Lecture 5 - Java Graphical User Interface (GUI): JavaFX

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

Outline

- Java Abstract Window Toolkit (AWT)
- Java Swing
- JavaFX

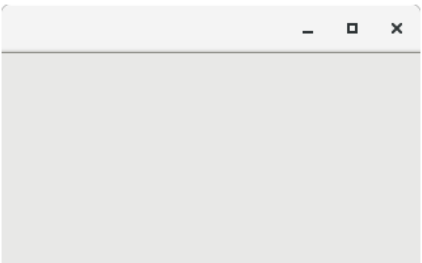
JavaFX

- JavaFX is a Java library used to develop Desktop applications as well as Rich Internet Applications (RIA).
- The applications built in JavaFX, can run on multiple platforms including Web, Mobile and Desktops.
- JavaFX is intended to replace swing in Java applications as a GUI framework. However, It provides more functionalities than swing.

JavaFX Application Structure

JavaFX application is divided into Stages, Scenes and nodes.

- **Stage** in a JavaFX application is similar to the Frame in a Swing Application. It acts like a container for all the JavaFX objects.

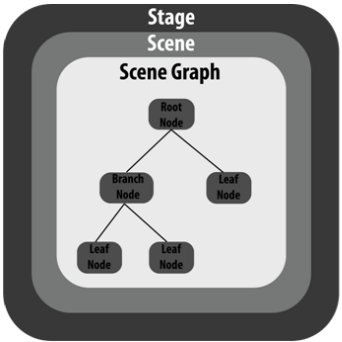


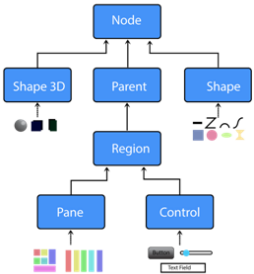
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Application Structure

JavaFX application is divided into Stages, Scenes and nodes.

- **Scene** actually holds all the physical contents (nodes) of a JavaFX application. The object of the primary stage is passed to the **start** method. We need to call **show** method on the **primary stage object** in order to show our primary stage.

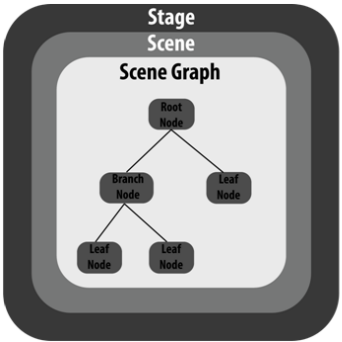


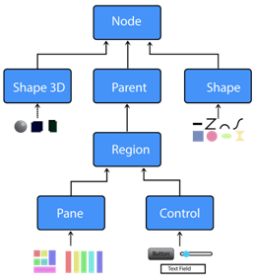


JavaFX Application Structure

JavaFX application is divided into Stages, Scenes and nodes.

- **Scene Graph** can be seen as the collection of various **nodes**. A node is an element that is visualized on the stage. It can be any button, text box, layout, image, radio button, check box, etc.





JavaFX Application Example

A JavaFX application that prints hello world on the console by clicking the button.

- Step 1: Extend javafx.application.Application and override start().

```
package application;
import javafx.application.Application;
import javafx.stage.Stage;
public class Hello_World extends Application{

   @Override
   public void start(Stage primaryStage) throws Exception {
      // TODO Auto-generated method stub

   }

}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Application Example

A JavaFX application that prints hello world on the console by clicking the button.

- Step 2: Create a Button.

```
package application;
import javafx.application.Application;
importjavafx.scene.control.Button;
import javafx.stage.Stage;
public class Hello_World extends Application{

   @Override
   public void start(Stage primaryStage) throws Exception {
      // TODO Auto-generated method stub
      Buttonbtn1=newButton("Say, Hello World");

   }

}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Application Example

A JavaFX application that prints hello world on the console by clicking the button.

- Step 3: Create a layout and add a button to it.

```
package application;
import javafx.application.Application;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{

  @Override
  public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Button btn1=new Button("Say, Hello World");
    StackPane root=new StackPane();
    root.getChildren().add(btn1);

  }

}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Application Example

A JavaFX application that prints hello world on the console by clicking the button.

- Step 4: Create a Scene.

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{

  @Override
  public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Button btn1=new Button("Say, Hello World");
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root);
  }
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Application Example

A JavaFX application that prints hello world on the console by clicking the button.

- Step 5: Prepare the Stage.

```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{

  @Override
  public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Button btn1=new Button("Say, Hello World");
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root);
    primaryStage.setScene(scene);
    primaryStage.setTitle("First JavaFX Application");
    primaryStage.show();
  }

}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Application Example

A JavaFX application that prints hello world on the console by clicking the button.

- Step 6: Create an event for the button.

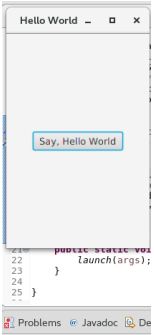
```
package application;
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.StackPane;
public class Hello_World extends Application{

  @Override
  publicvoid start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Button btn1=new Button("Say, Hello World");
    btn1.setOnAction(new EventHandler<ActionEvent>() {
      @Override
      publicvoid handle(ActionEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println("hello world");
      }
    });
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root,600,400);
    primaryStage.setScene(scene);
    primaryStage.setTitle("First JavaFX Application");
    primaryStage.show();
  }

}
```

JavaFX Application Example

A JavaFX application that prints hello world on the console by clicking the button.

```
public class Hello_World extends Application{

  @Override
  public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Button btn1=new Button("Say, Hello World");
    btn1.setOnAction(new EventHandler<ActionEvent>() {

      @Override
      public void handle(ActionEvent arg0) {
        // TODO Auto-generated method stub
        System.out.println("hello world");
      }
    });
```

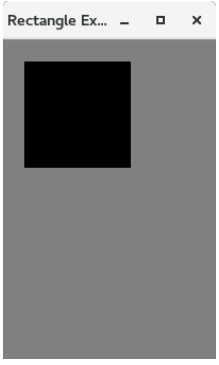
```
    StackPane root=new StackPane();
    root.getChildren().add(btn1);
    Scene scene=new Scene(root,600,400);
    primaryStage.setTitle("First JavaFX Application");
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  publicstaticvoid main (String[] args)
  {
    launch(args);
  }

}
```



JavaFX 2D Shapes: Rectangle

JavaFX library allows the developers to create a rectangle.

```
public class Shape_Example extends Application{

   @Override
   public void start(Stage primaryStage) throws Exception {
      // TODO Auto-generated method stub
   primaryStage.setTitle("Rectangle Example");
   Group group = new Group(); //creating Group
   Rectangle rect=new Rectangle(); //instantiating Rectangle
   rect.setX(20); //setting the X coordinate of upper left //corner of rectangle
   rect.setY(20); //setting the Y coordinate of upper left //corner of rectangle
   rect.setWidth(100); //setting the width of rectangle
   rect.setHeight(100); // setting the height of rectangle
   group.getChildren().addAll(rect); //adding rectangle to the //group
   Scene scene = new Scene(group,200,300,Color.GRAY);
   primaryStage.setScene(scene);
   primaryStage.show();
}
public static void main(String[] args) {
   launch(args);
}

}
```

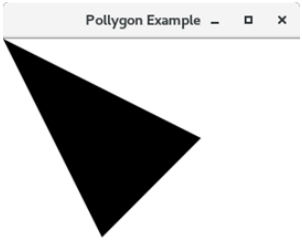


香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX 2D Shapes: Polygons

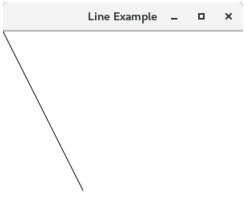
In JavaFX, Polygon can be created by instantiating javafx.scene.shape.Polygon class.

