JavaFX

1. Understanding User Interfaces and HCI (Human-Computer Interaction)

- **User Interfaces:** User interfaces allow users to interact with computers or devices. Types include:
  - **Graphical User Interface (GUI)**: Uses visual elements like windows, icons, and buttons.
  - **Command Line Interface (CLI)**: Text-based, accepting commands from a prompt.
  - **Menu-Driven Interface**: Presents a list of choices, often seen in ATMs.
  - **Touch User Interface**: Found in smartphones or tablets.
  - **Voice User Interface (VUI)**: Voice-based interaction, like Alexa or Siri.
  - **Form-Based and Natural Language Interfaces**: Used in surveys or AI chatbots, allowing for flexible user inputs.
- **HCI (Human-Computer Interaction):** HCI studies how humans interact with computers to design interfaces that are more intuitive and efficient. This field is crucial for developing user-friendly systems, which can be important in high-stakes situations, such as medical devices, cars, or emergency response systems.

2. Java GUI Libraries: AWT, Swing, and JavaFX

- **AWT (Abstract Window Toolkit)**
  - **Java's first GUI library**: AWT was platform-dependent, relying on the operating system's UI resources, which limited its flexibility and led to inconsistent behavior across platforms.
- **Swing:**
  - **Extension of AWT**: Introduced a more consistent set of components and behaviors across platforms, designed mainly for desktop applications.

Swing added flexibility and more complex UI components like tables and trees.

- **JavaFX:**
  - **Designed for Modern Interfaces**: JavaFX is optimized for desktop, web, and mobile, aiming for responsive design influenced by web technologies. It offers a robust set of tools for creating rich, interactive UIs, making it popular for modern Java applications.
  - JavaFX is built around the **Model-View-Controller (MVC)** design pattern and uses a hierarchical scene graph to represent UI components. It organizes GUIs into three main elements:
    1. **Application**: The entry point for any JavaFX program.
    2. **Stage**: Represents the window of the application.
    3. **Scene**: Represents the content of the window.

3. Setting up JavaFX

- JavaFX is a separate library and needs to be added as a dependency. It can be set up in development environments like Maven with the following dependency in the `pom.xml` file:

```
<dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>14</version>
</dependency>
```

- **Running JavaFX Programs:** In some IDEs, the typical “play” button might not run JavaFX programs due to integration with Maven. Instead:
  1. Double-click on `javafx:run` in Maven's left panel.
  2. For errors, check the "Run" panel in the IDE for detailed exception information.

4. JavaFX Application Structure: Stages, Scenes, and Nodes

A JavaFX application is structured as follows:

- **Stage**: Represents the main window.
- **Scene**: Contains all the UI elements.
- **Nodes**: Individual UI components, like buttons or text fields.

For example:

```
public class App extends Application {
    @Override
    public void start(Stage stage) {
        // Stage configuration
        stage.setTitle("Hello World!");

        // Pane and Button setup
        StackPane pane = new StackPane();
        Button btn = new Button("Say 'Hello World'");

        // Button action
        btn.setOnAction((ActionEvent event) -> {
            System.out.println("Hello World!");
        });

        // Add button to the pane and scene to the stage
        pane.getChildren().add(btn);
        Scene scene = new Scene(pane, 300, 250);
        stage.setScene(scene);

        // Display the stage
        stage.show();
    }

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

```
    }
}
```

Key Concepts:

1. **Extending Application**: Classes that run JavaFX apps extend `Application`.
2. **Overriding start(Stage stage)**: Where the main GUI setup occurs.
3. **Setting Up Stages and Scenes**: `Scene` contains `Pane`, which organizes UI elements (Nodes). `Stage` is used to display the Scene.

5. UI Components in JavaFX

Common JavaFX components include:

- **Text Labels**: Display static text.
- **Buttons**: Used for actions when clicked.
- **Images and Icons**: Display graphics.
- **Input Fields**: Collect user data, like `TextField` or `PasswordField`.
- **Dialogs and Modals**: Show additional information or choices without leaving the main window.

JavaFX supports creating responsive, interactive components by using event handlers. Here's an example of a button click handler with a lambda function:

```
btn.setOnAction((ActionEvent event) -> {
    System.out.println("Hello World!");
});
```

6. Essential JavaFX Classes

- **Application Class:** The `Application` class is the starting point of any JavaFX program. It:
  - Is extended to create your JavaFX application.
  - Has an overridden `start(Stage stage)` method to define the GUI setup.
  - Uses the `launch()` method to start the application.

- Example:

