ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

GUI Programming with JavaFX

Elearning Lectures

- JavaFx Tutorial For Beginners

https://www.youtube.com/watch?v=9YrmON6nlEw&list=PLS1QulWo1RIaUGP446_pWLgTZPiFizEMq

- Khóa học lập trình JavaFX

https://www.youtube.com/watch?v=zAq7Lmv46PE&list=PL33lvabfss1yRgFCgFXjtYaGAuDJjjH-j

Content

1. Introduction
2. JavaFX Installment
3. GUI components in JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Event handling models
7. “Drag and drop” GUI with SceneBuilder

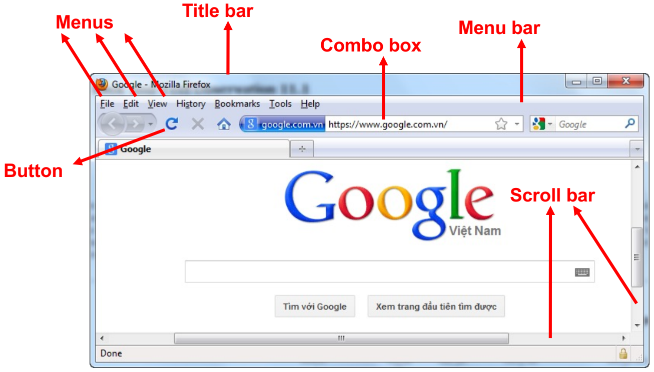
Content

1. **Introduction**
2. JavaFX Installment
3. GUI components in JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Event handling models
7. “Drag and drop” GUI with SceneBuilder

1. Introduction

- A graphical user interface - GUI (pronounced "GOO-ee"):
  - is a type of user interface
  - allows users to interact with electronic devices using images rather than text commands
- Why use term GUI?
  - The first interactive user interfaces to computers were not graphical

1. Introduction



Java APIs for GUI programming

- Two core sets of Java APIs for graphics programming:
  - AWT (Abstract Windowing Toolkit)
  - Swing
- AWT:
  - introduced in JDK 1.0
  - should be replaced by newer Swing components
- Swing:
  - enhances AWT
  - integrated into core Java since JDK 1.2
- Others:
  - Eclipse's Standard Widget Toolkit (SWT)
  - Google Web Toolkit (GWT)
  - 3D Graphics API such as Java bindings for OpenGL (JOGL) and Java3D.

JavaFX – Features

- **Written in Java** – The JavaFX library is written in Java and is available for the languages that can be executed on a JVM.
- **FXML** – JavaFX features a language known as FXML, which is a HTML like declarative markup language. The sole purpose of this language is to define a user Interface.
- **Scene Builder** – JavaFX provides an application named Scene Builder. The users can access a drag and drop design interface, which is used to develop FXML applications
- **Swing Interoperability** – In a JavaFX application, you can embed Swing content using the **Swing Node** class
- **Built-in UI controls** – JavaFX library caters UI controls using which we can develop a full-featured application.
- **CSS like Styling** – JavaFX provides a CSS like styling. By using this, you can improve the design of your application with a simple knowledge of CSS.

History of JavaFX

- JavaFX was originally developed by **Chris Oliver**, when he was working for a company named **See Beyond Technology Corporation**, which was later acquired by **Sun Microsystems** in the year 2005.
- In the year 2007, JavaFX was announced officially at **Java One**, a world wide web conference which is held yearly.
- In the year 2008, **Net Beans** integrated with JavaFX was available. In the same year, the Java **Standard Development Kit** for JavaFX 1.0 was released.
- The latest version, JavaFX8, was released as an integral part of Java on 18th of March 2014.
- 2018: JavaFX is taken out of Java SDK 11

Content

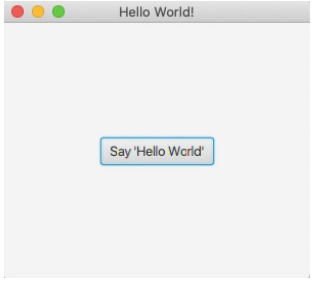
1. Introduction
2. **JavaFX Installment**
3. GUI components in JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Event handling models
7. “Drag and drop” GUI with SceneBuilder

2. JavaFX Installment

- JavaFX home page: https://openjfx.io/
- JavaFX downloading page: https://gluonhq.com/products/javafx/
- Download, decompress, copy files from the lib folder and add to project build path
- Note to run JavaFX in Eclipse
  - Go to runtime configuration and configure VM arguments as follows:
    - --module-path ${project_classpath:REPLACE_ME_WITH_YOUR_PROJECT_NAME} --add-modules javafx.controls,javafx.fxml
  - Uncheck: “Use the -XstartOnFirstThread argument when launching with SWT”

JavaFX Hello World

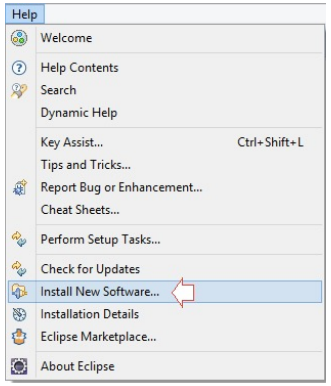
```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button btn = new Button();
        btn.setText("Say 'Hello World'");
        btn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello World!");
            }
        });

        StackPane root = new StackPane();
        root.getChildren().add(btn);

        Scene scene = new Scene(root, 300, 250);

        primaryStage.setTitle("Hello World!");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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



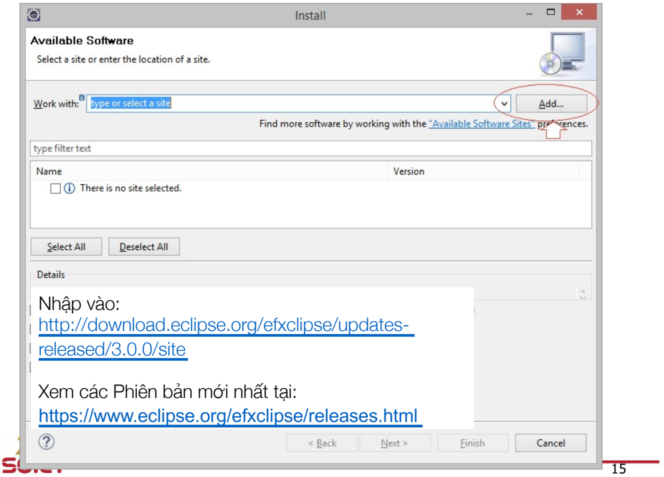
JavaFX plug-ins in Eclipse

- e(fx)clipse
  - https://www.eclipse.org/efxclipse/releases.html
  - Is an Eclipse plugin
  - Support JavaFX programming in Eclipse
- JavaFX Scene Builder
  - https://www.oracle.com/java/technologies/javafxscenebuilder-1x-archive-downloads.html
  - Is an independent tool, platform-independent, support visual programming
  - Users can drag and drop GUI components, change their properties, apply visual styles
  - Output: FXML file used in JavaFX programs

e(fx)clipse installment



e(fx)clipse installment


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the "Available Software Sites" preferences.

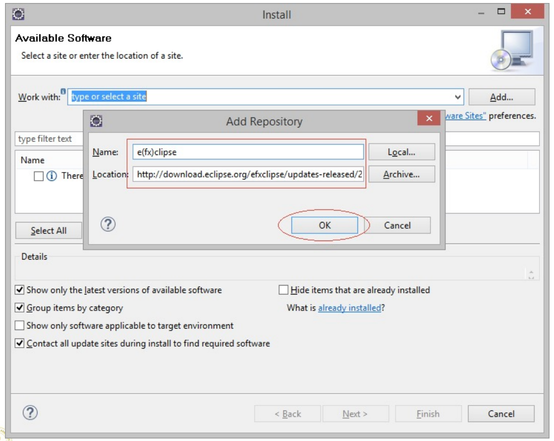
type filter text

Name	Version
□ There is no site selected.	

Nhập vào:
http://download.eclipse.org/efxclipse/updates-released/3.0.0/site

Xem các Phiên bản mới nhất tại:
https://www.eclipse.org/efxclipse/releases.html

e(fx)clipse installment


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

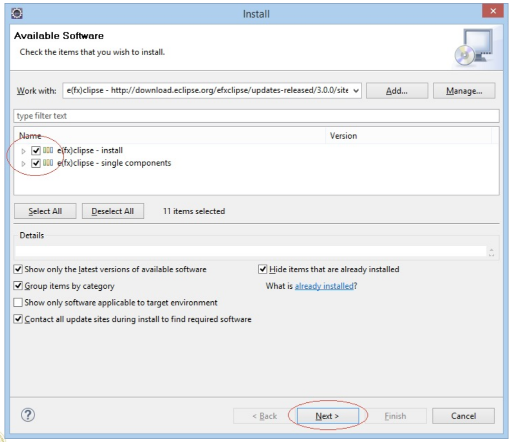
Add Repository

Name: e(fx)clipse

Location: http://download.eclipse.org/efxclipse/updates-released/2

☑ Show only the latest versions of available software
□ Hide items that are already installed
☑ Group items by category
What is already installed?
□ Show only software applicable to target environment
☑ Contact all update sites during install to find required software

e(fx)clipse installment



Available Software

Check the items that you wish to install.

Work with: e(fx)clipse - http://download.eclipse.org/efxclipse/updates-released/3.0.0/site

Name	Version
☑ e(fx)clipse - install	
☑ e(fx)clipse - single components	

Select All | Deselect All | 11 items selected

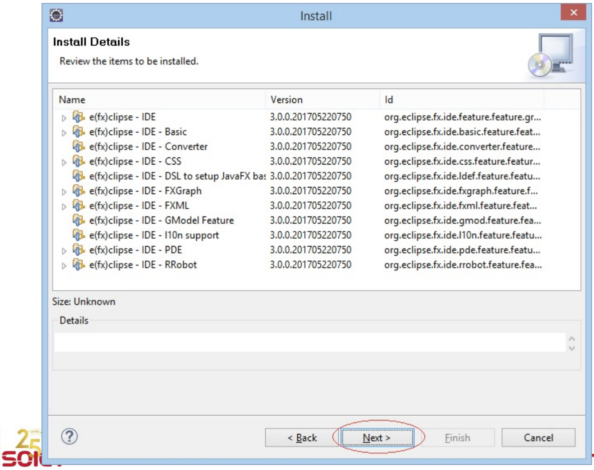
Details

- ☑ Show only the latest versions of available software
- ☑ Group items by category
- □ Show only software applicable to target environment
- ☑ Contact all update sites during install to find required software
- ☑ Hide items that are already installed

What is already installed?

17

e(fx)clipse installment



Install Details

Review the items to be installed.

Name	Version	Id
e(fx)clipse - IDE	3.0.0.201705220750	org.eclipse.fx.ide.feature.feature.gr...
e(fx)clipse - IDE - Basic	3.0.0.201705220750	org.eclipse.fx.ide.basic.feature.feat...
e(fx)clipse - IDE - Converter	3.0.0.201705220750	org.eclipse.fx.ide.converter.feature...
e(fx)clipse - IDE - CSS	3.0.0.201705220750	org.eclipse.fx.ide.css.feature.featur...
e(fx)clipse - IDE - DSL to setup JavaFX bas	3.0.0.201705220750	org.eclipse.fx.ide.ldef.feature.featu...
e(fx)clipse - IDE - FXGraph	3.0.0.201705220750	org.eclipse.fx.ide.fxgraph.feature.f...
e(fx)clipse - IDE - FXML	3.0.0.201705220750	org.eclipse.fx.ide.fxml.feature.feat...
e(fx)clipse - IDE - GModel Feature	3.0.0.201705220750	org.eclipse.fx.ide.gmod.feature.fea...
e(fx)clipse - IDE - I10n support	3.0.0.201705220750	org.eclipse.fx.ide.l10n.feature.featu...
e(fx)clipse - IDE - PDE	3.0.0.201705220750	org.eclipse.fx.ide.pde.feature.featu...
e(fx)clipse - IDE - RRobot	3.0.0.201705220750	org.eclipse.fx.ide.rrobot.feature.fea...

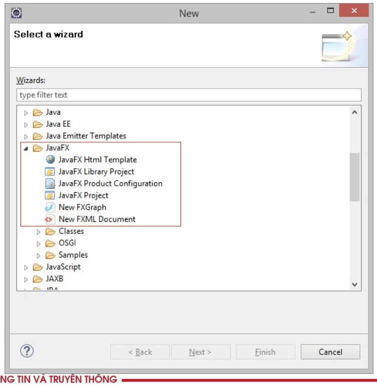
Size: Unknown

Details

18

e(fx)clipse installment

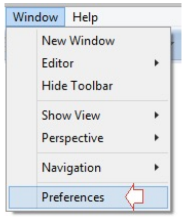
- After the instamment, we restart **Eclipse**, go to menu **File/New/Others** ... we will see **JavaFX Wizards**



19

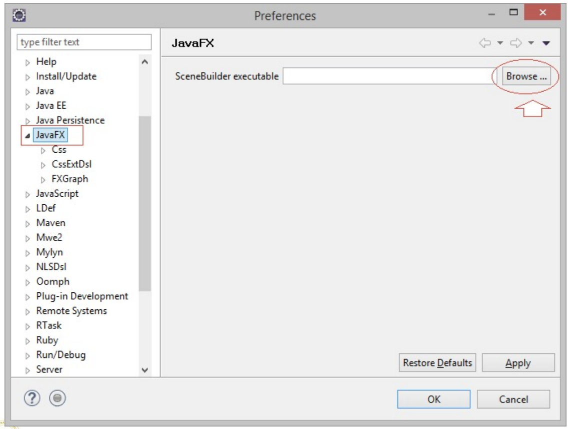
Add JavaFX Scene Builder to Eclipse

- Download and install JavaFX Scene Builder
- Open Eclipse, go to Window/Preferences



20

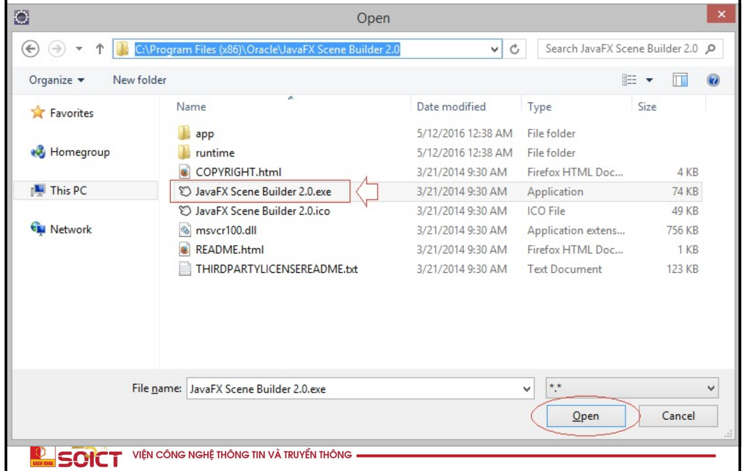
Add JavaFX Scene Builder to Eclipse



SceneBuilder executable

21

Add JavaFX Scene Builder to Eclipse


Open
C:\Program Files (x86)\Oracle\JavaFX Scene Builder 2.0
Search JavaFX Scene Builder 2.0
Organize
New folder
Favorites
Homegroup
This PC
Network
Open
Cancel

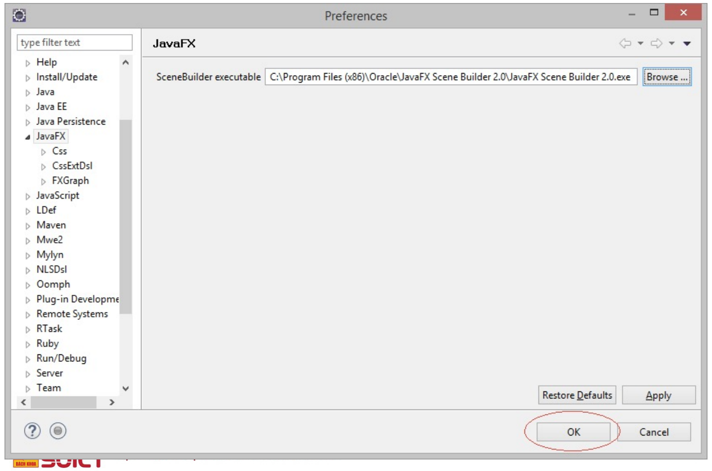
Name	Date modified	Type	Size
app	5/12/2016 12:38 AM	File folder	
runtime	5/12/2016 12:38 AM	File folder	
COPYRIGHT.html	3/21/2014 9:30 AM	Firefox HTML Doc...	4 KB
JavaFX Scene Builder 2.0.exe	3/21/2014 9:30 AM	Application	74 KB
JavaFX Scene Builder 2.0.ico	3/21/2014 9:30 AM	ICO File	49 KB
msvcr100.dll	3/21/2014 9:30 AM	Application extens...	756 KB
README.html	3/21/2014 9:30 AM	Firefox HTML Doc...	1 KB
THIRDPARTYLICENSEREADME.txt	3/21/2014 9:30 AM	Text Document	123 KB

File name: JavaFX Scene Builder 2.0.exe

.

22

Add JavaFX Scene Builder to Eclipse



SceneBuilder executable C:\Program Files (x86)\Oracle\JavaFX Scene Builder 2.0\JavaFX Scene Builder 2.0.exe

23

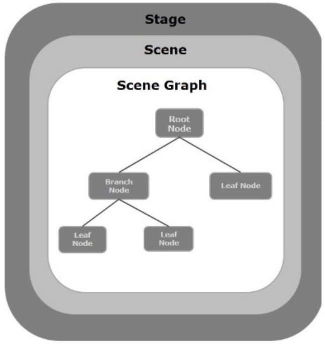
Content

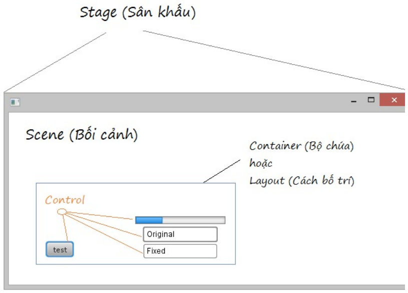
1. Introduction
2. JavaFX Installment
3. **GUI components in JavaFX**
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Event handling models
7. “Drag and drop” GUI with SceneBuilder

24

3. GUI components in JavaFX

- 3 main concepts of JavaFX applications: Stage, Scene, and Nodes



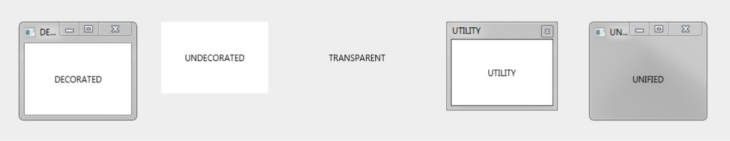


Stage

- A stage (a window) contains all the objects of a JavaFX application.
- It is represented by **Stage** class of the package **javafx.stage**
- **Stage** object is passed as an argument to the **start()** method of the **Application** class (See the **HelloWorld** JavaFX example)
- A stage has two parameters determining its position namely **Width** and **Height**.
- It is divided as Content Area and Decorations (Title Bar and Borders).
- You have to call the **show()** method to display the contents of a stage.
- There are five types of stages available: Decorated, Undecorated, Transparent, Unified, Utility

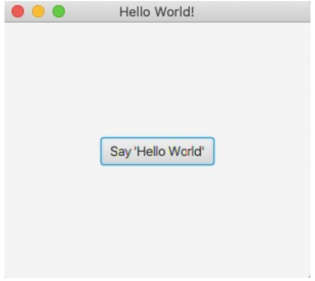
Stage – set the style