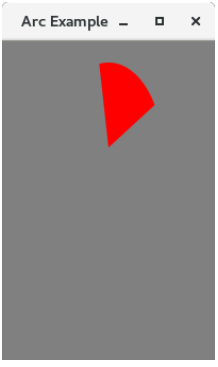
```
public class Shape_Example extends Application {

 @Override
 public void start(Stage primarystage) {
  Group root = new Group();
  primarystage.setTitle("Pollygon Example");

  Polygon polygon = new Polygon();
  polygon.getPoints().addAll(new Double[]{
     0.0, 0.0,
     100.0, 200.0,
     200.0, 100.0 });

  root.getChildren().add(polygon);
  Scene scene = new Scene(root,300,400);
  primarystage.setScene(scene);
  primarystage.show();
 }

 public static void main(String[] args) {
  launch(args);
 }
}
```

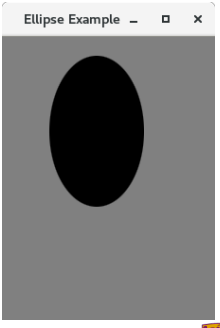


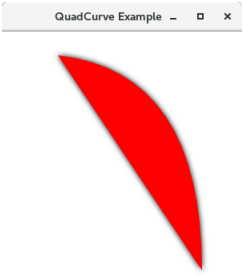
JavaFX 2D Shapes: Others

JavaFX library allows the developers to create line, arc, circle, and quadratic Curves of different colors (from left to right).









The Chinese University of Hong Kong, Shenzhen

JavaFX Effects

JavaFX provides the package named javafx.scene.effect which contains various classes that can be used to apply effects on the UI graphic components like images and shapes.

How to apply the effect to a node?

1. Create the node.
2. Create the object of the respective Effect class which is to be applied on the node.
3. Set the properties of the Effect.
4. Call setEffect() method through the node object and pass the Effect class object into it.

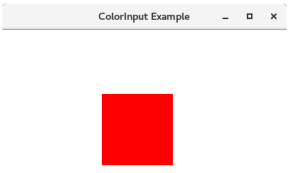
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Effects

ColorInput. This Effect makes a colored rectangle. This displays a rectangular box if applied to a node.

```
package application;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.effect.ColorInput;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;
public class ColorInputExample extends Application {
public static void main(String[] args) {
   launch(args);
}
@Override
   public void start(Stage primaryStage) throws Exception {
      // TODO Auto-generated method stub
```

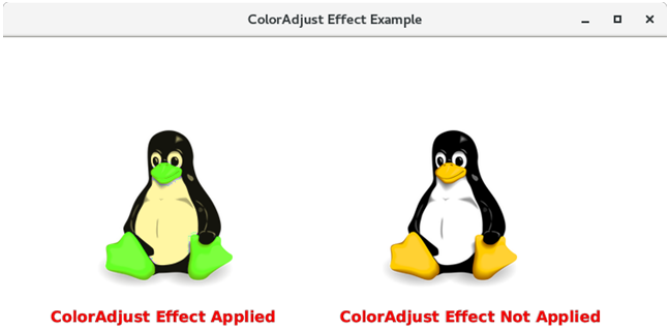
```
   ColorInput color = new ColorInput();
   color.setPaint(Color.RED);
   color.setHeight(100);
   color.setWidth(100);
   color.setX(140);
   color.setY(90);
   Rectangle rect = new Rectangle();
   rect.setEffect(color);
   Group root = new Group();
   Scene scene = new Scene(root,400,300);
   root.getChildren().add(rect);
   primaryStage.setScene(scene);
   primaryStage.setTitle("ColorInput Example");
   primaryStage.show();
   }
}
```



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

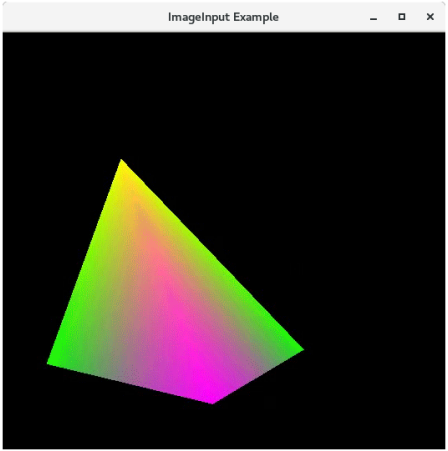
JavaFX Effects

ColorAdjust. This Effect adjusts the color of the node by varying the properties like Hue, Saturation, Brightness, contrast, etc.



JavaFX Effects

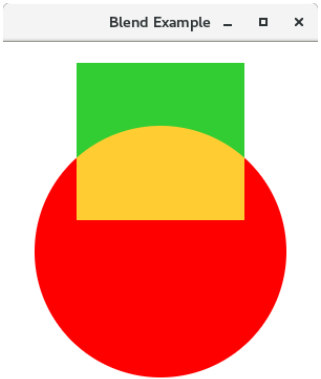
ImageInput. This effect is used to bind the image to the scene. It basically passes the specified image to some effect.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

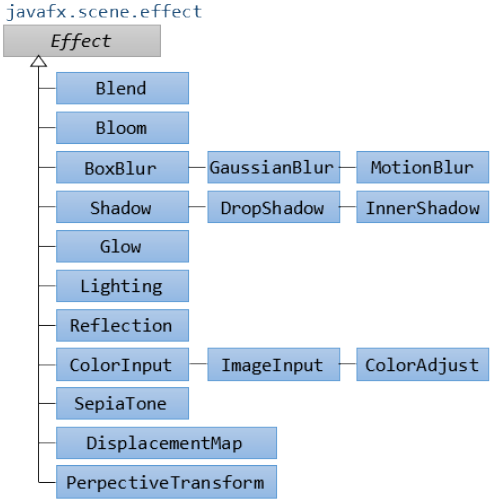
JavaFX Effects

Blend. This effect joins the pixels of two inputs and produces the combined output at the same location. There are various blend modes defined in the class which can alter the output appearance.



JavaFX Effects

Other effects. There a total of 18 effects summarized as below:



JavaFX Transformation

Transformation can be defined as the change in form, nature or appearance of the graphics. In JavaFX, the package named javafx.scene.transform represents all the transformations.

