برنامه نویسی پیشرفته

رفع اشکال: جلسه ۱۰

JavaFX Application Structure
UI Controls and Events
FXML and CSS
Properties and Bindings
Multimedia and animations
Concurrency

JAVAFX APPLICATION

JavaFX is Oracle's official Java library for building rich-client desktop applications with modern UI controls, 2D/3D graphics, and multimedia support.

A JavaFX application typically consists of the following components:

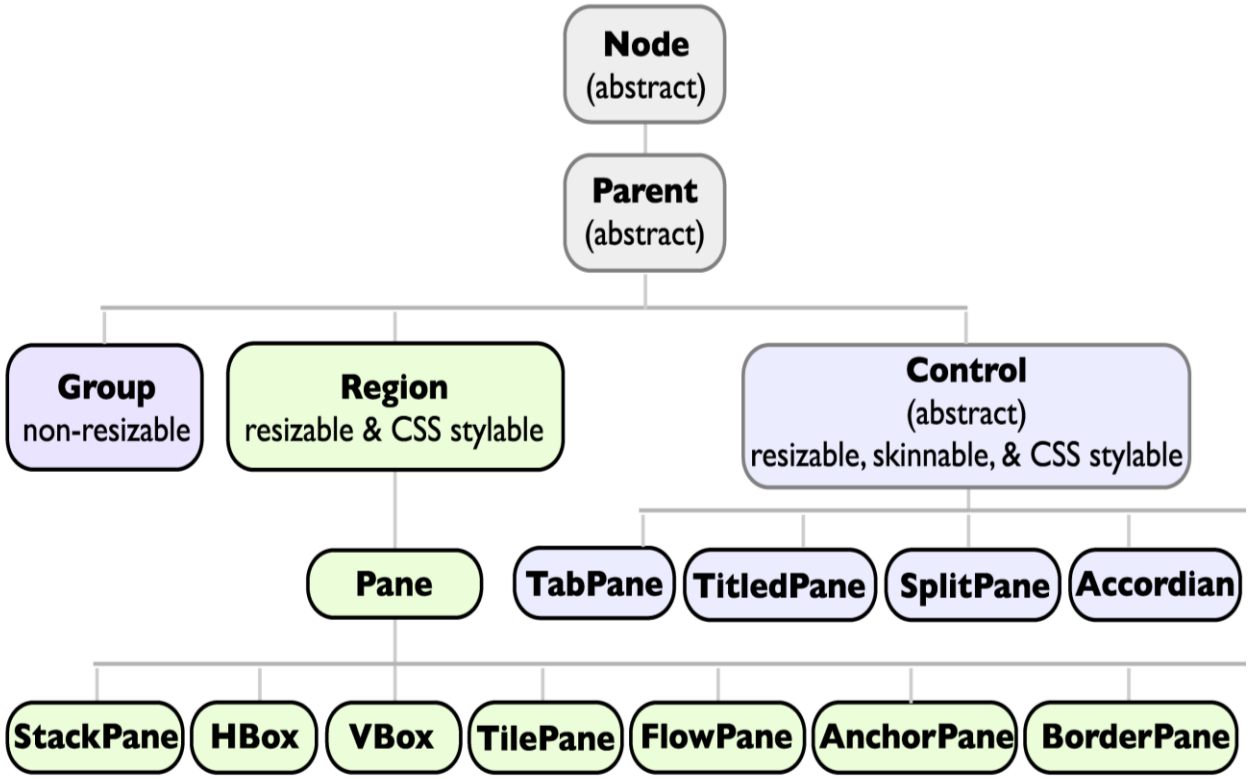
- Application Class: The entry point of any JavaFX application, extending the Application class.
- Stage: Represents the primary window of the application.
- Scene: Holds the UI elements (nodes) and is set on the stage.
- Nodes: The UI components like buttons, labels, etc.

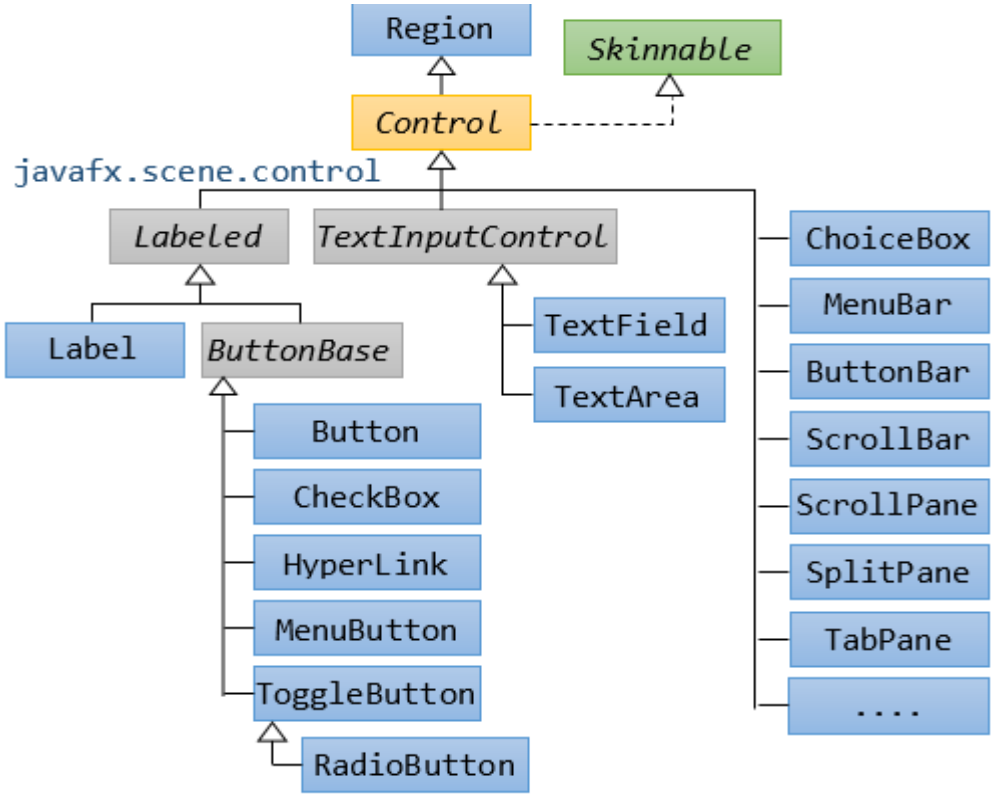
HELLO, JAVAFX!