```
stage.initStyle(StageStyle.DECORATED);
//stage.initStyle(StageStyle.UNDECORATED);
//stage.initStyle(StageStyle.TRANSPARENT);
//stage.initStyle(StageStyle.UNIFIED);
//stage.initStyle(StageStyle.UTILITY);
```



JavaFX Hello World

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.stage.Stage;

public class HelloWorld extends Application {
   @Override
   public void start(Stage primaryStage) {
      Button btn = new Button();
      btn.setText("Say 'Hello World'");
      btn.setOnAction(new EventHandler<ActionEvent>() {
         @Override
         public void handle(ActionEvent event) {
            System.out.println("Hello World!");
         }
      });

      StackPane root = new StackPane();
      root.getChildren().add(btn);

      Scene scene = new Scene(root, 300, 250);

      primaryStage.setTitle("Hello World!");
      primaryStage.setScene(scene);
      primaryStage.show();
   }

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



Scene

- A scene represents the physical contents of a JavaFX application. It contains all the contents of a scene graph.
- The class **Scene** of the package **javafx.scene** represents the scene object.
- At an instance, the scene object is added to only one stage.
- You can create a scene by instantiating the Scene Class. You can opt for the size of the scene by passing its dimensions (height and width) along with the **root node** to its constructor.
  - Scene(Parent root)
  - Scene(Parent root, double width, double height)
  - …

Scene Graph and Nodes

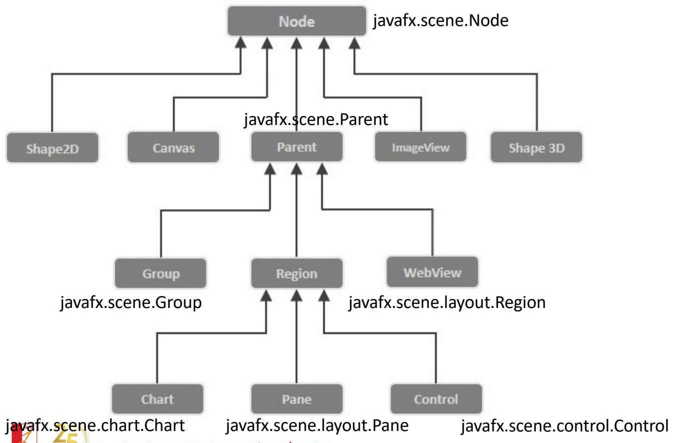
- **Scene graph**: is a tree-like data structure (hierarchical) representing the contents of a scene.
- In contrast, a **node** is a visual/graphical object of a scene graph. A **node** may include –
  - Geometrical (Graphical) objects (2D and 3D) such as – Circle, Rectangle, Polygon, etc.
  - UI Controls such as – Button, Checkbox, Choice Box, Text Area, etc.
  - Containers (Layout Panes) such as Border Pane, Grid Pane, Flow Pane, etc.
  - Media elements such as Audio, Video and Image Objects
- The **Node** Class of the package **javafx.scene** represents a node in JavaFX, this class is the super class of all the nodes

Scene Graph and Nodes

A node is of three types

- **Root Node** – The first Scene Graph is known as the Root node.
- **Branch Node/Parent Node** – The node with child nodes are known as branch/parent nodes. The abstract class named **Parent** of the package **javafx.scene** is the base class of all the parent nodes, and those parent nodes will be of the following types –
  - **Group** – A group node is a collective node that contains a list of children nodes. Whenever the group node is rendered, all its child nodes are rendered in order. Any transformation, effect state applied on the group will be applied to all the child nodes.
  - **Region** – It is the base class of all the JavaFX Node based UI Controls, such as Chart, Pane and Control.
  - **WebView** – This node manages the web engine and displays its contents.
- **Leaf Node** – The node without child nodes is known as the leaf node. Rectangle, Ellipse, Box, ImageView, MediaView are examples of leaf nodes.

JavaFX node class hierarchy



Node javafx.scene.Node

Shape2D

Canvas

javafx.scene.Parent

Parent

ImageView

Shape 3D

Group

javafx.scene.Group

Region

WebView

javafx.scene.layout.Region

Chart

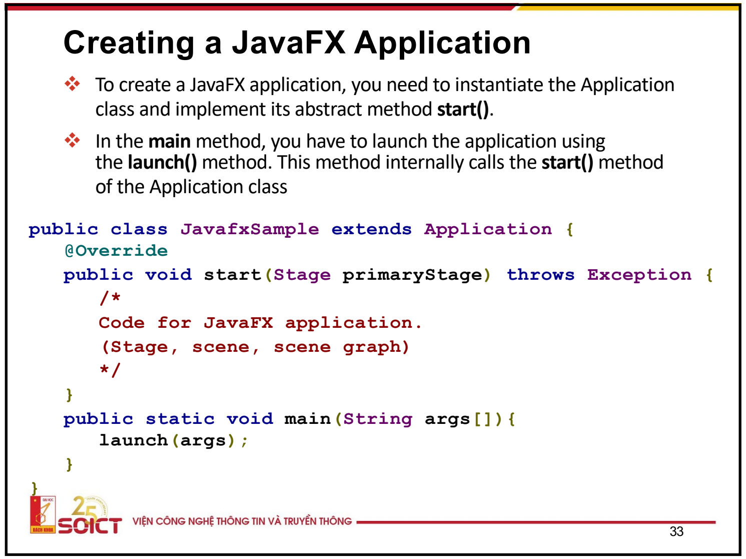
javafx.scene.chart.Chart

Pane

javafx.scene.layout.Pane

Control

javafx.scene.control.Control

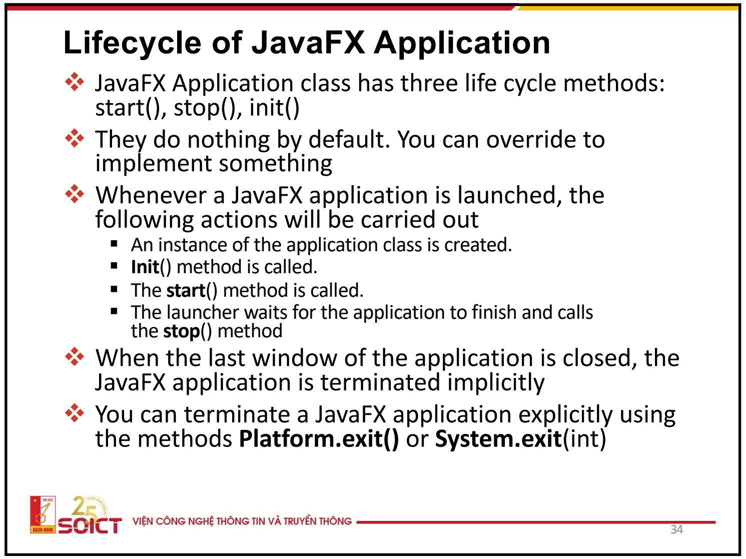


Creating a JavaFX Application

- To create a JavaFX application, you need to instantiate the Application class and implement its abstract method **start()**.
- In the **main** method, you have to launch the application using the **launch()** method. This method internally calls the **start()** method of the Application class

