**Comsats University Islamabad (Lahore Campus)**

****

**Assigment 2**

**Subject:** Object Oriented Programming

**Submitted by:** Aisha (SP24-BSE-142)

**Submitted to:** Sir Shahid

Libraries used in my code:

**javafx.application.Application**:

* Provides the base class for JavaFX applications. It's needed to create and launch the main application window.

**javafx.scene.paint.Color**:

* Defines color constants and methods for working with colors. Used to set colors for UI elements like backgrounds, text, etc.

**javafx.stage.Stage**:

* Represents the main window or "stage" in a JavaFX application. It’s the top-level container that holds all your UI components.

**javafx.scene.Scene**:

* Represents the content of the window. It contains the layout and all UI controls within the window.

**javafx.scene.control.\*** (e.g., Button, Label, TextField, ComboBox, etc.):

* These are used to create UI controls like buttons, labels, text fields, and combo boxes for user interaction.

**javafx.scene.layout.GridPane**:

* A layout manager that arranges components in a grid of rows and columns. Useful for structuring the UI in a tabular form.

**javafx.stage.FileChooser**:

* Provides a dialog for file selection (open/save). Useful for allowing users to select files from their system.

**javafx.scene.image.ImageView and javafx.scene.image.Image**:

* ImageView is used to display images in the UI, and Image represents the image data itself.

**javafx.geometry.Insets**:

* Defines padding or margins around UI components to control their spacing.

**javafx.scene.layout.VBox, javafx.scene.layout.HBox, javafx.scene.layout.BorderPane**:

* VBox and HBox are layout managers that arrange components vertically or horizontally, respectively.
* BorderPane organizes components into five regions (top, bottom, left, right, center) for flexible layout design.

**javafx.scene.text.Font, javafx.scene.text.FontWeight**:

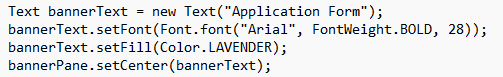
* Used to set fonts and font styles (like bold) for text in the UI.

**javafx.collections.FXCollections**:

* Provides utility methods for creating observable lists (lists that notify listeners when their content changes), useful for dynamic UIs.

**javafx.collections.ObservableList**:

* Represents a list that can be observed for changes, often used for managing dynamic data (e.g., populating a ComboBox or ListView).



**Text bannerText = new Text("Application Form");**

* This creates a text object with the words **"Application Form"**. It will be displayed in the application.

**bannerText.setFont(Font.font("Arial", FontWeight.BOLD, 28));**

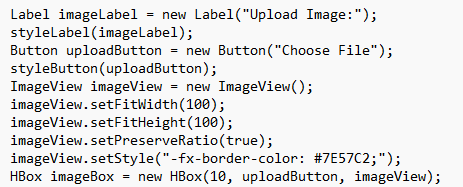
* This sets the font of the text to **Arial**, makes it **bold**, and sets the size to **28**. So, the text will appear larger and bold.

**bannerText.setFill(Color.LAVENDER);**

* This changes the color of the text to **lavender** (a light purple color).

**bannerPane.setCenter(bannerText);**

* This adds the text to the **center** of a layout pane called bannerPane. So, the "Application Form" text will be placed in the center of that area.



**Label imageLabel = new Label("Upload Image:");**

* This creates a **Label** with the text **"Upload Image:"**. The label is typically used to provide instructions or information to the user.

**styleLabel(imageLabel);**

* This calls a method (not shown here) to **style** the label. It likely changes the appearance of the label (e.g., font, color, padding, etc.).

**Button uploadButton = new Button("Choose File");**

* This creates a **Button** with the text **"Choose File"**. The button is used to allow the user to select an image file from their computer.

**styleButton(uploadButton);**

* Similar to the label, this calls a method (not shown here) to **style** the button. This could adjust the button’s appearance (e.g., background color, text color, size, etc.).

**ImageView imageView = new ImageView();**

* This creates an **ImageView** object, which is used to display an image. However, no image has been set yet.

**imageView.setFitWidth(100);**

* This sets the **width** of the image to **100 pixels**. The image will be resized to fit this width.

**imageView.setFitHeight(100);**

* This sets the **height** of the image to **100 pixels**. So, the image will be resized to have a 100px height.