How to apply the transformation on the node?

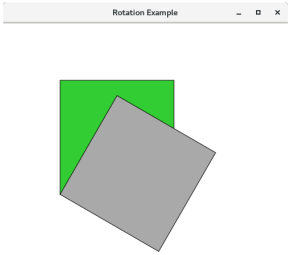
1. Instantiate the respective class.
2. Set the appropriate properties of the scale class object.
3. Apply the transformation to the respective node.

JavaFX Transformation

Rotation can be defined as the process of rotating an object by a certain angle Θ.

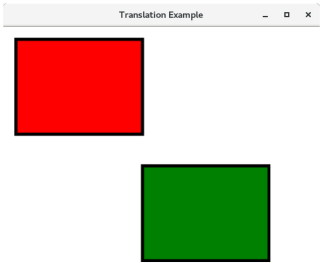
```
package application;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.scene.transform.Rotate;
import javafx.stage.Stage;
public class RotateExample extends Application{
@Override
public void start(Stage primaryStage) throws Exception {
  // TODO Auto-generated method stub
  // creating the rectangles
  Rectangle rect1 = new Rectangle(100,100,200,200);
  Rectangle rect2 = new Rectangle(100,100,200,200);

  // setting the color and stroke for the Rectangles
  rect1.setFill(Color.LIMEGREEN);
  rect2.setFill(Color.DARKGREY);
  rect1.setStroke(Color.BLACK);
  rect2.setStroke(Color.BLACK);
```

```
  // instantiating the Rotate class.
  Rotate rotate = new Rotate();

  //setting properties for the rotate object.
  rotate.setAngle(30);
  rotate.setPivotX(100);
  rotate.setPivotY(300);

  //rotating the 2nd rectangle.
  rect2.getTransforms().add(rotate);

  Group root = new Group();
  root.getChildren().addAll(rect1,rect2);
  Scene scene = new Scene(root,500,420);
  primaryStage.setScene(scene);
  primaryStage.setTitle("Rotation Example");
  primaryStage.show();
}
public static void main(String[] args) {
  launch(args);
}
}
```



JavaFX Transformation

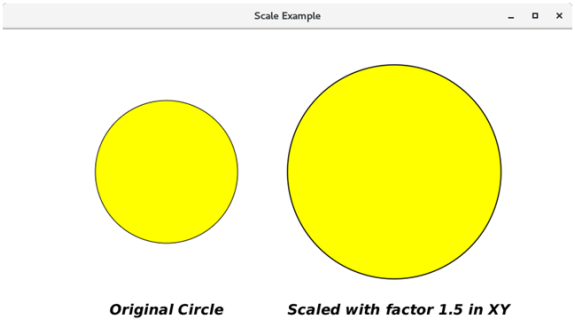
Translation. Translation can be defined as the change in the position of an object on the screen. The position of an object gets changed by moving it along the X-Y direction.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Transformation

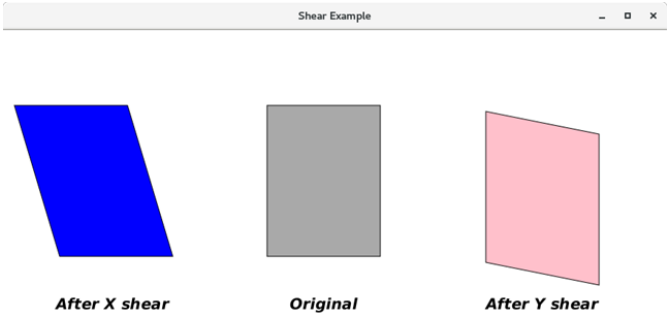
Scaling. Scaling changes the size of the object. The size can be altered by multiplying the coordinates of the object by a factor which is called the scaling factor.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Transformation

Shearing. Shearing is a kind of transformation that changes the slope of the object with respect to any of the axis. The X-shear transformation changes the X-coordinate values while the Y-shear changes the Y-coordinate values.



JavaFX Animation

In general, animation is the set of transformations applied to an object over the specified duration sequentially so that the object can be shown as it is in motion.

How to apply animations?

1. Create the target node and configure its properties.

```
Rectangle rect = new Rectangle(120,100,100,100);
rect.setFill(Color.RED);
```

JavaFX Animation

In general, animation is the set of transformations applied to an object over the specified duration sequentially so that the object can be shown as it is in motion.

How to apply animations?

2. Instantiate the respective transition class.

```
RotateTransition rotate = new RotateTransition();
```

JavaFX Animation

In general, animation is the set of transformations applied to an object over the specified duration sequentially so that the object can be shown as it is in motion.

How to apply animations?

3. Set the desired properties like duration, cycle-count, etc. for the transition.

```
rotate.setDuration(Duration.millis(1000));
rotate.setAxis(Rotate.Y_Axis);
rotate.setCycleCount(500);
```

JavaFX Animation

In general, animation is the set of transformations applied to an object over the specified duration sequentially so that the object can be shown as it is in motion.

How to apply animations?

4. Set the target node on which the transition will be applied. Use the following method for this purpose.

```
rotate.setNode(rect);
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Animation

In general, animation is the set of transformations applied to an object over the specified duration sequentially so that the object can be shown as it is in motion.

How to apply animations?

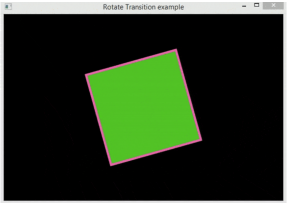
5. Finally, play the transition using the play() method.

```
rotate.play();
```

JavaFX Animation: Rotate Transition

This transition rotates the node along any of the three axes over the specified duration.

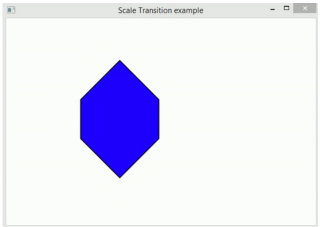
```
public class Rotate_Transition extends Application{

  @Override
  public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub

    //Creating Rectangle
    Rectangle rect = new Rectangle(200,100,200,200);
    rect.setFill(Color.LIMEGREEN);
    rect.setStroke(Color.HOTPINK);
    rect.setStrokeWidth(5);

    //Instantiating RotateTransition class
    RotateTransition rotate = new RotateTransition();

    //Setting Axis of rotation
    rotate.setAxis(Rotate.Z_AXIS);

    // setting the angle of rotation
    rotate.setByAngle(360);

    //setting cycle count of the rotation
    rotate.setCycleCount(500);

    //the transition will be auto reversed by setting this to true
    rotate.setAutoReverse(true);

    //setting Rectangle as the node onto which the
// transition will be applied
    rotate.setNode(rect);

    //playing the transition
    rotate.play();

    //Configuring Group and Scene
    Group root = new Group();
    root.getChildren().add(rect);
    Scene scene = new Scene(root,600,400,Color.BLACK);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Rotate Transition example");
    primaryStage.show();