```
public class JavafxSample extends Application {
   @Override
   public void start(Stage primaryStage)  throws Exception {
      /*
      Code for JavaFX application.
      (Stage, scene, scene graph)
      */
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

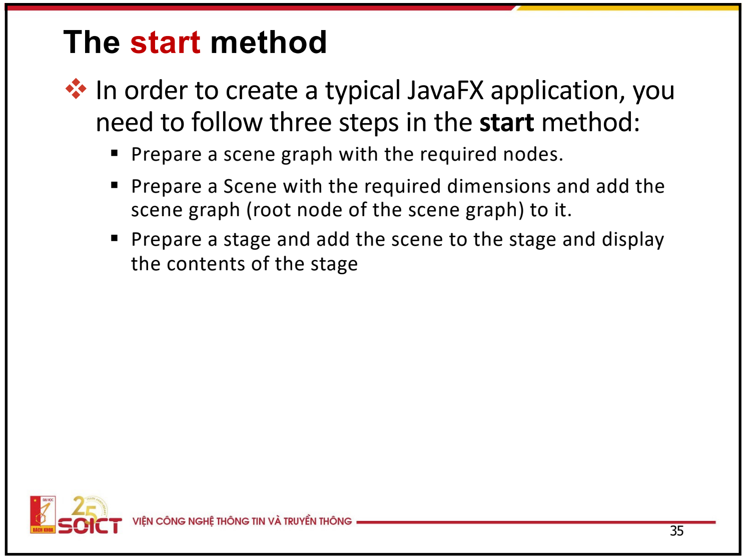
33



Lifecycle of JavaFX Application

- JavaFX Application class has three life cycle methods: start(), stop(), init()
- They do nothing by default. You can override to implement something
- Whenever a JavaFX application is launched, the following actions will be carried out
  - An instance of the application class is created.
  - **Init**() method is called.
  - The **start**() method is called.
  - The launcher waits for the application to finish and calls the **stop**() method
- When the last window of the application is closed, the JavaFX application is terminated implicitly
- You can terminate a JavaFX application explicitly using the methods **Platform.exit()** or **System.exit**(int)

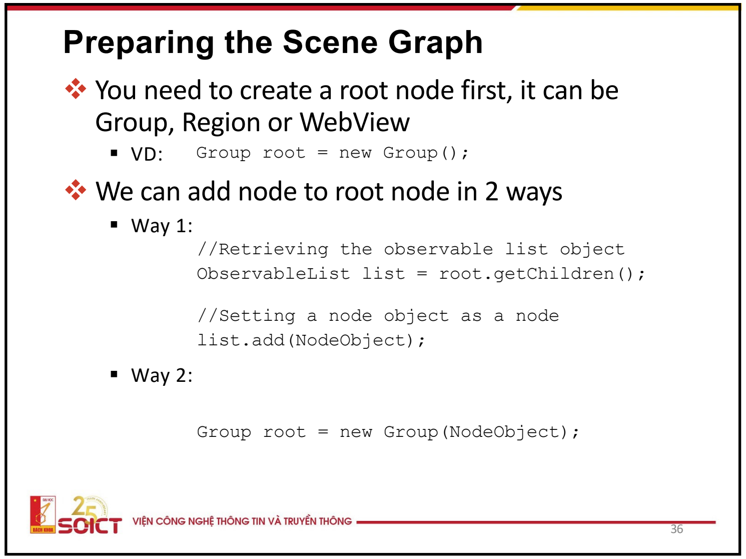
34



The start method

- In order to create a typical JavaFX application, you need to follow three steps in the **start** method:
  - Prepare a scene graph with the required nodes.
  - Prepare a Scene with the required dimensions and add the scene graph (root node of the scene graph) to it.
  - Prepare a stage and add the scene to the stage and display the contents of the stage

35



Preparing the Scene Graph

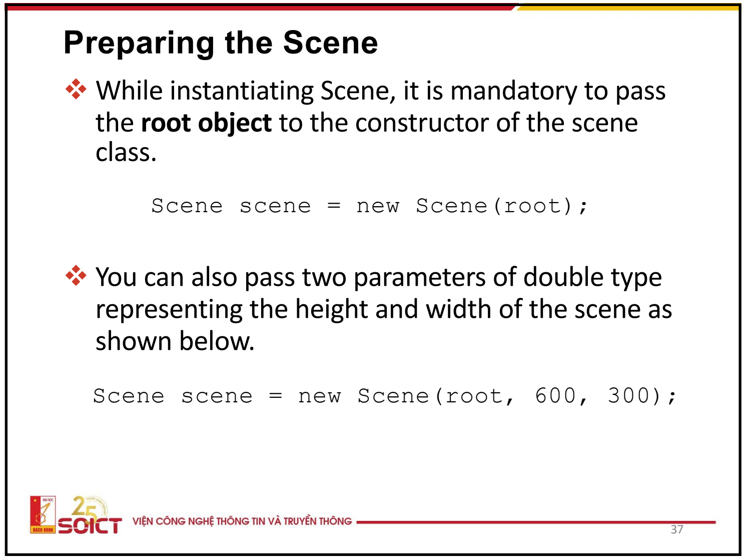
- You need to create a root node first, it can be Group, Region or WebView
  - VD: `Group root = new Group();`
- We can add node to root node in 2 ways
  - Way 1:

```
//Retrieving the observable list object
ObservableList list = root.getChildren();

//Setting a node object as a node
list.add(NodeObject);
```

  - Way 2:

```
Group root = new Group(NodeObject);
```

36



Preparing the Scene

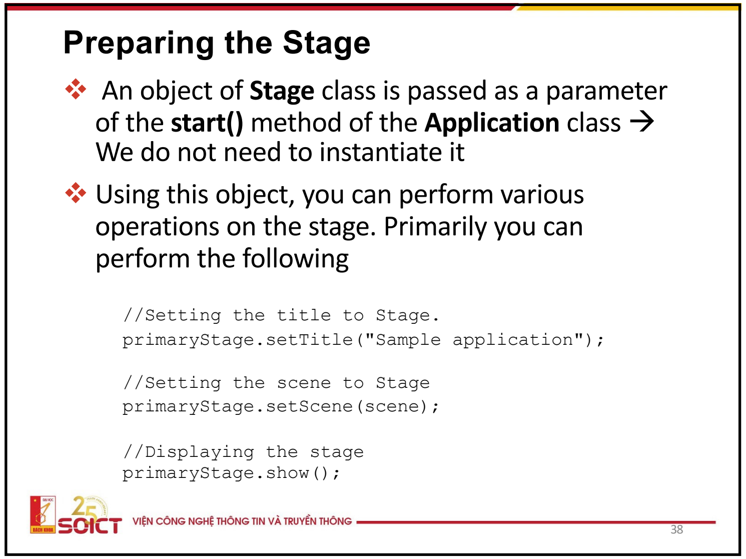
- While instantiating Scene, it is mandatory to pass the **root object** to the constructor of the scene class.

```
Scene scene = new Scene(root);
```

- You can also pass two parameters of double type representing the height and width of the scene as shown below.

```
Scene scene = new Scene(root, 600, 300);
```

37

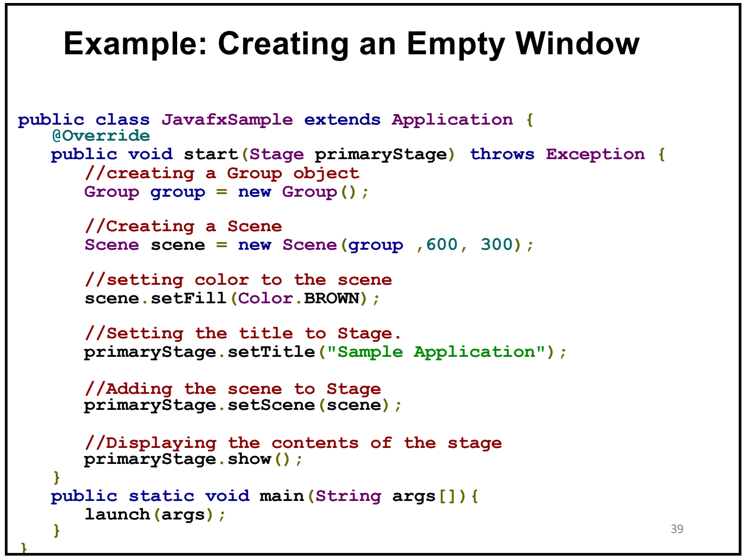


Preparing the Stage

- An object of **Stage** class is passed as a parameter of the **start()** method of the **Application** class → We do not need to instantiate it
- Using this object, you can perform various operations on the stage. Primarily you can perform the following

```
//Setting the title to Stage.
primaryStage.setTitle("Sample application");

//Setting the scene to Stage
primaryStage.setScene(scene);

//Displaying the stage
primaryStage.show();
```