**imageView.setPreserveRatio(true);**

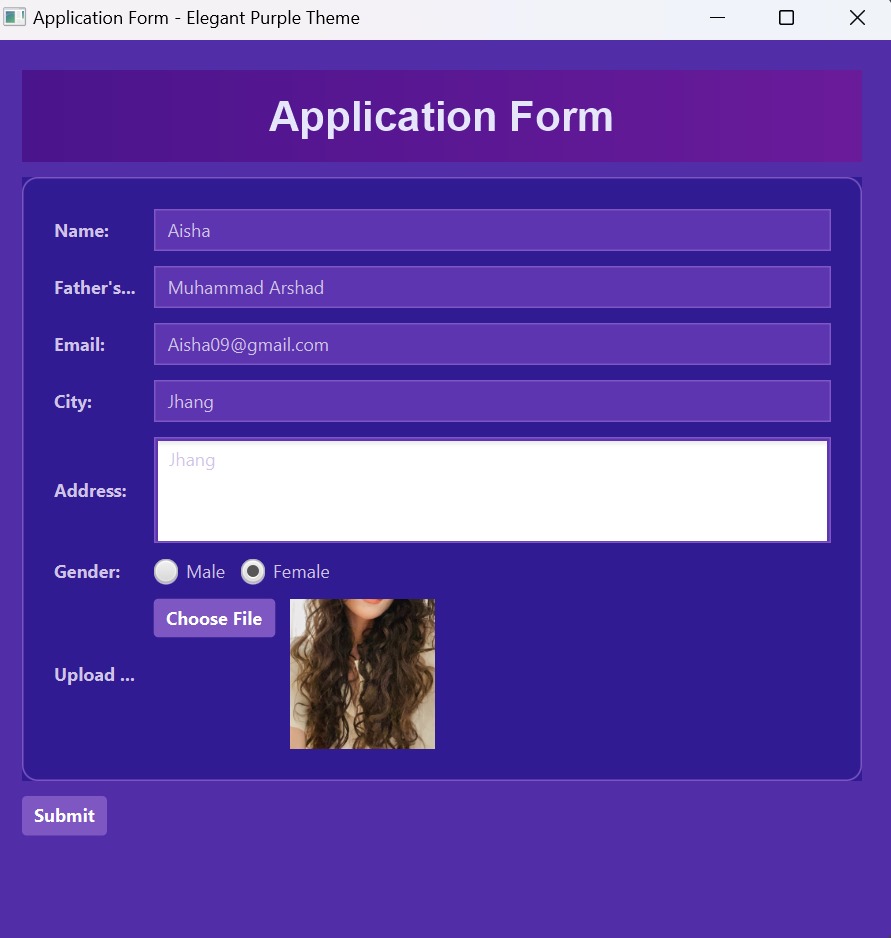
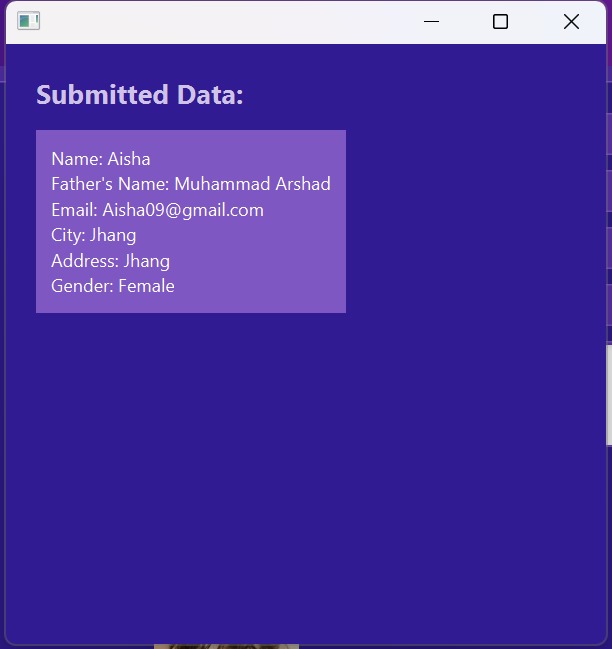
* This ensures that the **aspect ratio** (the proportion between width and height) of the image is preserved when resizing it. The image won’t be stretched or distorted.

**imageView.setStyle("-fx-border-color: #7E57C2;");**

* This adds a **border** around the ImageView with the color **#7E57C2** (a shade of purple). The border helps to visually separate the image from the rest of the UI.

**HBox imageBox = new HBox(10, uploadButton, imageView);**

* This creates an **HBox** (a horizontal layout container) that arranges the **uploadButton** and **imageView** side by side. The **10** is the space (in pixels) between the button and the image. So, the button and image are placed horizontally with a 10px gap between them.
* Source code: 

Output of my code: