

Introduction to JavaFX

Objectives

- ◆ Understand JavaFX Architecture
- ◆ JavaFX Core: Stage, Scene, FXML
- ◆ Understand JavaFX Properties and Bindings, Events, Layouts, Controls
- ◆ Create Java desktop applications with JavaFX

What is JavaFX?

- ◆ JavaFX is an open source, next generation client application platform for desktop, mobile and embedded systems built on Java.
- ◆ It is a collaborative effort by many individuals and companies with the goal of producing a modern, efficient, and fully featured toolkit for developing rich client applications.
- ◆ A powerful framework for building rich client applications in Java.
- ◆ Successor to Swing, offering a modern approach to GUI development.
- ◆ Open-source and available as a separate library from Java 11 onwards.

JavaFX - Key features

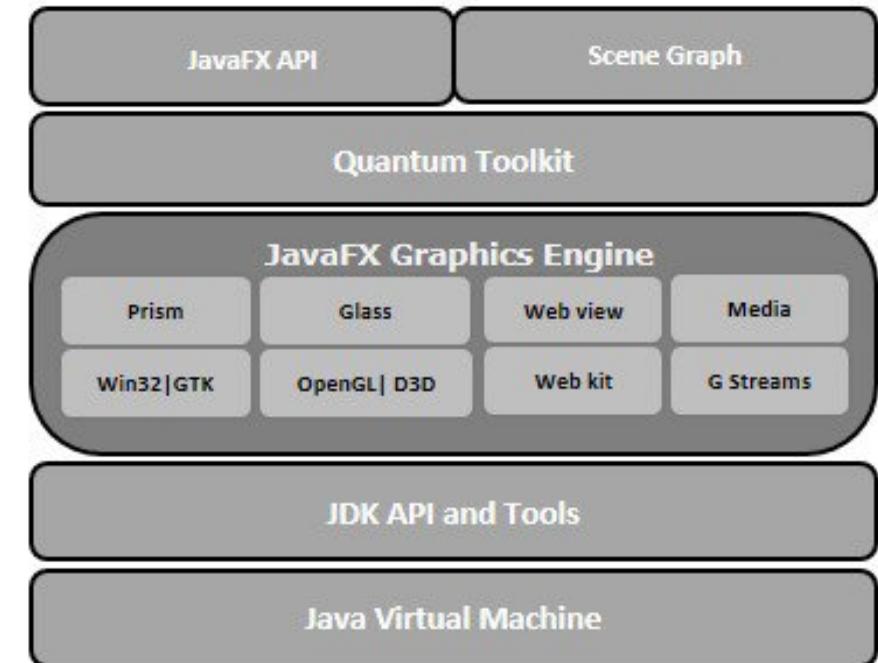
- Written in Java: Leverages the power and flexibility of the Java language, including multithreading, lambda expressions, etc.
- Declarative UI with FXML: Define your UI structure using XML, making it easy to separate UI design from application logic.
- Data Binding: Simplifies data synchronization between UI elements and the underlying data model.
- Rich UI Controls: Provides a diverse set of modern and customizable UI controls, including buttons, text fields, charts, and more.

JavaFX - Key features

- ◆ Media and Web Integration: Enables playback of audio and video content and embedding web components within your application.
- ◆ 2D and 3D Graphics: Offers robust support for both 2D and 3D graphics rendering, allowing you to create visually stunning applications.
- ◆ Animations and Effects: Create smooth animations and apply various visual effects to enhance user experience.
- ◆ CSS Styling: Style your application using CSS for consistent and flexible design.

JavaFX - Architecture

- ◆ **javafx.animation** – Contains classes to add transition based animations such as fill, fade, rotate, scale and translation, to the JavaFX nodes.
- ◆ **javafx.application** – Contains a set of classes responsible for the JavaFX application life cycle.
- ◆ **javafx.css** – Contains classes to add CSS-like styling to JavaFX GUI applications.



JavaFX - Architecture

- ◆ **javafx.event** – Contains classes and interfaces to deliver and handle JavaFX events.
- ◆ **javafx.geometry** – Contains classes to define 2D objects and perform operations on them.
- ◆ **javafx.stage** – This package holds the top level container classes for JavaFX application.
- ◆ **javafx.scene** – This package provides classes and interfaces to support the scene graph. In addition, it also provides sub-packages such as canvas, chart, control, effect, image, input, layout, media, paint, shape, text, transform, web, etc.

JavaFX Parts

- ◆ JavaFX has 3 parts
 - A GUI builder called **SceneBuilder** allows drag-and-drop manipulation of widgets.
 - A configuration language called **FXML** that records the widgets in the GUI, their visible attributes and their relationship to each other.
 - A **Controller** class that must be completed by the programmer to bring the GUI to life.
- ◆ A JavaFX application has some additional parts
 - A set of classes to describe the model, which is what the GUI allows the user to interact with.
 - A set of cascading style sheets (**CSS** files) to further specify “look-and-feel”.

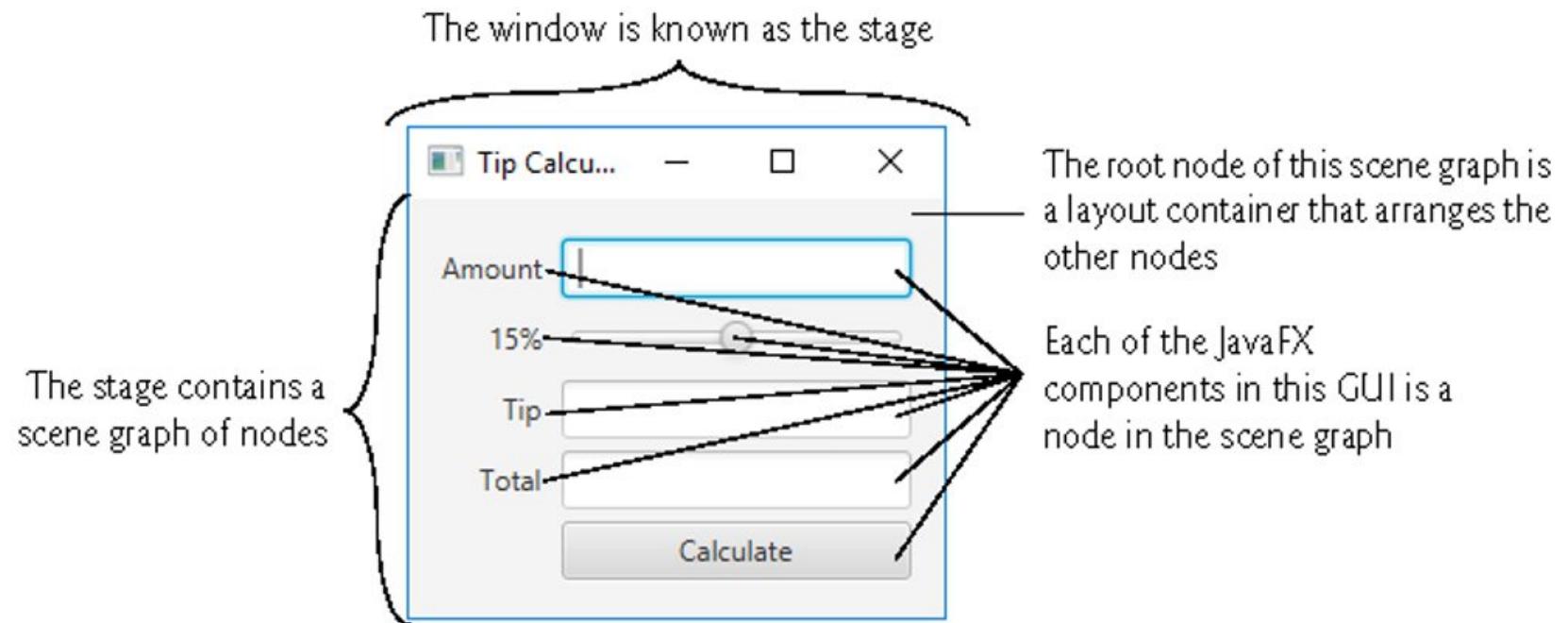
JavaFX Scene Builder

- JavaFX Scene Builder is a standalone JavaFX GUI visual layout tool that can also be used with various IDEs including Eclipse, NetBeans and IntelliJ.
- JavaFX Scene Builder enables you to create GUIs by dragging and dropping GUI components from Scene Builder's library onto a design area, then modifying and styling the GUI—all without writing any code.
- JavaFX Scene Builder generates FXML (FX Markup Language)—an XML vocabulary for defining and arranging JavaFX GUI controls without writing any Java code.

JavaFX Scene Builder

- The FXML code is separate from the program logic that's defined in Java source code—this separation of the interface (the GUI) from the implementation (the Java code) makes it easier to debug, modify and maintain JavaFX GUI apps.
- Placing GUI components in a window can be tedious. Being able to do it dynamically using a configuration file makes the job much easier.
- No additional compilation is needed unless actions need to be programmed in the Controller.java class.

JavaFX App Window Structure



JavaFX App Window Structure

- ◆ The **Stage** is the window in which a JavaFX app's GUI is displayed
 - It's an instance of class **Stage** (package `javafx.stage`).
- ◆ The **Stage** contains one active **Scene** that defines the GUI as a **scene graph** - a tree data structure of an app's visual elements, such as GUI controls, shapes, images, video, text and.
- ◆ The scene is an instance of class **Scene** (package `javafx.scene`).

- ◆ **Controls** are GUI components, such as
 - Labels that display text,
 - TextFields that enable a program to receive user input,
 - Buttons that users click to initiate actions, and more.

JavaFX Application Layout

- ◆ An application Window in JavaFX is known as a **Stage**.
 - package javafx.stage
- ◆ A **Stage** contains an active **Scene** which is set to a Layout container.
 - package javafx.scene
- ◆ The Scene may have other Layout containers for organizing **Controllers** in a Tree organization.
 - Nodes with children are layout containers.
 - Nodes without children are widgets.

JavaFX Application Layout

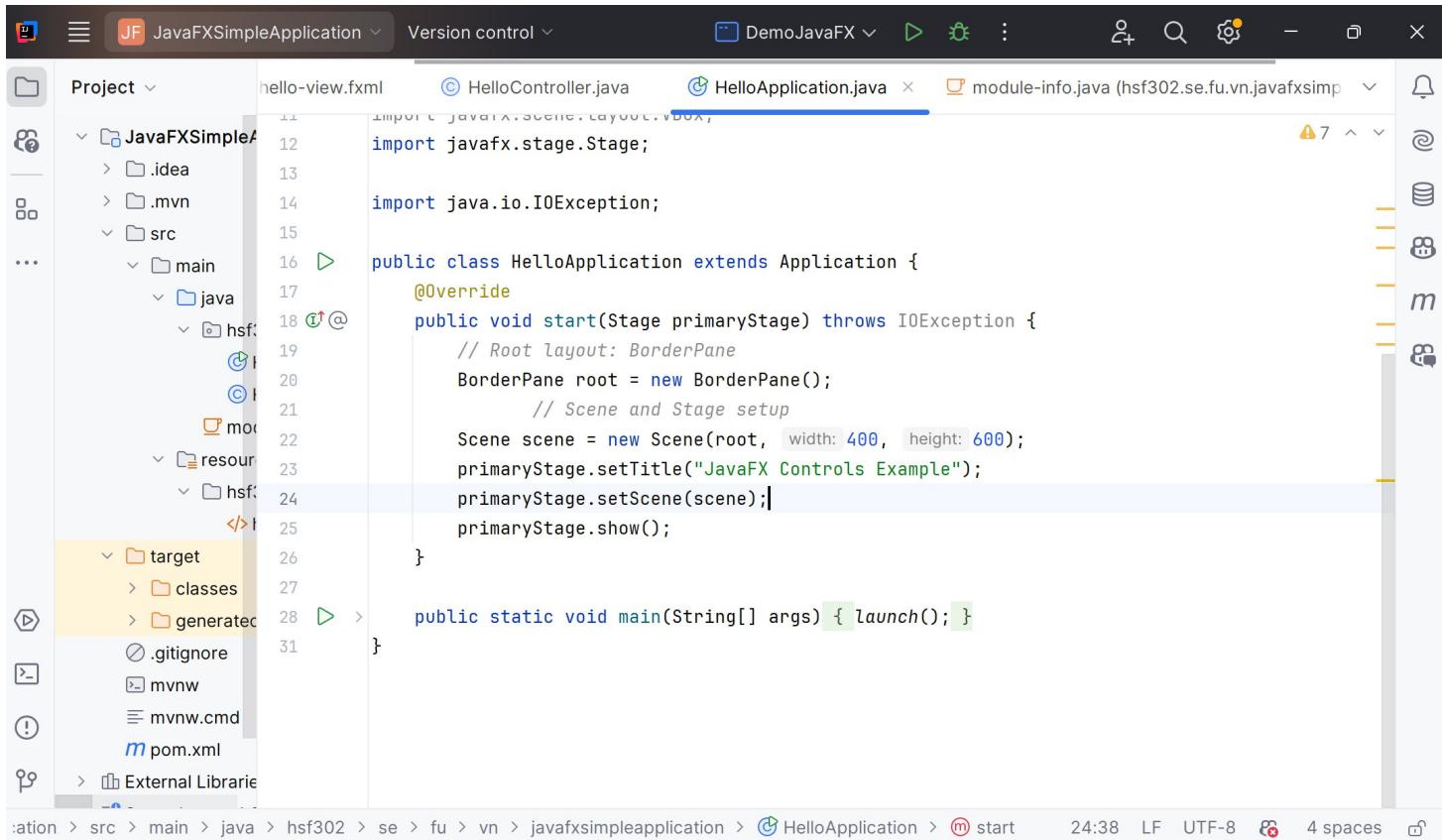
- Each visual element in the scene graph is a **node** - an instance of a subclass of **Node** (package `javafx.scene`), which defines common attributes and behaviors for all nodes
- With the exception of the first node in the scene graph - the **root node** - each node in the scene graph has one parent.
- Nodes can have transforms (e.g., moving, rotating and scaling), opacity (whether a node is transparent, partially transparent or opaque), effects (e.g., drop shadows, blurs, reflection and lighting) and more.

JavaFX Application Controls

- ◆ Nodes with children are typically **layout containers** that arrange their child nodes in the scene.
 - **Layout containers** contain **controls** that accept inputs or other layout containers.
- ◆ When the user interacts with a **control**, the control generates an **event**.
- ◆ Programs can respond to these events—known as event handling—to specify what should happen when each user interaction occurs.
- ◆ An **event handler** is a method that responds to a user interaction. An FXML GUI's event handlers are defined in a so-called **controller class**.

JavaFX Application Class

- JavaFX class must implement the abstract **start()** method of the **Application** class



The screenshot shows the IntelliJ IDEA interface with the project 'JavaFXSimpleApplication' open. The code editor displays the `HelloApplication.java` file, which contains the following Java code:

```
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;
import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage primaryStage) throws IOException {
        // Root layout: BorderPane
        BorderPane root = new BorderPane();
        // Scene and Stage setup
        Scene scene = new Scene(root, 400, 600);
        primaryStage.setTitle("JavaFX Controls Example");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

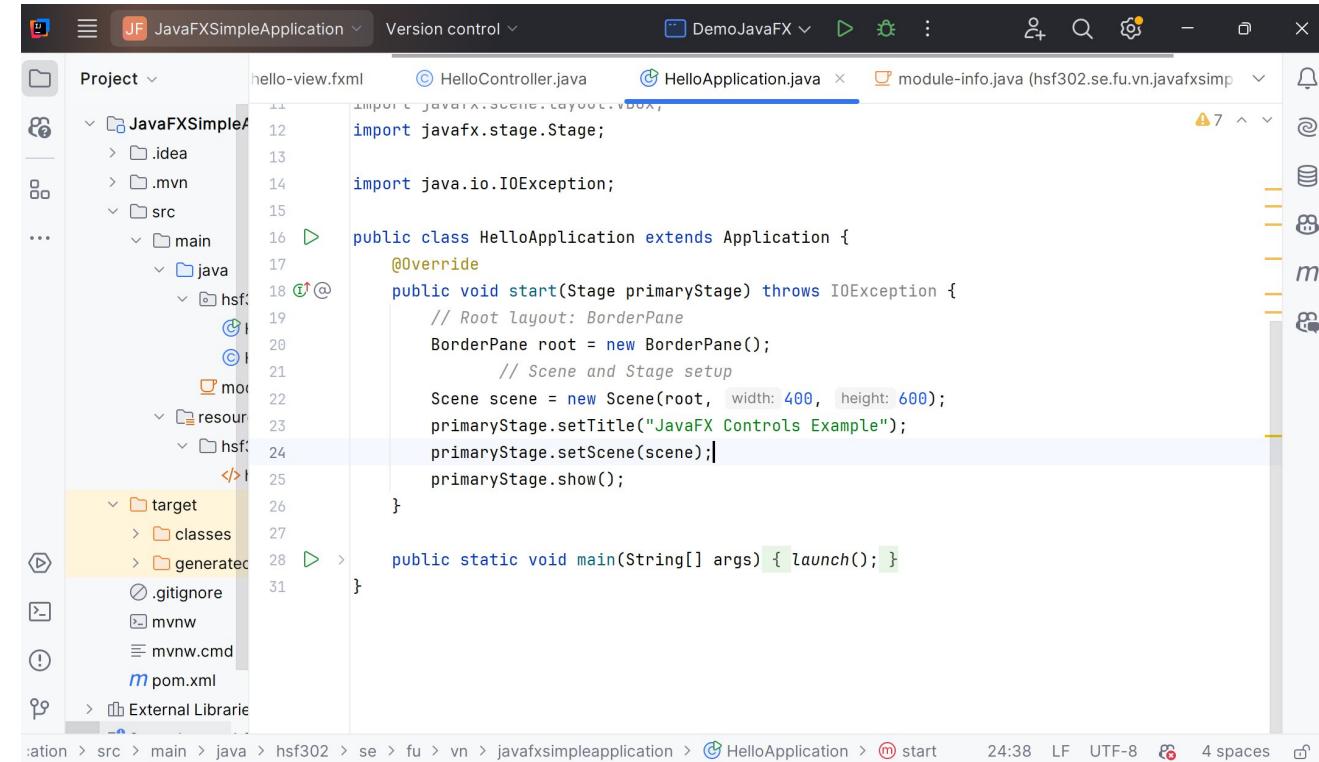
    public static void main(String[] args) { launch(); }
}
```

The code implements the `Application` interface and overrides the `start` method to set up a `BorderPane` root layout, a `Scene` with width 400 and height 600, and a `Stage` titled "JavaFX Controls Example". It also defines a `main` method to launch the application.

JavaFX Core Components, Concepts and Features

Scene to the Stage object

- To display something inside the JavaFX application window you must add a Scene to the Stage object.



The screenshot shows the IntelliJ IDEA interface with the following details:

- Title Bar:** JavaFXSimpleApplication
- Toolbars:** Version control, DemoJavaFX
- Project View:** Shows the project structure under JavaFXSimpleApplication, including .idea, .mvn, src, main, java, resources, target, and various configuration files like pom.xml and mvnw.
- Code Editor:** The HelloApplication.java file is open, displaying JavaFX code:

```
import javafx.scene.layout.VBox;
import javafx.stage.Stage;

import java.io.IOException;

public class HelloApplication extends Application {
    @Override
    public void start(Stage primaryStage) throws IOException {
        // Root layout: BorderPane
        BorderPane root = new BorderPane();
        // Scene and Stage setup
        Scene scene = new Scene(root, width: 400, height: 600);
        primaryStage.setTitle("JavaFX Controls Example");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) { launch(); }
}
```
- Status Bar:** Shows the current file path: location > src > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication > start, and system information: 24:38, LF, UTF-8, 4 spaces.

JavaFX Stage

- ◆ A JavaFX Stage, `javafx.stage.Stage`, represents a window in a JavaFX desktop application. Inside a JavaFX Stage, insert a JavaFX Scene which represents the content displayed inside a window - inside a Stage.
- ◆ Operations:
 - Creating a Stage
 - Showing a Stage (`show()` vs. `showAndWait()`)
 - Set a Scene on a Stage
 - Stage Title
 - Stage Position
 - Stage Width and Height
 - Stage Modality
 - Stage Owner
 - Stage Style (DECORATED, UNDECORATED, TRANSPARENT, UNIFIED, UTILITY)

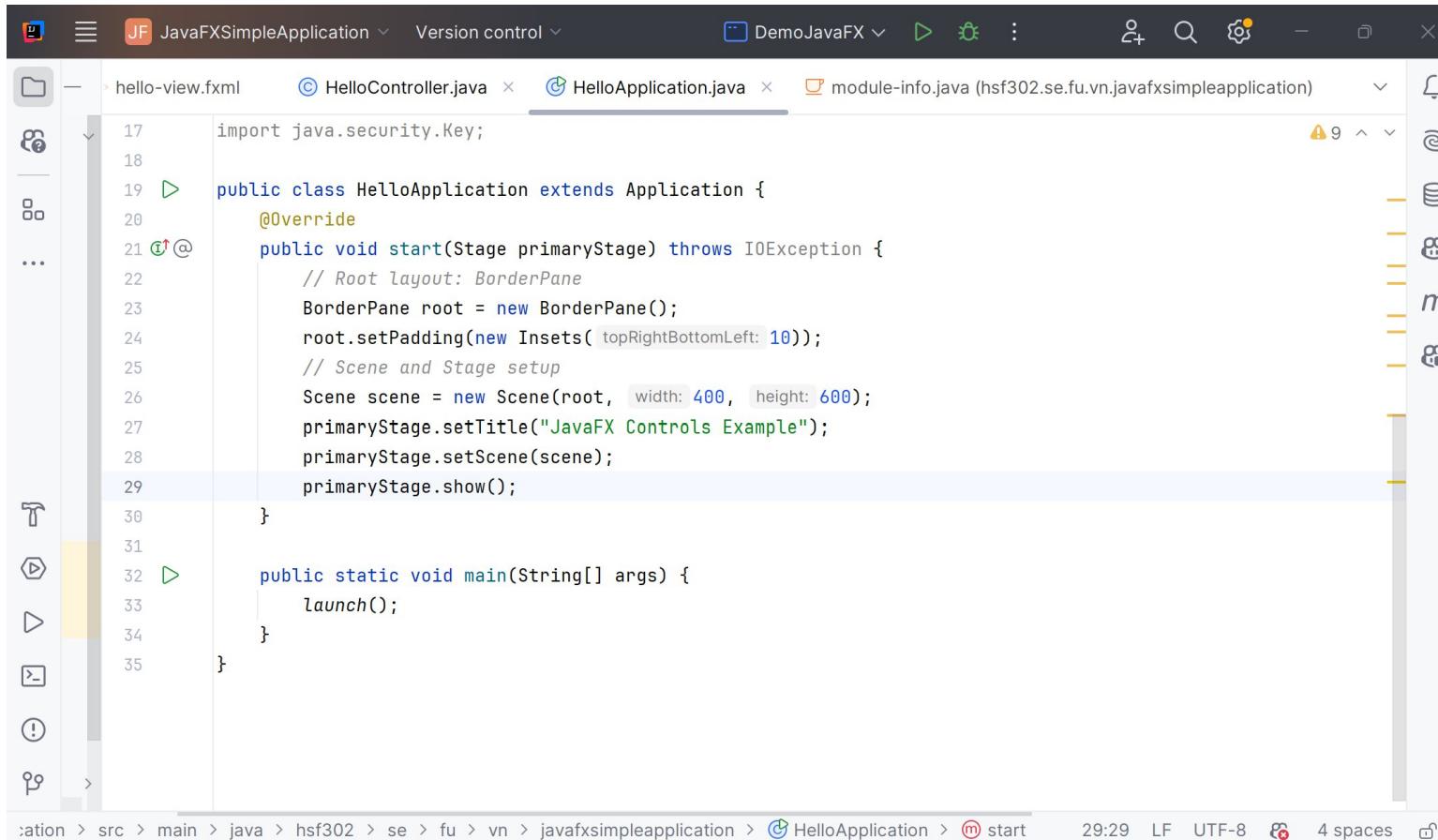
JavaFX Scene

- ◆ The JavaFX Scene object is the root of the JavaFX Scene graph.
- ◆ JavaFX Scene contains all the visual JavaFX GUI components inside it.
- ◆ A JavaFX Scene is represented by the class `javafx.scene.Scene`.
- ◆ A Scene object has to be set on a JavaFX Stage to be visible.