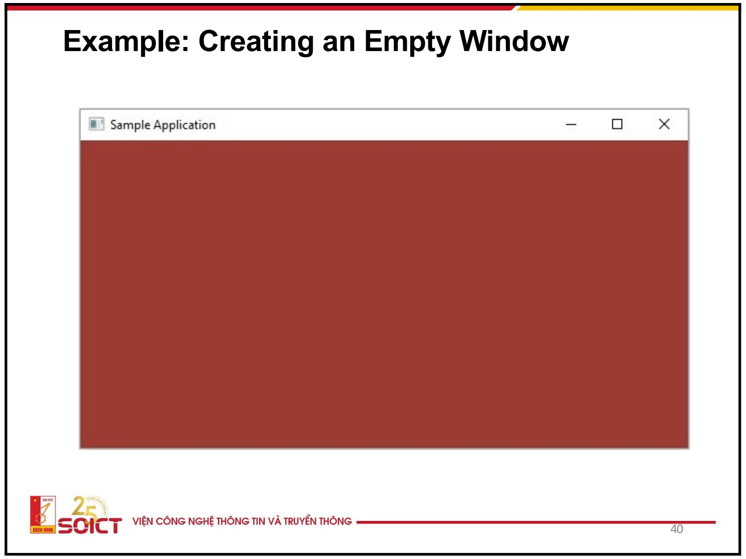
38



Example: Creating an Empty Window

```
public class JavafxSample extends Application {
   @Override
   public void start(Stage primaryStage) throws Exception {
      //creating a Group object
      Group group = new Group();

      //Creating a Scene
      Scene scene = new Scene(group ,600, 300);

      //setting color to the scene
      scene.setFill(Color.BROWN);

      //Setting the title to Stage.
      primaryStage.setTitle("Sample Application");

      //Adding the scene to Stage
      primaryStage.setScene(scene);

      //Displaying the contents of the stage
      primaryStage.show();
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

39



Example: Creating an Empty Window

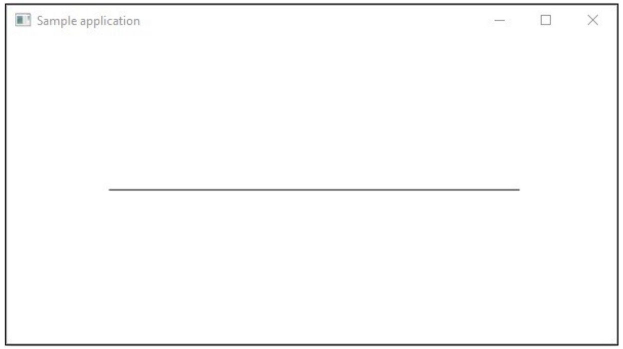
40

Example: Drawing a Straight Line

```
public class DrawingLine extends Application{
   @Override
   public void start(Stage stage) {
      //Creating a line object
      Line line = new Line();

      //Setting the properties to a line
      line.setStartX(100.0);
      line.setStartY(150.0);
      line.setEndX(500.0);
      line.setEndY(150.0);

      //Creating a Group
      Group root = new Group(line);

      //Creating a Scene
      Scene scene = new Scene(root, 600, 300);

      //Setting title to the scene
      stage.setTitle("Sample application");

      //Adding the scene to the stage
      stage.setScene(scene);

      //Displaying the contents of a scene
      stage.show();
   }
   public static void main(String args[]){
      launch(args);
   }
```

41

Example: Drawing a Straight Line



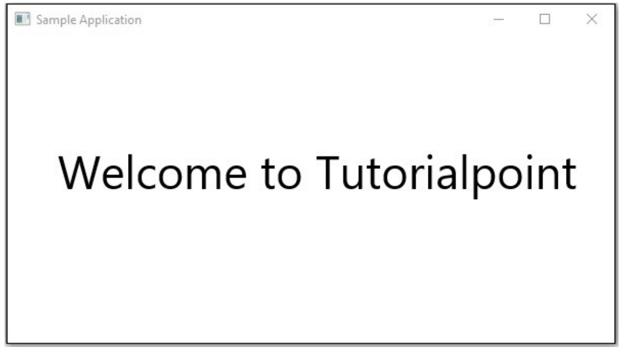
42

Example: Displaying Text

```
public class DisplayingText extends Application {
   @Override
   public void start(Stage stage) {
      //Creating a Text object
      Text text = new Text();

      //Setting font to the text
      text.setFont(new Font(45));

      //setting the position of the text
      text.setX(50);
      text.setY(150);

      //Setting the text to be added.
      text.setText("Welcome to Tutorialspoint");

      //Creating a Group object
      Group root = new Group();

      //Retrieving the observable list object
      ObservableList list = root.getChildren();

      //Setting the text object as a node to the group object
      list.add(text);

      //Creating a scene object
      Scene scene = new Scene(root, 600, 300);

      //Setting title to the Stage
      stage.setTitle("Sample Application");

      //Adding scene to the stage
      stage.setScene(scene);

      //Displaying the contents of the stage
      stage.show();
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

43

Example: Displaying Text



Welcome to Tutorialpoint

44

Example: Text decoration

```
public class DecorationsExample extends Application {
   @Override
   public void start(Stage stage) {
      //Creating a Text Example object
      Text text1 = new Text("Hi how are you");

      //Setting font to the text
      text1.setFont(
         Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 20)
      );

      //setting the position of the text
      text1.setX(50);
      text1.setY(75);

      //Striking through the text
      text1.setStrikethrough(true);

      //Creating a Text Example object
      Text text2 = new Text("Welcome to Tutorialspoint");

      //Setting font to the text
      text2.setFont(
         Font.font("verdana", FontWeight.BOLD, FontPosture.REGULAR, 20)
      );
```

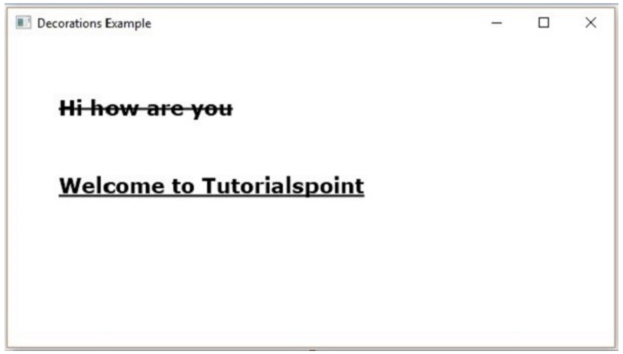
45

Example: Text decoration

```
      //setting the position of the text
      text2.setX(50);
      text2.setY(150);

      //underlining the text
      text2.setUnderline(true);

      //Creating a Group object
      Group root = new Group(text1, text2);

      //Creating a scene object
      Scene scene = new Scene(root, 600, 300);

      //Setting title to the Stage
      stage.setTitle("Decorations Example");

      //Adding scene to the stage
      stage.setScene(scene);

      //Displaying the contents of the stage
      stage.show();
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

46

Example: Text decoration



Decorations Example

~~Hi how are you~~

Welcome to Tutorialspoint

47

Content

1. Introduction
2. JavaFX Installment
3. GUI components in JavaFX
4. **JavaFX - UI controls**
5. JavaFX - Layout Panes
6. Event handling models
7. “Drag and drop” GUI with SceneBuilder

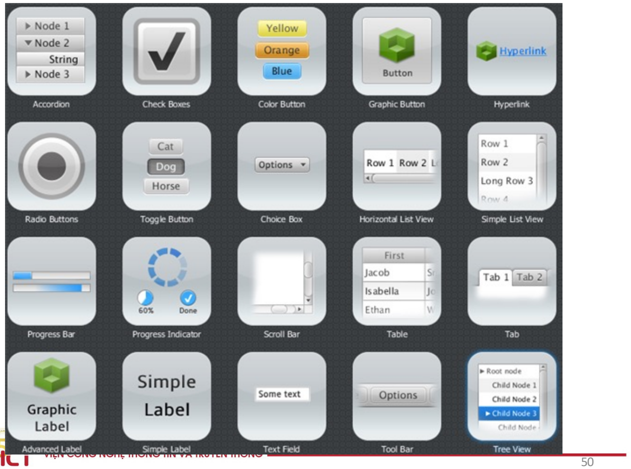
48

4. Java FX - UI Controls

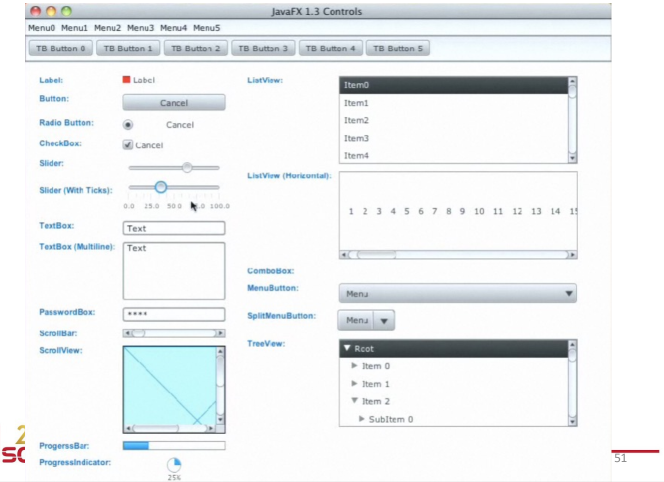
Every user interface considers the following three main aspects:

- **UI elements** – These are the core visual elements which the user eventually sees and interacts with. JavaFX provides a huge list of widely used and common elements varying from basic to complex
- **Layouts** – They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface).
- **Behavior** – These are events which occur when the user interacts with UI elements.

4. Java FX - UI Controls

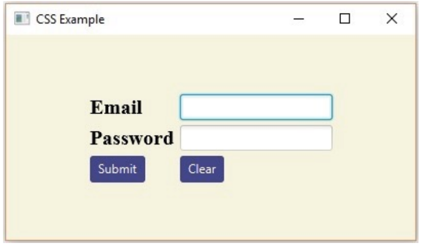


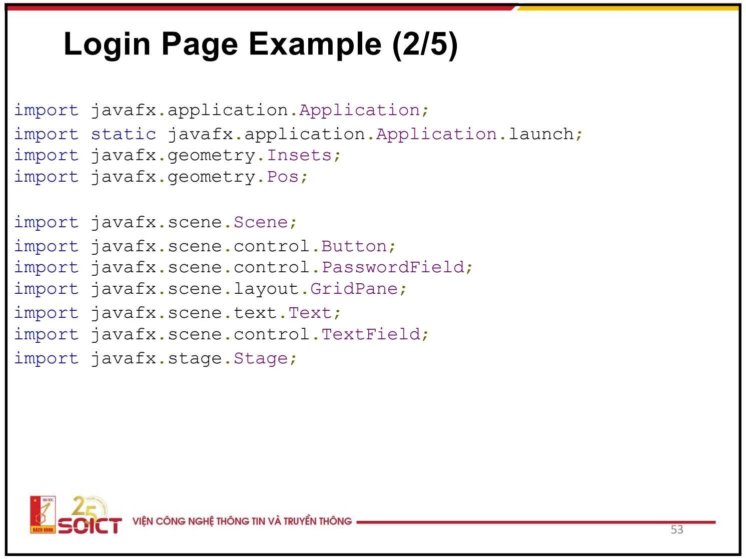
4. Java FX - UI Controls



Login Page Example (1/5)

- Develop a JavaFX application as shown belows:

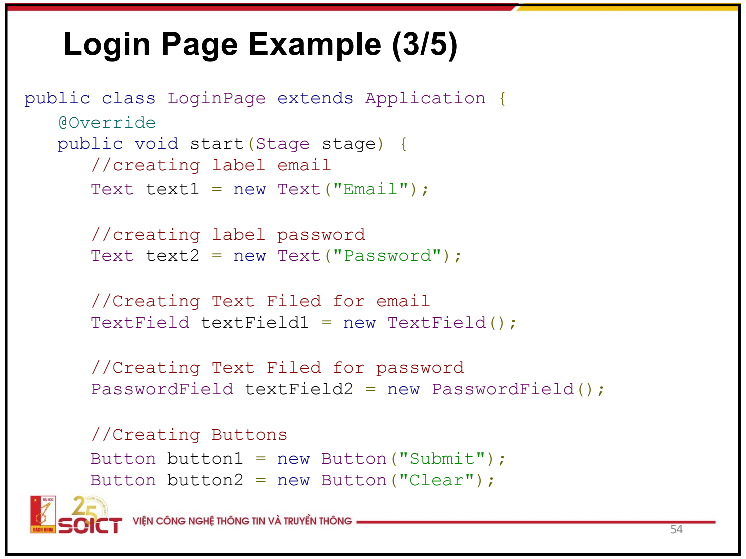




Login Page Example (2/5)

```java
import javafx.application.Application;
import static javafx.application.Application.launch;
import javafx.geometry.Insets;
import javafx.geometry.Pos;

import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.layout.GridPane;
import javafx.scene.text.Text;
import javafx.scene.control.TextField;
import javafx.stage.Stage;
```

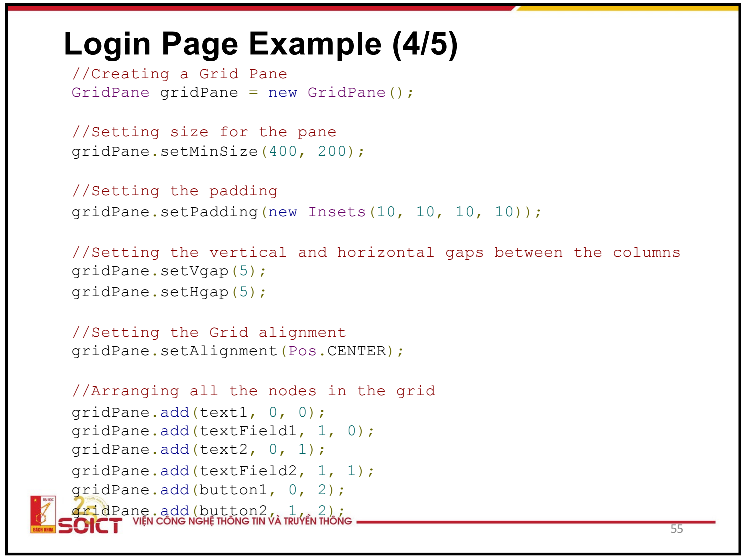
53



Login Page Example (3/5)