  }
  public static void main(String[] args) {
    launch(args);
  }

}
```

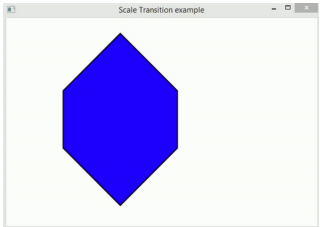


香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Animation: Others

- **Scale**: This transition animates the scaling of the node over the specified duration by the specified factor in any or all of the three directions X, Y, and Z.

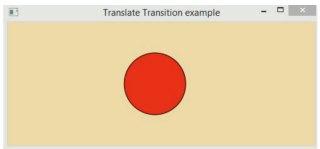


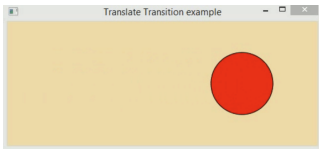


香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Animation: Others

- **Translate**: This transition translates the node from one position to another by updating the node's translateX and translateY properties at regular intervals.

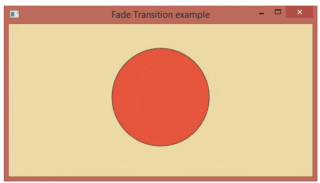


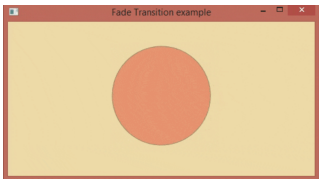


香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Animation: Others

- **Fade**: This transition animates the opacity of the node by keep decreasing the opacity of the fill color over a specified duration in order to reach a target opacity value.

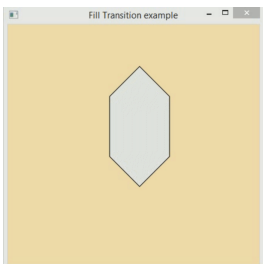


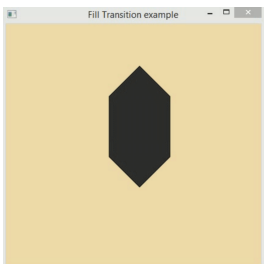


香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Animation: Others

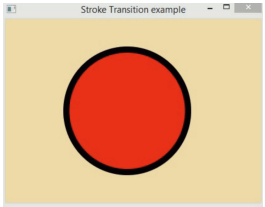
- **Fill**: This transition animates the node's fill color so that the fill color can fluctuate between the two color values over the specified duration.

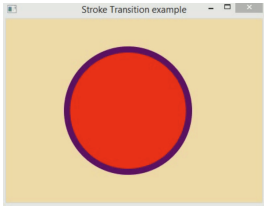




JavaFX Animation: Others

- **Stroke**: This transition animates the node's stroke color so that the stroke color can fluctuate between the two color values over the specified duration.

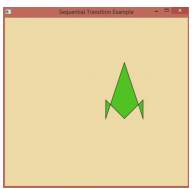
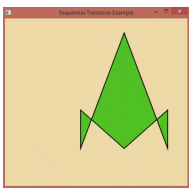
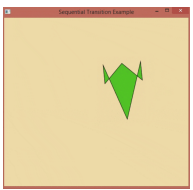
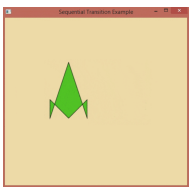




香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

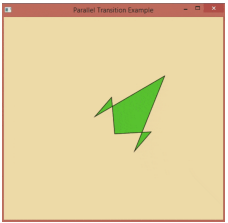
JavaFX Animation: Others

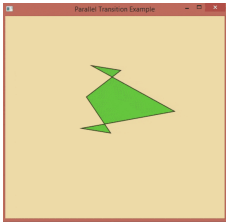
- **Sequential Transition**: This transition is used to apply the list of animations on a node in sequential order (from left to right, scale, rotate and translate).

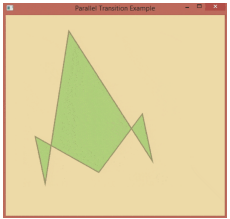





JavaFX Animation: Others

- **Parallel Transition**: This transition applies multiple animations on a node in parallel. It is similar to Sequential Transition except for the fact that it applies multiple transitions on a node at the same time.







JavaFX Animation: Others

- **Pause Transition**: This transition is used to pause between the multiple animations applied on a node in the sequential order. During the lifespan of this transition, the node remains unmovable on the screen.
- **Path Transition**: It allows the node to animate through a specified path over the specified duration.

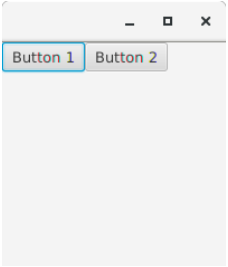
JavaFX Layouts

- Layouts are the top-level container classes that define the UI styles for scene graph objects.
- The layout can be seen as the parent node to all the other nodes.
- JavaFX provides various layout panes that support different styles of layouts.

JavaFX Layouts

- **JavaFX HBox**: HBox layout pane arranges the nodes in a single row.

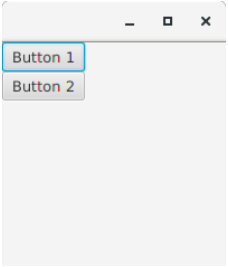
```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class Label_Test extends Application {

@Override
public void start(Stage primaryStage) throws Exception {
Button btn1 = new Button("Button 1");
Button btn2 = new Button("Button 2");
HBox root = new HBox();
Scene scene = new Scene(root,200,200);
root.getChildren().addAll(btn1,btn2);
primaryStage.setScene(scene);
primaryStage.show();
}
public static void main(String[] args) {
      launch(args);
}
```



JavaFX Layouts

- **JavaFX VBox**: This layout Pane arranges the nodes in a single vertical column.

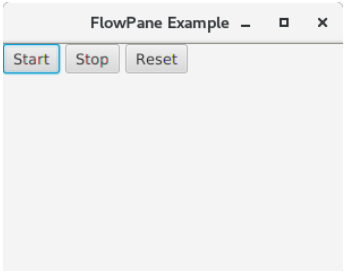
```
package application;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.VBox;
import javafx.stage.Stage;
public class Label_Test extends Application {

  @Override
  public void start(Stage primaryStage) throws Exception {
     Button btn1 = new Button("Button 1");
     Button btn2 = new Button("Button 2");
     VBox root = new VBox();
     Scene scene = new Scene(root,200,200);
     root.getChildren().addAll(btn1,btn2);
     primaryStage.setScene(scene);
     primaryStage.show();
  }
  public static void main(String[] args) {
     launch(args);
  }
```



JavaFX Layouts

- **JavaFX FlowPane**: FlowPane layout pane organizes the nodes in a flow that are wrapped at the FlowPane's boundary.

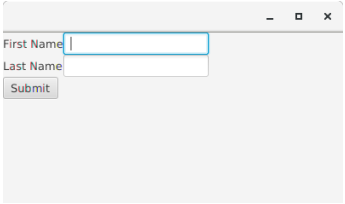
```
public class FlowPaneTest extends Application {

  @Override
  public void start(Stage primaryStage) {
    primaryStage.setTitle("FlowPane Example");
    FlowPane root = new FlowPane();
    root.setVgap(6);
    root.setHgap(5);
    root.setPrefWrapLength(250);
    root.getChildren().add(new Button("Start"));
    root.getChildren().add(new Button("Stop"));
    root.getChildren().add(new Button("Reset"));
    Scene scene = new Scene(root,300,200);

    primaryStage.setScene(scene);
    primaryStage.show();
  }

  public static void main(String[] args) {
    launch(args);
  }
```



JavaFX Layouts

- **JavaFX GridPane**: This layout pane provides a flexible grid of rows and columns where nodes can be placed in any cell of the grid.