- ◆ Operations
 - Create Scene
 - Set Scene on Stage
 - The Scene Graph
 - Scene Mouse Cursor

JavaFX Scene

- The example shows how to set a specific mouse cursor

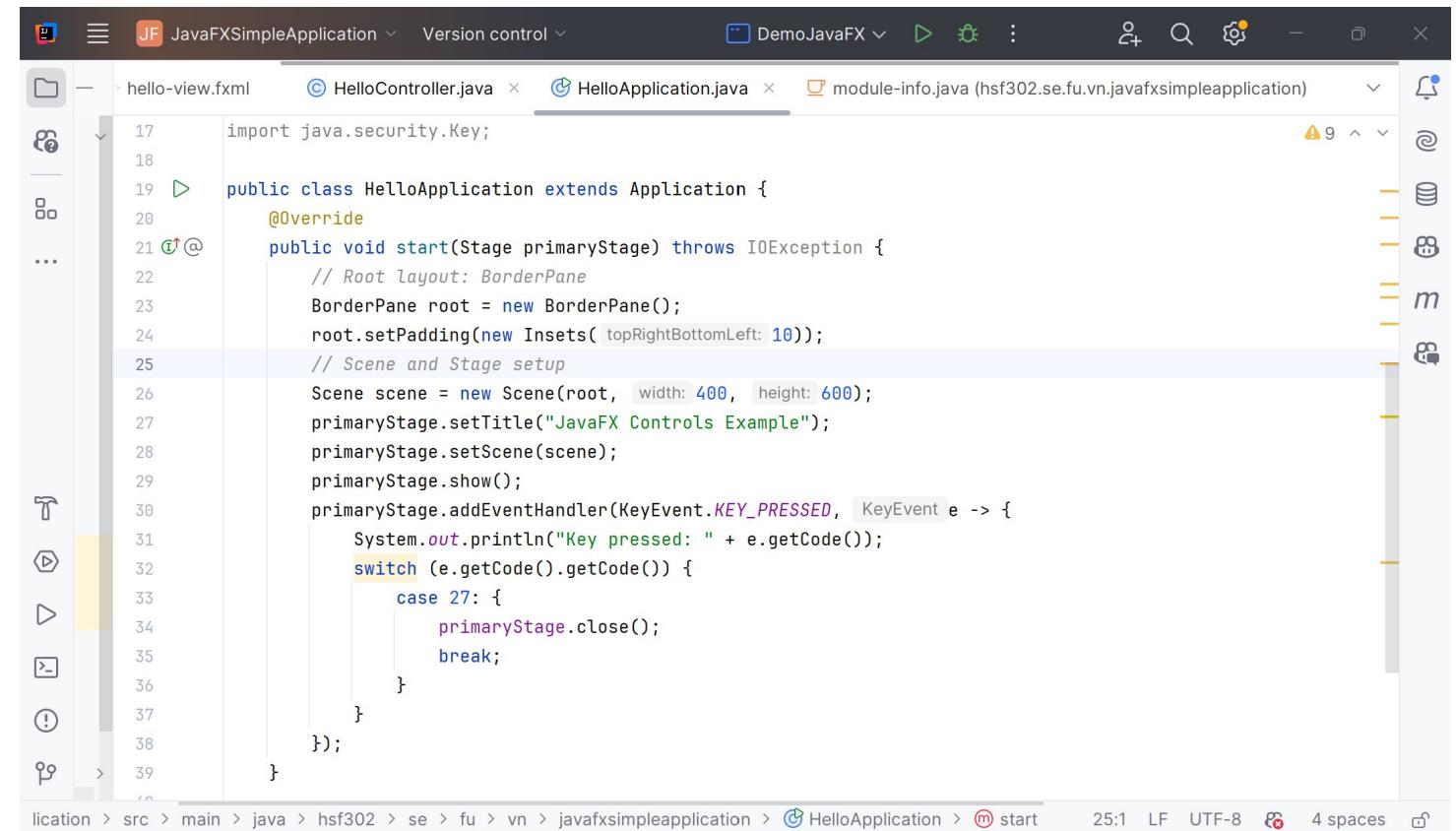


A screenshot of a Java IDE (IntelliJ IDEA) showing the code for a JavaFX application. The code defines a `HelloApplication` class that extends `Application`. It overrides the `start` method to create a `BorderPane` root layout, set its padding, and show a scene with a title. The `main` method calls `launch`. The code editor shows syntax highlighting and a vertical bar indicating code completion.

```
import java.security.Key;
public class HelloApplication extends Application {
    @Override
    public void start(Stage primaryStage) throws IOException {
        // Root layout: BorderPane
        BorderPane root = new BorderPane();
        root.setPadding(new Insets( topRightBottomLeft: 10));
        // Scene and Stage setup
        Scene scene = new Scene(root, width: 400, height: 600);
        primaryStage.setTitle("JavaFX Controls Example");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

Stage Life Cycle Events

- ◆ The Stage events: Close Request, Hiding, Hidden, Showing, Shown
- ◆ Stage keyboard events



The screenshot shows a Java IDE interface with the following details:

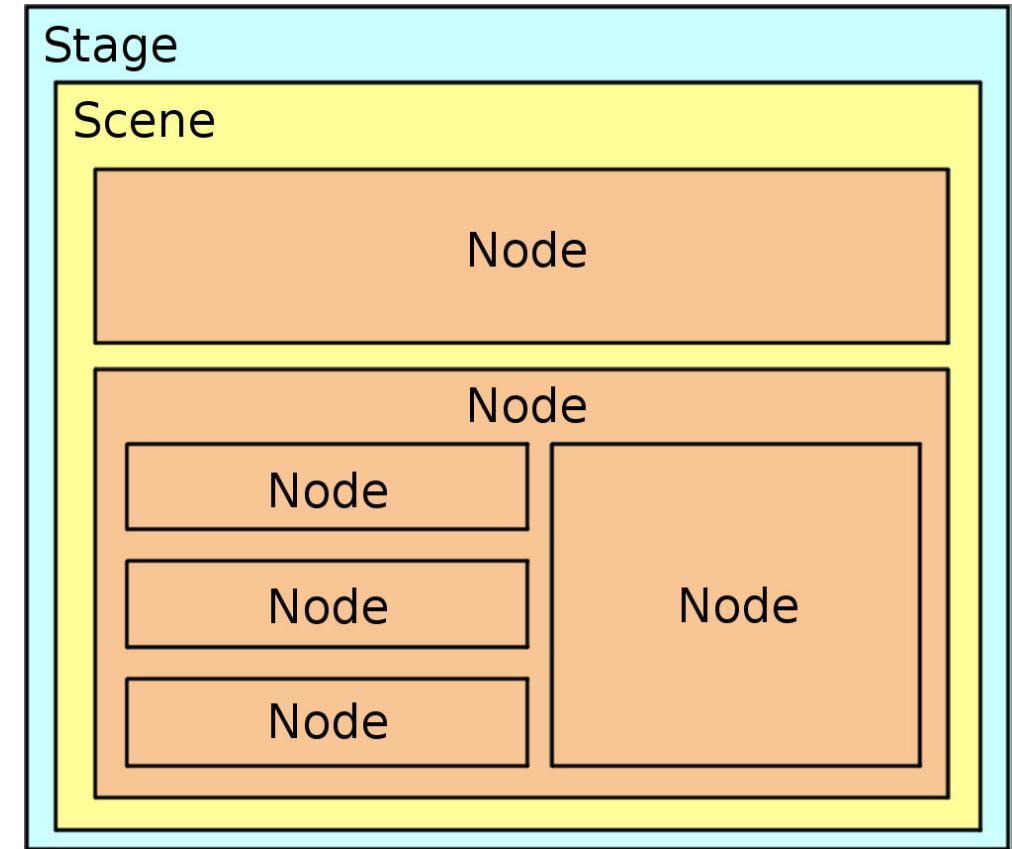
- Title Bar:** JF JavaFXSimpleApplication, Version control.
- Toolbar:** DemoJavaFX, navigation icons.
- Left Sidebar:** Shows files like hello-view.fxml, HelloController.java, HelloApplication.java (selected), and module-info.java.
- Status Bar:** Location: application > src > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication > start, 25:1 LF, UTF-8, 4 spaces.
- Code Editor:** Displays the following Java code:

```
import java.security.Key;

public class HelloApplication extends Application {
    @Override
    public void start(Stage primaryStage) throws IOException {
        // Root layout: BorderPane
        BorderPane root = new BorderPane();
        root.setPadding(new Insets( topRightBottomLeft: 10));
        // Scene and Stage setup
        Scene scene = new Scene(root, width: 400, height: 600);
        primaryStage.setTitle("JavaFX Controls Example");
        primaryStage.setScene(scene);
        primaryStage.show();
        primaryStage.addEventHandler(KeyEvent.KEY_PRESSED, KeyEvent e -> {
            System.out.println("Key pressed: " + e.getCode());
            switch (e.getCode().getCode()) {
                case 27: {
                    primaryStage.close();
                    break;
                }
            }
        });
    }
}
```

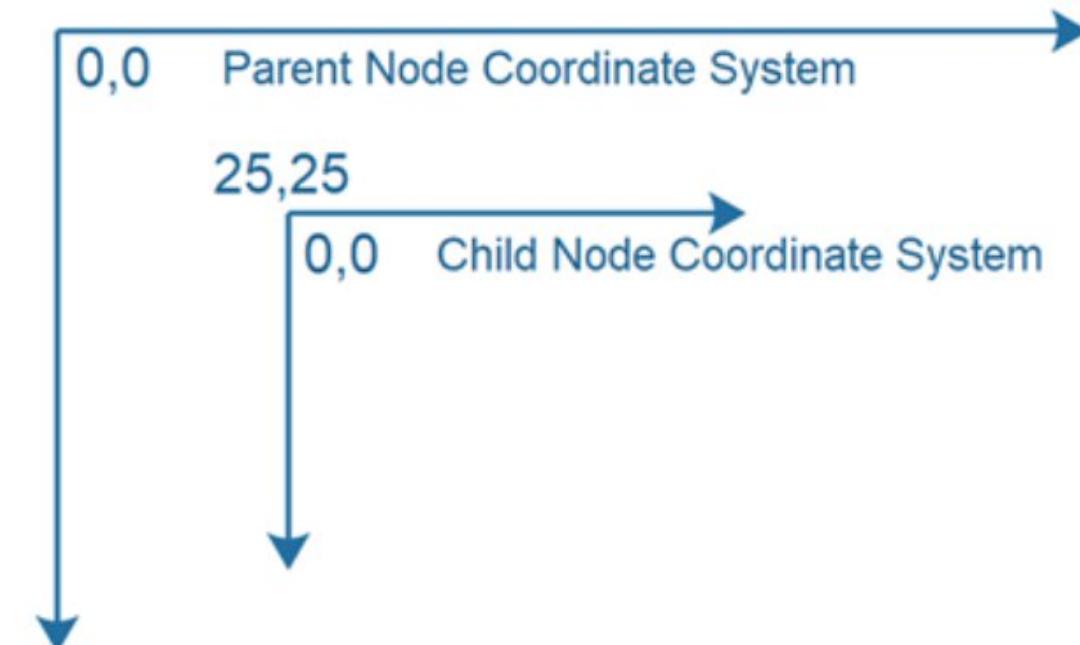
JavaFX Node

- ◆ Each JavaFX Node (subclass) instance can only be added to the JavaFX scene graph once.
- ◆ Node instance can only appear in one place in the scene graph.



JavaFX Node Properties

- ◆ A cartesian coordinate system
- ◆ A bounding box delimited by: Layout bounds, Bounds in local, Bounds in parent
- ◆ layoutX
- ◆ layoutY
- ◆ Preferred height
- ◆ Preferred width
- ◆ Minimum height
- ◆ Minimum width
- ◆ Maximum height
- ◆ Maximum width
- ◆ User data



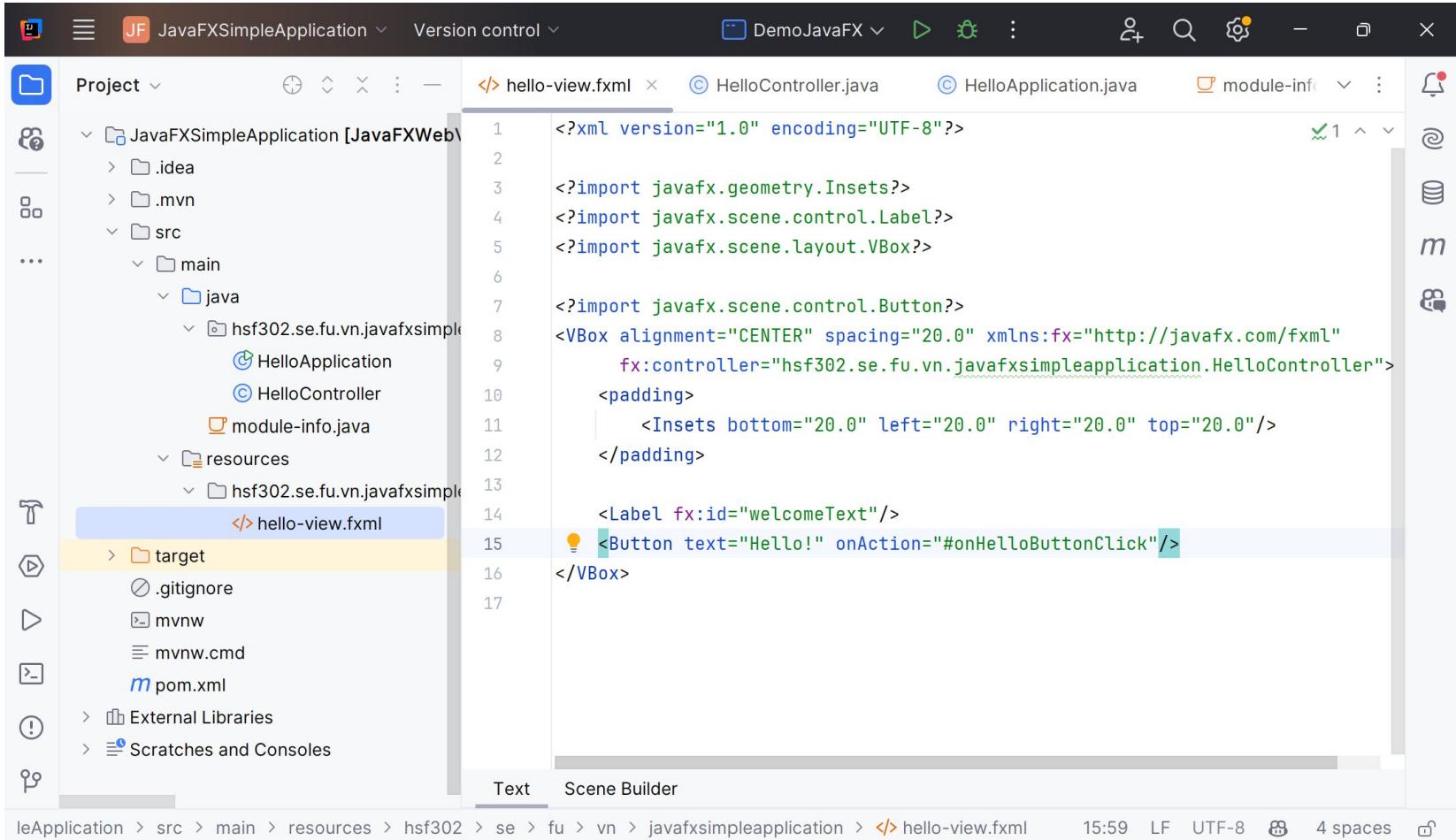
JavaFX Properties

- ◆ A JavaFX Property is a special kind member variable of JavaFX controls.
- ◆ JavaFX properties are typically used to contain control properties such as X and Y position, width and height, text, children and other central properties of JavaFX controls.
- ◆ Can attach change listeners to JavaFX properties so other components can get notified when the value of the property changes, and can bind properties to each other so when one property value changes, so does the other.

JavaFX FXML

- ◆ JavaFX FXML is an XML format that enables you to compose JavaFX GUIs in a fashion similar to how you compose web GUIs in HTML.
- ◆ FXML enables to separate the JavaFX layout code from the rest of the application code. This cleans up both the layout code and the rest of the application code.
- ◆ FXML can be used both to compose the layout of a whole application GUI, or just part of an application GUI, e.g. the layout of one part of a form, tab, dialog etc.

JavaFX FXML



The screenshot shows the IntelliJ IDEA interface with the following details:

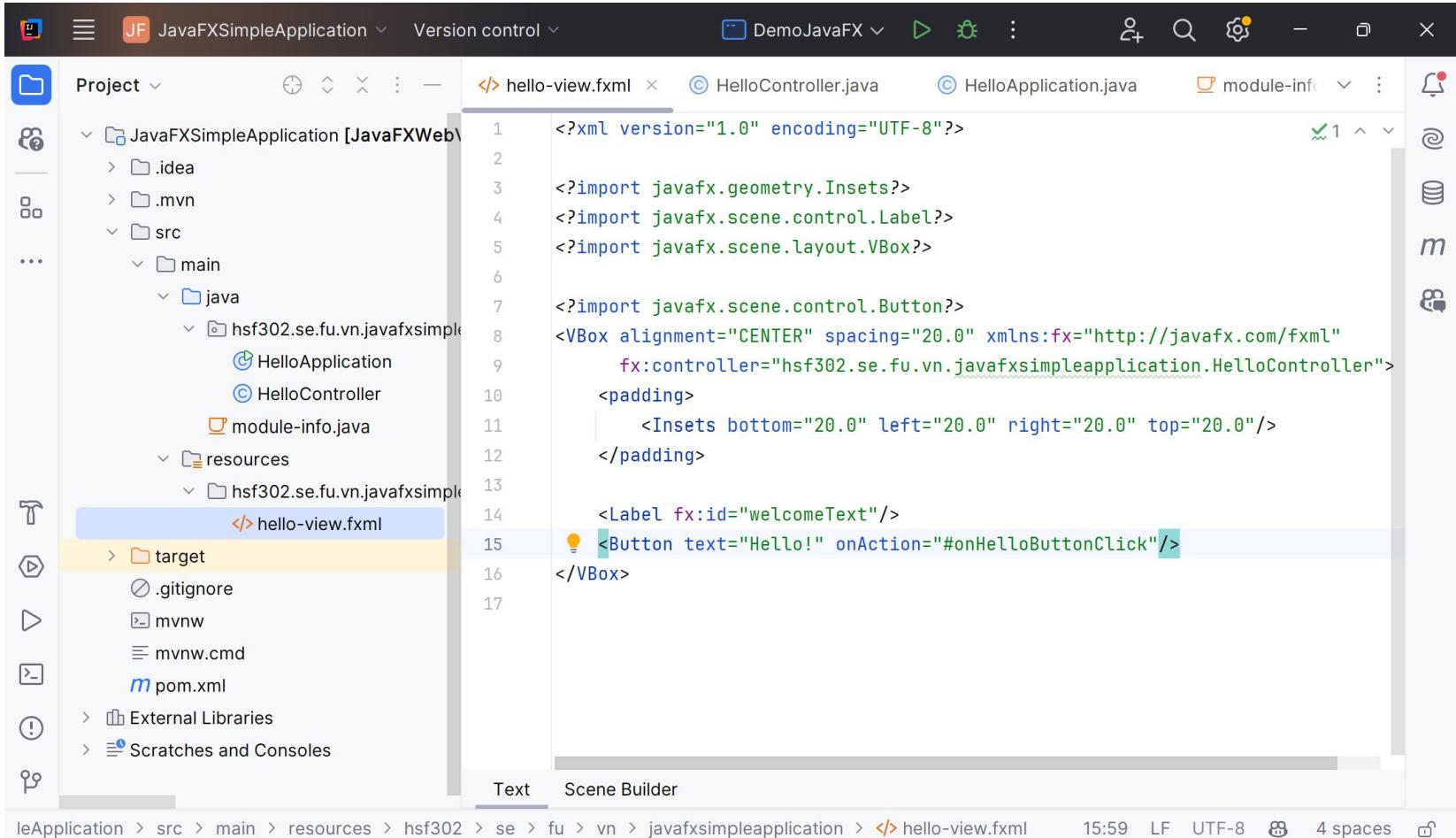
- Title Bar:** JF JavaFXSimpleApplication
- Toolbars:** Version control, DemoJavaFX, and several icons for navigation and search.
- Project Tool Window:** Shows the project structure:
 - JavaFXSimpleApplication [JavaFXWeb]
 - .idea
 - .mvn
 - src
 - main
 - java
 - hsf302.se.fu.vn.javafxsimpleapplication
 - HelloApplication
 - HelloController
 - module-info.java
 - resources
 - hsf302.se.fu.vn.javafxsimpleapplication
 - hello-view.fxml
 - target
 - .gitignore
 - mvnw
 - mvnw.cmd
 - pom.xml
 - External Libraries
 - Scratches and Consoles
- Editor:** The "hello-view.fxml" file is open, displaying JavaFX FXML code. The code defines a VBox with a Label and a Button.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<?import javafx.scene.control.Button?>
<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
      fx:controller="hsf302.se.fu.vn.javafxsimpleapplication.HelloController">
    <padding>
      <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
    </padding>

    <Label fx:id="welcomeText"/>
    <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```
- Bottom Status Bar:** Shows the file path (hello-view.fxml), timestamp (15:59), and encoding (UTF-8).

JavaFX Controller



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** Shows "JF JavaFXSimpleApplication" and "Version control".
- Toolbars:** Standard IntelliJ toolbars for file operations.
- Editor:** The main window displays the XML content of `hello-view.fxml`. The code defines a `VBox` with a `Label` and a `Button` with an action listener.

