CSCI 2120:
Software Design & Development II

UNIT4: UI management

GUI framework

JavaFX Element Node: Text

Overview

1. Introduction
2. JavaFX Text Example
3. Set Text
4. Set Font
5. Set Fill Color
6. Set Stroke Color
7. Text X and Y Position
8. Text Origin
9. Multiline Text
10. Text Strikethrough
11. Text Underline
12. Font Smoothing Techniques

Introduction

- The *JavaFX* `Text` element can display a text inside a JavaFX GUI.
- The JavaFX `Text` element is represented by the JavaFX class `javafx.scene.text.Text` .
- You can set the font to be used by the `Text` element, text size, font decorations and many other things.
- Since the JavaFX `Text` is a subclass of the JavaFX `Shape` class, the `Text` class has all the same methods available that other JavaFX `Shape` objects do - *e.g. fill and stroke color and style.*
- The JavaFX `Text` is also a subclass of the JavaFX `Node` class, so the `Text` class also has all the same methods available as any other JavaFX `Node` has, meaning you can set effects on it etc.

JavaFX Text Example

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;

public class TextExample extends Application {

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

   public void start(Stage primaryStage) {

       Text text = new Text("This is a JavaFX text.");

       Scene scene = new Scene(new VBox(text), 300, 250);
       primaryStage.setScene(scene);
       primaryStage.show();

   }
}
```

Set Text

You can set the text of a JavaFX `Text` object via its `setText()` method.

Here is an example of setting the text of a JavaFX `Text` element via `setText()`:

```
Text text = new Text();

text.setText("This is the text to display");
```

Set Font → Font Family

You can set the font of a JavaFX `Text` element via its `setFont()` method.

Here is an example of setting the font of a JavaFX `Text` object via its `setFont()` method:

```
Text text = new Text("Some Text");

text.setFont(Font.font("Arial"));
```

- This example sets the font to be used by the `Text` element to render the text to be of the font family Arial.
- The **JavaFX Font** class used in this example is the `javafx.scene.text.Font` class

Set Font → Font Weight & Size

The `Font` class actually also lets you specify the font weight and font size.

Here is the example, modified to also set font weight and font size for the JavaFX Text element:

```
Text text = new Text("Some Text");

text.setFont(Font.font("Arial", FontWeight.BOLD, 36));
```

- This example sets the font weight to bold and font size to 36.
- The `FontWeight` class used in this example is the `javafx.scene.text.FontWeight` class.

Set Fill Color

Being a `Shape`, you can set the fill color of a JavaFX Text control. The fill color is the "inside" color used to draw the text. You set the fill color of a `Text` element via its `setFill()` method which takes a **JavaFX Color** object as parameter.

Here is an example of setting the fill color of a JavaFX `Text` element via `setFill()`:

```
Text text = new Text("Some Text");

text.setFill(Color.YELLOW);
```

Set Stroke Color

Being a `Shape`, you can also set the stroke color of a JavaFX `Text` element. The stroke color is the "outline" or "boundary" color used to draw the text. By default, text is rendered only using the fill color, but setting a stroke color can add a nice effect. You set the stroke color of a `Text` element via its `setStroke()` method which takes a **JavaFX Color** object as parameter.

Here is an example of setting the stroke color of a JavaFX `Text` element via `setStroke()`:

```
Text text = new Text("Some Text");

text.setStroke(Color.GREEN);
```

Text X and Y Position

The X and Y position of a JavaFX `Text` element determines where inside its parent container element the `Text` element is displayed - provided the parent container respects this position (**Pane** does, **VBox** does not). You can set the X and Y position of a `Text` element via its methods setX() and setY().

Here is an example of setting the X and Y position of a JavaFX `Text` element:

```
Text text = new Text("Some Text");

text.setX(50);
text.setY(25);
```

Text Origin

The JavaFX `Text` element has an *origin* which controls how the text is displayed relative to the Y position of the `Text` element. You set the origin using the `Text setTextOrigin()` method.

Here is an example of setting the Text origin:

```
Text text = new Text("Some Text");

text.setTextOrigin(VPos.BASELINE);
```

The `setTextOrigin()` method takes a `VPos` parameter. The `VPos` class contains the following constants you can choose between:

`VPos.BASELINE`	The Y position of the Text is interpreted to mean the Y baseline of the displayed text. The text is displayed just above the baseline, with some characters extending below the baseline.
`VPos.BOTTOM`	The Y position of the Text is interpreted to mean the bottom the displayed text. This is lower than BASELINE.
`VPos.CENTER`	The Y position of the Text is interpreted to mean the center of the text vertically.
`VPos.TOP`	The Y position of the Text is interpreted to mean the top of the text vertically.

Multiline Text → line break

The JavaFX `Text` element will break the text it displays on to multiple lines based on these rules:

- If the text contains a line break (`\n`) .
- If the text width exceeds a wrapping width set on the `Text` element.

Here is first an example showing a text that contains a line break:

```
Text text = new Text("This is a JavaFX text.\nLine 2");
```

The `Text` element will break the text before `"Line"` because the **String** contains a line break character.

Multiline Text → wrapping width

The JavaFX `Text` element will break the text it displays on to multiple lines based on these rules:

- If the text contains a line break (`\n`) .
- If the text width exceeds a wrapping width set on the `Text` element.

Here is an example of setting a text wrapping width on the JavaFX `Text` element:

```
Text text = new Text("This is a longer JavaFX text.");
text.setWrappingWidth(80);
```

The JavaFX `Text` element will attempt to break the text between words. Thus, if after a specific word the text width is wider than the wrapping width, the `Text` element will wrap the text before that word that makes the text wider than the wrapping width.

Text Strikethrough

The JavaFX `Text` element enables you to apply a *strikethrough* decoration to the text it displays. You enable the strikethrough decoration via the `Text setStrikethrough()` method, passing a value of true as parameter. A parameter value of `false` will disable the strikethrough effect.

Here is an example of enabling the JavaFX `Text` *strikethrough* decoration via the `setStrikethrough()` method:

```
Text text = new Text("This is JavaFX text.");
text.setStrikethrough(true);
```

Text Underline

The JavaFX `Text` element enables you to apply an *underline* decoration to the text it displays. You enable the underline decoration via the `Text setUnderline()` method, passing a value of `true` as parameter. A parameter value of `false` will disable the underline decoration.

Here is an example of enabling the JavaFX `Text` *underline* decoration via the `setUnderline()` method:

```
Text text = new Text("This is JavaFX text.");
text.setUnderline(true);
```

Font Smoothing Techniques

The JavaFX `Text` element contains two different font smoothing (antialiasing) techniques you can choose between:

1. The first technique is called LCD
2. The second is called GRAY.

You can choose which font smoothing technique the `Text` element should use via the `setFontSmoothingType()` method.

Here are examples of setting both LCD and GRAY as font smoothing technique on a JavaFX `Text` element:

```
Text text = new Text("This is JavaFX text.");

text.setFontSmoothingType(FontSmoothingType.GRAY);
text.setFontSmoothingType(FontSmoothingType.LCD);
```

Which of the two font smoothing techniques fits best to your application you will have to experiment with.

END