```
import javafx.application.Application;
import javafx.stage.Stage;

public class App extends Application {
    @Override
    public void start(Stage stage) {
        // GUI setup goes here
        stage.setTitle("My JavaFX Application");
        stage.show();
    }

    public static void main(String[] args) {
        launch(); // Starts the JavaFX application
    }
}
```

- **Stage Class:** The `Stage` object:
  - Represents the main window of the application.
  - Can display one `Scene` at a time.
  - Has methods to configure window properties like size, title, and full-screen mode.
  - **Key Methods:**
    - `setTitle(String title)`: Sets the window's title.
    - `setScene(Scene scene)`: Attaches a `Scene` to the stage.
    - `setResizable(boolean value)`: Allows/disallows resizing of the window.
  - Example:

```
@Override
public void start(Stage stage) {
    stage.setTitle("Hello, JavaFX!");
```

```
    Scene scene = new Scene(new Pane(), 300, 250); // Crea
ting a scene with root node
    stage.setScene(scene);
    stage.show();
}
```

- **Scene Class:** The `Scene` object:
  - Represents all content inside a `Stage`.
  - Acts as a container for UI elements arranged in a hierarchy (scene graph).
  - Can be swapped to change the application's appearance.
  - **Key Methods:**
    - `setRoot(Parent root)`: Sets the root of the scene graph.
    - Scene dimensions can be specified during initialization.
  - Example:

```
@Override
public void start(Stage stage) {
    Pane root = new Pane();
    Scene scene = new Scene(root, 400, 300); // Root node,
width, height
    stage.setScene(scene);
    stage.show();
}
```

7. Scene Graph

The **scene graph** is a tree-like structure of UI components where:

- `Node` is the base class for all components.
- `Parent` is an internal node that can have children.
- `Region` can be styled with CSS.
- `Pane` is a container for child nodes with layout management capabilities.

- **Hierarchy:**

```
Node -> Parent -> Region -> Pane
```

- **Example:**

```
Stage
 └─ Scene
     └─ Pane (Root)
         ├─ Button
         ├─ Label
         └─ TextField
```

8. JavaFX Layout Panes

- **Layout Panes** organize the positioning of child components within the UI. Common types include:

1. **BorderPane**: Divides the layout into five regions (top, bottom, left, right, center).

```
BorderPane borderPane = new BorderPane();
borderPane.setTop(new Label("Top Label"));
borderPane.setCenter(new Button("Center Button"));
```

2. **HBox**: Aligns children horizontally.

```
HBox hbox = new HBox();
hbox.getChildren().addAll(new Label("Label 1"), new Lab
el("Label 2"));
```

3. **VBox**: Aligns children vertically.

```
VBox vbox = new VBox();
vbox.getChildren().addAll(new Button("Button 1"), new B
```

```
utton("Button 2"));
```

4. **StackPane**: Stacks children on top of each other.

```
StackPane stackPane = new StackPane();
stackPane.getChildren().addAll(new Label("On Top"), new
Rectangle(100, 100));
```

5. **GridPane**: Arranges children in a grid (rows and columns).

```
GridPane gridPane = new GridPane();
gridPane.add(new Button("Button 1"), 0, 0); // (col, ro
w)
gridPane.add(new Button("Button 2"), 1, 0);
```

6. **FlowPane**: Aligns children in a row or column, wrapping when space runs out.

```
FlowPane flowPane = new FlowPane();
flowPane.getChildren().addAll(new Button("Button 1"), n
ew Button("Button 2"));
```

7. **TilePane**: Similar to `FlowPane`, but all children have equal dimensions.
8. **AnchorPane**: Anchors children to specific edges of the pane.

9. Styling with CSS

- JavaFX supports CSS for styling components, allowing separation of style and logic.
- Example of CSS (`main.css`):

```
.instructions {
    -fx-text-fill: #766f65;
    -fx-padding: 10 0 10 0; /* top, right, bottom, left */
}
```

- Linking CSS to JavaFX code:

```
Label label = new Label("Styled Label");
label.getStyleClass().add("instructions");

Scene scene = new Scene(new VBox(label), 300, 200);
scene.getStylesheets().add("style/main.css"); // Link styl
esheet
stage.setScene(scene);
```