```
<?xml version="1.0" encoding="UTF-8"?>
<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.layout.VBox?>

<?import javafx.scene.control.Button?>
<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
      fx:controller="hsf302.se.fu.vn.javafxsimpleapplication.HelloController">
  <padding>
    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
  </padding>

  <Label fx:id="welcomeText"/>
  <Button text="Hello!" onAction="#onHelloButtonClick"/>
</VBox>
```

- Sidebar:** Shows the project structure with `JavaFXSimpleApplication [JavaFXWeb]`, `src/main/java`, and `resources` containing `hello-view.fxml`.
- Bottom:** Shows the file path `hsf302/se/fu/vn/javafxsimpleapplication/hello-view.fxml`, status bar with time `15:59`, file type `LF`, encoding `UTF-8`, and code style settings.

JavaFX Layout, Control Components

JavaFX Layout Components

- ◆ Layout components to help organize and structure the user interface
- ◆ JavaFX layouts are components which contains other components inside them. The layout component manages the layout of the components nested inside it.
- ◆ JavaFX layout components are also sometimes called parent components because they contain child components, and because layout components are subclasses of the JavaFX class **javafx.scene.Parent**.
- ◆ Common layout components:
 - Group
 - Region
 - Pane
 - **VBox** and **HBox**: Arranges its child nodes in a single vertical column or in a single horizontal row.

JavaFX Layout Components

- ◆ **BorderPane**: Layout component divides the application window into five regions: top, bottom, left, right, and center.
- ◆ **FlowPane**: Arranges its child nodes in a flow that wraps at the boundary of the layout. It is useful for creating dynamically laid out content.
- ◆ **GridPane**: Allows for a flexible grid of rows and columns, where child nodes can be placed at specific row and column indices.
- ◆ **StackPane**: Layout component stacks its child nodes on top of each other.
- ◆ **AnchorPane**: Allows the positioning of nodes relative to the edges of the pane or relative to each other. It is useful for creating precise layouts.
- ◆ **TilePane**: Layout its children in a grid of equally sized cells.

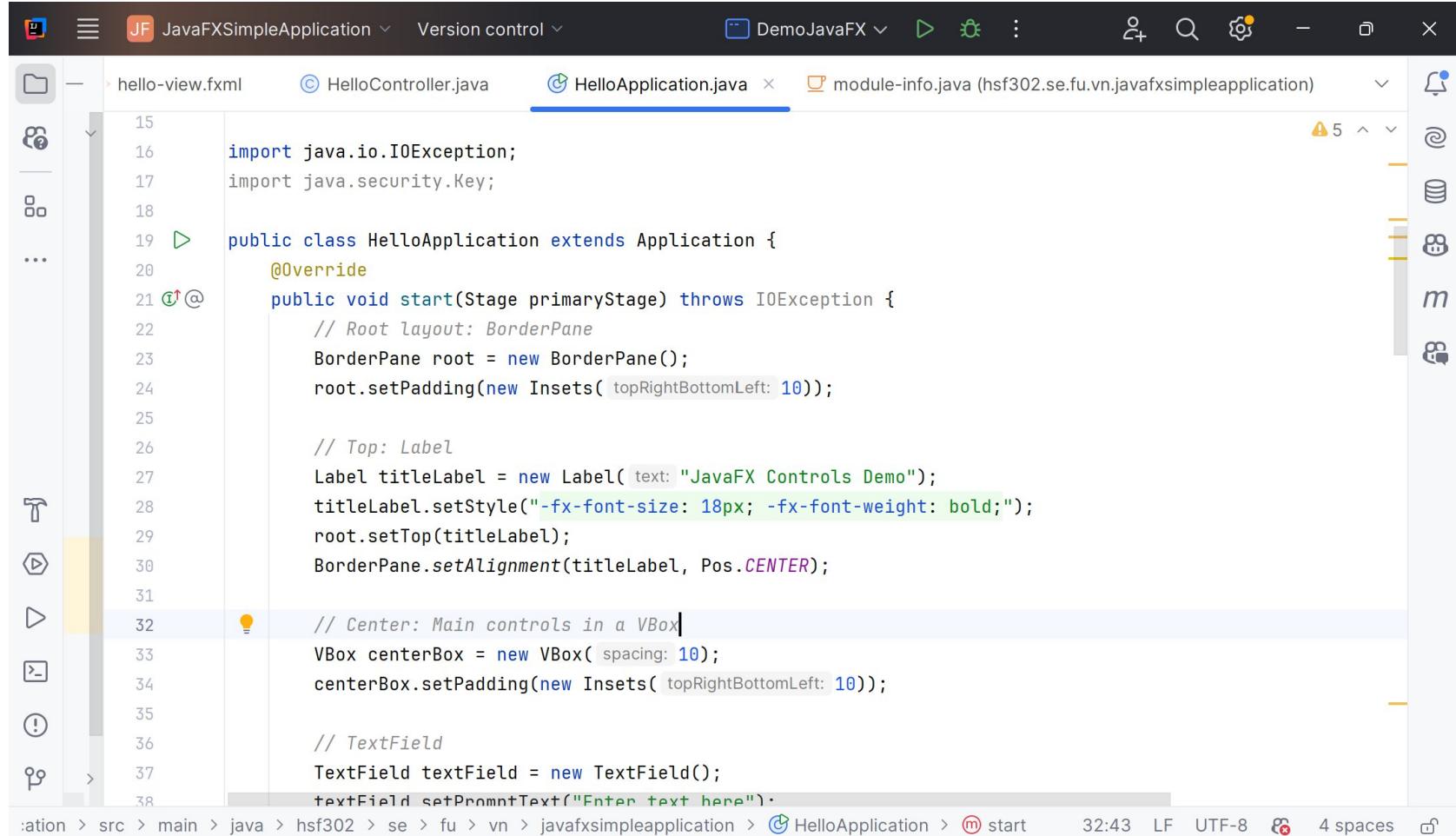
JavaFX Controls

- ◆ JavaFX controls are JavaFX components which provide some kind of control functionality inside a JavaFX application.
- ◆ For a control to be visible it must be attached to the scene graph of some Scene object.
- ◆ Controls are usually nested inside some JavaFX layout component that manages the layout of controls relative to each other.

JavaFX Controls

- ◆ Common JavaFX Controls
 - Accordion, Button, CheckBox, ChoiceBox, ColorPicker,
 - ComboBox, DatePicker, Label, ListView, Menu,MenuBar, PasswordField
 - ProgressBar, RadioButton, Slider, Spinner, SplitMenuItem, TableView
 - TextArea, TextField, ToggleButton, ToolBar, TreeTableView, TreeView

Example : BorderPane



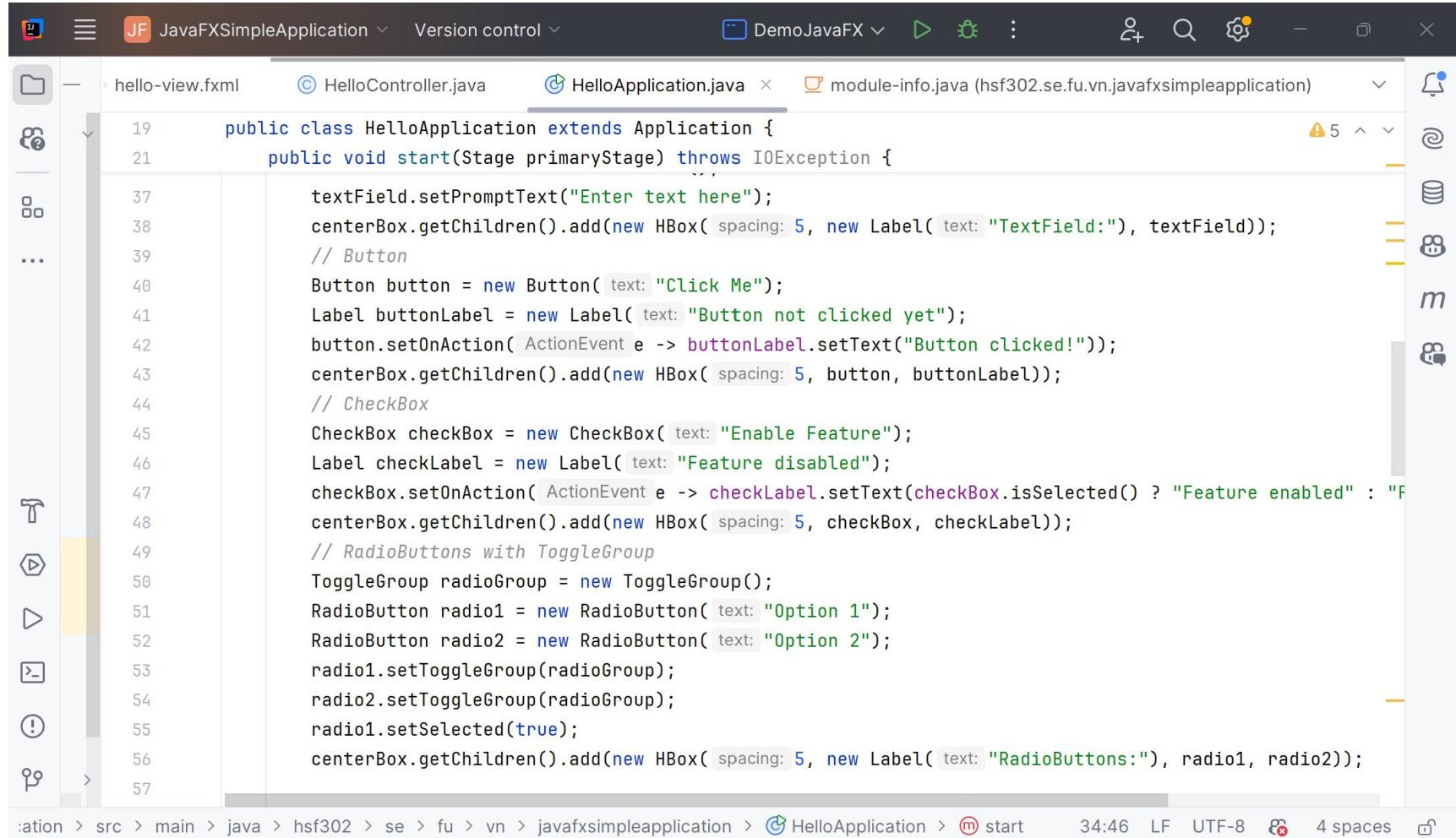
The screenshot shows the IntelliJ IDEA IDE interface with the following details:

- Title Bar:** Shows "JavaFXSimpleApplication" and "HelloApplication.java" as the active file.
- Toolbars:** Standard IntelliJ toolbars for file operations, search, and navigation.
- Left Sidebar:** Includes icons for file, folder, search, and other project-related functions.
- Code Editor:** Displays Java code for `HelloApplication.java`. The code creates a `BorderPane` root node and adds a `Label` to the top center. It also creates a `VBox` for main controls and a `TextField`.

```
15 import java.io.IOException;
16 import java.security.Key;
17
18
19 public class HelloApplication extends Application {
20     @Override
21     public void start(Stage primaryStage) throws IOException {
22         // Root layout: BorderPane
23         BorderPane root = new BorderPane();
24         root.setPadding(new Insets( topRightBottomLeft: 10));
25
26         // Top: Label
27         Label titleLabel = new Label( text: "JavaFX Controls Demo");
28         titleLabel.setStyle("-fx-font-size: 18px; -fx-font-weight: bold;");
29         root.setTop(titleLabel);
30         BorderPane.setAlignment(titleLabel, Pos.CENTER);
31
32         // Center: Main controls in a VBox
33         VBox centerBox = new VBox( spacing: 10);
34         centerBox.setPadding(new Insets( topRightBottomLeft: 10));
35
36         // TextField
37         TextField textField = new TextField();
38         textField.setPromptText("Enter text here").
```

- Bottom Status Bar:** Shows the current file path, file type, encoding, and code style settings (4 spaces).

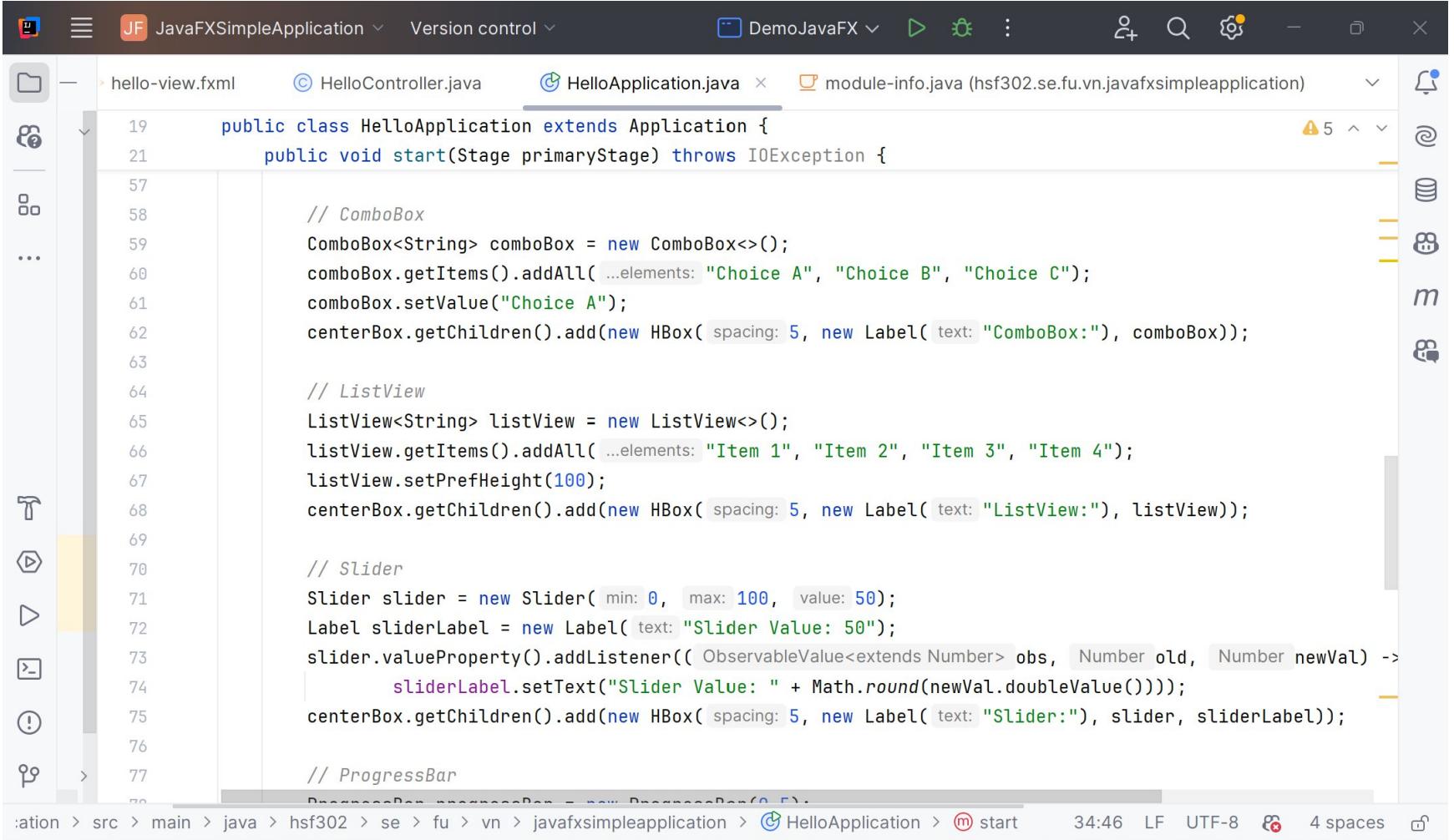
Example : Button



The screenshot shows a Java IDE interface with the following details:

- Title Bar:** JF JavaFXSimpleApplication, Version control, DemoJavaFX, search, settings, close.
- File Explorer:** Shows files: hello-view.fxml, HelloController.java, HelloApplication.java (selected), module-info.java (hsf302.se.fu.vn.javafxsimpleapplication).
- Code Editor:** Displays the `HelloApplication.java` file with the following code:public class HelloApplication extends Application {
 public void start(Stage primaryStage) throws IOException {
 textField.setPromptText("Enter text here");
 centerBox.getChildren().add(new HBox(spacing: 5, new Label(text: "TextField:"), textField));
 // Button
 Button button = new Button(text: "Click Me");
 Label buttonLabel = new Label(text: "Button not clicked yet");
 button.setOnAction(ActionEvent e -> buttonLabel.setText("Button clicked!"));
 centerBox.getChildren().add(new HBox(spacing: 5, button, buttonLabel));
 // CheckBox
 CheckBox checkBox = new CheckBox(text: "Enable Feature");
 Label checkLabel = new Label(text: "Feature disabled");
 checkBox.setOnAction(ActionEvent e -> checkLabel.setText(checkBox.isSelected() ? "Feature enabled" : "Feature disabled"));
 centerBox.getChildren().add(new HBox(spacing: 5, checkBox, checkLabel));
 // RadioButtons with ToggleGroup
 ToggleGroup radioGroup = new ToggleGroup();
 RadioButton radio1 = new RadioButton(text: "Option 1");
 RadioButton radio2 = new RadioButton(text: "Option 2");
 radio1.setToggleGroup(radioGroup);
 radio2.setToggleGroup(radioGroup);
 radio1.setSelected(true);
 centerBox.getChildren().add(new HBox(spacing: 5, new Label(text: "RadioButtons:"), radio1, radio2));
 }
}
- Toolbars:** On the left, there are icons for file operations (New, Open, Save, etc.) and a vertical toolbar with icons for text, play, forward, back, and other navigation functions.
- Bottom Status Bar:** Shows the file path: location > src > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication > start, and status: 34:46, LF, UTF-8, 4 spaces.

Example : Combobox



The screenshot shows an IDE interface with the following details:

- Title Bar:** Shows "JF JavaFXSimpleApplication" and "HelloApplication.java".
- Project Explorer:** Shows files like "hello-view.fxml", "HelloController.java", and "HelloApplication.java".
- Code Editor:** Displays Java code for a JavaFX application. The code includes imports for `javafx.scene.control.ComboBox`, `javafx.scene.control.ListView`, `javafx.scene.control.Slider`, and `javafx.scene.control.ProgressBar`. It creates a `ComboBox` with items "Choice A", "Choice B", and "Choice C", sets its value to "Choice A", and adds it to a `HBox`. It also creates a `ListView` with items "Item 1" through "Item 4", sets its preferred height to 100, and adds it to a `HBox`. A `Slider` is created with min 0, max 100, and value 50, with a value listener that updates a `Label` to show the rounded slider value. Finally, a `ProgressBar` is created.
- Sidebar:** Includes icons for file operations (New, Open, Save, Find, etc.) and a search bar.
- Status Bar:** Shows the file path "location > src > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication.java", line numbers (34:46), encoding (UTF-8), and code style settings (4 spaces).

Example : ProgressBar

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** Shows "JF JavaFXSimpleApplication" and "Version control".
- Toolbars:** Standard JavaFX toolbars for file operations.
- Code Editor:** The main window displays the `HelloApplication.java` file. The code implements a JavaFX application with a central progress bar, a bottom hyperlink, and key event handling. The code uses Java 8 features like `Optional` and `Stream`.
- Sidebar:** Includes a tree view of the project structure, a search bar, and various navigation icons.

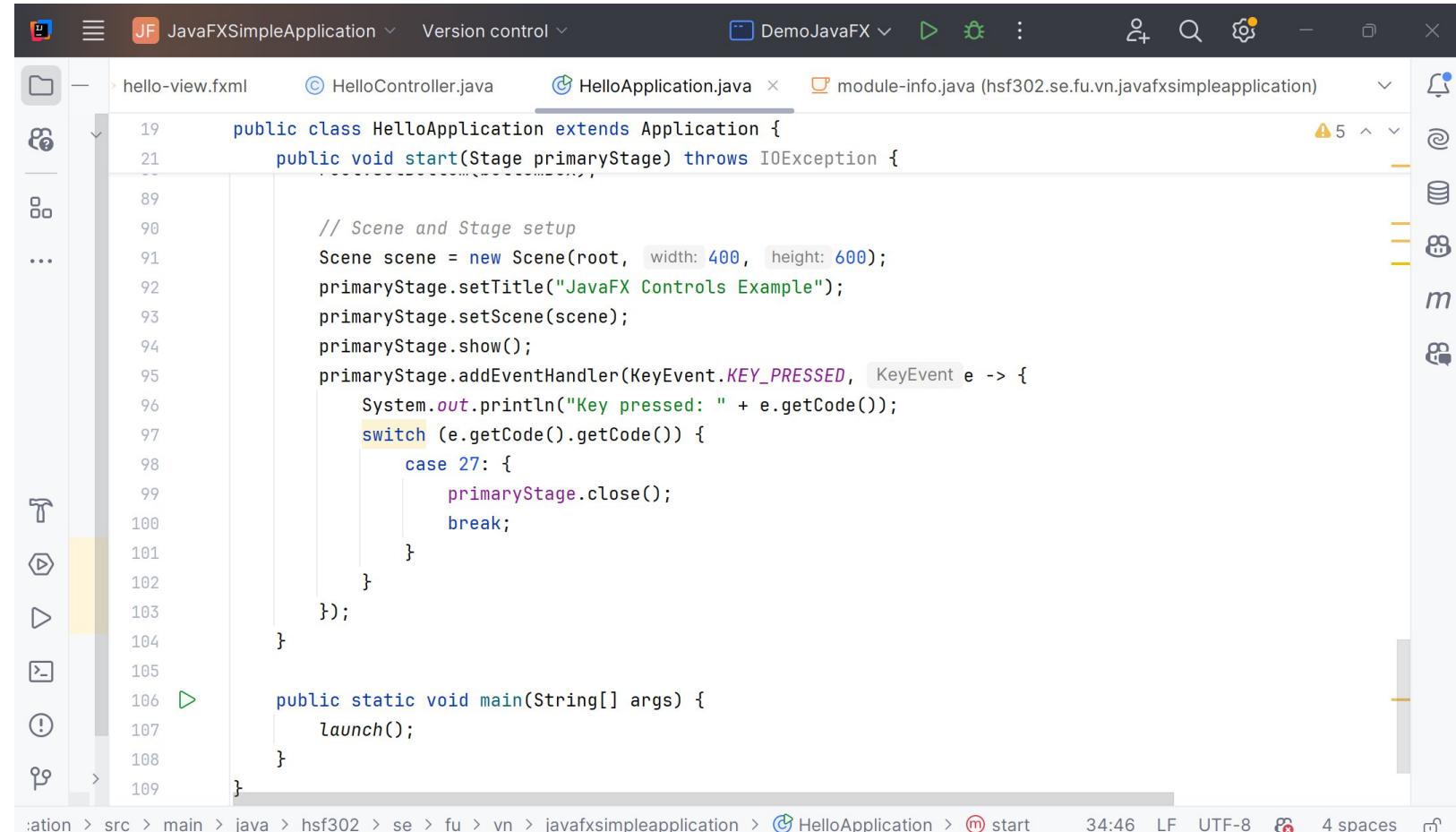
```
public class HelloApplication extends Application {
    public void start(Stage primaryStage) throws IOException {
        // ProgressBar
        ProgressBar progressBar = new ProgressBar(0.5);
        centerBox.getChildren().add(new HBox(spacing: 5, new Label(text: "ProgressBar:"), progressBar));

        // Add centerBox to the center of BorderPane
        root.setCenter(centerBox);

