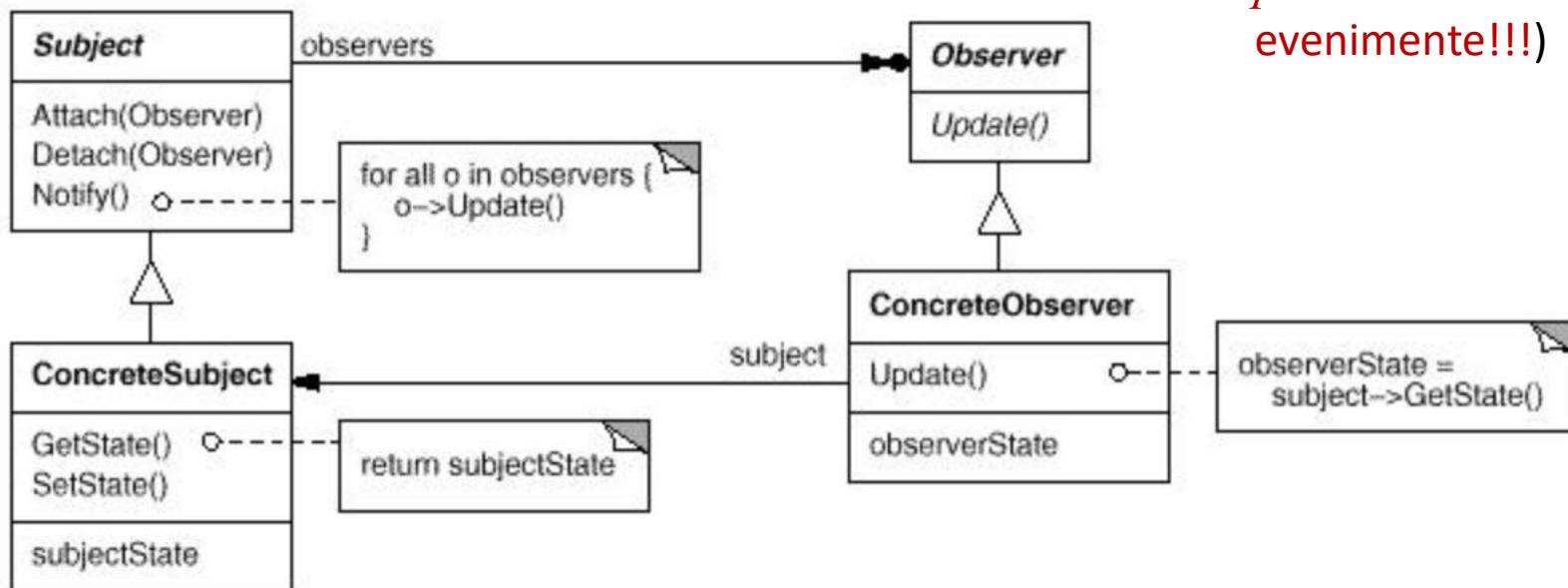
(In brief, Observer Pattern = publisher + subscriber.)

- Şablonul *Observer* defineşte **o relaţie de dependenţă 1 la n** între obiecte: când un obiect îşi schimbă starea, toţi dependentii lui sunt **notificaţi** şi **actualizaţi automat**.
 - Roluri obiecte: *subject(observat)* și *observator*
 - **Utilitate**: mai multe clase(*observatori*) depind de comportamentul unei alte clase(*subject*), în situaţii de tipul:
 - o clasă implementează/reprezintă logica, componenta de bază, iar alte clase doar folosesc rezultate ale acesteia (monitorizare).
 - o clasă efectuează acţiuni care apoi pot fi reprezentate în mai multe feluri de către alte clase (view-uri)
- Practic în toate aceste situaţii clasele Observer **observă** modificările/acţiunile clasei Subject. Observarea se implementează prin **notificări iniţiate din metodele clasei Subject**.