```java
public class LoginPage extends Application {
   @Override
   public void start(Stage stage) {
      //creating label email
      Text text1 = new Text("Email");

      //creating label password
      Text text2 = new Text("Password");

      //Creating Text Filed for email
      TextField textField1 = new TextField();

      //Creating Text Filed for password
      PasswordField textField2 = new PasswordField();

      //Creating Buttons
      Button button1 = new Button("Submit");
      Button button2 = new Button("Clear");
```

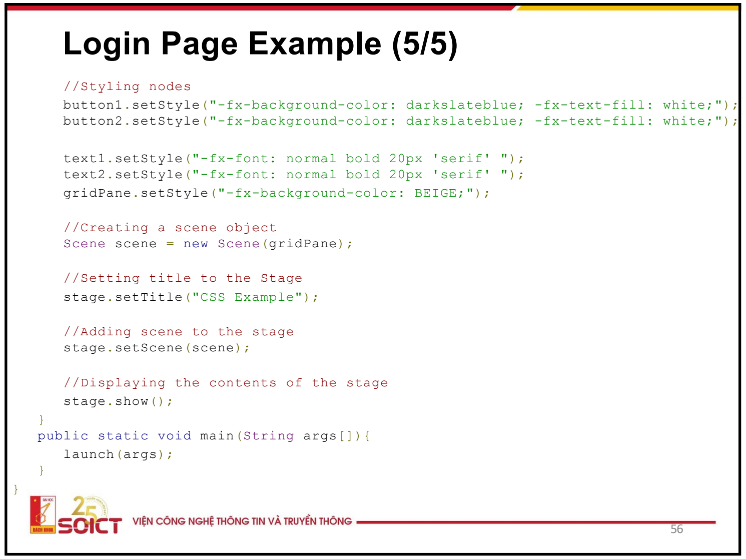
54



Login Page Example (4/5)

```java
//Creating a Grid Pane
GridPane gridPane = new GridPane();

//Setting size for the pane
gridPane.setMinSize(400, 200);

//Setting the padding
gridPane.setPadding(new Insets(10, 10, 10, 10));

//Setting the vertical and horizontal gaps between the columns
gridPane.setVgap(5);
gridPane.setHgap(5);

//Setting the Grid alignment
gridPane.setAlignment(Pos.CENTER);

//Arranging all the nodes in the grid
gridPane.add(text1, 0, 0);
gridPane.add(textField1, 1, 0);
gridPane.add(text2, 0, 1);
gridPane.add(textField2, 1, 1);
gridPane.add(button1, 0, 2);
gridPane.add(button2, 1, 2);
```

55



Login Page Example (5/5)

```java
      //Styling nodes
      button1.setStyle("-fx-background-color: darkslateblue; -fx-text-fill: white;");
      button2.setStyle("-fx-background-color: darkslateblue; -fx-text-fill: white;");

      text1.setStyle("-fx-font: normal bold 20px 'serif' ");
      text2.setStyle("-fx-font: normal bold 20px 'serif' ");
      gridPane.setStyle("-fx-background-color: BEIGE;");

      //Creating a scene object
      Scene scene = new Scene(gridPane);

      //Setting title to the Stage
      stage.setTitle("CSS Example");

      //Adding scene to the stage
      stage.setScene(scene);

      //Displaying the contents of the stage
      stage.show();
   }
   public static void main(String args[]){
      launch(args);
   }
}
```

56

Content

1. Introduction
2. JavaFX Installment
3. GUI components in JavaFX
4. JavaFX - UI controls
5. **JavaFX - Layout Panes**
6. Event handling models
7. “Drag and drop” GUI with SceneBuilder

5. JavaFX - Layout Panes (Container)

- After constructing all the required nodes in a scene, we will generally arrange them in order.
- This arrangement of the components within the container is called the Layout of the container.
- JavaFX provides several predefined layouts such as **HBox, VBox, Border Pane, Stack Pane, Text Flow, Anchor Pane, Title Pane, Grid Pane, Flow Panel**, etc.
- Each of the above mentioned layout is represented by a class and all these classes belongs to the package **javafx.layout**. The class named **Pane** is the base class of all the layouts in JavaFX.

Creating a Layout

- To create a layout, you need to
  - Create node.
  - Instantiate the respective class of the required layout.
  - Set the properties of the layout.
  - Add all the created nodes to the layout.

HBox layout

- HBox layout: all the nodes are set in a single horizontal row.
- Important properties:
  - **alignment** – represents the alignment of the nodes in the bounds of the HBox.
  - **spacing** – is of double type and represents the space between the children of the HBox.
- Khởi tạo HBox

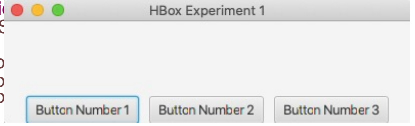
```
// Khởi tạo rỗng
HBox hbox = new HBox();

// Khởi tạo với các node
Button button1 = new Button("Button Number 1");
Button button2 = new Button("Button Number 2");
HBox hbox = new HBox(button1, button2);
```

HBox layout example

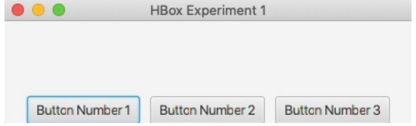
```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class HBoxExperiments extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");

        Button button1 = new Button("Button Number 1");
        Button button2 = new Button("Button Number 2");
        Button button3 = new Button("Button Number 3");

        HBox hbox = new HBox(button1, button2);
        hbox.setSpacing(10);
        hbox.setAlignment(Pos.BOTTOM_CENTER);
        hbox.getChildren().add(button3);

        Scene scene = new Scene(hbox, 400, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

HBox layout example

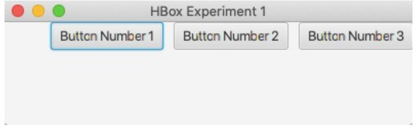
```
import javafx.application.Application;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
public class HBoxExperiments extends Application {
    @Override
    public voi
        primaryS

        Button b
        Button b
        Button b
        HBox hbox = new HBox(button1, button2);
        hbox.setSpacing(10);
        hbox.setAlignment(Pos.BOTTOM_CENTER);
        hbox.getChildren().add(button3);

        Scene scene = new Scene(hbox, 400, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



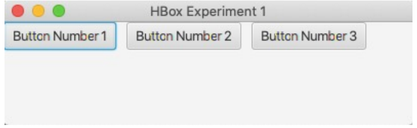
HBox layout example



BOTTOM_CENTER



BASELINE_RIGHT

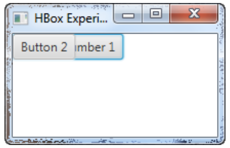


TOP_LEFT

Group layout

- Group layout do not arrange its components. All are in (0, 0)

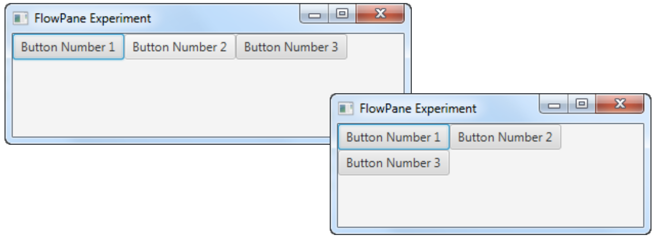
```
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
public class GroupExperiments extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("HBox Experiment 1");

        Button button1 = new Button("Button Number 1");
        Button button2 = new Button("Button 2");

        Group group = new Group();
        group.getChildren().add(button1);
        group.getChildren().add(button2);