        // Bottom: Hyperlink and Separator
        Hyperlink hyperlink = new Hyperlink(text: "Visit GoF");
        hyperlink.setOnAction(ActionEvent e -> System.out.println("Hyperlink clicked!"));
        HBox bottomBox = new HBox(spacing: 10, new Separator(), hyperlink);
        root.setBottom(bottomBox);

        // Scene and Stage setup
        Scene scene = new Scene(root, width: 400, height: 600);
        primaryStage.setTitle("JavaFX Controls Example");
        primaryStage.setScene(scene);
        primaryStage.show();
        primaryStage.addEventHandler(KeyEvent.KEY_PRESSED, KeyEvent e -> {
            System.out.println("Key pressed: " + e.getCode());
            switch (e.getCode().getCode()) {
                case F1:
                    System.out.println("F1 key pressed!");
                    break;
            }
        });
    }
}
```

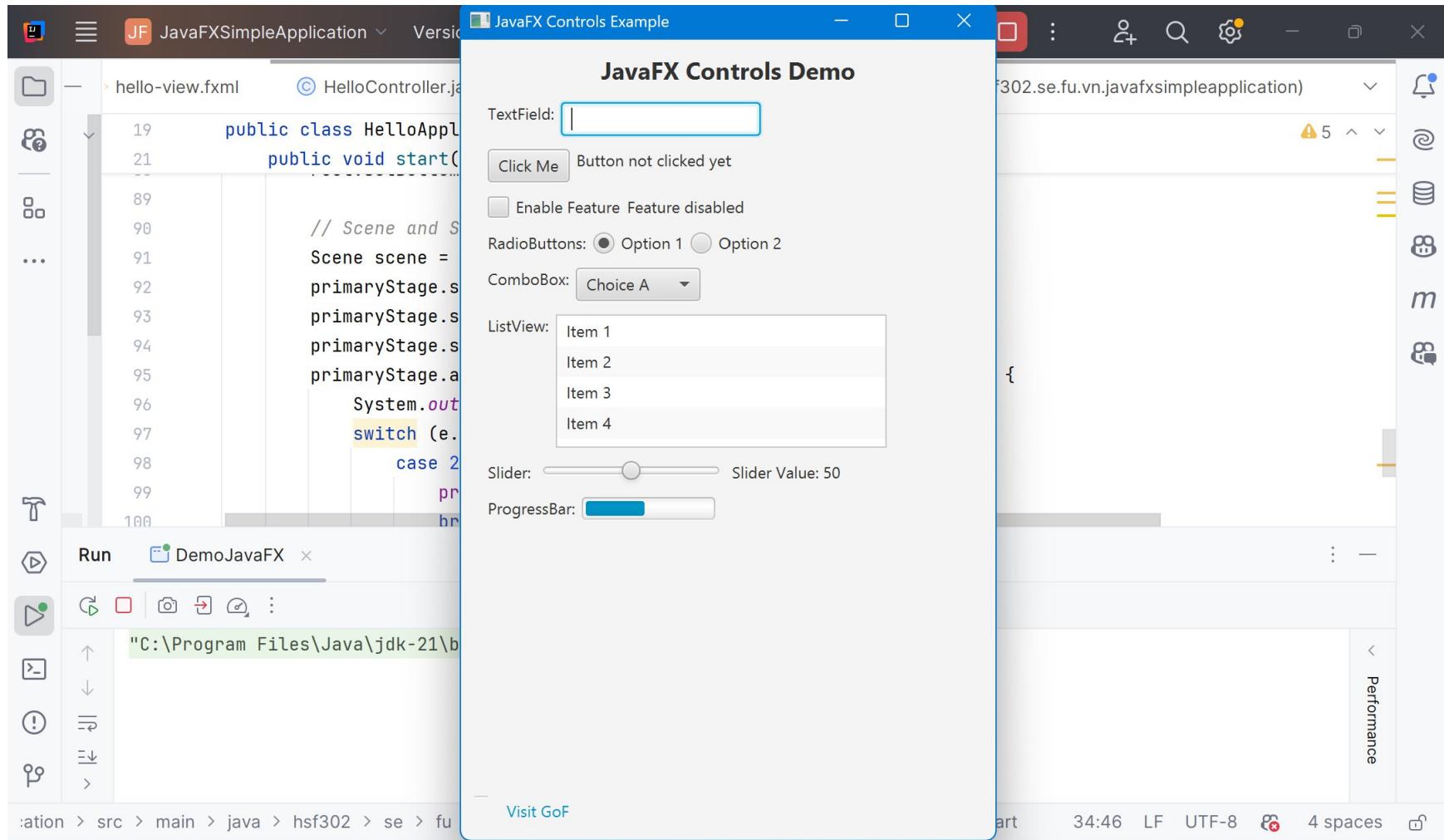
Example : Close



```
public class HelloApplication extends Application {
    public void start(Stage primaryStage) throws IOException {
        // Scene and Stage setup
        Scene scene = new Scene(root, width: 400, height: 600);
        primaryStage.setTitle("JavaFX Controls Example");
        primaryStage.setScene(scene);
        primaryStage.show();
        primaryStage.addEventHandler(KeyEvent.KEY_PRESSED, KeyEvent e -> {
            System.out.println("Key pressed: " + e.getCode());
            switch (e.getCode().getCode()) {
                case 27: {
                    primaryStage.close();
                    break;
                }
            }
        });
    }

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

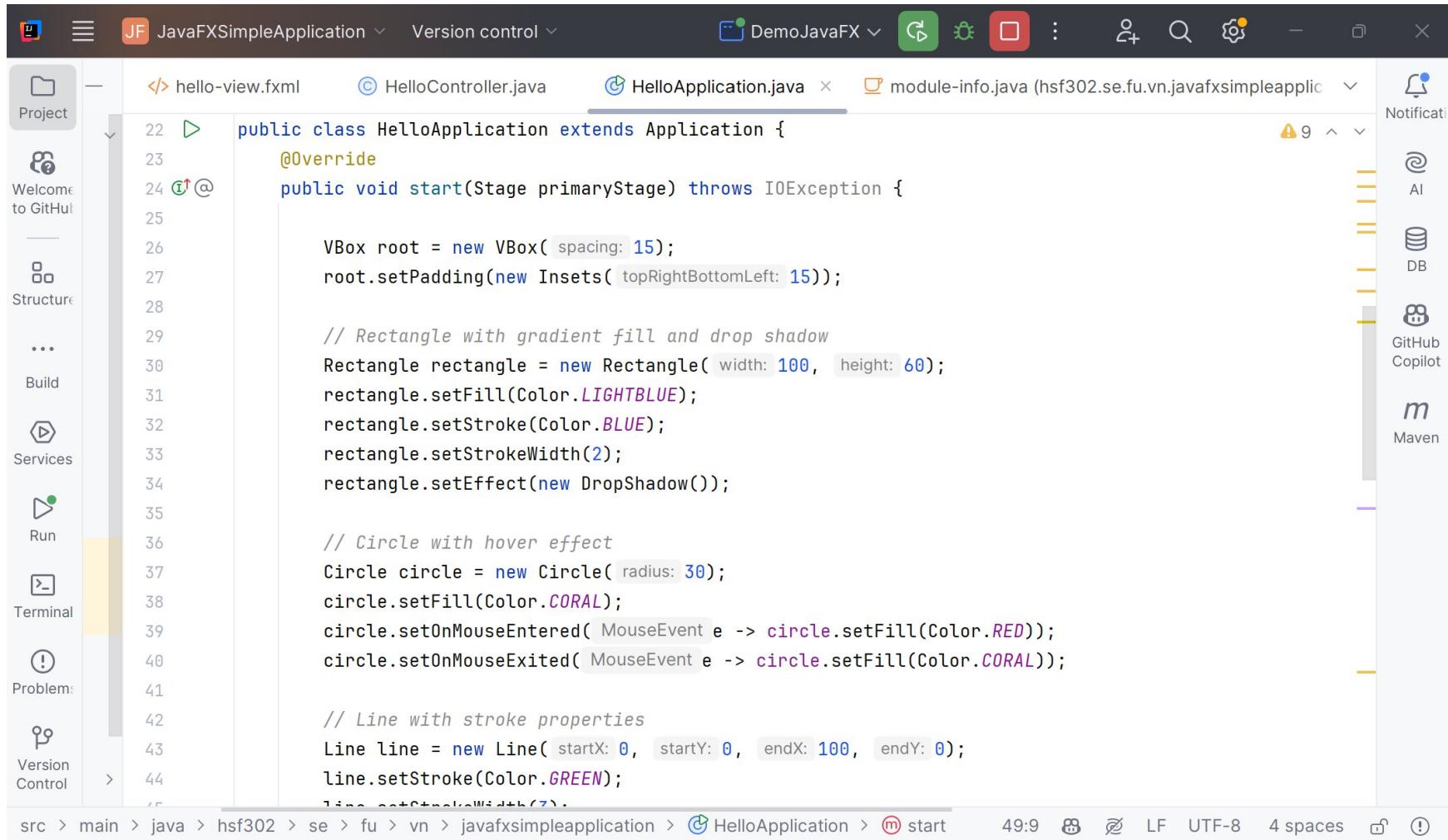
Result



JavaFX 2D Graphics

- ◆ JavaFX contains features that makes it easy to draw 2D graphics on the screen.
- ◆ 2D shape is represented by a class and all these classes belongs to the package **javafx.scene.shape**.
- ◆ Predefined shapes such as Line, Rectangle, Circle, Ellipse, Polygon, Polyline, Cubic Curve, Quad Curve, Arc.
- ◆ Path elements such as MoveTO Path Element, Line, Horizontal Line, Vertical Line, Cubic Curve, Quadratic Curve, Arc.
- ◆ In addition to these, you can also draw a 2D shape by parsing SVG path.

Example 2D Graphics



The screenshot shows the IntelliJ IDEA IDE interface with the following details:

- Title Bar:** JavaFXSimpleApplication > Version control > DemoJavaFX
- Project Tool Window:** Shows files like hello-view.fxml, HelloController.java, and HelloApplication.java.
- Code Editor:** The HelloApplication.java file is open, displaying Java code for creating a JavaFX application with 2D graphics elements (VBox, Rectangle, Circle, Line).
- Right Sidebar:** Contains icons for Notifications, AI, DB, GitHub Copilot, and Maven.
- Bottom Status Bar:** Shows the file path (src > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication.java), line numbers (49:9), and various encoding and font settings.

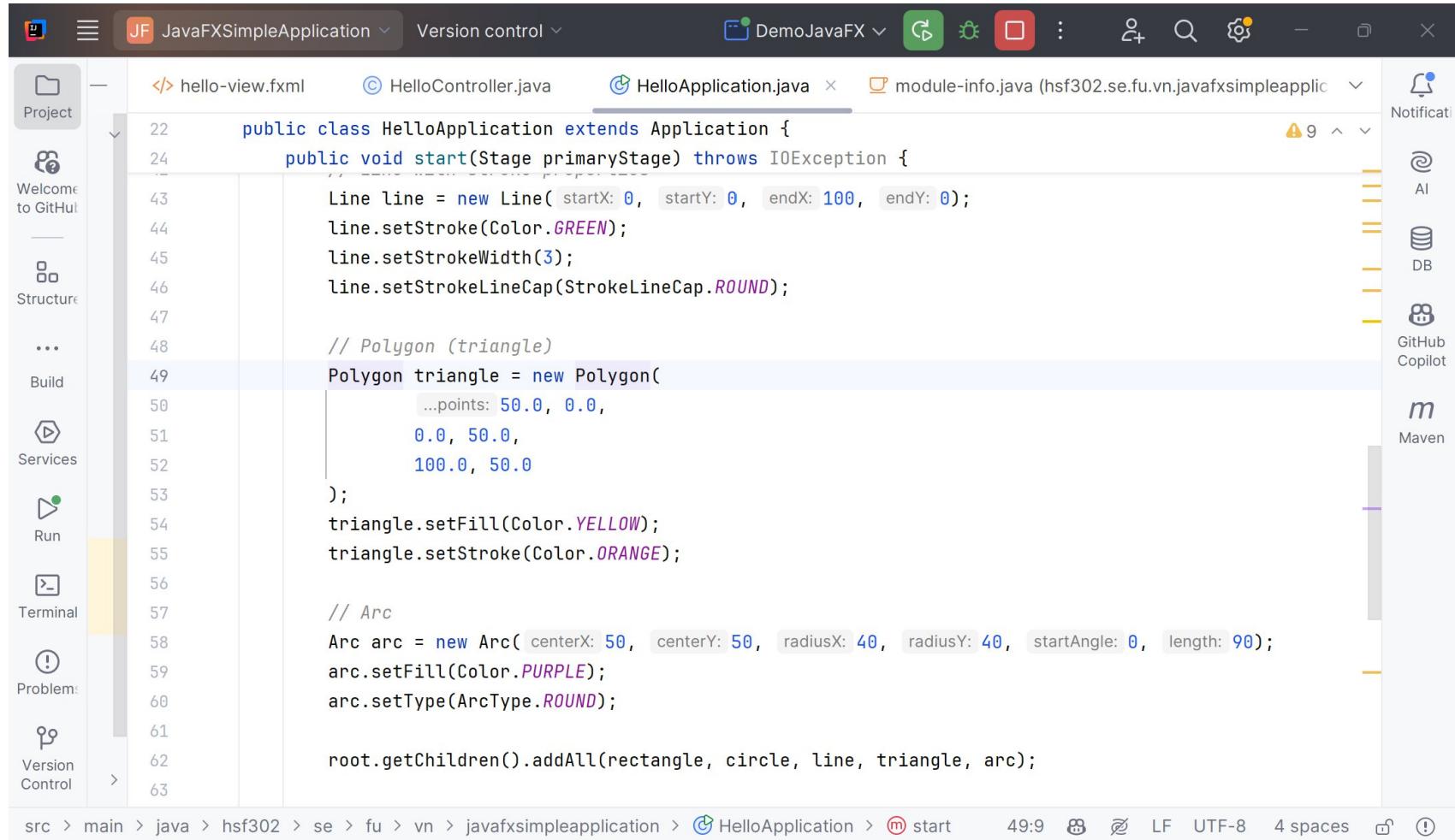
```
public class HelloApplication extends Application {
    @Override
    public void start(Stage primaryStage) throws IOException {
        VBox root = new VBox( spacing: 15 );
        root.setPadding(new Insets( topRightBottomLeft: 15 ));

        // Rectangle with gradient fill and drop shadow
        Rectangle rectangle = new Rectangle( width: 100, height: 60 );
        rectangle.setFill(Color.LIGHTBLUE);
        rectangle.setStroke(Color.BLUE);
        rectangle.setStrokeWidth(2);
        rectangle.setEffect(new DropShadow());

        // Circle with hover effect
        Circle circle = new Circle( radius: 30 );
        circle.setFill(Color.CORAL);
        circle.setOnMouseEntered( MouseEvent e -> circle.setFill(Color.RED));
        circle.setOnMouseExited( MouseEvent e -> circle.setFill(Color.CORAL));

        // Line with stroke properties
        Line line = new Line( startX: 0, startY: 0, endX: 100, endY: 0 );
        line.setStroke(Color.GREEN);
        line.setStrokeWidth(7);
    }
}
```

Example 2D Graphics



The screenshot shows a JavaFX application being developed in an IDE. The code in `HelloApplication.java` creates a scene with several graphical elements:

```
public class HelloApplication extends Application {
    public void start(Stage primaryStage) throws IOException {
        Line line = new Line( 0, 0, 100, 0);
        line.setStroke(Color.GREEN);
        line.setStrokeWidth(3);
        line.setStrokeLineCap(StrokeLineCap.ROUND);

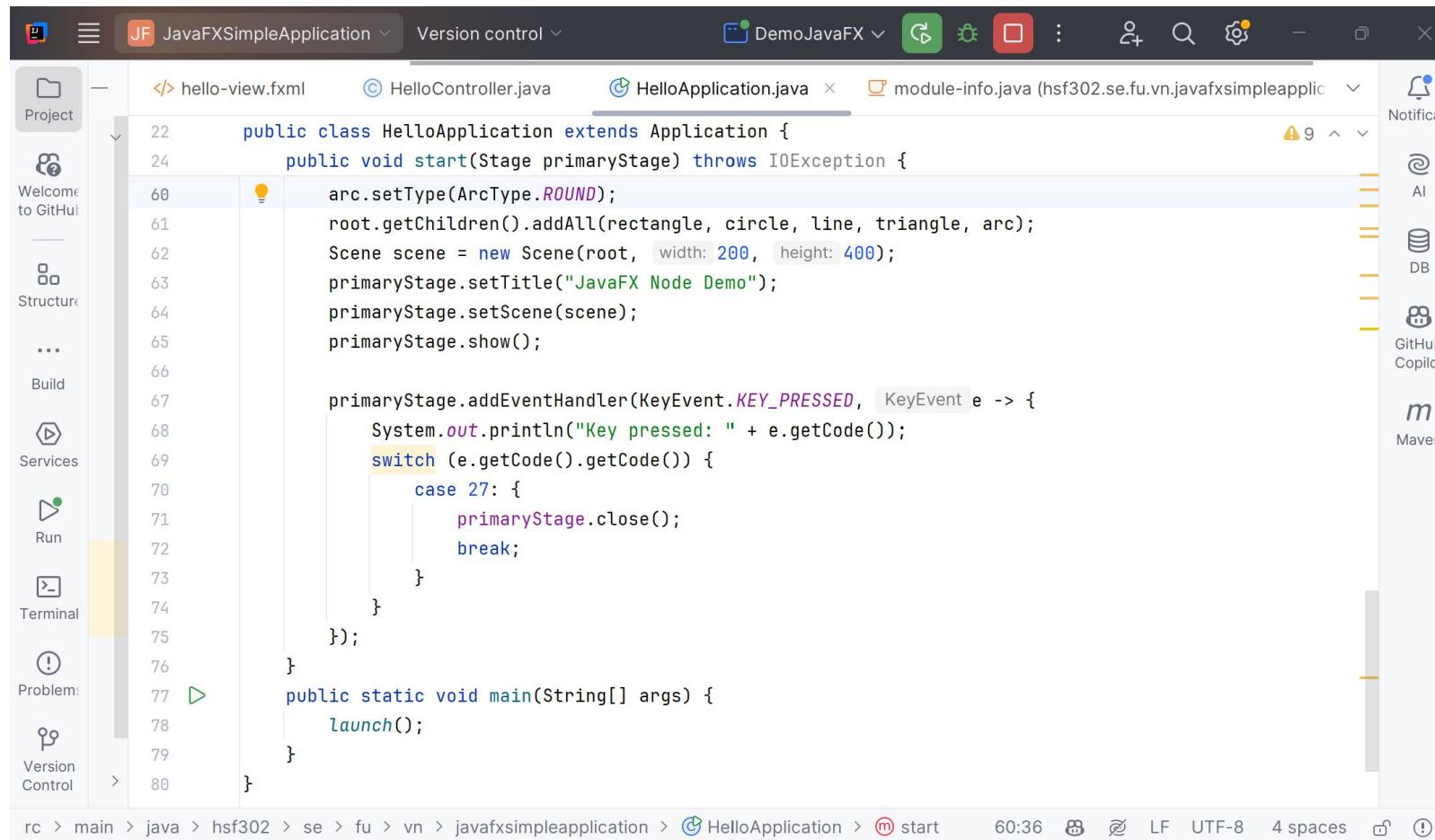
        // Polygon (triangle)
        Polygon triangle = new Polygon(
            ...points: 50.0, 0.0,
            0.0, 50.0,
            100.0, 50.0
        );
        triangle.setFill(Color.YELLOW);
        triangle.setStroke(Color.ORANGE);

        // Arc
        Arc arc = new Arc( centerX: 50, centerY: 50, radiusX: 40, radiusY: 40, startAngle: 0, length: 90);
        arc.setFill(Color.PURPLE);
        arc.setType(ArcType.ROUND);

        root.getChildren().addAll(rectangle, circle, line, triangle, arc);
    }
}
```

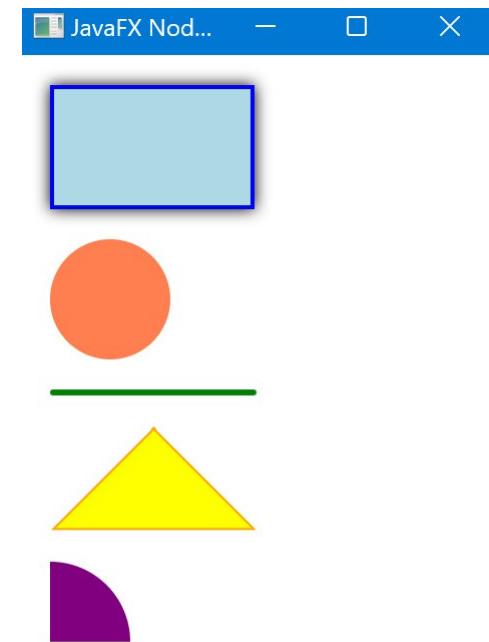
The IDE interface includes a Project sidebar, a central code editor with syntax highlighting, and various toolbars and panels on the right.