Şablonul Observer continuare

Subiect:

- menține o listă de referințe cu observatori fără să știe ce fac observatorii cu datele
- oferă metode de înregistrare/deînregistrare a unui *Observer*
- când apar modificări (e.g. se schimbă starea sa, valorile unor variabile etc) **notifică toți observatorii**



Observator:

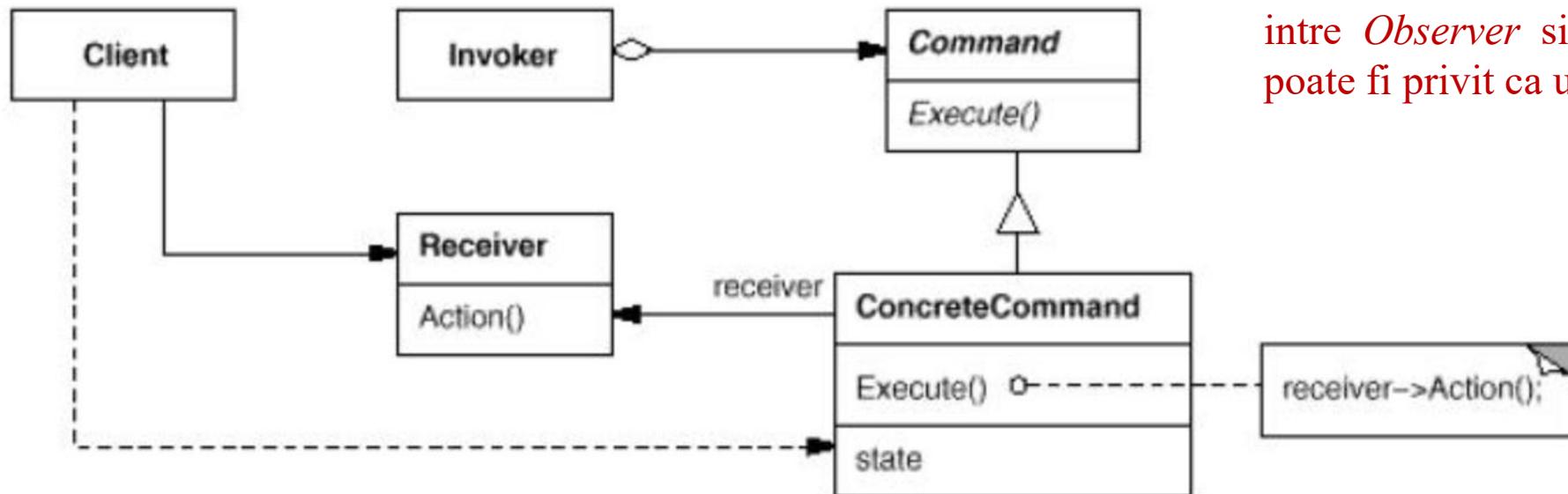
- definește o interfață `Observer` despre schimbări în subiec-
- toți observatorii pentru un anumit subiect trebuie să implementeze această interfață
- oferă una sau mai multe metode care să poată fi invocate de către *Subject* pentru a notifica o schimbare. Ca argumente se poate primi chiar instanța subiectului sau **obiecte speciale care reprezintă evenimentul ce a provocat schimbarea**. (Vezi exemplu seminar cu evenimente!!!)

Şablonul command

- Când se foloseşte: atunci când dorim să încapsulăm o comandă într-un obiect
- *Utilitate:*
Decuplare între **entitatea care dispune executarea comenzi** și **entitatea care o executa**.
Efectul unei comenzi poate fi schimbat dinamic.

Şablonul command

- *Command*
 - obiectul comanda
- *ConcreteCommand*
 - implementarea particulara a comenzi
 - apeleaza metode ale obiectului receptor
- *Invoker*
 - declanseaza comanda



