Advanced Programming
Java FX
What Is JavaFX?

- A set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications.
- High-performance, modern user interface that features audio, video, graphics, and animation.
- Deployed across multiple platforms: desktop, browsers, mobile, etc.
- JavaFX 8 is part of JDK 8
- Coexists with Swing – however, it may replace Swing as the standard GUI library;
JavaFX Key Features

- **FXML** → MVC Pattern Support
- **WebView** (embed web pages within a JavaFX application)
- Built-in UI controls, **CSS** and **Themes** (Modena, Caspian, etc.)
- **3D Graphics** Features (*Shape3D*)
- Multi-touch Support, Hi-DPI support, Rich Text Support
- **Hardware-accelerated** graphics (uses optimally the GPU)
- High-performance media engine (playback of web multimedia content)
- Self-contained application deployment model
- IDEs offer tools for **rapid application development**
  → JavaFX Scene Builder
Quantum Toolkit - the interface which sits between the “top half” of the JavaFX platform (which includes all of the public, supported API) and the “bottom half”. The bottom half of the platform is essentially made up of the windowing code, media engine, web engine, and graphics engine. The Toolkit APIs abstract away the implementation details of these engines from the code sitting above it. Prism processes render jobs. It can run on both hardware and software renderers, including 3-D. Glass Windowing Toolkit - native operating services, such as managing the windows, timers, surfaces
Hello World

// The main class extends Application
public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {
        // The main entry point
        Button helloBtn = new Button();
        helloBtn.setText("Hello World!");

        FlowPane root = new FlowPane();
        root.getChildren().add(helloBtn);

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

        // The UI is defined by a stage and a scene.
        // Stage class is the top-level JavaFX container.
        // The Scene class is the container for all content.

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

    public static void main(String[] args) {
        launch(args); // not required for JavaFX applications...
    }
}
The Scene Graph

The JavaFX scene graph is a retained mode API

```java
Group group = new Group();
Rectangle blueSquare = new Rectangle(50, 50);
blueSquare.setFill(Color.BLUE);
group.getChildren().add(blueSquare);

Circle redCircle = new Circle(50, new Color(1,0,0,0.5f));
group.getChildren().add(redCircle);
```
Each item in the scene graph is called a Node. Each node in the scene graph can be given a unique id. Each node has a bounding rectangle and a style. Any Node can have transformations applied to it: translation, rotation, scaling, or shearing.
Layout Management

Setting the position and size for UI elements.

- A “combo” of a Swing JPanel + LayoutManager
- `javafx.scene.layout.Pane` - Base class for layout panes; used directly in cases where absolute positioning of children is required.
- Uses preferred, minimum and maximum properties
- `FlowPane, BorderPane, AnchorPane, StackPane, TilePane, GridPane, TextFlow, HBox, VBox, etc.`

```
borderPane.setCenter(new ListView());
borderPane.setBottom(new Label("Hello"));
```
public class HelloWorld extends Application {
    @Override
    public void start(Stage primaryStage) {
        Button helloBtn = new Button();
        helloBtn.setText("Hello World!");
        helloBtn.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                System.out.println("Hello Button was clicked!");
            }
        });

        // The anonymous inner class
        // can be turned into a lambda expression

        Button ciaoBtn = new Button("Ciao Mondo!");
        ciaoBtn.setOnAction((ActionEvent event) -> {
            System.out.println("Ciao Mondo e stato cliccato!");
        });
    }
}
JavaFX Events

An event represents an occurrence of something of interest to the application.

`javafx.event.Event` - Base class for FX events.

- **source** → *origin* of the event
- **target** → *the path* through which the event will travel when posted.
- **type** → *hierarchy*

![Event Hierarchy Diagram](image-url)
Event Delivery Process

- **Target Selection**
  - the node that has focus,
  - the node location of the cursor, etc.