```
public class Label_Test extends Application {

  @Override
  public void start(Stage primaryStage) throws Exception {
    Label first_name=new Label("First Name");
    Label last_name=new Label("Last Name");
    TextField tf1=new TextField();
    TextField tf2=new TextField();
    Button Submit=new Button ("Submit");
    GridPane root=new GridPane();
    Scene scene = new Scene(root,400,200);
    root.addRow(0, first_name,tf1);
    root.addRow(1, last_name,tf2);
    root.addRow(2, Submit);
    primaryStage.setScene(scene);
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
```



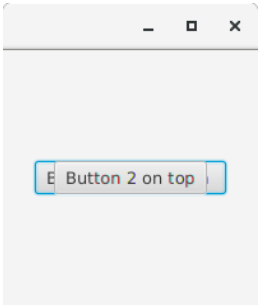
JavaFX Layouts

- **JavaFX BorderPane**: This layout arranges the nodes at the left, right, center, top, and bottom of the screen.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Layouts

- **JavaFX StackPane**: This layout places all the nodes into a single stack where every new node gets placed on top of the previous node.



JavaFX UI Controls

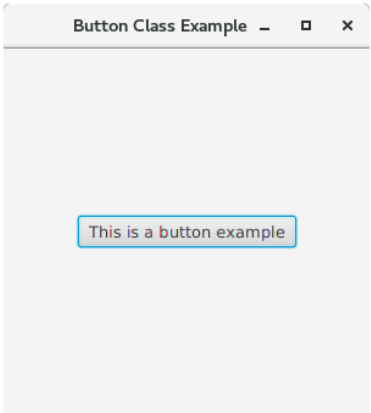
- **Label** is a component that is used to define a simple text on the screen.

```
public class LabelTest extends Application {

  @Override
  public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    Label my_label=new Label("This is an example of Label");
    StackPane root = new StackPane();
    Scene scene=new Scene(root,300,300);
    root.getChildren().add(my_label);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Label Class Example");
    primaryStage.show();

  }
  public static void main(String[] args) {
    launch(args);
  }
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

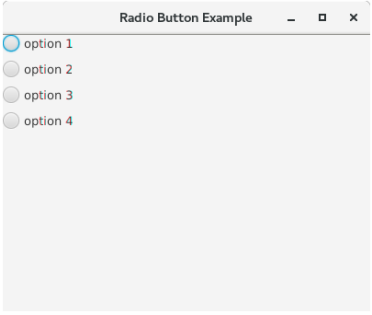
- **Button** is a component that controls the function of the application. Button class is used to create a labeled button.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

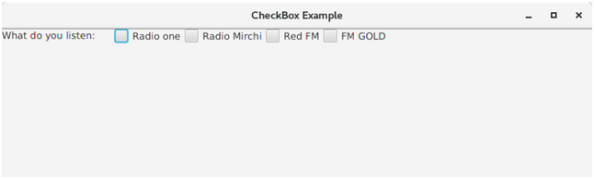
- **The Radio Button** is used to provide various options to the user. The user can only choose one option from all. A radio button is either selected or deselected.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

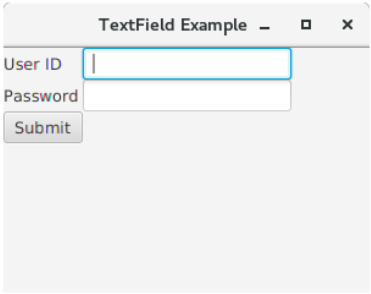
- **Check Box** is used to get the kind of information from the user which contains various choices. The user marked the checkbox either on (true) or off(false).



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

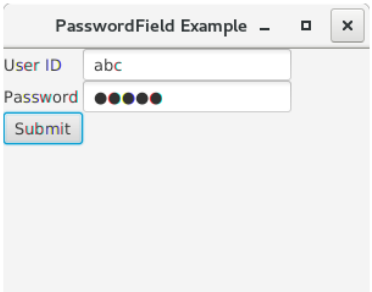
- **Text Field** is basically used to get input from the user in the form of text.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

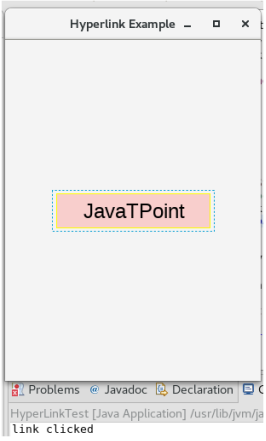
- **PasswordField** is used to get the user's password. Whatever is typed in the password field is not shown on the screen to anyone.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

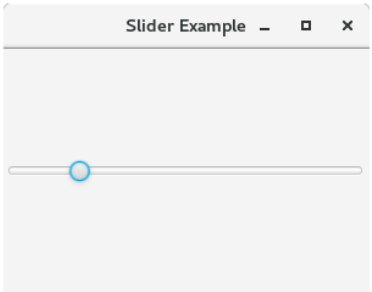
- **HyperLink** is used to refer to any of the web pages through your application.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

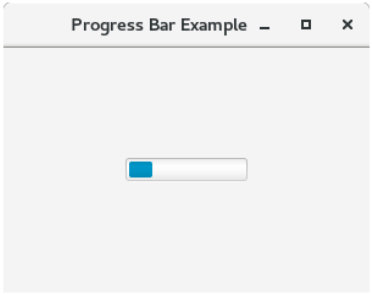
- **Slider** is used to provide a pane of options to the user in a graphical form where the user needs to move a slider over the range of values to select one of them.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

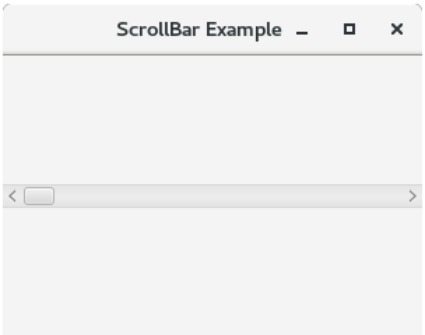
- **Progress Bar** is used to show the work progress to the user.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

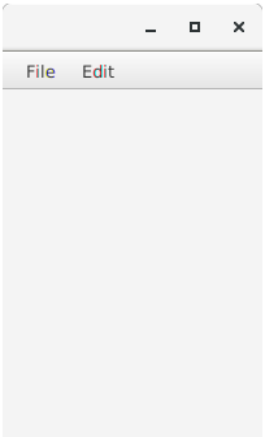
- **ScrollBar** is used to provide a scroll bar to the user so that the user can scroll down the application pages.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

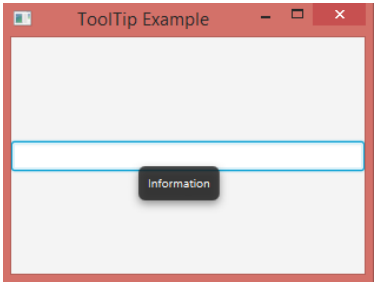
- **Menu** implement menus. The menu is the main component of any application.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX UI Controls

- **ToolTip** is used to provide hint to the user about any component. It is mainly used to provide hints about the text fields or password fields being used in the application.



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Charts

How to create Charts in JavaFX?

- **Step 1** Configure the Axes. In order to create the Xaxis and Yaxis, we need to instantiate the respective class. However, this step is not necessary for pie charts.

```
NumberAxis xaxis = new NumberAxis();
CategoryAxis yaxis = new CategoryAxis();
xaxis.setLabel(?X-Axis?);
yaxis.setLabel(?Y-Axis?);
```

JavaFX Charts

How to create Charts in JavaFX?

- **Step 2** Create the chart. We just need to instantiate the respective class in order to create the chart.

```
LineChart linechart = new LineChart(xaxis,yaxis);
linechart.setTitle(?Line Chart Example?);
```

JavaFX Charts

How to create Charts in JavaFX?

- **Step 3** Passing Data to the Chart by:
  1. Instantiate XYChart.Series:

```
XYChart.Series series = new XYChart.Series();
series.setName(?Value type 1?);
```

  2. Adding Data to the series:

```
series.getData().add(new XYChart.Data(2010,25)); series.getData().add(new XYChart.Data(2011,15));
series.getData().add(new XYChart.Data(2012,78))
series.getData().add(new XYChart.Data(2013,60));
```

  3. Adding Series to the Chart.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Charts

How to create Charts in JavaFX?

- **Step 4** Configure Group and Scene. Here, we will create the group and add the line chart to the group. The group object is passed into the scene class constructor. The scene class object is passed to the setScene Method.

```
Group group = new Group();
group.getChildren().add(linechart);
Scene scene = new Scene(group,600,400);
primaryStage.setScene(scene);
primaryStage.showTitle(?Chart Example?);
primaryStage.show();
```

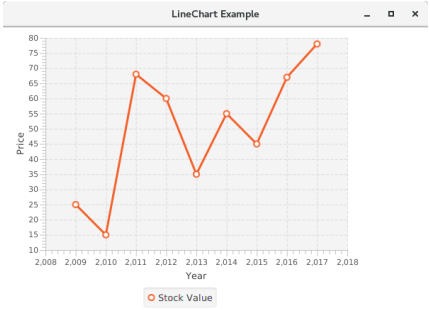
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX LineChart

The Line Chart displays the group of data points called markers.

```
public class LineChartTest extends Application
{

  @Override
  public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub
    //Defining Axis
    final NumberAxis xaxis = new NumberAxis(2008,2018,1);
    final NumberAxis yaxis = new NumberAxis(10,80,5);

    //Defining Label for Axis
    xaxis.setLabel("Year");
    yaxis.setLabel("Price");

    //Creating the instance of linechart with the specified axis
    LineChart linechart = new LineChart(xaxis,yaxis);

    //creating the series
    XYChart.Series series = new XYChart.Series();
```

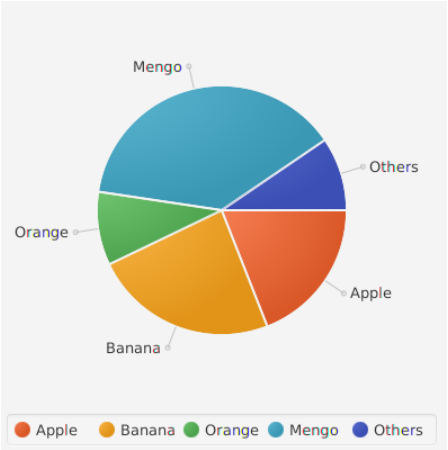
```
    //setting name and the date to the series
    series.setName("Stock Analysis");
    series.getData().add(new XYChart.Data(2009,25));
    series.getData().add(new XYChart.Data(2010,15));
    series.getData().add(new XYChart.Data(2011,68));
    series.getData().add(new XYChart.Data(2012,60));
    series.getData().add(new XYChart.Data(2013,35));
    series.getData().add(new XYChart.Data(2014,55));
    series.getData().add(new XYChart.Data(2015,45));
    series.getData().add(new XYChart.Data(2016,67));
    series.getData().add(new XYChart.Data(2017,78));

    //adding series to the linechart
    linechart.getData().add(series);

    //setting Group and Scene
    Group root = new Group(linechart);
    Scene scene = new Scene(root,600,400);
    primaryStage.setScene(scene);
    primaryStage.setTitle("LineChart Example");
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }
```



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

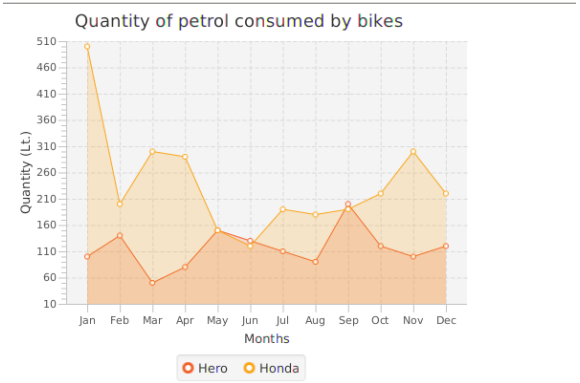
JavaFX PieChart

A Pie chart is a type of graph or diagram in which the sectors of a circle are used to represent different proportions of the whole information.



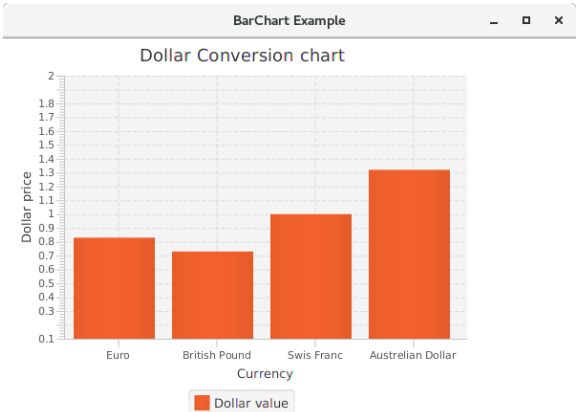
JavaFX AreaChart

The area chart is used to display the graphically quantitative data. It basically plots the area for the set of points on an XY Plane.