        Scene scene = new Scene(group, 200, 100);
        primaryStage.setScene(scene);
        primaryStage.show();
    }
```



2 button đều ở tọa độ (0, 0), đè lên nhau

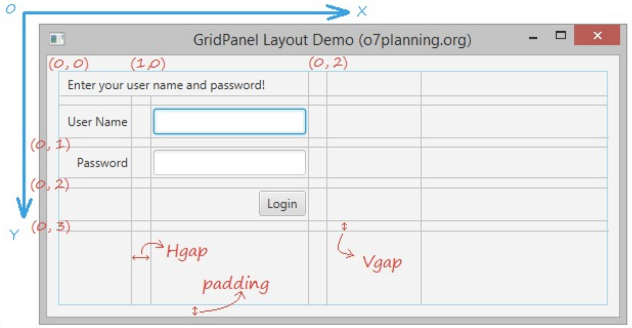
Other layouts

- **FlowPane**: wraps all the nodes in a flow. A horizontal flow pane wraps the elements of the pane at its height, while a vertical flow pane wraps the elements at its width



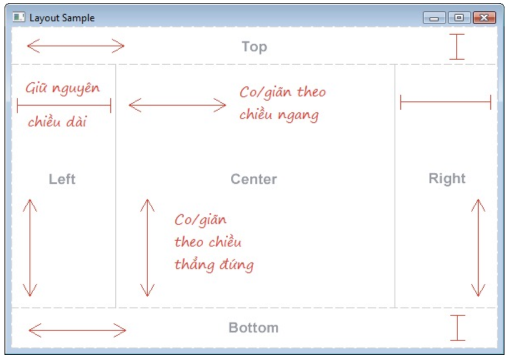
Other layouts

- **GridPane** layout: arranges the nodes in our application as a grid of rows and columns. This layout comes handy while creating forms using JavaFX.



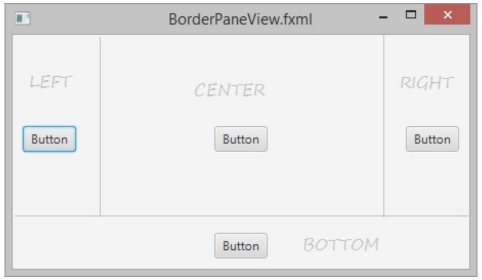
Other layouts

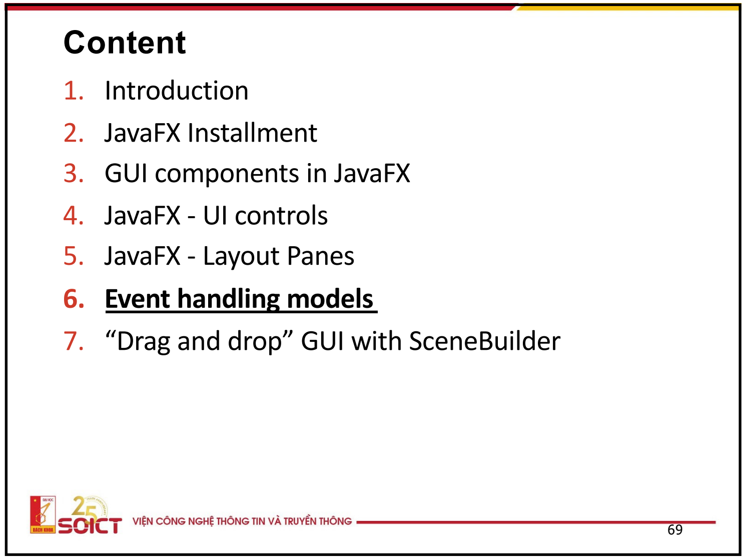
- **BorderPane** layout: arranges the nodes in our application in top, left, right, bottom and center positions.



Other layouts

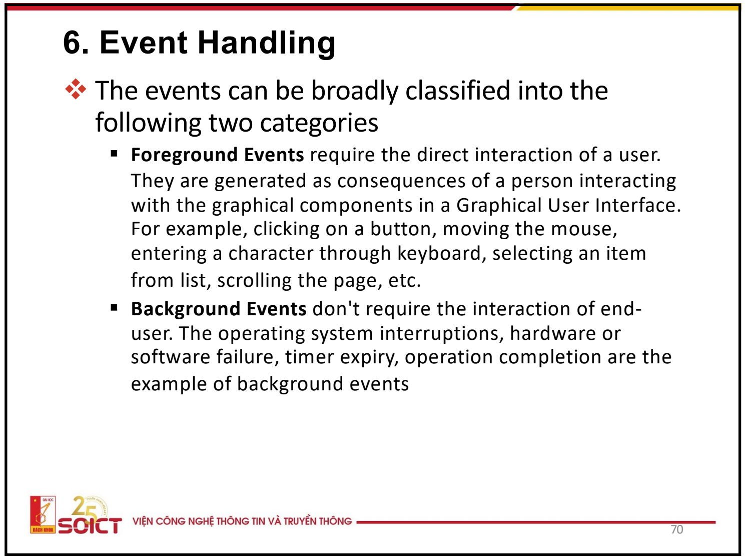
- BorderPane: if an area is empty, the other will take its space
- Example: empty TOP area





Content

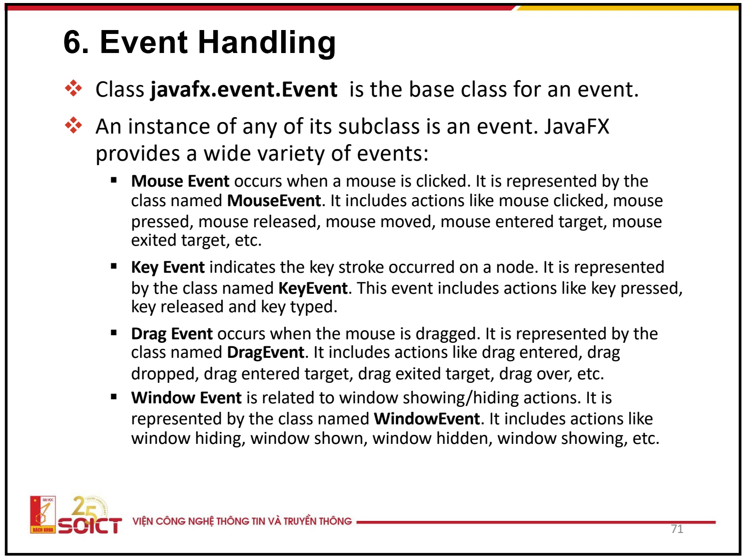
1. Introduction
2. JavaFX Installment
3. GUI components in JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. **Event handling models**
7. “Drag and drop” GUI with SceneBuilder



6. Event Handling

❖ The events can be broadly classified into the following two categories

- **Foreground Events** require the direct interaction of a user. They are generated as consequences of a person interacting with the graphical components in a Graphical User Interface. For example, clicking on a button, moving the mouse, entering a character through keyboard, selecting an item from list, scrolling the page, etc.
- **Background Events** don't require the interaction of end-user. The operating system interruptions, hardware or software failure, timer expiry, operation completion are the example of background events

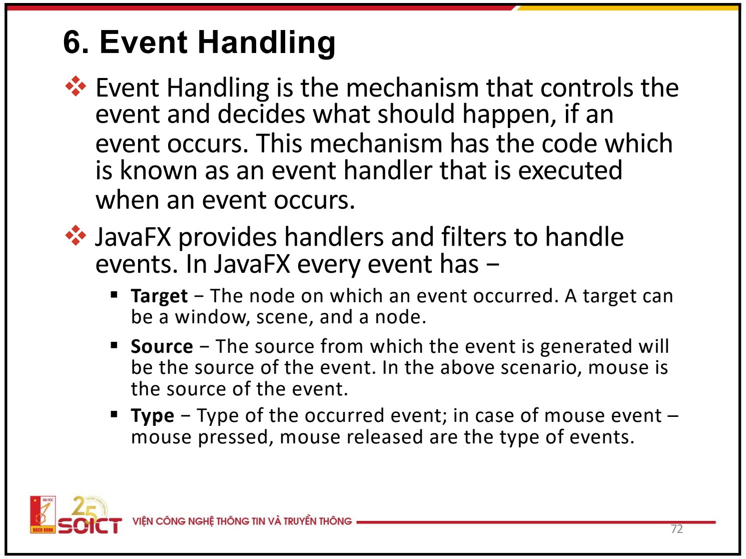


6. Event Handling

❖ Class **javafx.event.Event** is the base class for an event.

❖ An instance of any of its subclass is an event. JavaFX provides a wide variety of events:

- **Mouse Event** occurs when a mouse is clicked. It is represented by the class named **MouseEvent**. It includes actions like mouse clicked, mouse pressed, mouse released, mouse moved, mouse entered target, mouse exited target, etc.
- **Key Event** indicates the key stroke occurred on a node. It is represented by the class named **KeyEvent**. This event includes actions like key pressed, key released and key typed.
- **Drag Event** occurs when the mouse is dragged. It is represented by the class named **DragEvent**. It includes actions like drag entered, drag dropped, drag entered target, drag exited target, drag over, etc.
- **Window Event** is related to window showing/hiding actions. It is represented by the class named **WindowEvent**. It includes actions like window hiding, window shown, window hidden, window showing, etc.



6. Event Handling

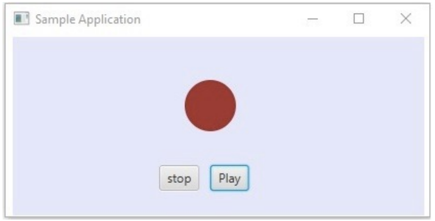
❖ Event Handling is the mechanism that controls the event and decides what should happen, if an event occurs. This mechanism has the code which is known as an event handler that is executed when an event occurs.

❖ JavaFX provides handlers and filters to handle events. In JavaFX every event has –

- **Target** – The node on which an event occurred. A target can be a window, scene, and a node.
- **Source** – The source from which the event is generated will be the source of the event. In the above scenario, mouse is the source of the event.
- **Type** – Type of the occurred event; in case of mouse event – mouse pressed, mouse released are the type of events.

Example

- If we click on the play button, the source will be the mouse, the target node will be the play button and the type of the event generated is the mouse click.



Event Delivery Process

- The event delivery process contains the following steps:
  - Target selection
  - Route construction
  - Event capturing
  - Event bubbling

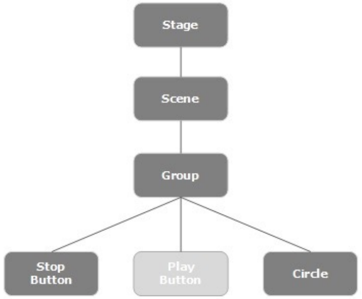
https://docs.oracle.com/javafx/2/events/processing.htm

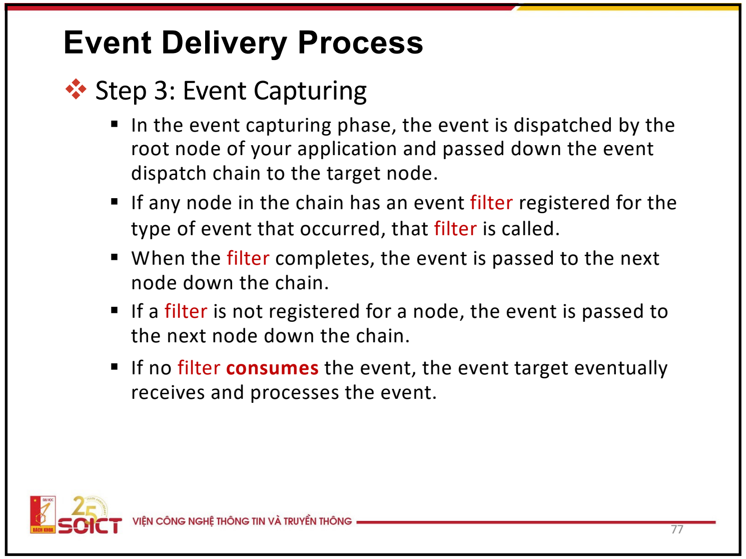
Event Delivery Process

- Step 1: Target selection (identify target node). When an action occurs, the system determines which node is the target based on internal rules:
  - For key events, the target is the node that has focus.
  - For mouse events, the target is the node at the location of the cursor. For synthesized mouse events, the touch point is considered the location of the cursor.
  - For swipe events that are generated by a swipe on a touch screen, the target is the node at the center of the entire path of all of the fingers. For indirect swipe events, the target is the node at the location of the cursor.
  - For touch events, the default target for each touch point is the node at the location first pressed. A different target can be specified using the ungrab(), grab(), or grab(*node*) methods for a touch point in an event filter or event handler.

Event Delivery Process

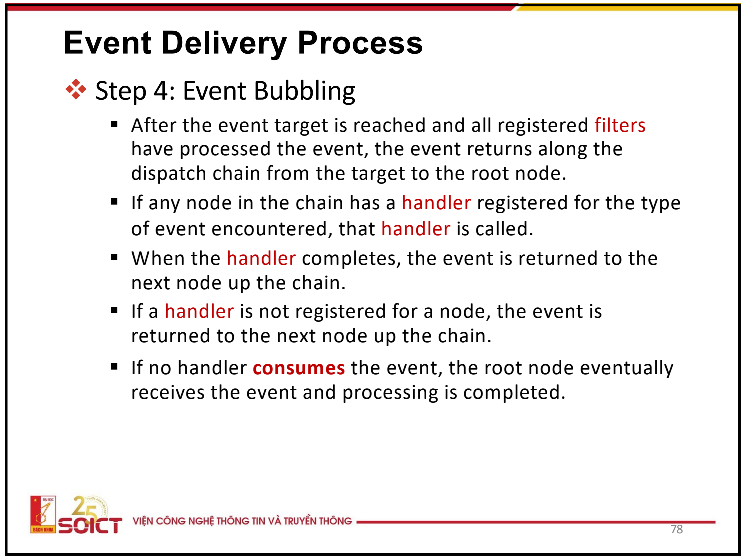
- Step 2: Route Construction – create the Event Dispatch chain: the route from stage to target node





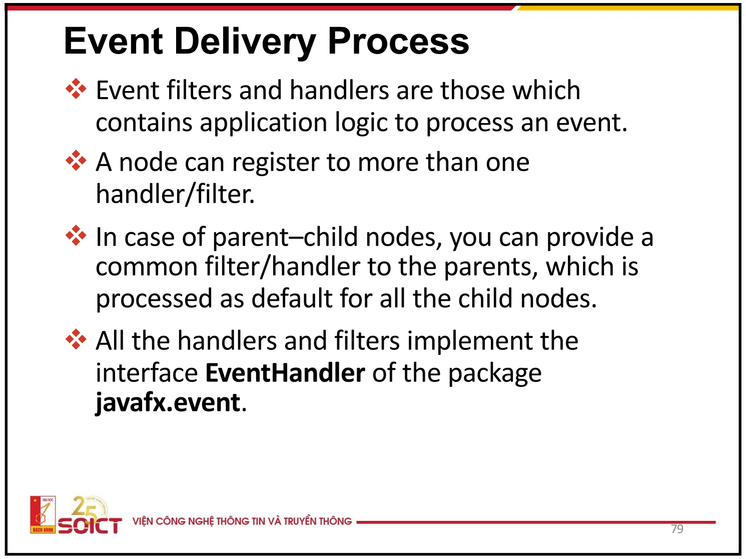
Event Delivery Process

- Step 3: Event Capturing
  - In the event capturing phase, the event is dispatched by the root node of your application and passed down the event dispatch chain to the target node.
  - If any node in the chain has an event filter registered for the type of event that occurred, that filter is called.
  - When the filter completes, the event is passed to the next node down the chain.
  - If a filter is not registered for a node, the event is passed to the next node down the chain.
  - If no filter **consumes** the event, the event target eventually receives and processes the event.



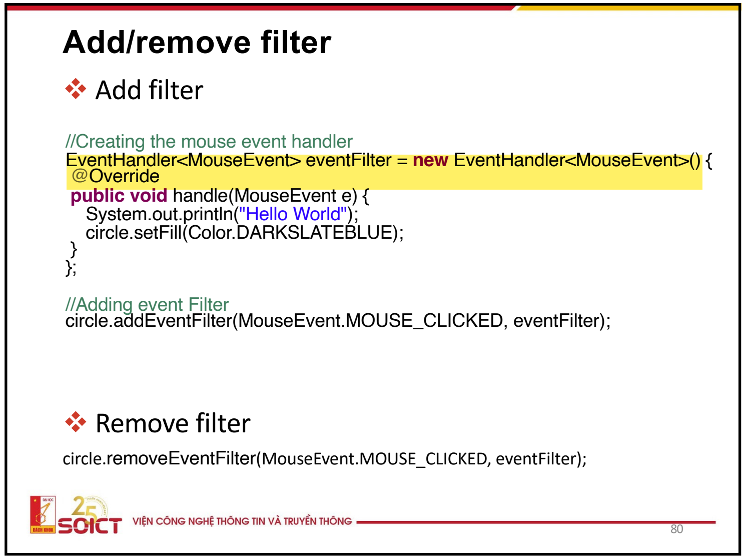
Event Delivery Process

- Step 4: Event Bubbling
  - After the event target is reached and all registered filters have processed the event, the event returns along the dispatch chain from the target to the root node.
  - If any node in the chain has a handler registered for the type of event encountered, that handler is called.
  - When the handler completes, the event is returned to the next node up the chain.
  - If a handler is not registered for a node, the event is returned to the next node up the chain.
  - If no handler **consumes** the event, the root node eventually receives the event and processing is completed.



Event Delivery Process

- Event filters and handlers are those which contains application logic to process an event.
- A node can register to more than one handler/filter.
- In case of parent–child nodes, you can provide a common filter/handler to the parents, which is processed as default for all the child nodes.
- All the handlers and filters implement the interface **EventHandler** of the package **javafx.event**.



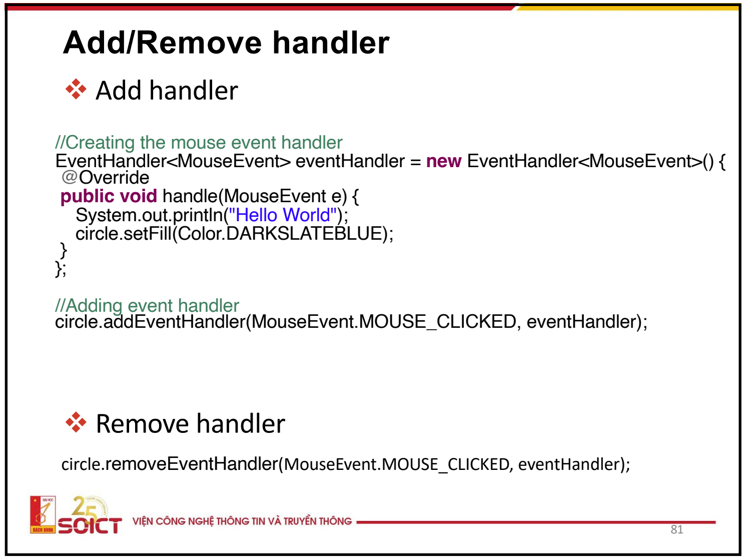
Add/remove filter

- Add filter