- **Route Construction**
  - *the event dispatch chain* →

- **Event Capturing**
  - passed **down** to the target
  - **filters** are invoked

- **Event Bubbling**
  - the event returns **up** from the target to the root
  - **handlers** are invoked
Event Handling

Intercepting Filter Design Pattern

- **EventHandler** functional interface

- **Filters**
  ```java
  redCircle.addEventFilter(
      MouseEvent.MOUSE_CLICKED, (MouseEvent e) -> {
        System.out.println("Click: going down");
        //e.consume();
    });
  ```

- **Handlers** (going up...)
  ```java
  redCircle.addEventHandler(
      MouseEvent.MOUSE_CLICKED, (MouseEvent e) -> {
        System.out.println("Click: going up");
    });
  ```

- **Convenience methods**
  ```java
  setOnEvent-type(EventHandler<? super event-class> value)
  helloBtn.setOnAction(new EventHandler<ActionEvent>() {...});
  redCircle.setOnMouseEntered(new EventHandler<MouseEvent>() {...});
  ```
Transitions and Animations

`TranslateTransition` translate =
  new TranslateTransition(Duration.millis(750));
translate.setToX(300); translate.setToY(250);

`FillTransition` fill = new FillTransition(Duration.millis(750));
fill.setToValue(Color.RED);

`RotateTransition` rotate = new
  RotateTransition(Duration.millis(750));
rotate.setToAngle(360);

`ScaleTransition` scale =
  new ScaleTransition(Duration.millis(750));
scale.setToX(0.1); scale.setToY(0.1);

`ParallelTransition` transition =
  new ParallelTransition(blueSquare,
    translate, fill, rotate, scale);
transition.setCycleCount(Timeline.INDEFINITE);
transition.setAutoReverse(true);
transition.play();
Pulse

- A **pulse** is an event that indicates to the JavaFX scene graph that it is time to synchronize the state of the elements on the scene graph with Prism.

- A pulse is throttled at **60 frames per seconds (fps) maximum** and is fired whenever animations are running or when something in the scene graph is changed. For example, if a position of a button is changed, a pulse is scheduled.

- When a pulse is fired, the state of the elements on the scene graph is synchronized down to the rendering layer.

- A pulse enables application developers a way to handle events **asynchronously**. This important feature allows the system to batch and execute events on the pulse.

- The Glass Windowing Toolkit is responsible for executing the pulse events. It uses the high-resolution native timers to make the execution.
Styling with CSS

Cascading Style Sheets

• Define Style Sheets Files

```css
.root {
    -fx-background-image: url("background.jpg");
}
.label {
    -fx-font-size: 12px;
    -fx-font-weight: bold;
    -fx-text-fill: #333333;
}
```

• Specify the CSS

```java
scene.getStylesheets().add("path/stylesheet.css");
```

• Inline

```java
helloBtn.setStyle("-fx-background-color: slateblue; " + "-fx-text-fill: white;"泓
```

**FXML**

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

- **Java (Programatic)**
  
  ```java
  BorderPane border = new BorderPane();
  Label helloLabel = new Label("Hello");
  border.setTop(helloLabel);
  Label worldLabel = new Label("World");
  border.setCenter(worldLabel);
  
  ```

- **FXML (Declarative)**
  
  ```xml
  <BorderPane>
  <top>
    <Label text="Hello"/>
  </top>
  <center>
    <Label text="World"/>
  </center>
  </BorderPane>
  ```

**JavaFX Scene Builder**
Using FXML to Create UI

• FXML Loader

```java
Parent root = FXMLLoader.load(
    getClass().getResource("example.fxml"));
Scene scene = new Scene(root, 300, 275);
```

• Create the link between view and control

```xml
<GridPane fx:controller="FXMLExampleController">
    <Button text="Sign In"
        onAction="#handleSubmitButtonAction"/>
    <Text fx:id="actiontarget" />
</GridPane>
```

• Define the code to handle events

```java
public class FXMLExampleController {
    @FXML
    private Text actiontarget;

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

- **Swing**
  - Maturity, Stability
  - Component Libraries and Frameworks
  - Large amount of resources

- **JavaFX**
  - Modern, MVC friendly, CSS, FXML
  - Spectacular (3D, Animations, etc.)
  - May not be “rock-solid” in production, yet
  - Not so many resources
Resources

• **Java Client Technologies**
  
  [http://docs.oracle.com/javase/8/javase-clienttechnologies.htm](http://docs.oracle.com/javase/8/javase-clienttechnologies.htm)

• **JavaFX API**
  
  [https://docs.oracle.com/javase/8/javafx/api/toc.htm](https://docs.oracle.com/javase/8/javafx/api/toc.htm)