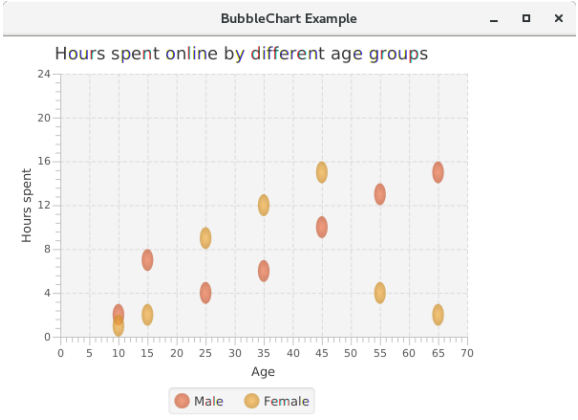
JavaFX BarChart

A bar chart can be defined as a diagram that uses rectangular bars for data representation. The length of the bars represents the exact numeric data values plotted on one of the axes.



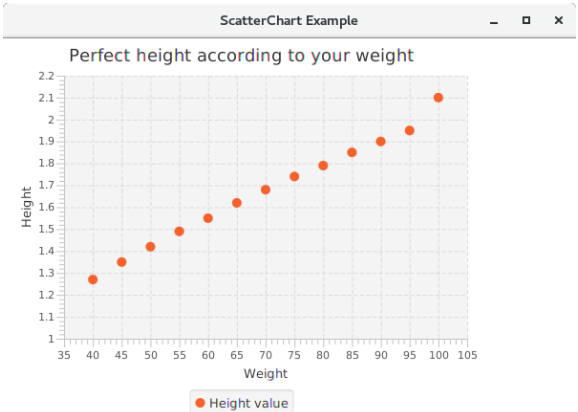
JavaFX Bubble Chart

A Bubble Chart can be defined as a diagram that is used to display three-dimensional data. Each entity is identified by a bubble that contains three triplets (v1, v2, v3).



JavaFX ScatterChart

In Scatter Chart, the data points are scattered along the graph. Each data point displays the mapping between both axis. It is mainly used to plot the relationship between the axes.



JavaFX FXML

JavaFX FXML is an **XML-based** language that provides the structure for building a user interface **separate from** the application logic of your code.

- **Step 1:** Load the FXML source file with the java file.

Example 4-1 FXMLExample.java

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

    Scene scene = new Scene(root, 300, 275);

    stage.setTitle("FXML Welcome");
    stage.setScene(scene);
    stage.show();
}
```

JavaFX FXML

JavaFX FXML is an **XML-based** language that provides the structure for building a user interface **separate from** the application logic of your code.

- **Step 2:** Modify the import statements in the fxml file.

Example 4-2 XML Declaration and Import Statements

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX FXML

JavaFX FXML is an **XML-based** language that provides the structure for building a user interface **separate from** the application logic of your code.

- **Step 3:** Create a gridPane layout in the fxml file.

Example 4-3 GridPane Layout

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

</GridPane>
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX FXML

JavaFX FXML is an **XML-based** language that provides the structure for building a user interface **separate from** the application logic of your code.

- **Step 4:** Add text and password Fields in the fxml file.

Example 4-4 Text, Label, TextField, and Password Field Controls

```
<Text text="Welcome"
    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 fx:id="passwordField"
    GridPane.columnIndex="1" GridPane.rowIndex="2"/>
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX FXML

JavaFX FXML is an **XML-based** language that provides the structure for building a user interface **separate from** the application logic of your code.

- **Step 5:** Add a button and text in the fxml file.

Example 4-5 HBox, Button, and Text

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

<Text fx:id="actiontarget"
       GridPane.columnIndex="1" GridPane.rowIndex="6"/>
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX FXML

JavaFX FXML is an **XML-based** language that provides the structure for building a user interface **separate from** the application logic of your code.

- **Step 6:** Add code to handle an event in the java file.

Example 4-6 FXMLExampleController.java

```
package fxmlexample;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.text.Text;

public class FXMLExampleController {
    @FXML private Text actiontarget;

    @FXML protected void handleSubmitButtonAction(ActionEvent event) {
        actiontarget.setText("Sign in button pressed");
    }

}
```

JavaFX FXML

JavaFX FXML is an **XML-based** language that provides the structure for building a user interface **separate from** the application logic of your code.

- **Example:** The following figure shows the results.

Before adding css.

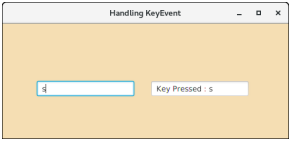
After adding css.

JavaFX Event Handling: Convenient Methods

JavaFX provides convenient methods to handle events (create and register event handlers to respond to KeyEvent, MouseEvent, Action Event, Drag and Drop Events).

```
package application;
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyEvent;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
public class JavaFX_KeyEvent extends Application{

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

    // TODO Auto-generated method stub

    //Creating TextFields and setting position for them
    TextField tf1 = new TextField();
    TextField tf2 = new TextField();
    tf1.setTranslateX(100);
    tf1.setTranslateY(100);
    tf2.setTranslateX(300);
    tf2.setTranslateY(100);
```

```
    //Handling KeyEvent for textfield 1
    tf1.setOnKeyPressed(new EventHandler<KeyEvent>() {

      @Override
      public void handle(KeyEvent key) {
        // TODO Auto-generated method stub
        tf2.setText("Key Pressed :"+" "+key.getText());
      }

    });

    //setting group and scene
    Group root = new Group();
    root.getChildren().addAll(tf2,tf1);
    Scene scene = new Scene(root,500,200,Color.WHEAT);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Handling KeyEvent");
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }

}
```



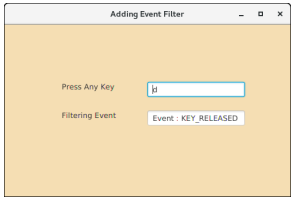
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

JavaFX Event Handling: Event Filters

Event Filters process the events **in the Event capturing phase**. JavaFX enables us to register a single event filter **for more than one node and more than one type**.

```
public class JavaFX_EventFilter extends Application{

  @Override
  public void start(Stage primaryStage) throws Exception {
    // TODO Auto-generated method stub

    //Adding Labels and TextFileds to the scene
    Label l1 = new Label("Press Any Key ");
    Label l2 = new Label("Filtering Event ");

    l1.setTranslateX(100);
    l1.setTranslateY(100);

    l2.setTranslateX(100);
    l2.setTranslateY(150);

    TextField tf1 = new TextField();
    TextField tf2 = new TextField();

    tf1.setTranslateX(250);
    tf1.setTranslateY(100);
    tf2.setTranslateX(250);
    tf2.setTranslateY(150);
```

```
    //Creating EventHandler Object
    EventHandler<KeyEvent> filter = new EventHandler<KeyEvent>() {
      @Override
      public void handle(KeyEvent event) {
        // TODO Auto-generated method stub
        tf2.setText("Event : "+event.getEventType());
        tf1.setText(event.getText());
        event.consume();
      }
    };

    //Registering Event Filter for the event generated on text field
    tf1.addEventFilter(KeyEvent.ANY,filter );

    //Setting Group and Scene
    Group root = new Group();
    root.getChildren().addAll(l1,l2,tf1,tf2);
    Scene scene = new Scene(root,500,300,Color.WHEAT);
    primaryStage.setScene(scene);
    primaryStage.setTitle("Adding Event Filter");
    primaryStage.show();
  }
  public static void main(String[] args) {
    launch(args);
  }

}
```