```
//Creating the mouse event handler
EventHandler<MouseEvent> eventFilter = new EventHandler<MouseEvent>() {
 @Override
 public void handle(MouseEvent e) {
   System.out.println("Hello World");
   circle.setFill(Color.DARKSLATEBLUE);
 }
};

//Adding event Filter
circle.addEventFilter(MouseEvent.MOUSE_CLICKED, eventFilter);
```

- Remove filter

```
circle.removeEventFilter(MouseEvent.MOUSE_CLICKED, eventFilter);
```



Add/Remove handler

❖ Add handler

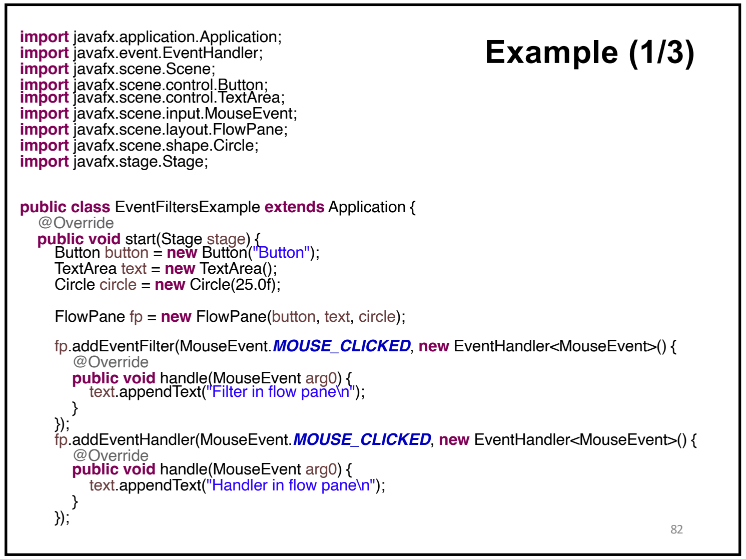
```
//Creating the mouse event handler
EventHandler<MouseEvent> eventHandler = new EventHandler<MouseEvent>() {
 @Override
 public void handle(MouseEvent e) {
   System.out.println("Hello World");
   circle.setFill(Color.DARKSLATEBLUE);
 }
};

//Adding event handler
circle.addEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);
```

❖ Remove handler

```
circle.removeEventHandler(MouseEvent.MOUSE_CLICKED, eventHandler);
```

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

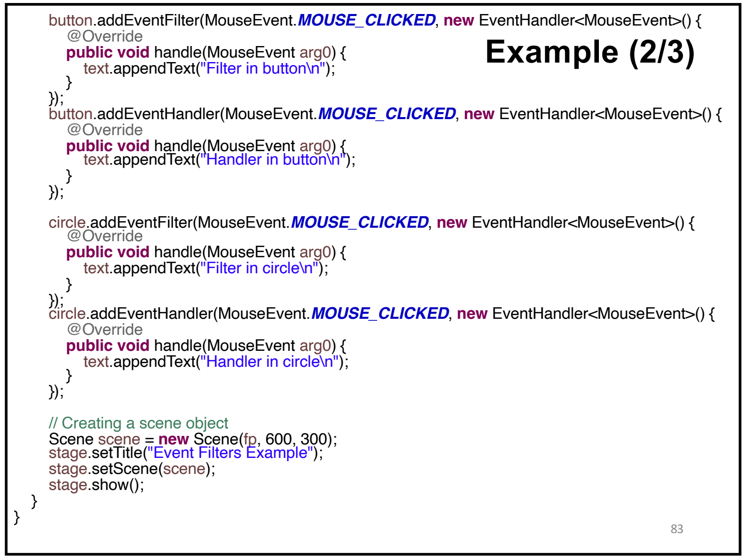
81



Example (1/3)

```
import javafx.application.Application;
import javafx.event.EventHandler;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.scene.input.MouseEvent;
import javafx.scene.layout.FlowPane;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;


public class EventFiltersExample extends Application {
  @Override
  public void start(Stage stage) {
    Button button = new Button("Button");
    TextArea text = new TextArea();
    Circle circle = new Circle(25.0f);

    FlowPane fp = new FlowPane(button, text, circle);

    fp.addEventFilter(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent arg0) {
        text.appendText("Filter in flow pane\n");
      }
    });
    fp.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent arg0) {
        text.appendText("Handler in flow pane\n");
      }
    });
```

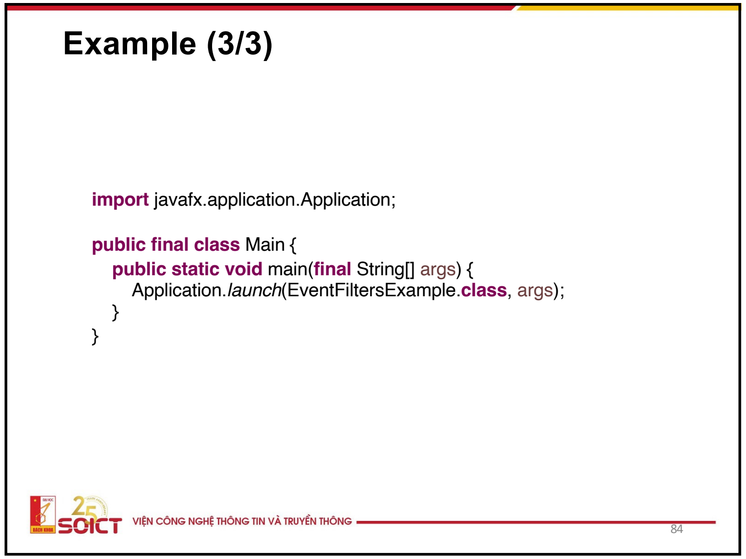
82



Example (2/3)

```
    button.addEventFilter(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent arg0) {
        text.appendText("Filter in button\n");
      }
    });
    button.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent arg0) {
        text.appendText("Handler in button\n");
      }
    });

    circle.addEventFilter(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent arg0) {
        text.appendText("Filter in circle\n");
      }
    });
    circle.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
      @Override
      public void handle(MouseEvent arg0) {
        text.appendText("Handler in circle\n");
      }
    });

    // Creating a scene object
    Scene scene = new Scene(fp, 600, 300);
    stage.setTitle("Event Filters Example");
    stage.setScene(scene);
    stage.show();
  }
}
```

83



Example (3/3)

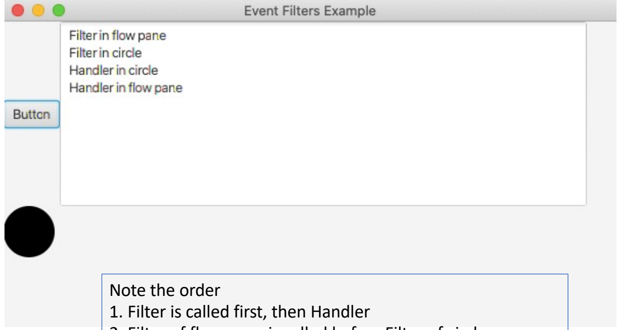
```
import javafx.application.Application;

public final class Main {
  public static void main(final String[] args) {
    Application.launch(EventFiltersExample.class, args);
  }
}
```

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

84

Example

❖ When click on **the circle**

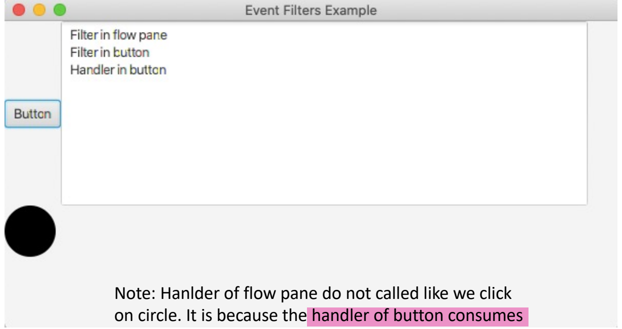


Filter in flow pane
Filter in circle
Handler in circle
Handler in flow pane

Note the order
1. Filter is called first, then Handler
2. Filter of flow pane is called before Filter of circle
3. Handler of circle is called before Handler of flow pane

Example

❖ When click on **Button**



Filter in flow pane
Filter in button
Handler in button

Note: Hanlder of flow pane do not called like we click on circle. It is because the handler of button consumes event by default

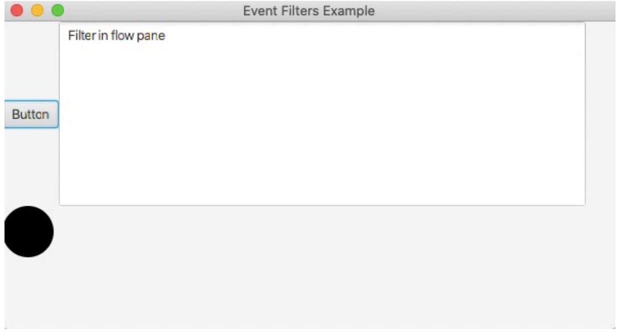
Example

❖ Update the source code of flow pane as follows