Example 2D Graphics



The screenshot shows an IDE interface with the following details:

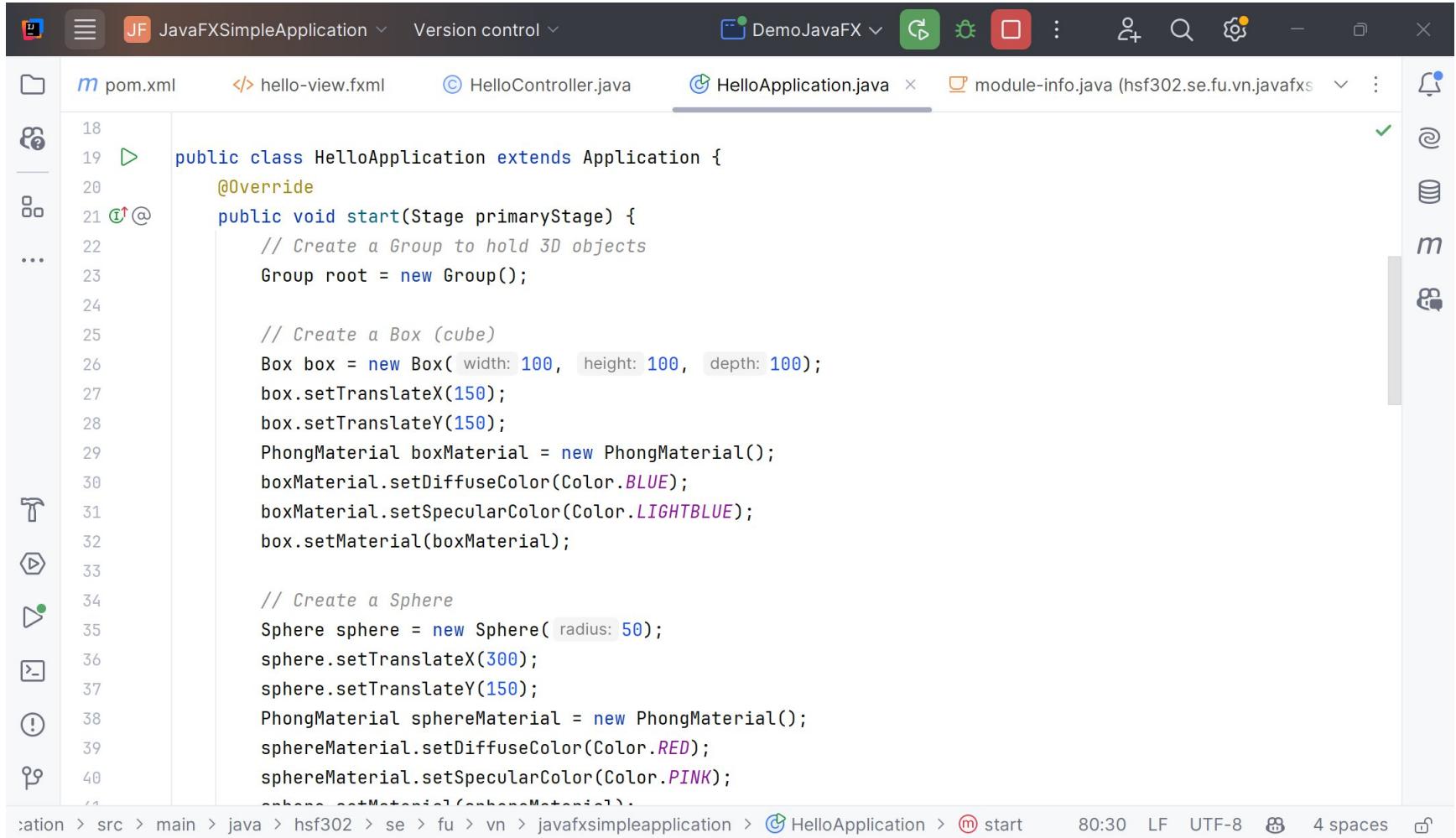
- Title Bar:** JF JavaFXSimpleApplication, Version control.
- Project Explorer:** Shows files: hello-view.fxml, HelloController.java, HelloApplication.java, and module-info.java.
- Code Editor:** Displays the `HelloApplication.java` file with JavaFX code. The code initializes a stage, adds various shapes (rectangle, circle, line, triangle, arc) to a scene, sets the stage title to "JavaFX Node Demo", and handles key events for closing the window.
- Toolbars and Status Bar:** Includes standard icons for file operations, search, and help. The status bar at the bottom shows the path: rc > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication > start, and system information like time (60:36), encoding (UTF-8), and font size (4 spaces).



JavaFX 3D Graphics

- ◆ JavaFX contains features that makes it easy to draw 3D graphics on the screen.
- ◆ In general, a 3D shape is a geometrical figure that can be drawn on the XYZ plane.
- ◆ They are defined by two or more dimensions, commonly length, width and depth.
- ◆ 3D shapes supported by JavaFX include a Cylinder, Sphere and a Box.
- ◆ All 3D shape classes belong to the package **javafx.scene.shape**.

Example 3D Graphics



The screenshot shows a Java IDE interface with the following details:

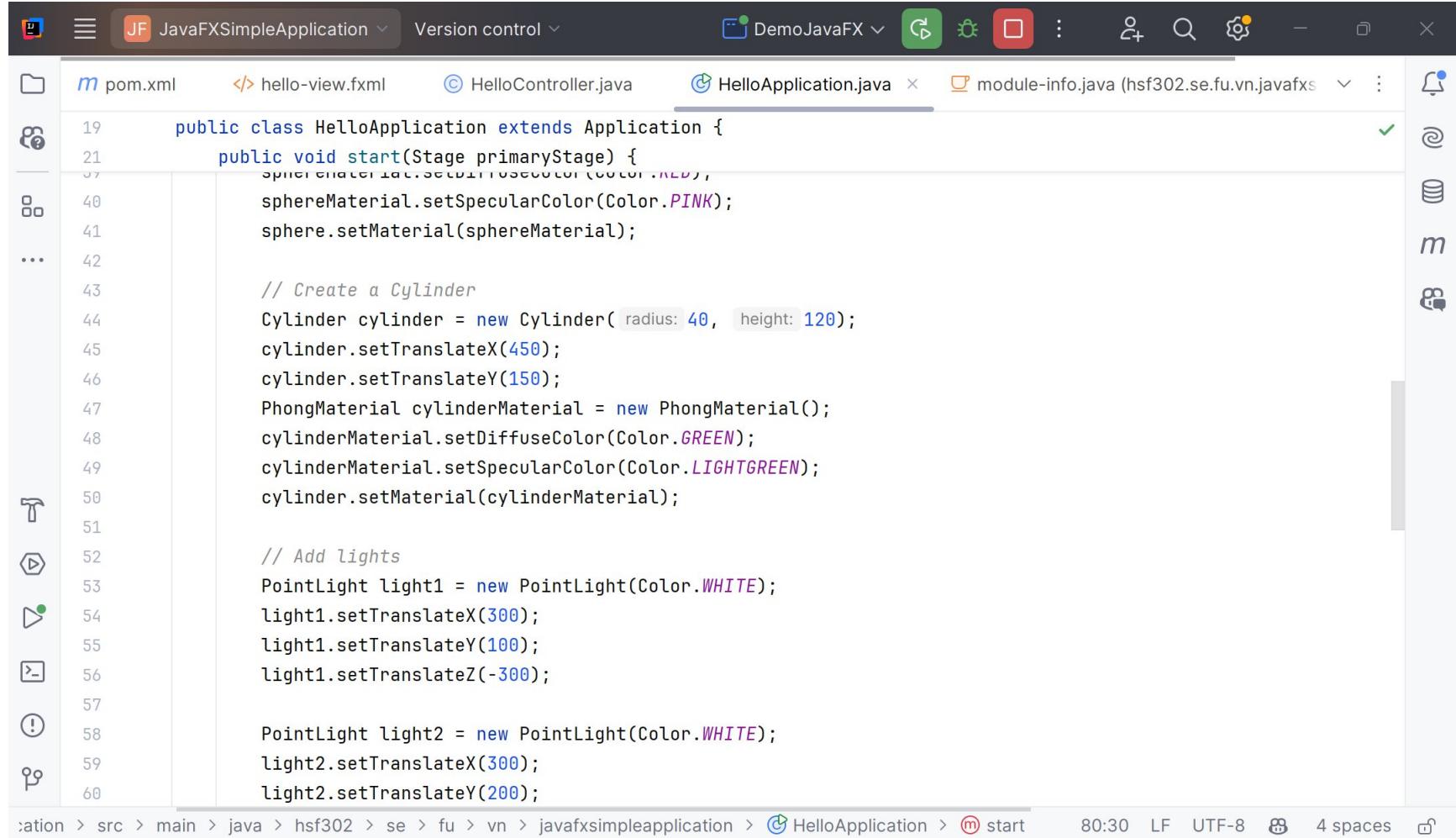
- Project Structure:** JavaFXSimpleApplication (selected), DemoJavaFX.
- Code Editor:** HelloApplication.java (active tab). The code implements a JavaFX application that creates a 3D scene with a blue cube and a red sphere.
- Code Snippet:**

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Create a Group to hold 3D objects
        Group root = new Group();

        // Create a Box (cube)
        Box box = new Box( width: 100, height: 100, depth: 100);
        box.setTranslateX(150);
        box.setTranslateY(150);
        PhongMaterial boxMaterial = new PhongMaterial();
        boxMaterial.setDiffuseColor(Color.BLUE);
        boxMaterial.setSpecularColor(Color.LIGHTBLUE);
        box.setMaterial(boxMaterial);

        // Create a Sphere
        Sphere sphere = new Sphere( radius: 50);
        sphere.setTranslateX(300);
        sphere.setTranslateY(150);
        PhongMaterial sphereMaterial = new PhongMaterial();
        sphereMaterial.setDiffuseColor(Color.RED);
        sphereMaterial.setSpecularColor(Color.PINK);
        sphere.setMaterial(sphereMaterial);
    }
}
```
- Toolbars and Status Bar:** Version control, pom.xml, hello-view.fxml, HelloController.java, module-info.java, file paths (src/main/java/hsf302/se/fu/vn/javafxsimpleapplication/HelloApplication.java), and status indicators (80:30, LF, UTF-8, 4 spaces).

Example 3D Graphics



The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows "JF JavaFXSimpleApplication" and "DemoJavaFX".
- Toolbars:** Version control, file operations, and search.
- Left Sidebar:** Shows icons for pom.xml, hello-view.fxml, HelloController.java, HelloApplication.java (selected), module-info.java, and other files.
- Code Editor:** Displays the `HelloApplication.java` file with Java code for creating a 3D scene with a sphere and a cylinder, and adding lights.

```
public class HelloApplication extends Application {
    public void start(Stage primaryStage) {
        Sphere sphere = new Sphere();
        sphereMaterial.setSpecularColor(Color.PINK);
        sphere.setMaterial(sphereMaterial);

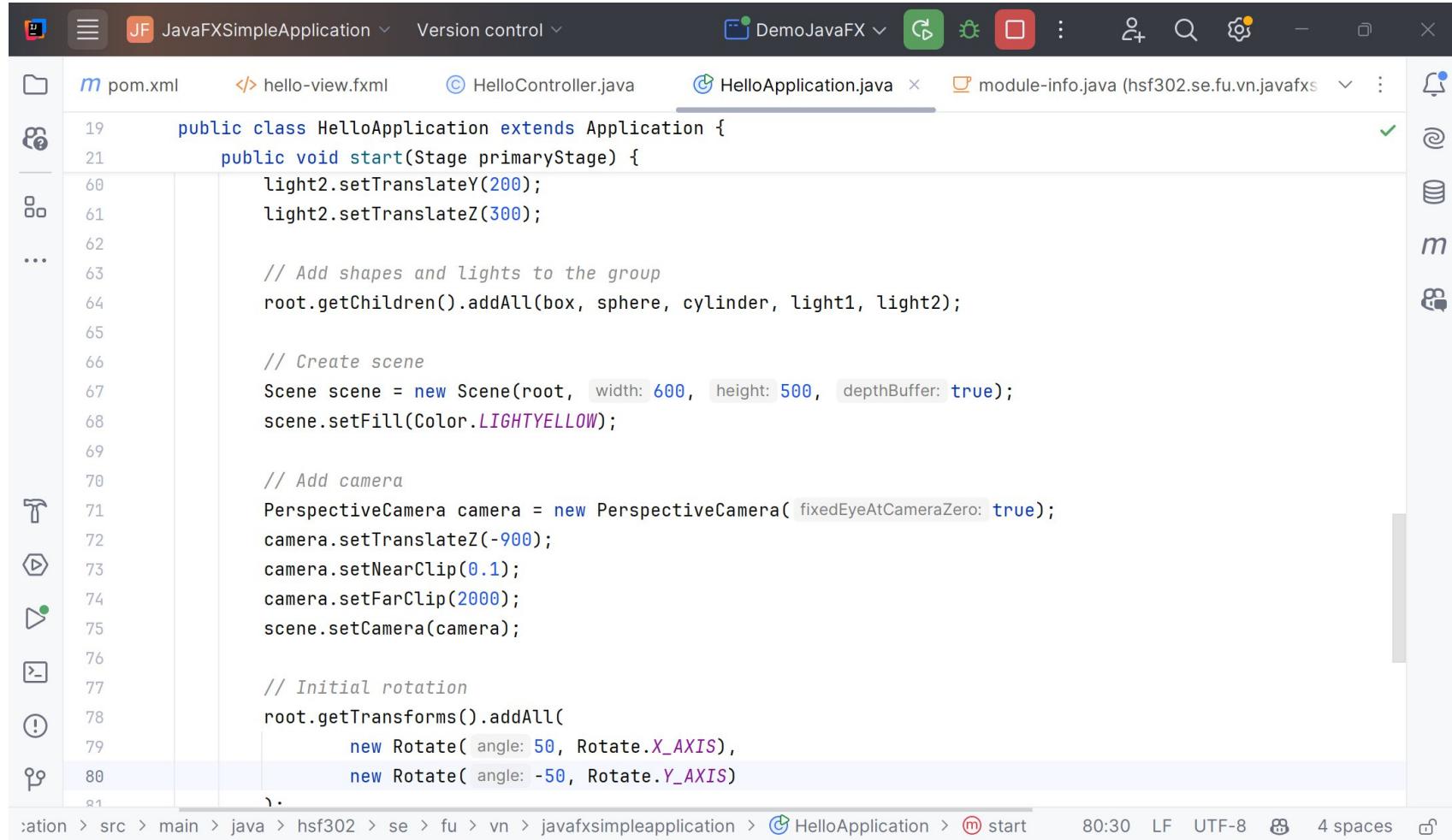
        // Create a Cylinder
        Cylinder cylinder = new Cylinder( radius: 40, height: 120 );
        cylinder.setTranslateX(450);
        cylinder.setTranslateY(150);
        PhongMaterial cylinderMaterial = new PhongMaterial();
        cylinderMaterial.setDiffuseColor(Color.GREEN);
        cylinderMaterial.setSpecularColor(Color.LIGHTGREEN);
        cylinder.setMaterial(cylinderMaterial);

        // Add lights
        PointLight light1 = new PointLight(Color.WHITE);
        light1.setTranslateX(300);
        light1.setTranslateY(100);
        light1.setTranslateZ(-300);

        PointLight light2 = new PointLight(Color.WHITE);
        light2.setTranslateX(300);
        light2.setTranslateY(200);
    }
}
```

- Bottom Status Bar:** Shows navigation (location, src, main, java, etc.), file status (start), time (80:30), encoding (UTF-8), and code style (4 spaces).

Example 3D Graphics



The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows the project name "JavaFXSimpleApplication" and the file "HelloApplication.java" is the active tab.
- File Explorer:** On the left, it lists files like pom.xml, hello-view.fxml, HelloController.java, and HelloApplication.java.
- Code Editor:** The main area displays the Java code for "HelloApplication.java". The code initializes a scene with a light yellow fill, adds 3D shapes (box, sphere, cylinder) and lights (light1, light2) to the root node, creates a perspective camera, and applies initial rotations to the root node's transforms.
- Status Bar:** At the bottom, it shows the file path: location > src > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication > start, and various system information like date, time, and encoding.

```
public class HelloApplication extends Application {
    public void start(Stage primaryStage) {
        light2.setTranslateY(200);
        light2.setTranslateZ(300);

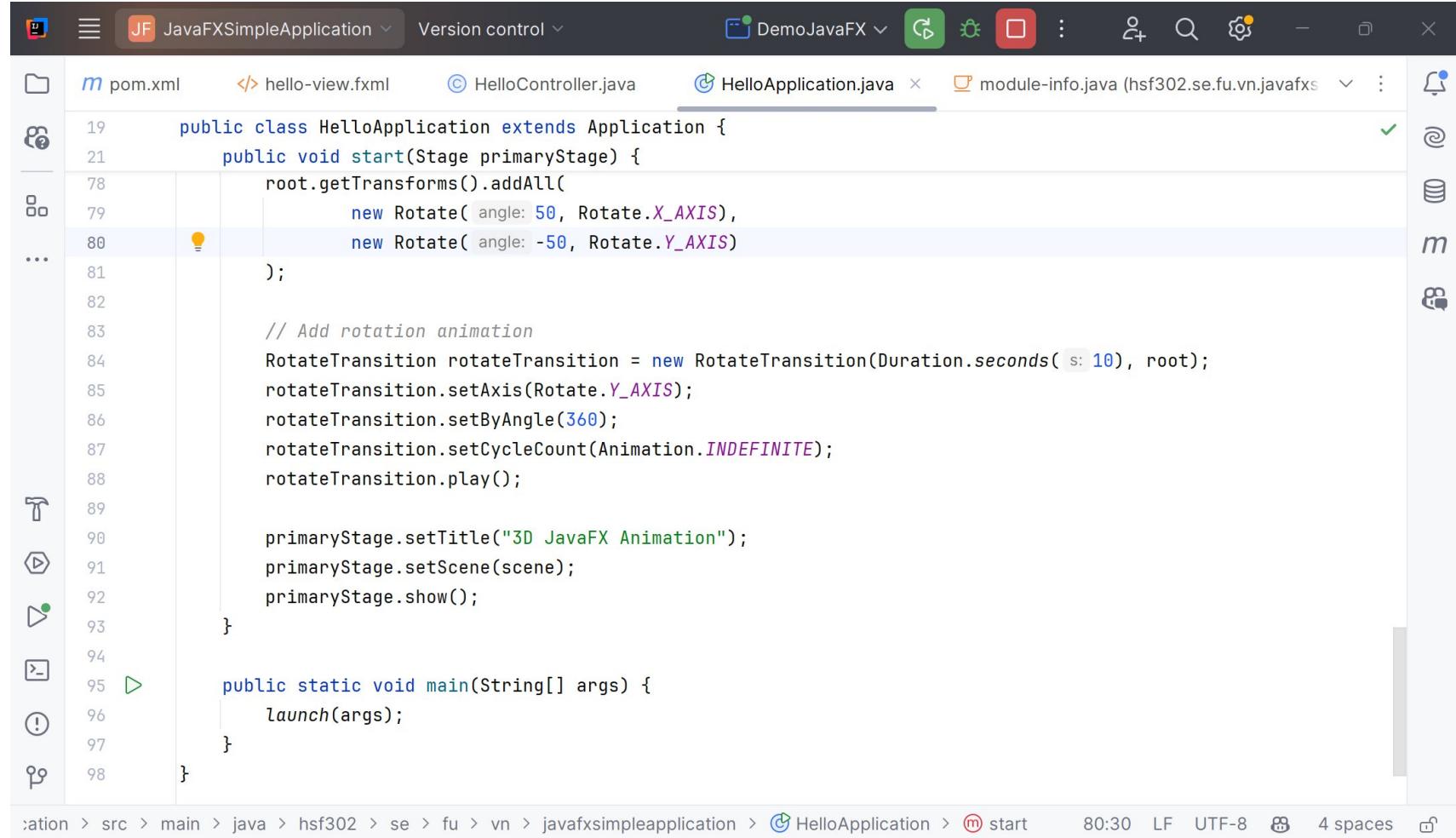
        // Add shapes and lights to the group
        root.getChildren().addAll(box, sphere, cylinder, light1, light2);

        // Create scene
        Scene scene = new Scene(root, width: 600, height: 500, depthBuffer: true);
        scene.setFill(Color.LIGHTYELLOW);

        // Add camera
        PerspectiveCamera camera = new PerspectiveCamera(fixedEyeAtCameraZero: true);
        camera.setTranslateZ(-900);
        camera.setNearClip(0.1);
        camera.setFarClip(2000);
        scene.setCamera(camera);

        // Initial rotation
        root.getTransforms().addAll(
            new Rotate(angle: 50, Rotate.X_AXIS),
            new Rotate(angle: -50, Rotate.Y_AXIS)
        );
    }
}
```

Example 3D Graphics



The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows "JF JavaFXSimpleApplication" and "DemoJavaFX".
- File Explorer:** Displays files: pom.xml, hello-view.fxml, HelloController.java, HelloApplication.java (selected), and module-info.java.
- Code Editor:** The "HelloApplication.java" file is open, containing the following Java code:

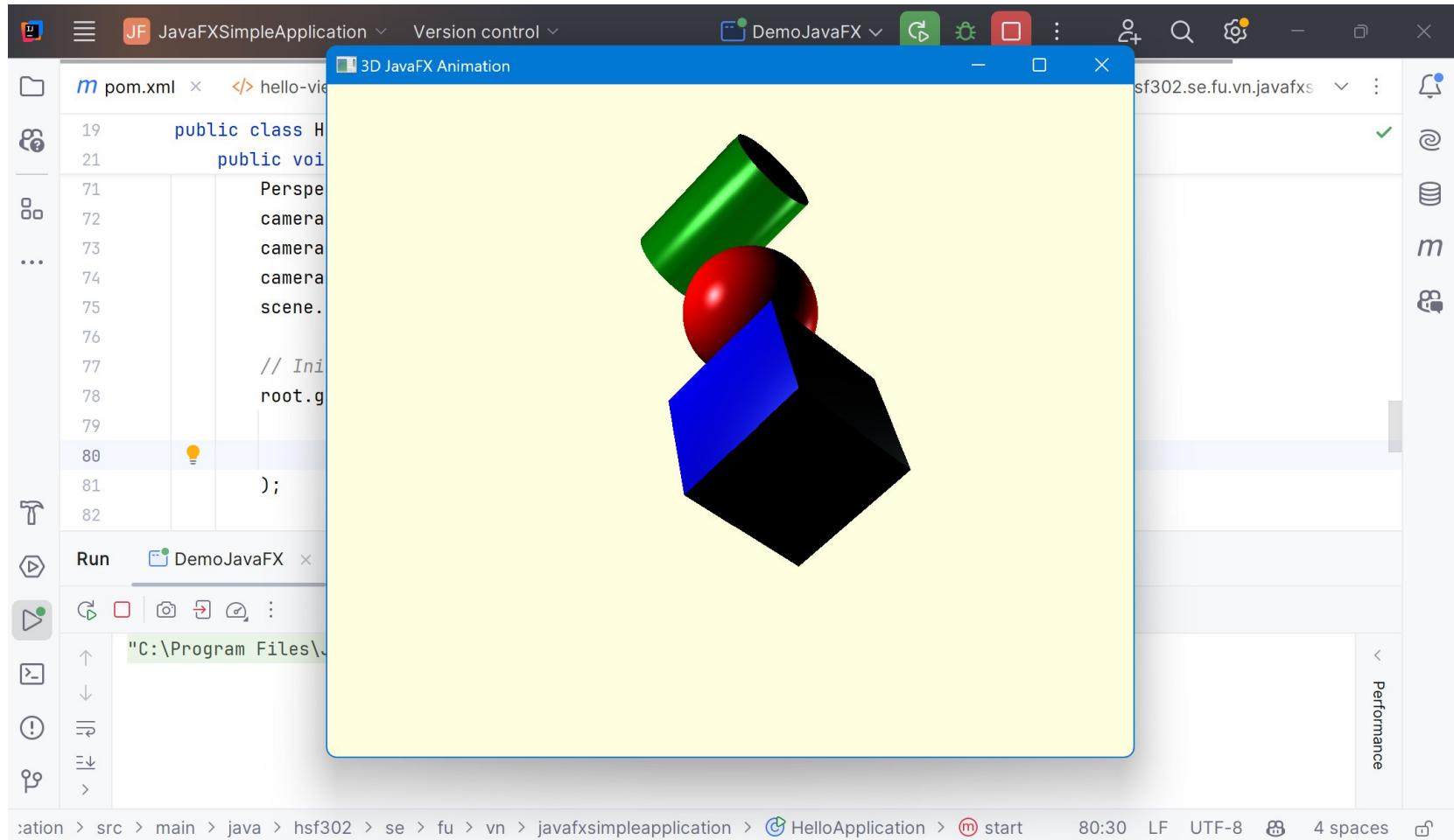
```
public class HelloApplication extends Application {
    public void start(Stage primaryStage) {
        root.getTransforms().addAll(
            new Rotate( angle: 50, Rotate.X_AXIS),
            new Rotate( angle: -50, Rotate.Y_AXIS)
        );

        // Add rotation animation
        RotateTransition rotateTransition = new RotateTransition(Duration.seconds( s: 10 ), root);
        rotateTransition.setAxis(Rotate.Y_AXIS);
        rotateTransition.setByAngle(360);
        rotateTransition.setCycleCount(Animation.INDEFINITE);
        rotateTransition.play();

        primaryStage.setTitle("3D JavaFX Animation");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```
- Toolbars and Status Bar:** Includes icons for file operations, search, and settings. The status bar at the bottom shows the path "location > src > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication", and the time "80:30" along with other system information.