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

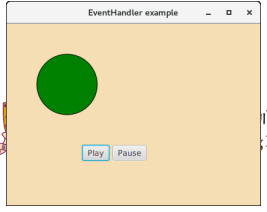
JavaFX Event Handling: Event Handlers

Event Handlers are used to handle the events in the **event bubbling phase**. One can register multiple handlers for a single node (or one handler for multiple nodes.).

```
public void start(Stage primaryStage) throws Exception {
  // TODO Auto-generated method stub
  //Creating Circle and setting the color and stroke in the circle
  Circle c = new Circle(100,100,50);
  c.setFill(Color.GREEN);
  c.setStroke(Color.BLACK);

  //creating play button and setting coordinates for the button
  Button btn = new Button("Play");
  btn.setTranslateX(125);
  btn.setTranslateY(200);

  // creating pause button and setting coordinate for the pause button
  Button btn1 = new Button("Pause");
  btn1.setTranslateX(175);
  btn1.setTranslateY(200);

  //Instantiating TranslateTransition class to create the animation
  TranslateTransition trans = new TranslateTransition();

  //setting attributes for the TranslateTransition
  trans.setAutoReverse(true);
  trans.setByX(200);
  trans.setCycleCount(100);
  trans.setDuration(Duration.millis(500));
  trans.setNode(c);
```

```
//Creating EventHandler
EventHandler<MouseEvent> handler = new EventHandler<MouseEvent>() {

  @Override
  public void handle(MouseEvent event) {
    // TODO Auto-generated method stub

    if(event.getSource()==btn)
    {
    trans.play(); //animation will be played when the play button is clicked
    }
    if(event.getSource()==btn1)
    {
      trans.pause(); //animation will be paused when the pause button is clicked
    }
    event.consume();
  }

};

//Adding Handler for the play and pause button
btn.setOnMouseClicked(handler);
btn1.setOnMouseClicked(handler);
```

```
  //Creating Group and scene
  Group root = new Group();
  root.getChildren().addAll(c,btn,btn1);
  Scene scene = new Scene(root,420,300,Color.WHEAT);
  primaryStage.setScene(scene);
  primaryStage.setTitle("EventHandler example");
  primaryStage.show();
}
public static void main(String[] args) {
  launch(args);
}
}
```



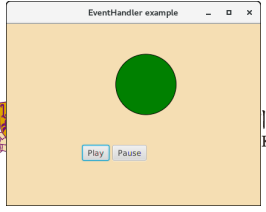
JavaFX Event Handling: Event Handlers

Event Handlers are used to handle the events in the **event bubbling phase**. One can register multiple handlers for a single node (or one handler for multiple nodes.).

```
public void start(Stage primaryStage) throws Exception {
  // TODO Auto-generated method stub
  //Creating Circle and setting the color and stroke in the circle
  Circle c = new Circle(100,100,50);
  c.setFill(Color.GREEN);
  c.setStroke(Color.BLACK);

  //creating play button and setting coordinates for the button
  Button btn = new Button("Play");
  btn.setTranslateX(125);
  btn.setTranslateY(200);

  // creating pause button and setting coordinate for the pause button
  Button btn1 = new Button("Pause");
  btn1.setTranslateX(175);
  btn1.setTranslateY(200);

  //Instantiating TranslateTransition class to create the animation
  TranslateTransition trans = new TranslateTransition();

  //setting attributes for the TranslateTransition
  trans.setAutoReverse(true);
  trans.setByX(200);
  trans.setCycleCount(100);
  trans.setDuration(Duration.millis(500));
  trans.setNode(c);
```

```
//Creating EventHandler
EventHandler<MouseEvent> handler = new EventHandler<MouseEvent>() {

  @Override
  public void handle(MouseEvent event) {
    // TODO Auto-generated method stub

    if(event.getSource()==btn)
    {
    trans.play(); //animation will be played when the play button is clicked
    }
    if(event.getSource()==btn1)
    {
      trans.pause(); //animation will be paused when the pause button is clicked
    }
    event.consume();
  }

};

//Adding Handler for the play and pause button
btn.setOnMouseClicked(handler);
btn1.setOnMouseClicked(handler);
```

```
  //Creating Group and scene
  Group root = new Group();
  root.getChildren().addAll(c,btn,btn1);
  Scene scene = new Scene(root,420,300,Color.WHEAT);
  primaryStage.setScene(scene);
  primaryStage.setTitle("EventHandler example");
  primaryStage.show();
}
public static void main(String[] args) {
  launch(args);
}
}
```



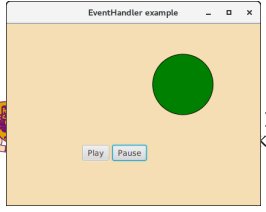
JavaFX Event Handling: Event Handlers

Event Handlers are used to handle the events in the **event bubbling phase**. One can register multiple handlers for a single node (or one handler for multiple nodes.).

```
public void start(Stage primaryStage) throws Exception {
  // TODO Auto-generated method stub
  //Creating Circle and setting the color and stroke in the circle
  Circle c = new Circle(100,100,50);
  c.setFill(Color.GREEN);
  c.setStroke(Color.BLACK);

  //creating play button and setting coordinates for the button
  Button btn = new Button("Play");
  btn.setTranslateX(125);
  btn.setTranslateY(200);

  // creating pause button and setting coordinate for the pause button
  Button btn1 = new Button("Pause");
  btn1.setTranslateX(175);
  btn1.setTranslateY(200);

  //Instantiating TranslateTransition class to create the animation
  TranslateTransition trans = new TranslateTransition();

  //setting attributes for the TranslateTransition
  trans.setAutoReverse(true);
  trans.setByX(200);
  trans.setCycleCount(100);
  trans.setDuration(Duration.millis(500));
  trans.setNode(c);
```

```
//Creating EventHandler
EventHandler<MouseEvent> handler = new EventHandler<MouseEvent>() {

  @Override
  public void handle(MouseEvent event) {
    // TODO Auto-generated method stub

    if(event.getSource()==btn)
    {
    trans.play(); //animation will be played when the play button is clicked
    }
    if(event.getSource()==btn1)
    {
      trans.pause(); //animation will be paused when the pause button is clicked
    }
    event.consume();
  }

};

//Adding Handler for the play and pause button
btn.setOnMouseClicked(handler);
btn1.setOnMouseClicked(handler);
```

```
  //Creating Group and scene
  Group root = new Group();
  root.getChildren().addAll(c,btn,btn1);
  Scene scene = new Scene(root,420,300,Color.WHEAT);
  primaryStage.setScene(scene);
  primaryStage.setTitle("EventHandler example");
  primaryStage.show();
}
public static void main(String[] args) {
  launch(args);
}
}
```



)

Kong, Shenzhen