• *Receiver*

- realizeaza, efectiv, operatiile aferente comenzi generate

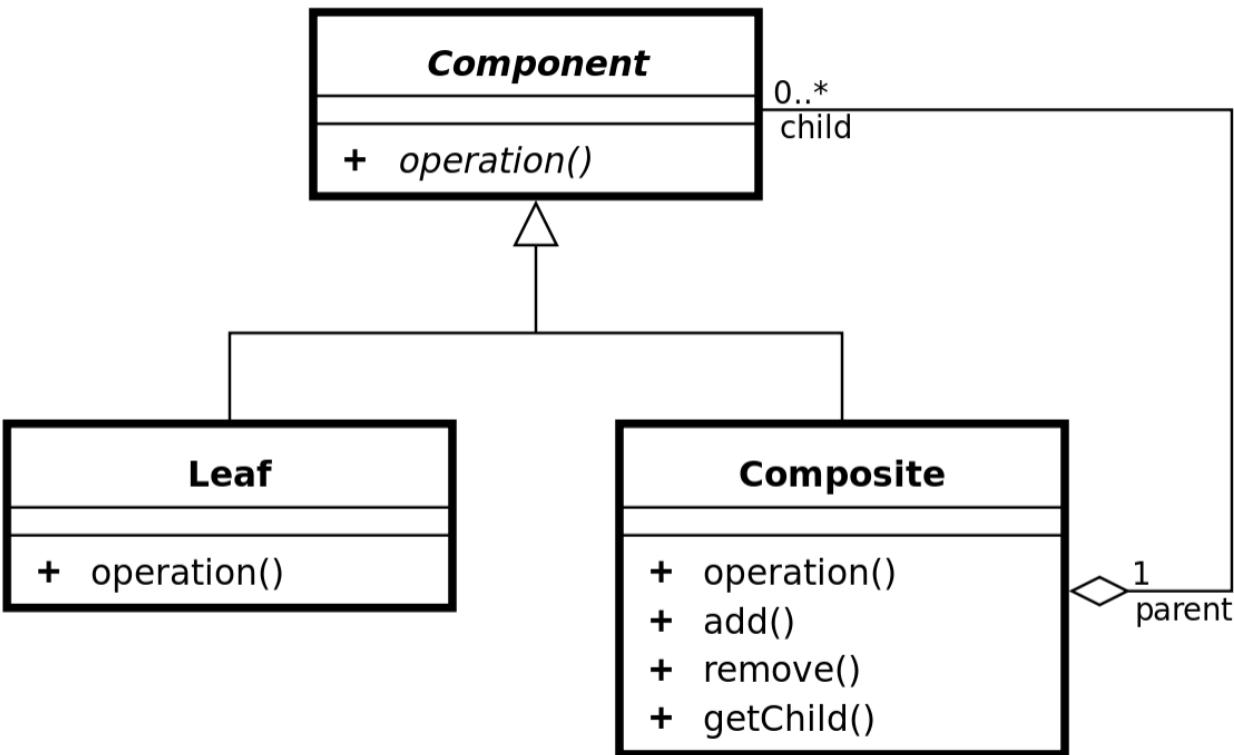
• *Client*

- defineste obiectul comanda si efectul ei

Observatie: Nu exista o delimitare clara intre *Observer* si *Command*. Un observator poate fi privit ca un obiect comanda.

Composite Pattern

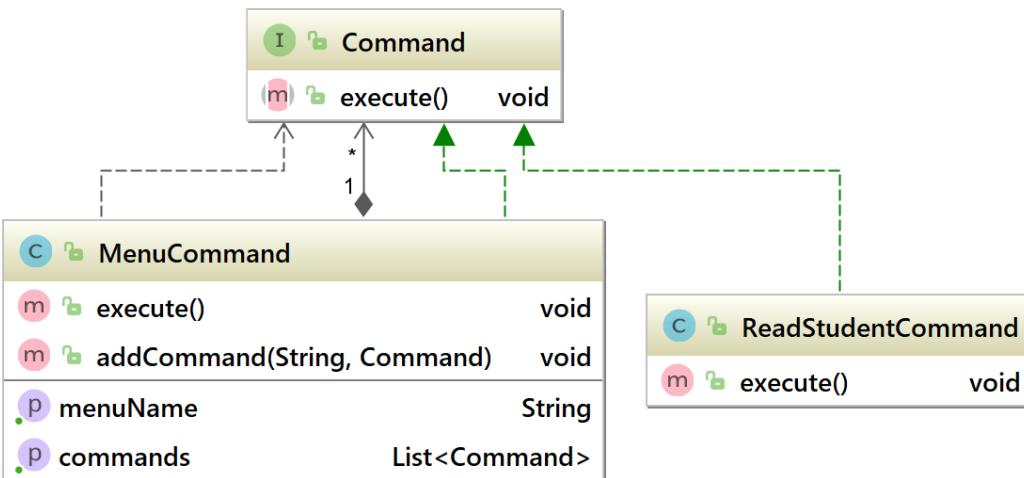
Compune mai multe obiecte similare a.i ele pot fi manipulate ca un singur obiect



TextMenuCommand

- O combinație de Command Pattern și Composite Pattern

```
public interface Command{  
    void execute();  
}
```



```
public class MenuCommand implements Command {  
  
    private String menuName;  
    private Map<String, Command> map= new TreeMap<>();  
  
    public MenuCommand(String menuName) {  
        this.menuName = menuName;  
    }  
    @Override  
    public void execute() {  
        map.keySet().forEach(x-> System.out.println(x));  
    }  
  
    public void addCommand(String desc, Command c){  
        map.put(desc, c);  
    }  
  
    public List<Command> getCommands(){  
        return map.values().stream().collect(Collectors.toList());  
    }  
  
    public String getMenuName() {  
        return menuName;  
    }  
}
```

TextMenuCommand

