CSCI 2120:
Software Design & Development II

UNIT4: UI management

GUI framework

JavaFX Core: Stage

Overview

1. Introduction
2. Creating a Stage
3. Showing a Stage
4. Set a Scene on a Stage
5. Stage Title
6. Stage Position
7. Stage Width and Height
8. Stage Modality
9. Stage Owner
10. Stage Style
11. Stage Full Screen Mode
12. Stage Keyboard Events

Introduction

A JavaFX *Stage*, `javafx.stage.Stage`, represents a window in a JavaFX desktop application. Inside a JavaFX `Stage` you can insert a JavaFX `Scene` which represents the content displayed inside a window - inside a `Stage`.

When a JavaFX application starts up, it creates a root `Stage` object which is passed to the `start(Stage primaryStage)` method of the root class of your JavaFX application. This `Stage` object represents the primary window of your JavaFX application. You can create new `Stage` objects later in your application's life time, in case your application needs to open more windows.

Creating a Stage

You create a JavaFX `Stage` object just like any other Java object: Using the `new` command and the `Stage` constructor.

Here is an example of creating a JavaFX `Stage` object.

```
Stage stage = new Stage();
```

Showing a Stage

Simple creating a JavaFX `Stage` object will not show it. In order to make the `Stage` visible you must call either its `show()` or `showAndWait()` method. Here is an example of showing a JavaFX `Stage`:

```
Stage stage = new Stage();
stage.show()
```

show() vs. showAndWait()

The difference between the JavaFX `Stage` methods `show()` and `showAndWait()` is, that `show()` makes the `Stage` visible and then exits the `show()` method immediately, whereas the `showAndWait()` shows the `Stage` object and then blocks (stays inside the `showAndWait()` method) until the `Stage` is closed.

Set a Scene on a Stage

In order to display anything inside a JavaFX `Stage`, you must set a JavaFX `Scene` object on the `Stage`. The content of the `Scene` will then be displayed inside the `Stage` when the `Stage` is shown.

Here is an example of setting a `Scene` on a JavaFX `Stage`:

```
VBox vBox = new VBox(new Label("A JavaFX Label"));
Scene scene = new Scene(vBox);

Stage stage = new Stage();
stage.setScene(scene);
```

Stage Title

You can set the JavaFX `Stage` title via the `Stage setTitle()` method. The `Stage` title is displayed in the title bar of the `Stage` window.

Here is an example of setting the title of a JavaFX `Stage`:

```
stage.setTitle("JavaFX Stage Window Title");
```

Stage Position

You can set the position (X,Y) of a JavaFX `Stage` via its `setX()` and `setY()` methods. The `setX()` and `setY()` methods set the position of the upper left corner of the window represented by the `Stage`.

Here is an example of setting the X and Y position of a JavaFX `Stage` object:

```
Stage stage = new Stage();

stage.setX(50);
stage.setY(50);
```

Note:
It might be necessary to also set the `width` and `height` of the `Stage` if you set the X and Y position, or the stage window might become very small.

Stage Width and Height

You can set the width and of a JavaFX `Stage` via its `setWidth()` and `setHeight()` methods.

Here is an example of setting the width and height of a JavaFX `Stage`:

```
Stage stage = new Stage();

stage.setWidth(600);
stage.setHeight(300);
```

Stage Modality

You can set window modality of a JavaFX `Stage`. The `Stage` modality determines if the window representing the `Stage` will block other windows opened by the same JavaFX application. You set the window modality of a JavaFX Stage via its `initModality()` method.

Here is an example of setting the JavaFX `Stage` modality:

```
public class StageExamples extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("JavaFX App");

        Stage stage = new Stage();
        stage.initModality(Modality.APPLICATION_MODAL);
        //stage.initModality(Modality.WINDOW_MODAL);
        //stage.initModality(Modality.NONE);

        primaryStage.show();
        stage.showAndWait();
    }
}
```

Code Explanation:

This is a full JavaFX application. The `start()` method is executed when the JavaFX application is launched (first `main()` is called which calls launch() which later calls `start()`).

A new `Stage` object is created, its modality mode set, and then both the primary and the new `Stage` objects are made visible (shown). The second `Stage` has its modality set to `Modality.APPLICATION_MODAL` meaning it will block all other windows (stages) opened by this JavaFX application. You cannot access any other windows until this `Stage` window has been closed.

The `Modality.WINDOW_MODAL` modality option means that the newly created `Stage` will block the `Stage` window that "owns" the newly created `Stage`, but only that. Not all windows in the application.

The `Modality.NONE` modality option means that this `Stage` will not block any other windows opened in this application.

Stage Owner

A JavaFX `Stage` can be *owned* by another `Stage`. You set the owner of a `Stage` via its `initOwner()` method. Here is an example of initializing the owner of a JavaFX `Stage`, plus set the modality of the `Stage` to `Modality.WINDOW_MODAL`:

```
public class StageExamples extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("JavaFX App");

        Stage stage = new Stage();
        stage.initModality(Modality.WINDOW_MODAL);

        stage.initOwner(primaryStage);

        primaryStage.show();
        stage.showAndWait();
    }
}
```

Code Explanation:

This example will open a new `Stage` which will block the `Stage` owning the newly created `Stage` (which is set to the primary stage).

Stage Style

You can set the style of a JavaFX `Stage` via its `initStyle()` method. There are a set of different styles you can choose from:

DECORATED	A decorated Stage is a standard window with OS decorations (title bar and minimize / maximize / close buttons), and a white background.
UNDECORATED	An undecorated Stage is a standard window without OS decorations, but still with a white background.
TRANSPARENT	A transparent Stage is an undecorated window with transparent background.
UNIFIED	A unified Stage is like a decorated stage, except it has no border between the decoration area & the main content area
UTILITY	A utility Stage is a decorated window, but with minimal decorations.

Here is an example of setting the style of a JavaFX Stage:

```
stage.initStyle(StageStyle.DECORATED);

//stage.initStyle(StageStyle.UNDECORATED);
//stage.initStyle(StageStyle.TRANSPARENT);
//stage.initStyle(StageStyle.UNIFIED);
//stage.initStyle(StageStyle.UTILITY);
```

Code Explanation:

Only the first line is actually executed. The rest are commented out. They are just there to show how to configure the other options.

Stage Full Screen Mode

You can switch a JavaFX `Stage` into full screen mode via the `Stage setFullScreen()` method. Please note, that you may not get the expected result (a window in full screen mode) unless you set a `Scene` on the `Stage`.

Here is an example of setting a JavaFX `Stage` to full screen mode:

```
VBox vbox = new VBox();
Scene scene = new Scene(vbox);

primaryStage.setScene(scene);
primaryStage.setFullScreen(true);
primaryStage.show();
```

Stage Life Cycle Events

The JavaFX `Stage` can emit a few events you can listen for. These `Stage` events are:

- Close Request
- Hiding
- Hidden
- Showing
- Shown

Stage Life Cycle Events → Close Stage Event Listener

You can listen for close events on a JavaFX `Stage`, meaning you can be notified when the user clicks the button with the X on, in the upper right corner of the `Stage` window. Listening for the `Stage` close event can be useful if you need to clean up some resources after the main `Stage` window is closed, or e.g. need to stop some threads etc.

Here is an example of listening for `Stage` close events:

```
primaryStage.setOnCloseRequest((event) -> {
       System.out.println("Closing Stage");
});
```

Stage Life Cycle Events → Hiding Stage Event Listener

You can attach a `Stage` hiding event listener to a JavaFX stage. The `Stage` hiding event listener is called before the `Stage` is being hidden, but after it has been requested hidden.

Here is an example of attaching a `Stage` hiding event listener to a JavaFX `Stage`:

```
primaryStage.setOnHiding((event) -> {
       System.out.println("Hiding Stage");
});
```

Stage Life Cycle Events → Hidden Stage Event Listener

You can attach a `Stage` hidden event listener to a JavaFX stage. The `Stage` hidden event listener is called after the `Stage` is hidden.

Here is an example of attaching a `Stage` hidden event listener to a JavaFX `Stage`:

```
primaryStage.setOnHidden((event) -> {
       System.out.println("Stage hidden");
});
```

Stage Life Cycle Events → Showing Stage Event Listener

You can attach a `Stage` showing event listener to a JavaFX stage. The `Stage` showing event listener is called after the `Stage` is requested shown, but before the `Stage` is shown.

Here is an example of attaching a `Stage` showing event listener to a JavaFX `Stage`:

```
primaryStage.setOnShowing((event) -> {
       System.out.println("Showing Stage");
});
```

Stage Life Cycle Events → Shown Stage Event Listener

You can attach a `Stage` shown event listener to a JavaFX stage. The `Stage` shown event listener is called after the `Stage` is shown.

Here is an example of attaching a `Stage` shown event listener to a JavaFX `Stage`:

```
primaryStage.setOnShown((event) -> {
       System.out.println("Stage Shown");
});
```

Stage Keyboard Events

It is possible to listen for keyboard events on a JavaFX `Stage`. That way you can catch all keyboard events that occur while the `Stage` has focus.

Here is an example that listens for the ESC and Return keys on the keyboard - when a JavaFX `Stage` has focus:

```
primaryStage.addEventHandler(KeyEvent.KEY_PRESSED,  (event) -> {
    System.out.println("Key pressed: " + event.toString());

    switch(event.getCode().getCode()) {
        case 27 : { // 27 = ESC key
            primaryStage.close();
            break;
        }
        case 10 : { // 10 = Return
            primaryStage.setWidth( primaryStage.getWidth() * 2);
       }
       default:  {
           System.out.println("Unrecognized key");
       }
   }
});
```

END