Example 3D Graphics



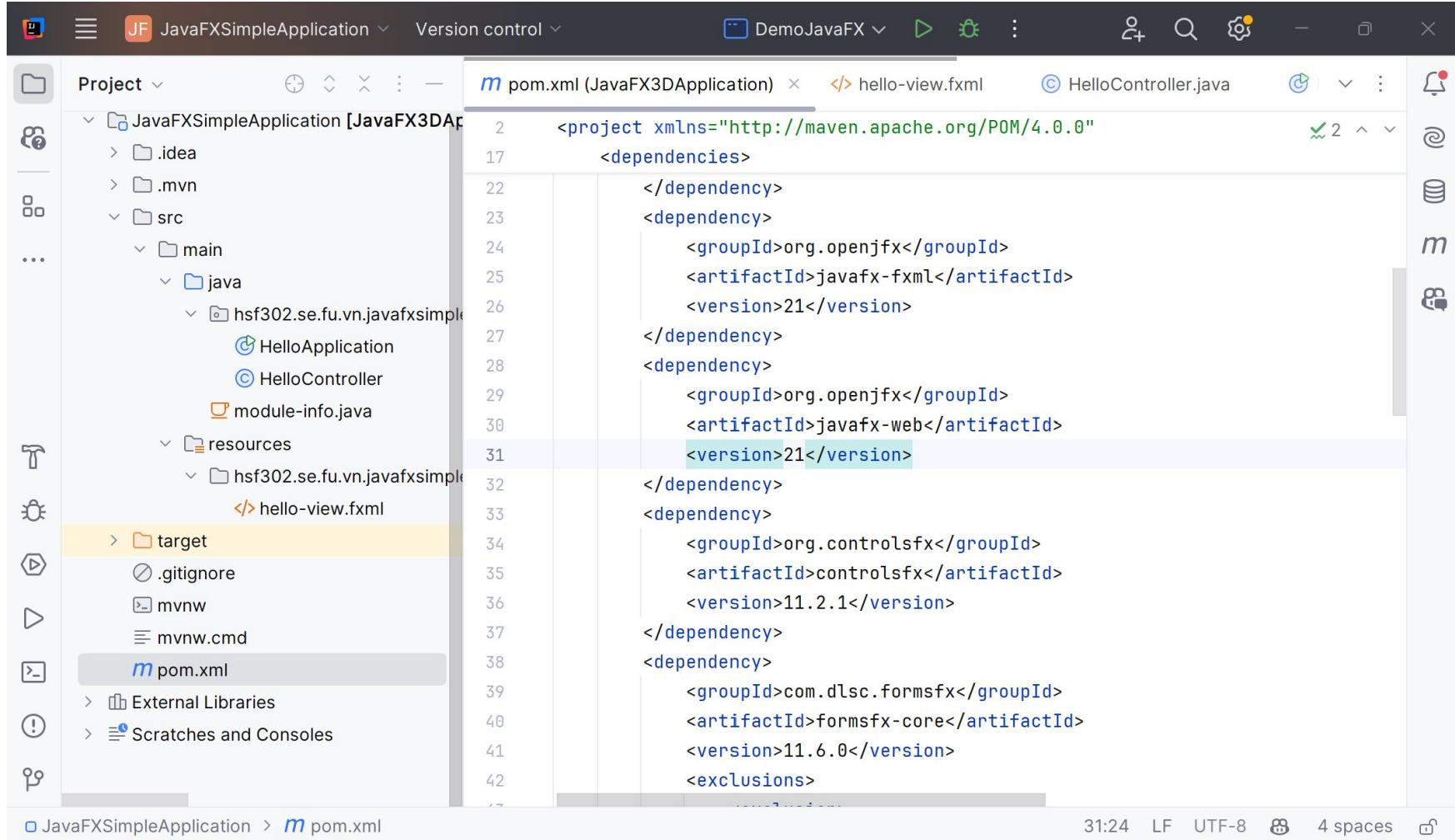
JavaFX Audio, Video

- ◆ JavaFX contains features that makes it easy to play audio, video in JavaFX applications.
- ◆ The package **javafx.scene.media** contains the classes and interfaces to provide media functionality in JavaFX.
- ◆ It is provided in the form of three components
 - Media Object – This represents a media file
 - Media Player – To play media content.
 - Media View – To display media.

JavaFX WebView

- ◆ The JavaFX WebView (`javafx.scene.web.WebView`) component is capable of showing web pages (HTML, CSS, SVG, JavaScript) inside a JavaFX application. As such, the JavaFX WebView is a mini browser.
- ◆ The JavaFX WebView uses the WebKit open source browser engine internally to render the web pages.
- ◆ The JavaFX WebView WebEngine (`javafx.scene.web.WebEngine`) is an internal component used by the WebView to load the data that is to be displayed inside the WebView.
- ◆ To make the WebView WebEngine load data, you must first obtain the WebEngine instance from the WebView.

Update pom.xml

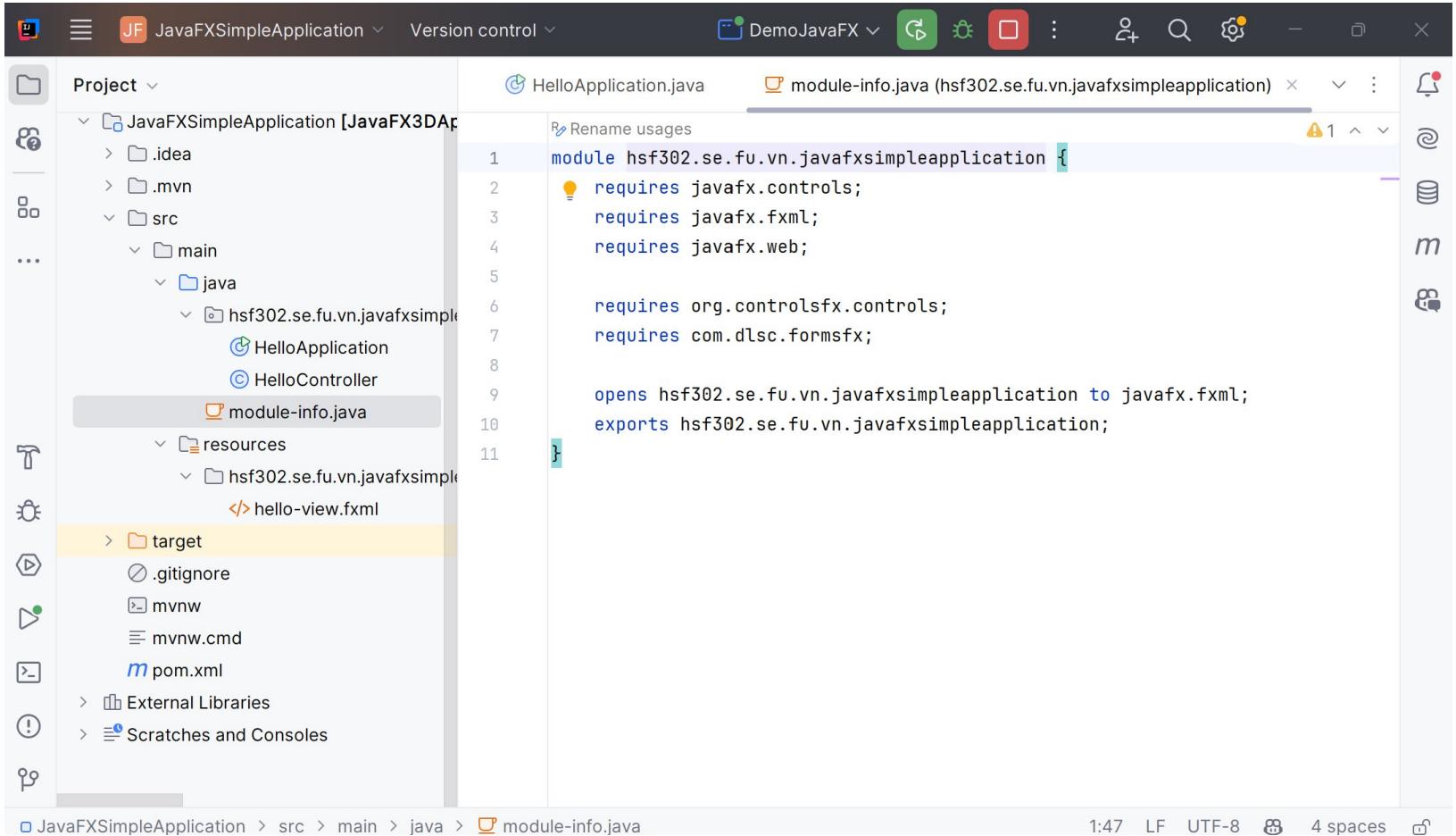


The screenshot shows the IntelliJ IDEA interface with the project `JavaFXSimpleApplication` selected in the left sidebar. The `pom.xml` file is open in the main editor window, displaying Maven dependency code. The code includes dependencies for `javafx-fxml`, `javafx-web`, `controlsfx`, and `formsfx-core`. The `target` directory is highlighted in the project tree. The status bar at the bottom indicates the file is 31:24, LF, UTF-8, with 4 spaces.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>hsf302.se.fu.vn.javafxsimple</groupId>
    <artifactId>JavaFXSimpleApplication</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-fxml</artifactId>
            <version>21</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-web</artifactId>
            <version>21</version>
        </dependency>
        <dependency>
            <groupId>org.controlsfx</groupId>
            <artifactId>controlsfx</artifactId>
            <version>11.2.1</version>
        </dependency>
        <dependency>
            <groupId>com.dlsc.formsfx</groupId>
            <artifactId>formsfx-core</artifactId>
            <version>11.6.0</version>
        </dependency>
        <exclusions>
            <exclusion>
                <groupId>org.controlsfx</groupId>
                <artifactId>controlsfx</artifactId>
            </exclusion>
        </exclusions>
    </dependencies>

```

Update module-info.java

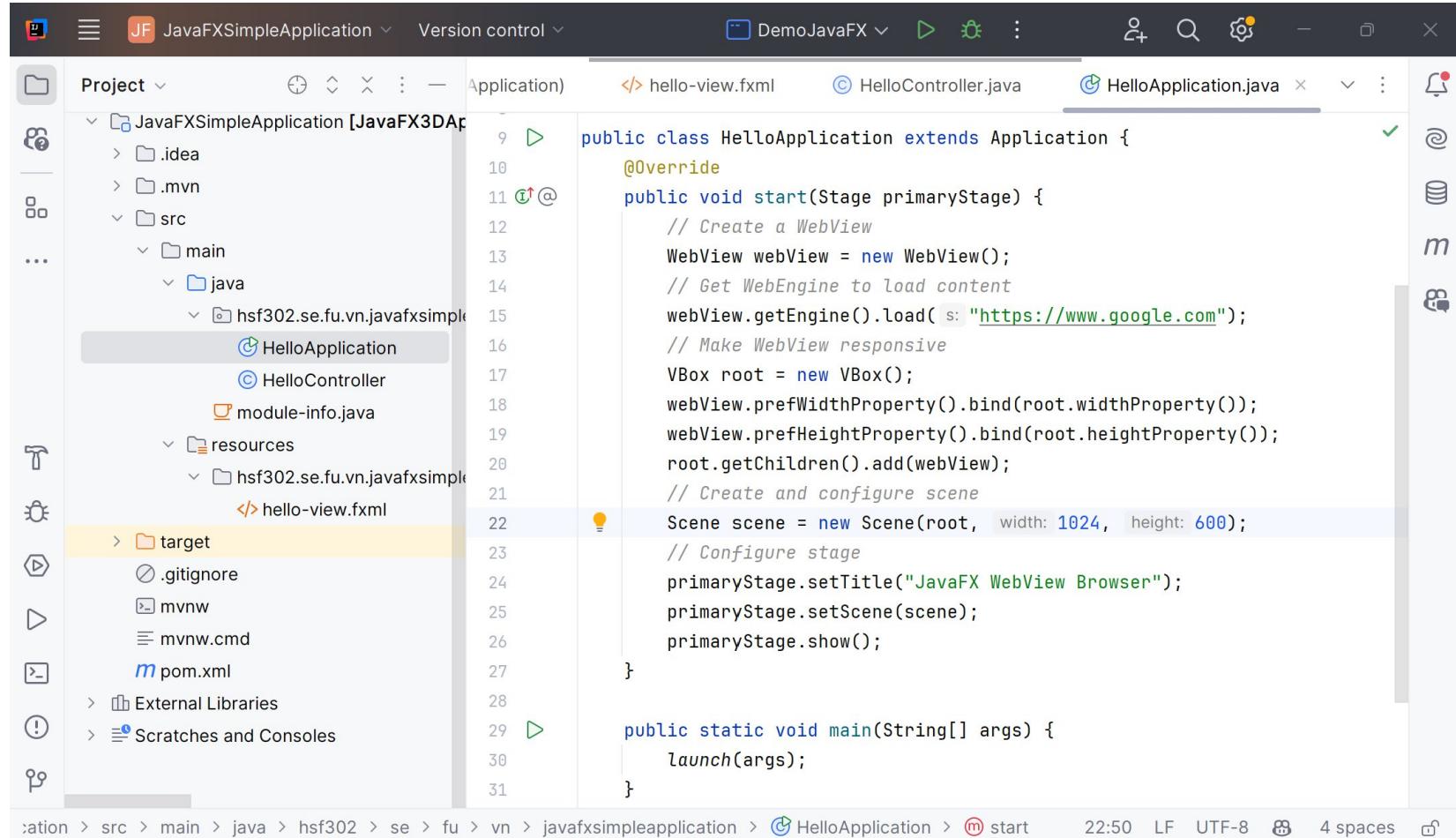


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** Shows "JF JavaFXSimpleApplication" and "Version control".
- Toolbars:** Standard IntelliJ toolbars for navigation and search.
- Left Panel (Project View):** Shows the project structure under "JavaFXSimpleApplication [JavaFX3DAp]". It includes ".idea", ".mvn", "src" (with "main" and "java" subfolders), "hsf302.se.fu.vn.javafxsimpleapplication" package (containing "HelloApplication", "HelloController", and "module-info.java"), "resources" folder, and "target" folder (which is highlighted).
- Right Panel (Editor):** Displays the content of "module-info.java" (hsf302.se.fu.vn.javafxsimpleapplication). The code is as follows:

```
module hsf302.se.fu.vn.javafxsimpleapplication {  
    requires javafx.controls;  
    requires javafx.fxml;  
    requires javafx.web;  
  
    requires org.controlsfx.controls;  
    requires com.dlsc.formsfx;  
  
    opens hsf302.se.fu.vn.javafxsimpleapplication to javafx.fxml;  
    exports hsf302.se.fu.vn.javafxsimpleapplication;
```
- Bottom Status Bar:** Shows the path "JavaFXSimpleApplication > src > main > java > module-info.java", the timestamp "1:47", and file settings like "LF", "UTF-8", "4 spaces".

Update main.java

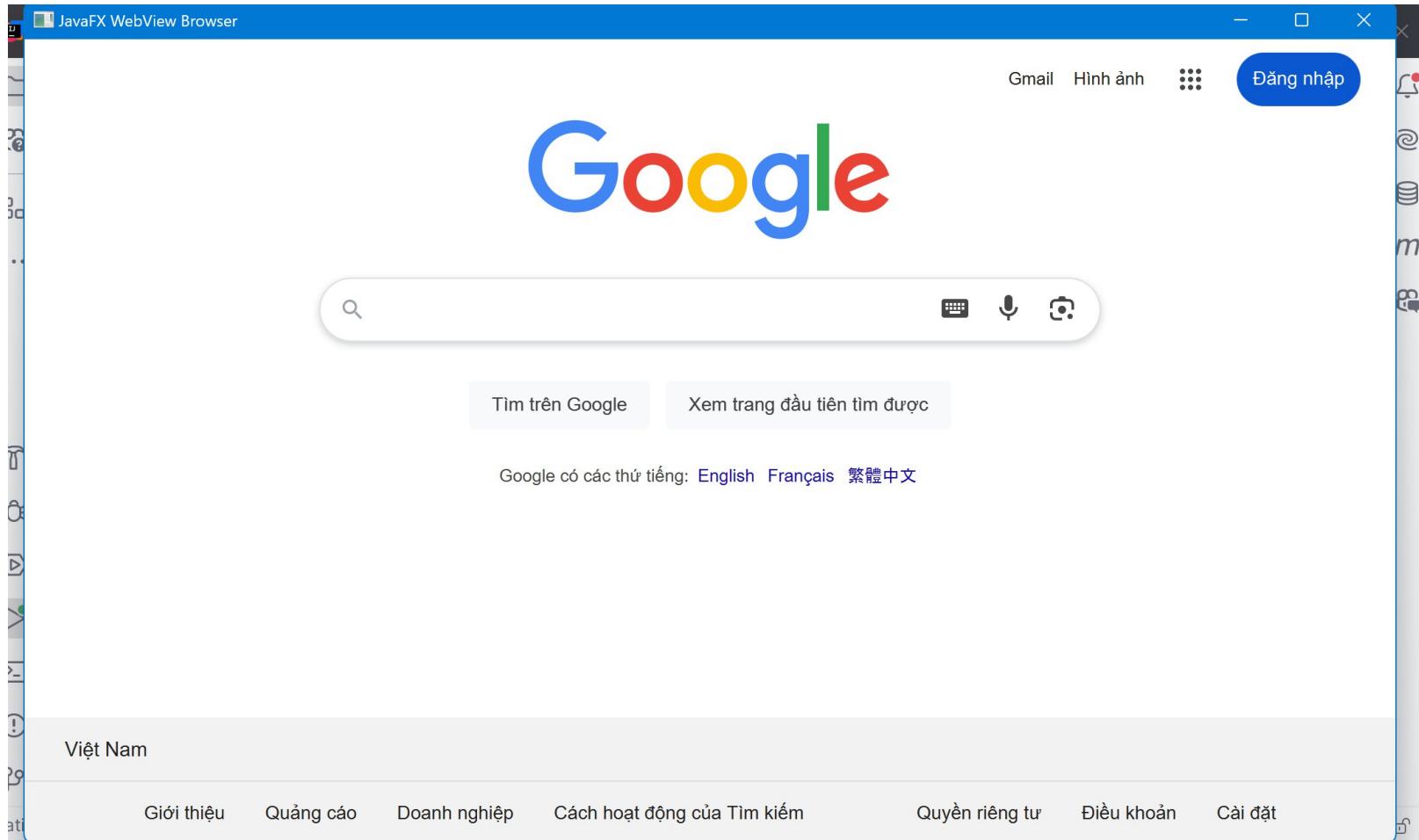


The screenshot shows a Java IDE interface with the following details:

- Title Bar:** Shows "JavaFXSimpleApplication" and "Version control".
- Toolbar:** Includes icons for file operations, search, and settings.
- Project Explorer:** Shows the project structure:
 - JavaFXSimpleApplication [JavaFX3DAp]
 - .idea
 - .mvn
 - src
 - main
 - java
 - hsf302.se.fu.vn.javafxsimple
 - HelloApplication
 - HelloController
 - module-info.java
 - resources
 - hsf302.se.fu.vn.javafxsimple
 - hello-view.fxml
 - target
 - .gitignore
 - mvnw
 - mvnw.cmd
 - pom.xml
 - External Libraries
 - Scratches and Consoles
 - Code Editor:** Displays the content of HelloApplication.java:

```
public class HelloApplication extends Application {
    @Override
    public void start(Stage primaryStage) {
        // Create a WebView
        WebView webView = new WebView();
        // Get WebEngine to load content
        webView.getEngine().load( s: "https://www.google.com");
        // Make WebView responsive
        VBox root = new VBox();
        webView.prefWidthProperty().bind(root.widthProperty());
        webView.prefHeightProperty().bind(root.heightProperty());
        root.getChildren().add(webView);
        // Create and configure scene
        Scene scene = new Scene(root, width: 1024, height: 600);
        // Configure stage
        primaryStage.setTitle("JavaFX WebView Browser");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```
 - Bottom Status Bar:** Shows the file path "location > src > main > java > hsf302 > se > fu > vn > javafxsimpleapplication > HelloApplication > start", and system information like "22:50 LF UTF-8 4 spaces".

JavaFX WebView



JavaFX Charts

The JavaFX charts available in the **javafx.scene.chart** package.

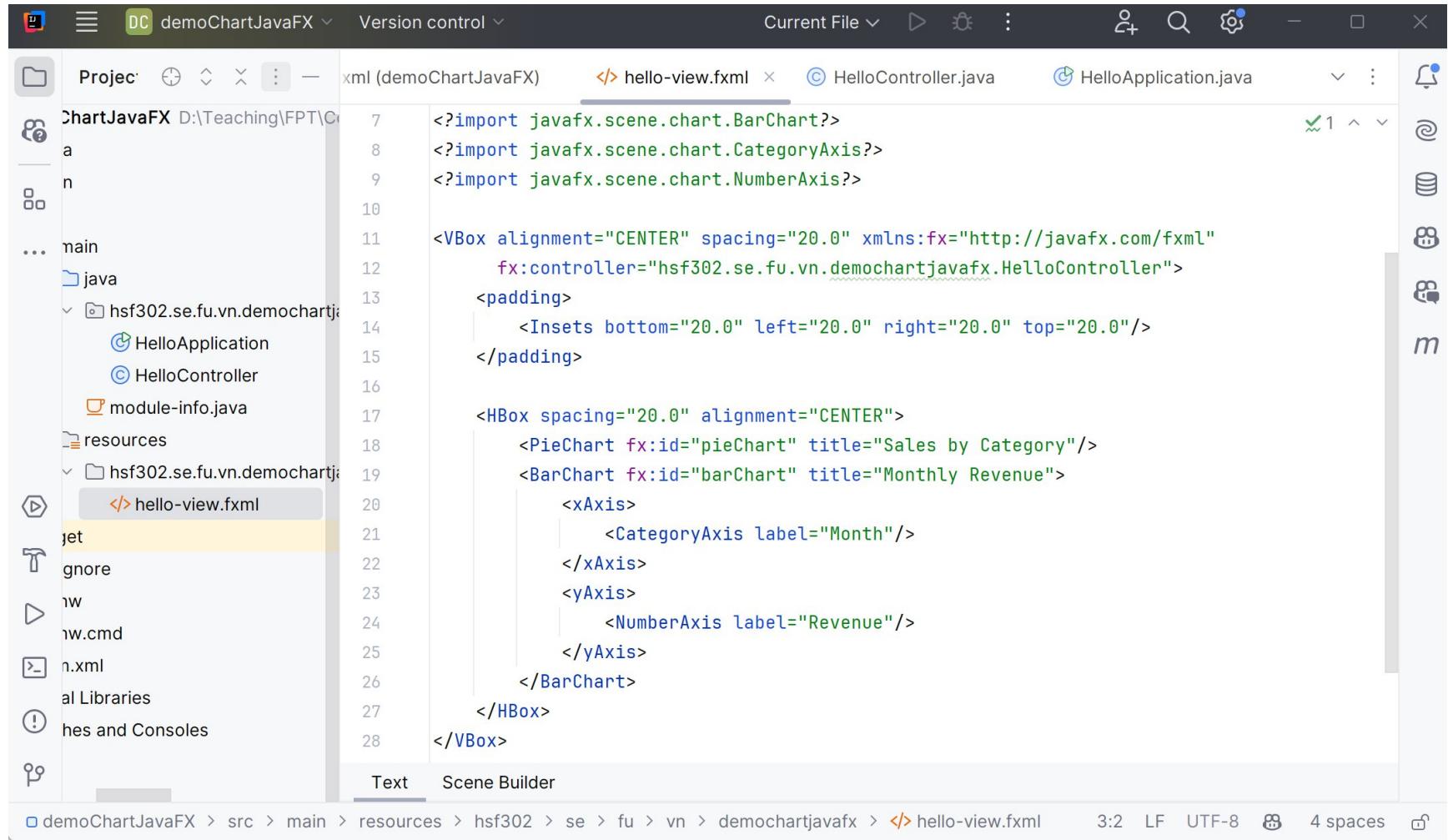
- ◆ AreaChart
- ◆ BarChart
- ◆ BubbleChart
- ◆ LineChart
- ◆ PieChart
- ◆ ScatterChart
- ◆ StackedAreaChart
- ◆ StackedBarChart



JavaFX Charts

- ◆ The JavaFX PieChart component is capable of drawing pie charts in your JavaFX application based on data you supply it.
- ◆ The PieChart component is really easy to use.
- ◆ The JavaFX PieChart component is represented by the class **javafx.scene.chart.PieChart**.