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.stage.Stage;

public class HelloFX extends Application {
    @Override
    public void start(Stage primaryStage) {
        Label label = new Label("Hello, JavaFX!");
        Scene scene = new Scene(label, 300, 200);
        primaryStage.setScene(scene);
        primaryStage.setTitle("HelloFX");
        primaryStage.show();
    }

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

UI NODES

JavaFX provides a rich set of UI controls, including buttons, labels, text fields, checkboxes, and more. These controls can handle user interactions through event handling mechanisms.





UI CONTROLS AND LAYOUTS

Component	Description	Component	Description
Button	A clickable control used to perform an action.	**Label**	Displays non-editable text, often used to describe other controls.
TextField	Allows single-line text input from the user.	**CheckBox**	Represents a boolean option that can be toggled on or off.
ComboBox	Provides a dropdown list for selecting one item from multiple choices.	**ListView**	Displays a scrollable list of items, allowing single or multiple selections.
HBox	Arranges its children in a single horizontal row.	**VBox**	Arranges its children in a single vertical column.
GridPane	Lays out its children within a flexible grid of rows and columns.	**BorderPane**	Divides the layout into five regions: top, bottom, left, right, and center.

FXML AND SCENE BUILDER

FXML is an XML-based language that allows you to define the UI structure separately from the application logic in a controller class.

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

<?import javafx.scene.control.Button?>
<?import javafx.scene.layout.StackPane?>

<StackPane xmlns:fx="http://javafx.com/fxml"
stylesheets="@style.css">
    <Button text="Click Me!" fx:id="myButton"
styleClass="custom-button"/>
</StackPane>
```

```
public class MyController {
    @FXML
    private Button myButton;

    @FXML
    private void initialize() {
        myButton.setOnAction(e ->
System.out.println("Button Clicked from FXML!"));
    }}
// To load the fxml:
FXMLLoader loader = new
FXMLLoader(getClass().getResource("layout.fxml"));
Parent root = loader.load();
Scene scene = new Scene(root);
```

CSS

JavaFX allows you to style your UI components using CSS, similar to web development.

```
/* style.css for previous page example */
.custom-button {
    -fx-font-size: 16px;
    -fx-background-color: linear-gradient(to bottom, #00c6ff, #0072ff);
    -fx-text-fill: white;
    -fx-padding: 10 20 10 20;
    -fx-background-radius: 8;
}
```

MEDIA

JavaFX supports playing audio and video files using the Media and MediaPlayer classes.

```
String path = "path_to_video.mp4";
Media media = new Media(new File(path).toURI().toString());
MediaPlayer mediaPlayer = new MediaPlayer(media);
MediaView mediaView = new MediaView(mediaPlayer);

StackPane root = new StackPane(mediaView);
Scene scene = new Scene(root, 640, 480);
primaryStage.setScene(scene);
primaryStage.setTitle("Media Example");
primaryStage.show();

mediaPlayer.play();
```

ANIMATIONS

JavaFX provides several classes for creating animations, such as FadeTransition, TranslateTransition, RotateTransition, and more.

```
Label label = new Label("Fading Text");
FadeTransition fade = new FadeTransition(Duration.seconds(2), label);
fade.setFromValue(1.0);
fade.setToValue(0.0);
fade.setCycleCount(FadeTransition.INDEFINITE);
fade.setAutoReverse(true);
fade.play();

StackPane root = new StackPane(label);
Scene scene = new Scene(root, 300, 200);
primaryStage.setScene(scene);
primaryStage.setTitle("Animation Example");
primaryStage.show();
```

JAVAFX THREADS

JavaFX uses a multi-threaded architecture to keep UIs responsive by separating rendering, event handling, and background work across distinct threads. The **JavaFX Application Thread** handles all UI updates and event dispatching.

Thread	Role	UI Access?	Primary Use
Application	Live scene-graph & events	Yes	Scene graph updates
Launcher	Startup & init()	No	Bootstrap FX runtime
Prism Render	Frame rendering	No	Graphics pipeline
Media	Audio/video decode	No	Media playback
Pulse	60 Hz heartbeat	No	Layout/CSS/animation tick
Background	Worker threads	No	I/O, tasks, computation

CONCURRENCY

In JavaFX, long-running tasks should not be executed on the JavaFX Application Thread, as this can make the UI unresponsive. Instead use:

- Task<V> – For Background Work
  Task is a JavaFX class for running long or blocking operations off the UI thread.

- Platform.runLater(Runnable) – For UI Updates from Any Thread
  Note that JavaFX UI can only be safely updated from its main thread.

```
Platform.runLater(() -> button.setText("Updated!"));
```

TIME TO CODE

Do a full JavaFX Snake game example, complete with:

- Initialization of game state
- Animation loop
- directional movement
- Collision detection
- game-over logic
- Score display
- automatic restart

پایان