```
fp.addEventFilter(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent arg0) {
        text.appendText("Filter in flow pane\n");
        arg0.consume();
    }
});
```

Example

❖ When click on **circle** (or **button**), because the event is consumed, the result will be as follows



Filter in flow pane

Content

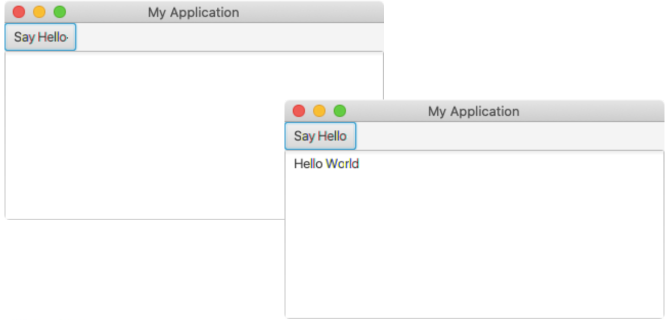
1. Introduction
2. JavaFX Installment
3. GUI components in JavaFX
4. JavaFX - UI controls
5. JavaFX - Layout Panes
6. Event handling models
7. **“Drag and drop” GUI with SceneBuilder**

7. “Drag and drop” GUI with SceneBuilder

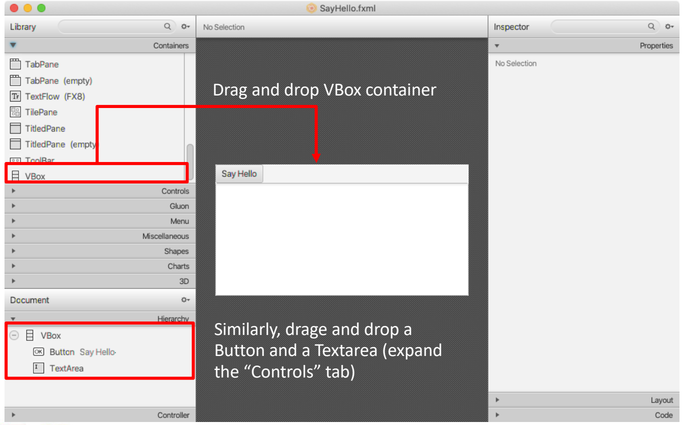
- ❖ Idea: seperate the interface with the business logic
  - Interface: defined in file fxml
  - Business Logic (controller): is separated in Java source code
- ❖ To work with SceneBuilder:
  - Install SceneBuilder
  - Create interface (and then generate fxml file), define the component properties (name of component, event handling methods)
  - Create a JavaFX project
  - Copy fxml file to JavaFX project
  - Create controller
  - Connect the fxml file with the controller
  - Create JavaFX application, load fxml file

Example

- ❖ Develop an application with the following interface. When we click on the button “Say Hello”, the text Hello World is printed in the textbox



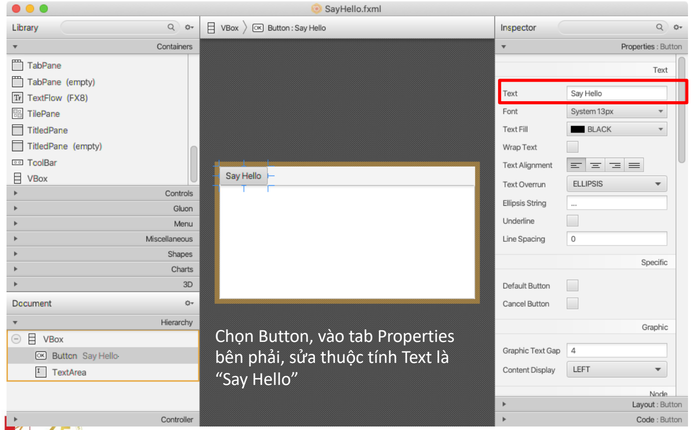
Create SayHello.fxml file with SceneBuilder



Drag and drop VBox container

Similarly, drage and drop a Button and a Textarea (expand the “Controls” tab)

Specify button properties



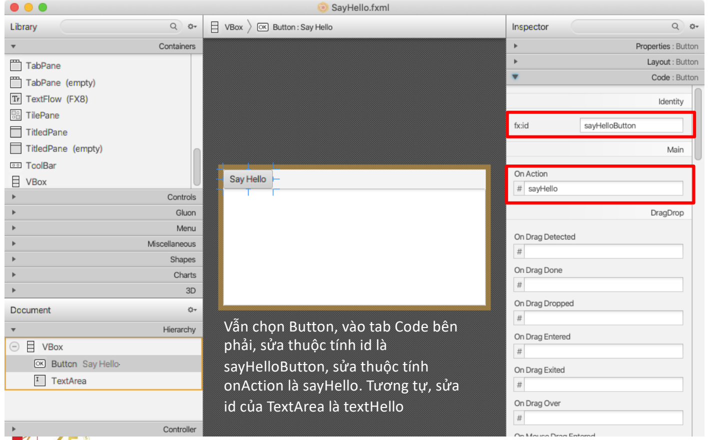
SayHello.fxml

Inspector — Properties : Button

Text: Say Hello
Font: System 13px
Text Fill: BLACK
Wrap Text: □
Text Overrun: ELLIPSIS
Ellipsis String: ...
Underline: □
Line Spacing: 0
Default Button: □
Cancel Button: □
Graphic Text Gap: 4
Content Display: LEFT

Chọn Button, vào tab Properties bên phải, sửa thuộc tính Text là "Say Hello"

Specify button properties



SayHello.fxml

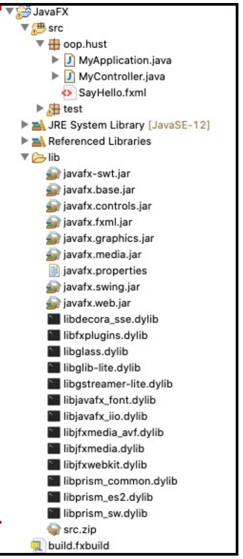
Inspector — Code : Button

fx:id: sayHelloButton
On Action: # sayHello

Vẫn chọn Button, vào tab Code bên phải, sửa thuộc tính id là sayHelloButton, sửa thuộc tính onAction là sayHello. Tương tự, sửa id của TextArea là textHello

Create project JavaFX

- Create a Java project as normal, then copy file SayHello.fxml to the project

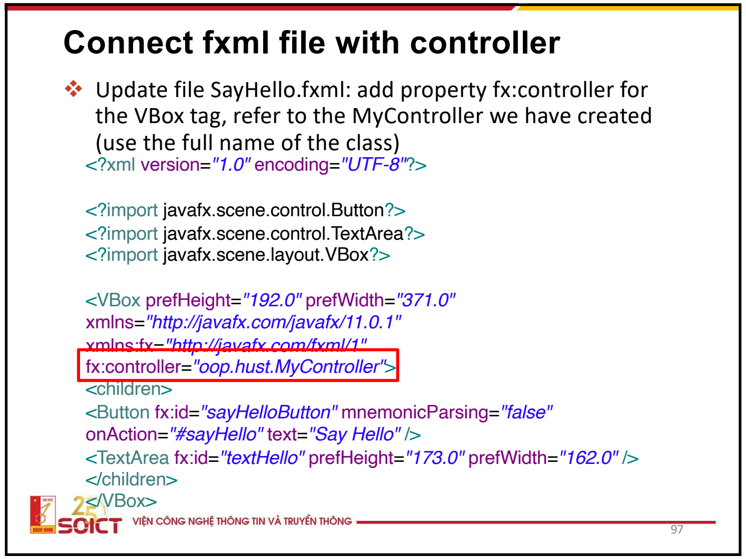


```
JavaFX
  src
    oop.hust
      MyApplication.java
      MyController.java
      SayHello.fxml
    test
  JRE System Library [JavaSE-12]
  Referenced Libraries
  lib
    javafx-swt.jar
    javafx.base.jar
    javafx.controls.jar
    javafx.fxml.jar
    javafx.graphics.jar
    javafx.media.jar
    javafx.properties
    javafx.swing.jar
    javafx.web.jar
    libdecora_sse.dylib
    libfxplugins.dylib
    libglass.dylib
    libglib-lite.dylib
    libgstreamer-lite.dylib
    libjavafx_font.dylib
    libjavafx_iio.dylib
    libjfxmedia_avf.dylib
    libjfxmedia.dylib
    libjfxwebkit.dylib
    libprism_common.dylib
    libprism_es2.dylib
    libprism_sw.dylib
    src.zip
  build.fxbuild
```

Create controller class: MyController

Note: names of Button and TextArea must correspond with the ids specified in SceneBuilder

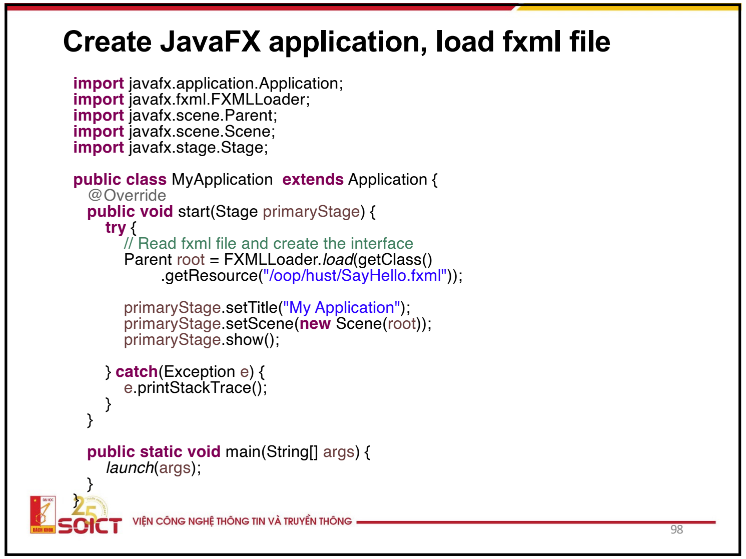
```
import java.net.URL;
import java.util.ResourceBundle;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;

public class MyController implements Initializable {
  @FXML
  private Button sayHelloButton;

  @FXML
  private TextArea textHello;

  @Override
  public void initialize(URL location, ResourceBundle resources) {
  }

  public void sayHello(ActionEvent event) {
     textHello.setText("Hello World");
  }
}
```



Connect fxml file with controller

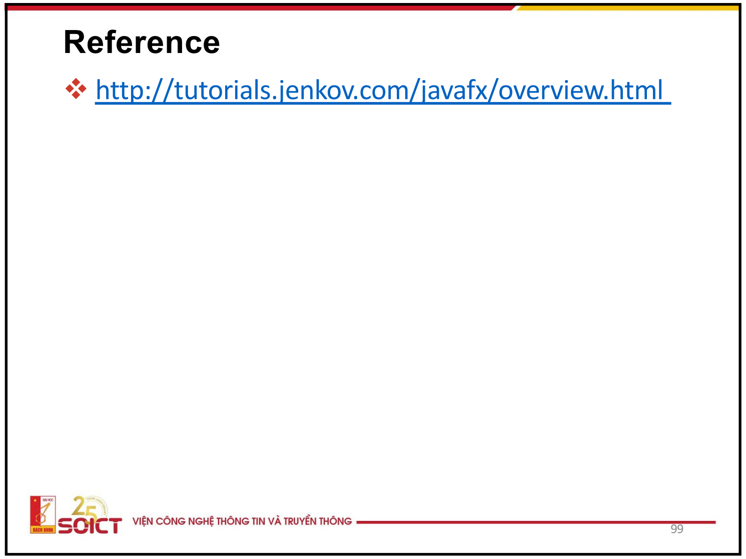
- Update file SayHello.fxml: add property fx:controller for the VBox tag, refer to the MyController we have created (use the full name of the class)

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.layout.VBox?>

<VBox prefHeight="192.0" prefWidth="371.0"
xmlns="http://javafx.com/javafx/11.0.1"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="oop.hust.MyController">
<children>
<Button fx:id="sayHelloButton" mnemonicParsing="false"
onAction="#sayHello" text="Say Hello" />
<TextArea fx:id="textHello" prefHeight="173.0" prefWidth="162.0" />
</children>
</VBox>
```



Create JavaFX application, load fxml file

```
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class MyApplication  extends Application {
  @Override
  public void start(Stage primaryStage) {
     try {
        // Read fxml file and create the interface
        Parent root = FXMLLoader.load(getClass()
              .getResource("/oop/hust/SayHello.fxml"));

        primaryStage.setTitle("My Application");
        primaryStage.setScene(new Scene(root));
        primaryStage.show();

     } catch(Exception e) {
        e.printStackTrace();
     }
  }

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



Reference

- http://tutorials.jenkov.com/javafx/overview.html