View of JavaFX Charts



The screenshot shows a JavaFX application interface with the following details:

- Project:** demoChartJavaFX
- File:** hello-view.fxml
- Code Content:**

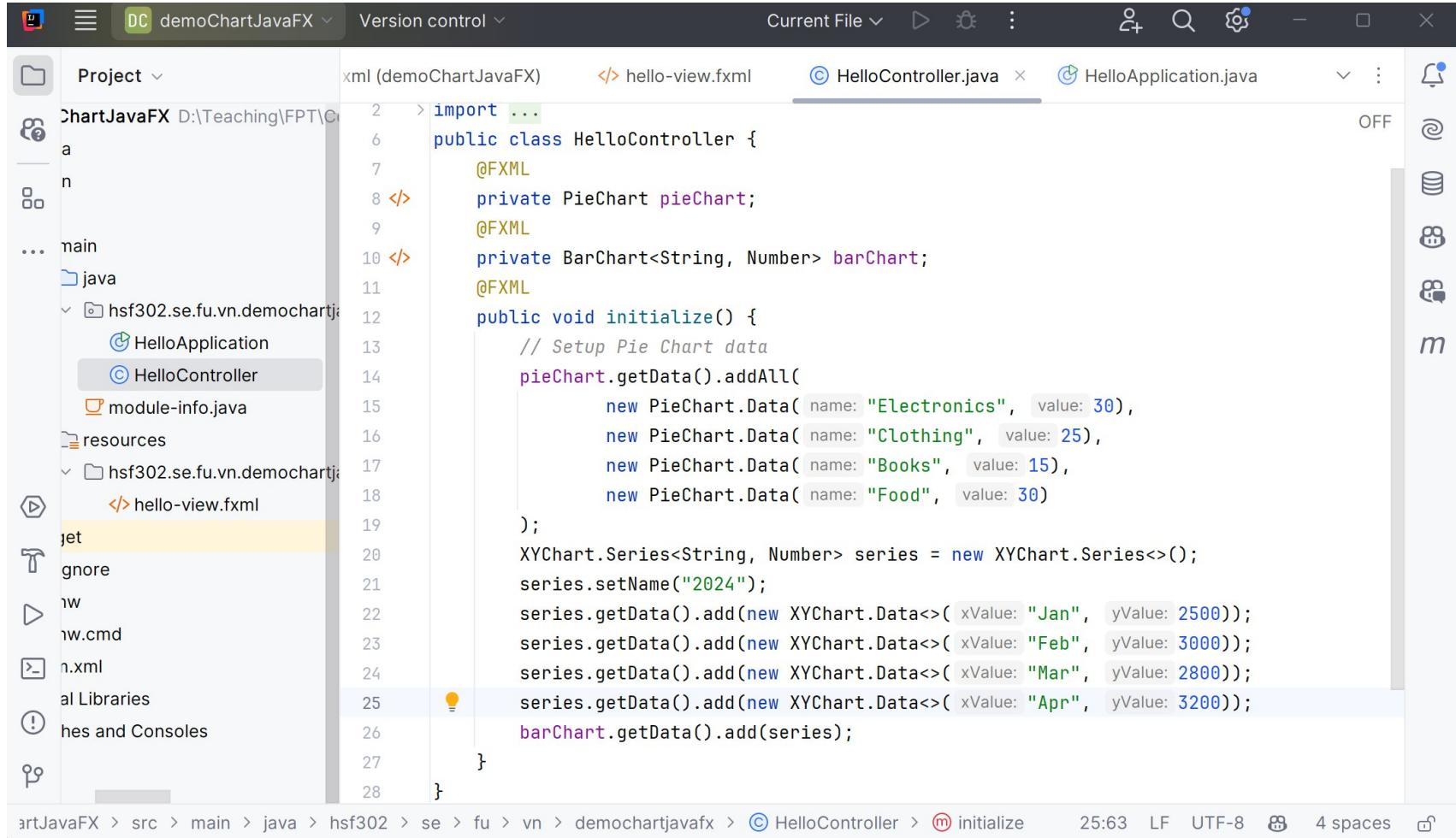
```
<?import javafx.scene.chart.BarChart?>
<?import javafx.scene.chart.CategoryAxis?>
<?import javafx.scene.chart.NumberAxis?>

<VBox alignment="CENTER" spacing="20.0" xmlns:fx="http://javafx.com/fxml"
      fx:controller="hsf302.se.fu.vn.demochartjavafx.HelloController">
  <padding>
    <Insets bottom="20.0" left="20.0" right="20.0" top="20.0"/>
  </padding>

  <HBox spacing="20.0" alignment="CENTER">
    <PieChart fx:id="pieChart" title="Sales by Category"/>
    <BarChart fx:id="barChart" title="Monthly Revenue">
      <xAxis>
        <CategoryAxis label="Month"/>
      </xAxis>
      <yAxis>
        <NumberAxis label="Revenue"/>
      </yAxis>
    </BarChart>
  </HBox>
</VBox>
```

- Toolbars and Menus:** Version control, Current File, etc.
- Sidebar:** Project tree showing files like HelloApplication.java, HelloController.java, and module-info.java.
- Status Bar:** demoChartJavaFX > src > main > resources > hsf302 > se > fu > vn > demochartjavafx > hello-view.fxml, 3:2 LF UTF-8 4 spaces

Controller of JavaFX Charts

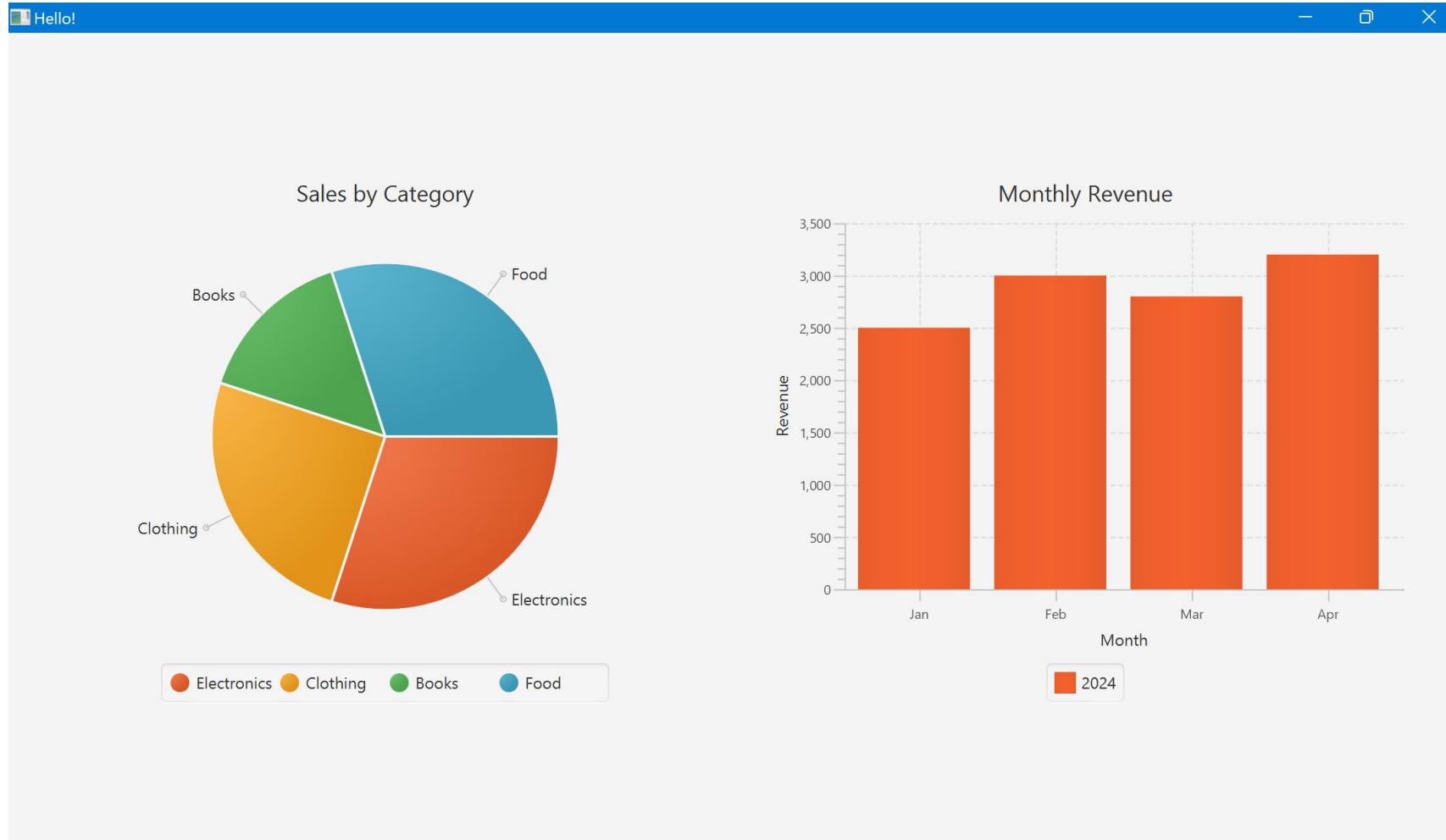


The screenshot shows a Java IDE interface with the following details:

- Title Bar:** demoChartJavaFX
- Project Explorer:** Shows a project structure with a main package containing `HelloApplication`, `HelloController`, and `module-info.java`. A file named `hello-view.fxml` is also listed.
- Code Editor:** The `HelloController.java` file is open, displaying Java code for initializing JavaFX charts.
- Code Content:**

```
import ...  
public class HelloController {  
    @FXML  
    private PieChart pieChart;  
    @FXML  
    private BarChart<String, Number> barChart;  
    @FXML  
    public void initialize() {  
        // Setup Pie Chart data  
        pieChart.getData().addAll(  
            new PieChart.Data("Electronics", 30),  
            new PieChart.Data("Clothing", 25),  
            new PieChart.Data("Books", 15),  
            new PieChart.Data("Food", 30)  
        );  
        XYChart.Series<String, Number> series = new XYChart.Series<>();  
        series.setName("2024");  
        series.getData().add(new XYChart.Data("Jan", 2500));  
        series.getData().add(new XYChart.Data("Feb", 3000));  
        series.getData().add(new XYChart.Data("Mar", 2800));  
        series.getData().add(new XYChart.Data("Apr", 3200));  
        barChart.getData().add(series);  
    }  
}
```
- Status Bar:** Shows the path `artJavaFX > src > main > java > hsf302 > se > fu > vn > demochartjavafx > HelloController > initialize`, file size 25:63, encoding LF, and 4 spaces.

JavaFX Chart

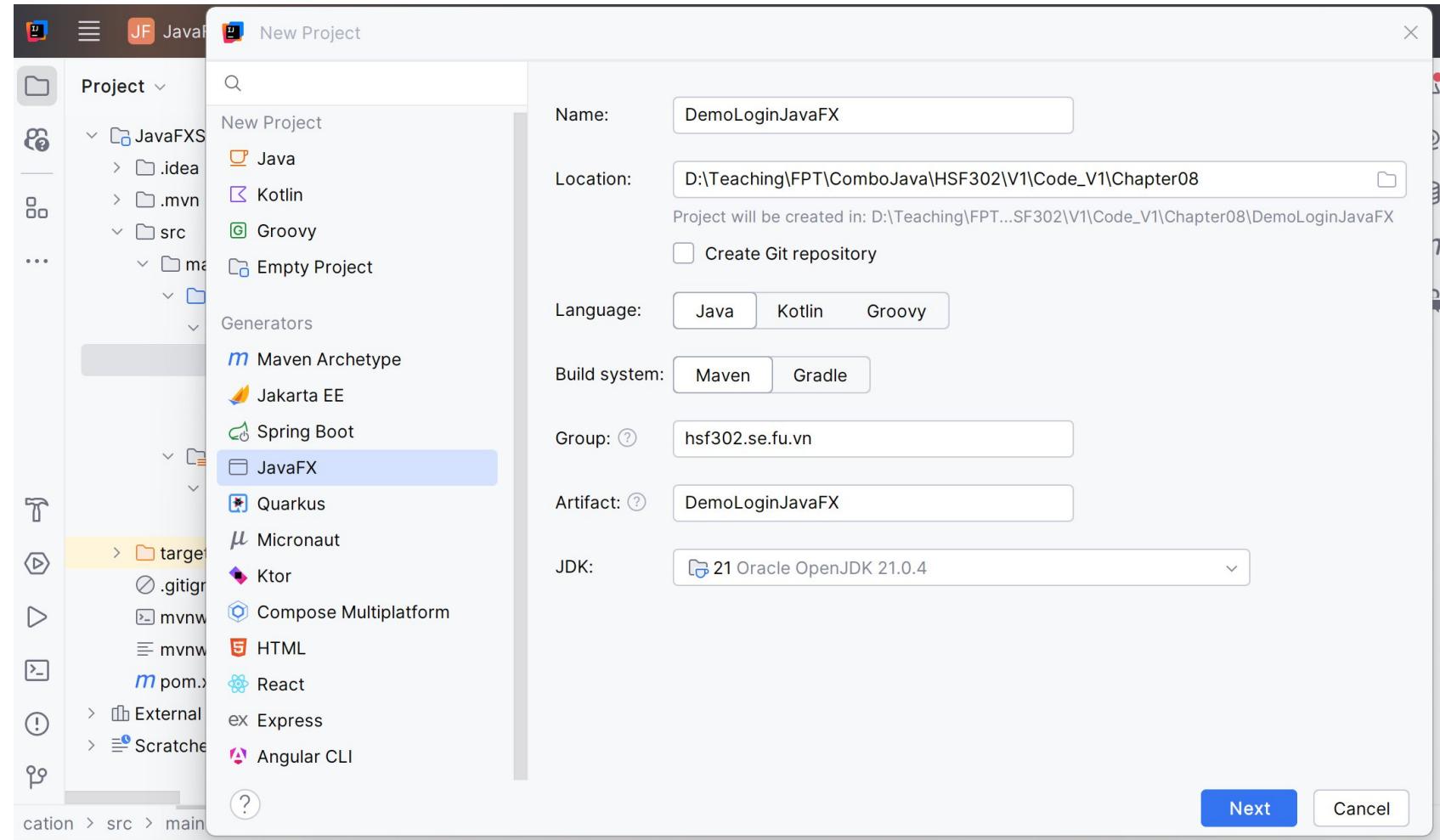


JavaFX Concurrency

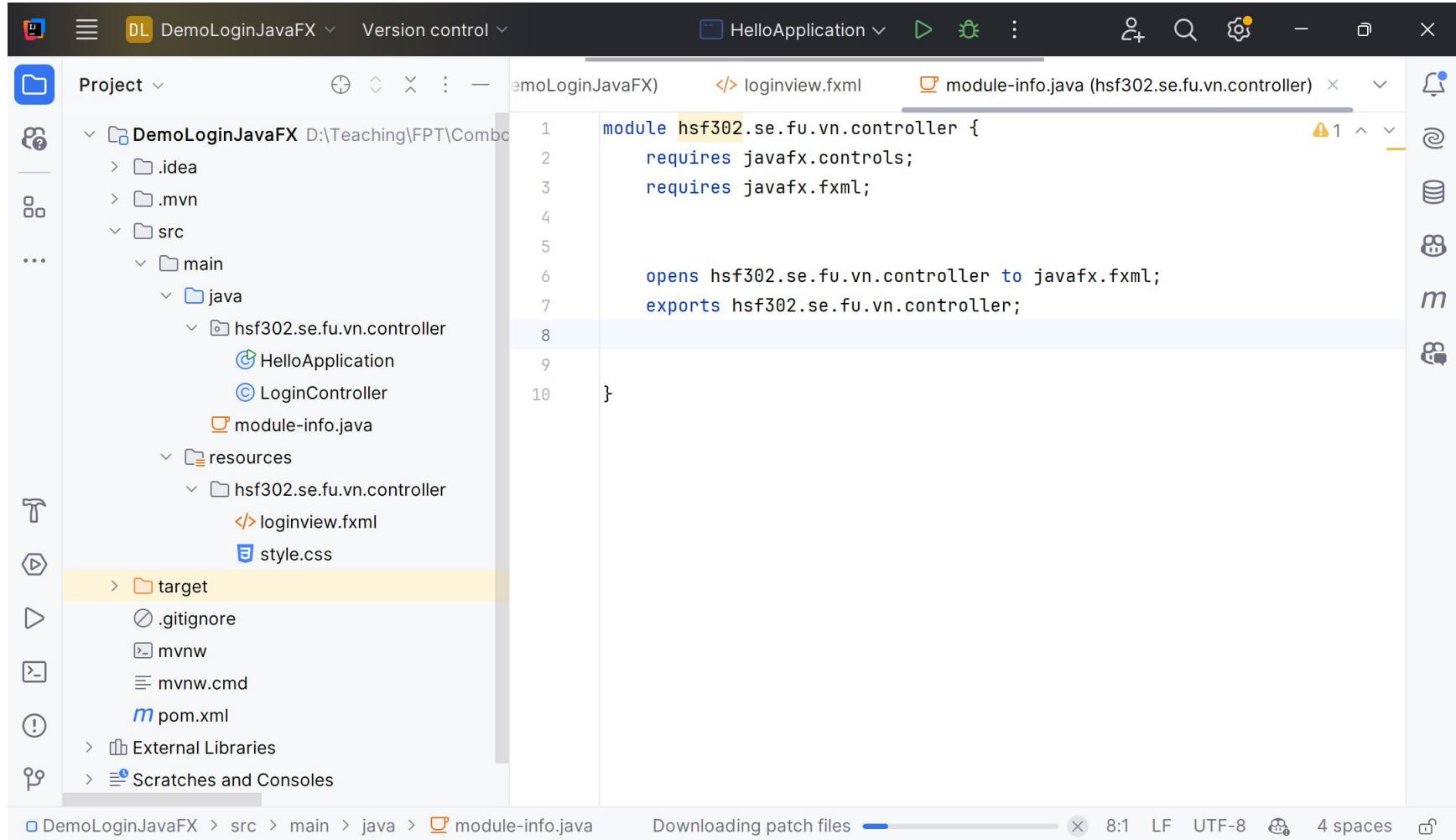
- ◆ *JavaFX concurrency* refers to how JavaFX is designed with respect to multithreading and concurrency.
- ◆ JavaFX uses a single-threaded rendering design, meaning only a single thread can render anything on the screen, and that is the JavaFX application thread. In fact, only the JavaFX application thread is allowed to make any changes to the JavaFX Scene Graph in general.
- ◆ A single-threaded rendering design is easier to implement correctly, but long-running tasks that run within the JavaFX application thread make the GUI unresponsive until the task is completed. No JavaFX GUI controls react to mouse clicks, mouse over, keyboard input while the JavaFX application thread is busy running that task.

JavaFX Demonstration

Create JavaFX Simple Application



Structure of JavaFX Application

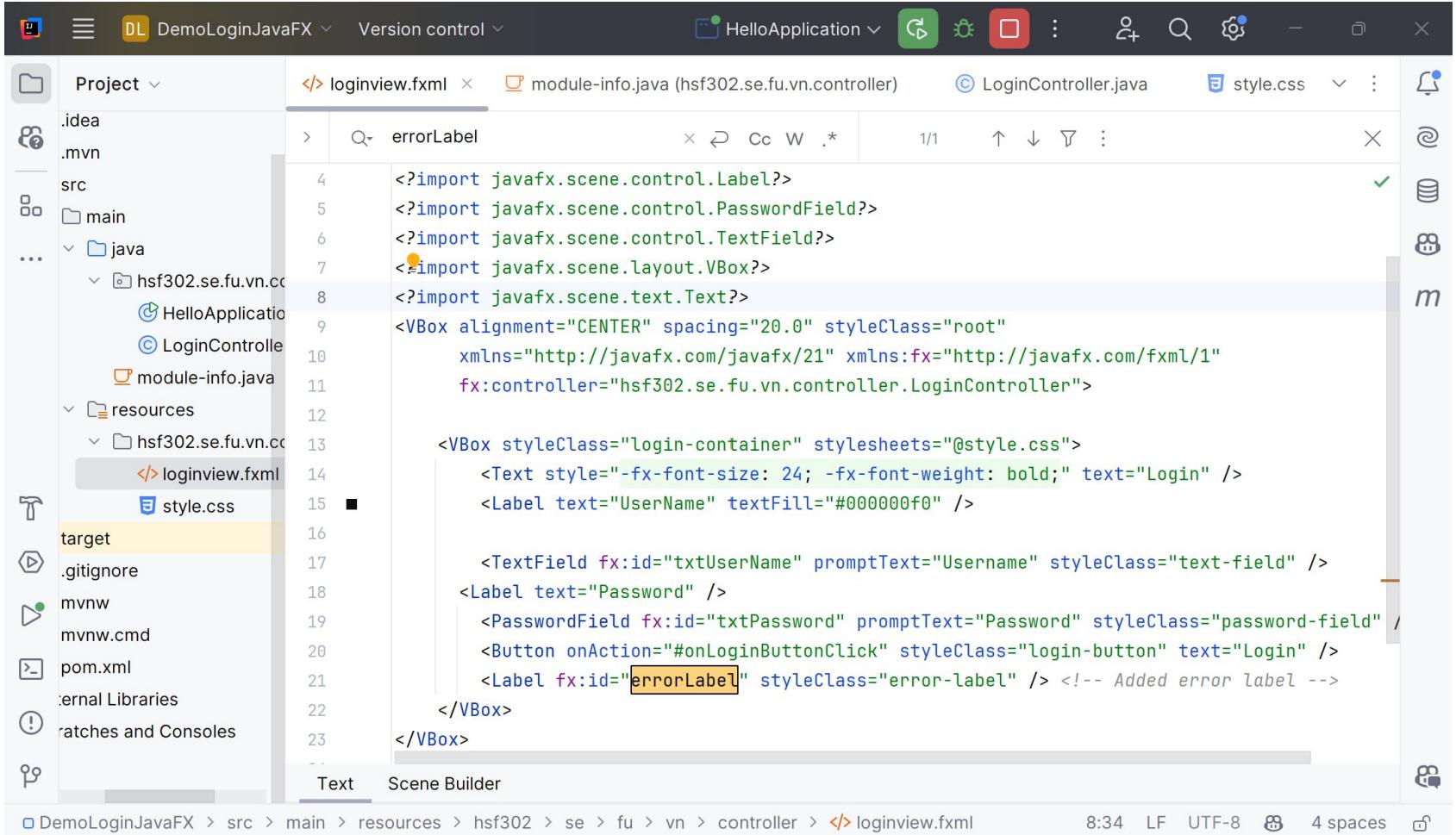


The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** DL DemoLoginJavaFX
- Toolbars:** Version control, HelloApplication, and several icons for navigation and search.
- Left Sidebar:** Shows the project structure under "Project".
 - DemoLoginJavaFX (selected)
 - .idea
 - .mvn
 - src
 - main
 - java
 - hsf302.se.fu.vn.controller
 - HelloApplication
 - LoginController
 - module-info.java
 - resources
 - hsf302.se.fu.vn.controller
 - loginview.fxml
 - style.css
 - target (highlighted with a yellow background)
 - .gitignore
 - mvnw
 - mvnw.cmd
 - pom.xml
 - External Libraries
 - Scratches and Consoles
- Central Editor:** The "module-info.java" file is open in the editor.

```
module hsf302.se.fu.vn.controller {  
    requires javafx.controls;  
    requires javafx.fxml;  
  
    opens hsf302.se.fu.vn.controller to javafx.fxml;  
    exports hsf302.se.fu.vn.controller;  
}
```
- Bottom Status Bar:** Shows the path "DemoLoginJavaFX > src > main > java > module-info.java", a progress bar for "Downloading patch files", and settings for 8:1 LF, UTF-8, 4 spaces, and a file icon.

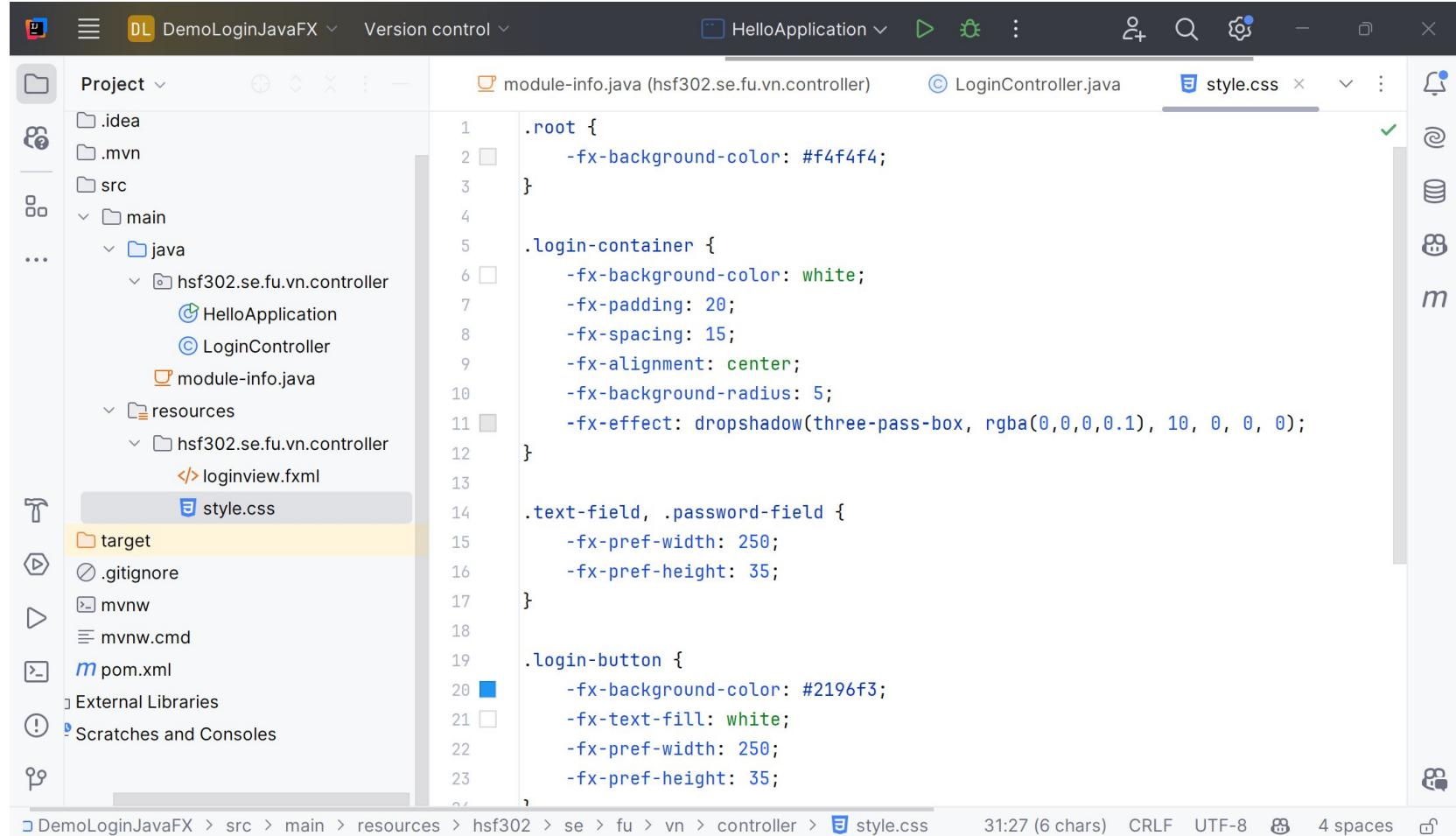
View of JavaFX



The screenshot shows the IntelliJ IDEA IDE interface with the following details:

- Project Structure:** The left sidebar shows the project structure for "DemoLoginJavaFX". It includes the ".idea" folder, ".mvn", "src" directory, and "main" directory which contains "java" and "resources" sub-directories. Inside "java", there are "HelloApplication.java", "LoginController.java", and "module-info.java". Inside "resources", there are "style.css" and "loginview.fxml".
- Code Editor:** The main editor window displays the FXML code for "loginview.fxml". The code defines a root `VBox` with alignment "CENTER" and spacing "20.0". It includes imports for `Label`, `PasswordField`, `TextField`, and `VBox`. The `styleClass` is set to "root".
The `loginview.fxml` also contains a `VBox` with style class "login-container" and a `styleSheet` of "@style.css". It contains a `Text` element with a large font size and bold weight, and a `Label` with the text "UserName".
Below that is a `TextField` with id "txtUserName", a `Label` with the text "Password", a `PasswordField` with id "txtPassword", a `Button` with id "onLoginButtonClick" and text "Login", and finally a `Label` with id "errorLabel" and style class "error-label". A comment indicates this is an "Added error label".
- Toolbars and Status Bar:** The top bar shows tabs for "HelloApplication", "Scene Builder", and "Text". The status bar at the bottom shows the file path "DemoLoginJavaFX > src > main > resources > hsf302 > se > fu > vn > controller > loginview.fxml", the time "8:34", and encoding "UTF-8".

Create the style.css



The screenshot shows the IntelliJ IDEA interface with a JavaFX project named "DemoLoginJavaFX". The project structure on the left includes ".idea", ".mvn", "src", and "main" folders. Inside "main", there are "java" and "resources" folders. The "java" folder contains "HelloApplication", "LoginController", and "module-info.java". The "resources" folder contains "loginview.fxml" and "style.css". The "style.css" file is currently selected in the editor. The code in "style.css" defines CSS rules for ".root", ".login-container", ".text-field", ".password-field", and ".login-button".

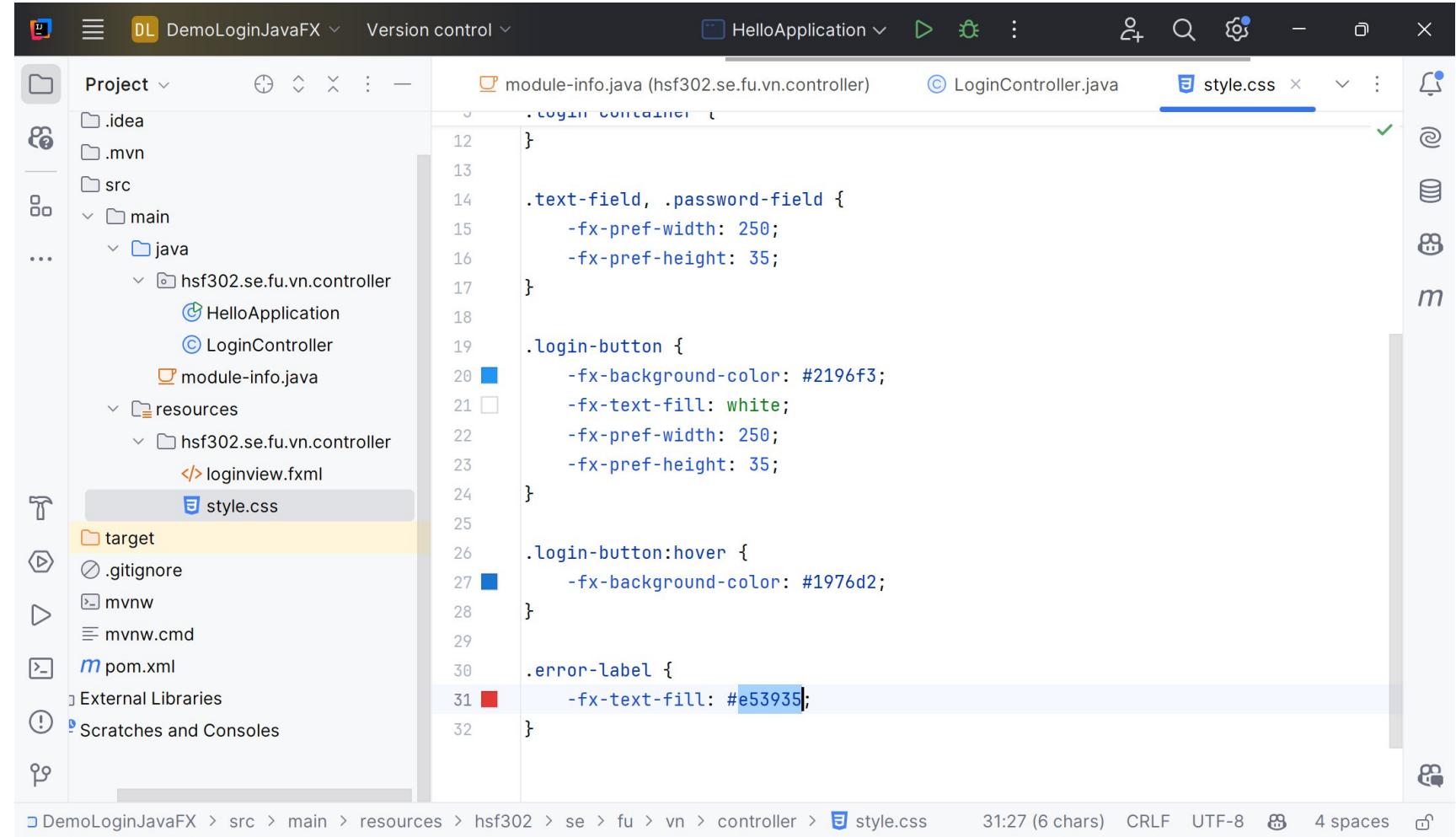
```
.root {
    -fx-background-color: #f4f4f4;
}

.login-container {
    -fx-background-color: white;
    -fx-padding: 20;
    -fx-spacing: 15;
    -fx-alignment: center;
    -fx-background-radius: 5;
    -fx-effect: dropshadow(three-pass-box, rgba(0,0,0,0.1), 10, 0, 0, 0);
}

.text-field, .password-field {
    -fx-pref-width: 250;
    -fx-pref-height: 35;
}

.login-button {
    -fx-background-color: #2196f3;
    -fx-text-fill: white;
    -fx-pref-width: 250;
    -fx-pref-height: 35;
}
```

Create the style.css

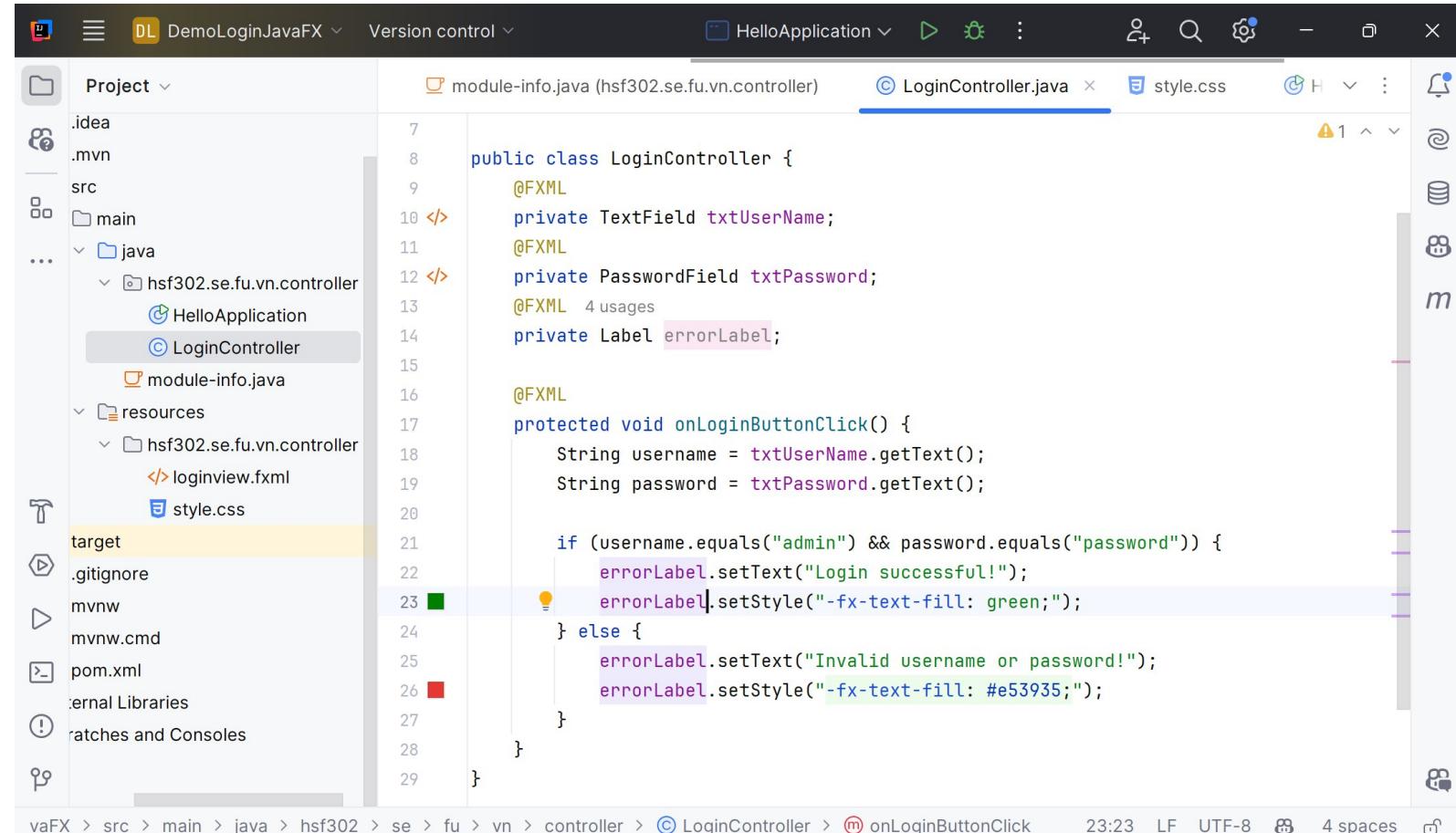


The screenshot shows the IntelliJ IDEA interface with a JavaFX project named "DemoLoginJavaFX". The project structure on the left includes ".idea", ".mvn", "src", and "main" folders, with "main/java" containing "HelloApplication", "LoginController", and "module-info.java", and "main/resources" containing "loginview.fxml" and "style.css". The "style.css" file is open in the editor, showing CSS code for styling login components:

```
.text-field, .password-field {  
    -fx-pref-width: 250;  
    -fx-pref-height: 35;  
}  
  
.login-button {  
    -fx-background-color: #2196f3;  
    -fx-text-fill: white;  
    -fx-pref-width: 250;  
    -fx-pref-height: 35;  
}  
  
.login-button:hover {  
    -fx-background-color: #1976d2;  
}  
  
.error-label {  
    -fx-text-fill: #e53935;  
}
```

The status bar at the bottom indicates the file path as "DemoLoginJavaFX > src > main > resources > hsf302 > se > fu > vn > controller > style.css", and the file details as "31:27 (6 chars) CRLF UTF-8 4 spaces".

Controller of JavaFX



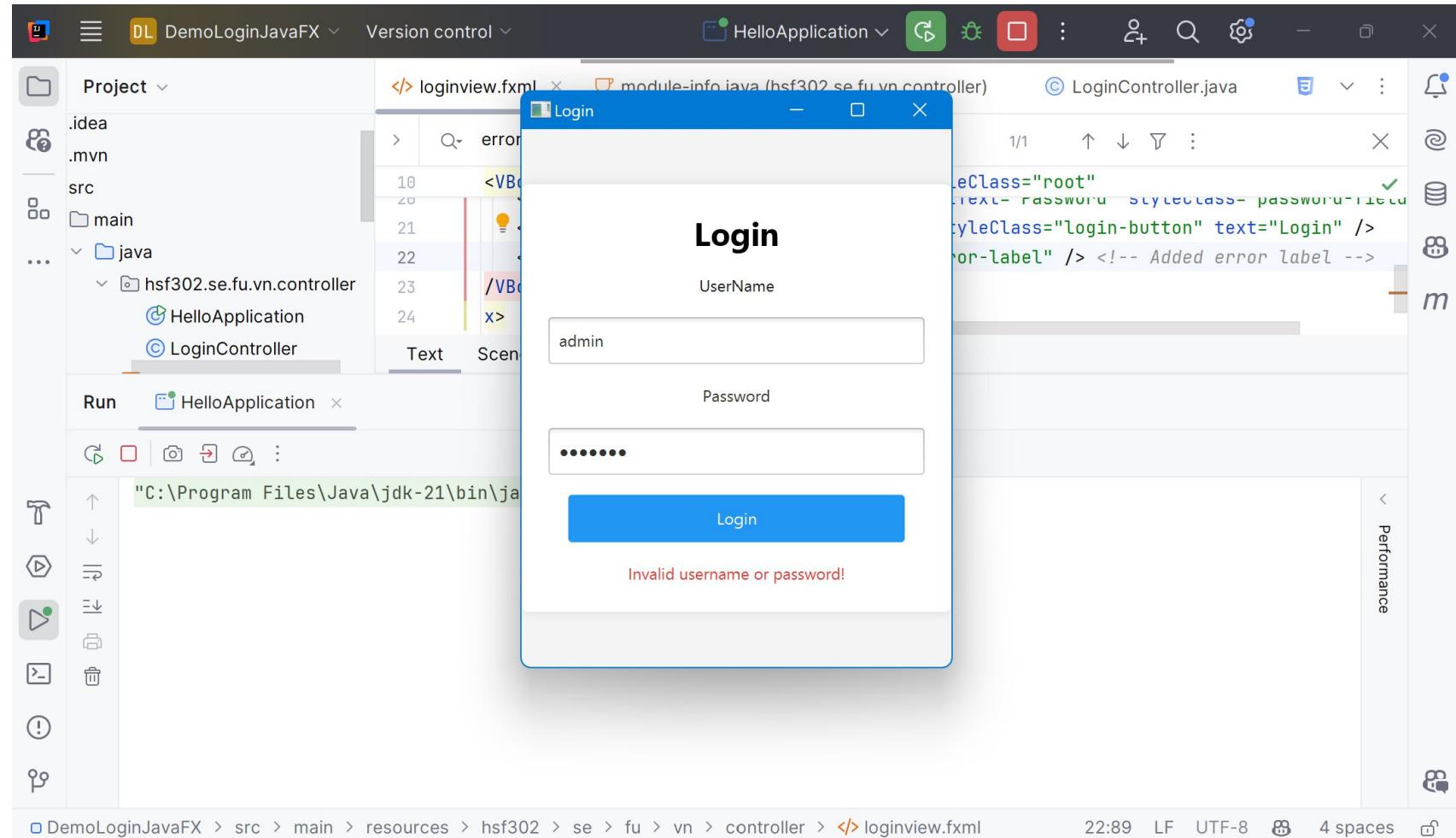
The screenshot shows the IntelliJ IDEA IDE interface. The left sidebar displays the project structure for a JavaFX application named "DemoLoginJavaFX". The "src" directory contains "main", which has "java" and "resources" sub-directories. Inside "java", there is a package "hsf302.se.fu.vn.controller" containing files "HelloApplication.java" and "LoginController.java", with "LoginController.java" currently selected. Inside "resources", there is also a package "hsf302.se.fu.vn.controller" containing "loginview.fxml" and "style.css". The right pane shows the code editor with the content of "LoginController.java". The code defines a controller class "LoginController" that handles button click events. It uses JavaFX annotations like @FXML and private fields for text fields and a label. The "onLoginButtonClick" method retrieves user input from the text fields and checks if the username is "admin" and the password is "password". If successful, it sets the error label's text to "Login successful!" and changes its style to green. Otherwise, it sets the text to "Invalid username or password!" and changes its style to red. The code editor shows syntax highlighting and some code completion suggestions.

```
public class LoginController {
    @FXML
    private TextField txtUserName;
    @FXML
    private PasswordField txtPassword;
    @FXML
    private Label errorLabel;

    protected void onLoginButtonClick() {
        String username = txtUserName.getText();
        String password = txtPassword.getText();

        if (username.equals("admin") && password.equals("password")) {
            errorLabel.setText("Login successful!");
            errorLabel.setStyle("-fx-text-fill: green;");
        } else {
            errorLabel.setText("Invalid username or password!");
            errorLabel.setStyle("-fx-text-fill: #e53935;");
        }
    }
}
```

Result



Summary

Concepts were introduced:

- ◆ JavaFX Architecture
- ◆ JavaFX Core: Stage, Scene, FXML
- ◆ Understand JavaFX Properties and Bindings, Events, Layouts, Controls
- ◆ Create Java desktop applications with JavaFX