Multimedia-Programmierung
Übung 4

Ludwig-Maximilians-Universität München
Sommersemester 2013
Today

- Scene Graph and Layouts
- Interaction
- CustomNodes
- Effects
- Animation
- Stylesheets
- MediaPlayer
JavaFX Scene Graph 1

- Scene graph is a tree data structure consisting of **nodes**
- Nodes can be the root, branches or leafs
- Branches have one or more children, while leafs have no children
JavaFX Scene Graph 2

- Nodes can be UI components, text, images …
- Nodes can be transformed, animated or applied with effects

```javascript
var counter = 0;
Stage {
    title: "My first App"
    width: 250
    height: 200

    scene: Scene {
        content: [
            Button {
                text: "press me"
                layoutX: 80, layoutY: 100
                action: function() { counter++; }
            }
            Text {
                font: Font { size: 24 }
                x: 100, y: 80
                content: bind "{counter}"
            }
        ]
    }
}
```
Order Matters

• Nodes are painted in their order
• Later nodes are painted on top of previous nodes
Grouping Nodes

- Nodes can be grouped together (javafx.scene.Group)
- Groups enable the manipulation of several nodes at the same time

```java
Stage {
    title: "My first Group", width: 200, height: 200
    scene: Scene {
        content: [
            Group {
                content: [
                    Circle {
                        centerX: 120, centerY: 120, radius: 20
                        fill: Color.MAGENTA, stroke: Color.BLACK
                        strokeWidth: 2
                    }
                    Rectangle {
                        x: 50, y: 40, width: 50, height: 50, fill: Color.BLUE
                        stroke: Color.BLACK, strokeWidth: 2
                    }
                ]
            }
            Rectangle {
                x: 10, y: 30, width: 50, height: 50, fill: Color.RED
                stroke: Color.BLACK, strokeWidth: 2
            }
        ]
    }
}
```
Changing Nodes

• Changes on a node (e.g. transformations) affect the node’s children in the same way
Layout Nodes

- Until now: layouts defined by absolute coordinates

- Now: layout nodes support relative layouts (javafx.scene.layout)
VBox and HBox Layouts

• Nodes are laid out horizontally (HBox) or vertically (VBox)
# HBox and VBox variables

## HBox

<table>
<thead>
<tr>
<th>access</th>
<th>name</th>
<th>type</th>
<th>Can Read</th>
<th>Can Init</th>
<th>Can Write</th>
<th>Default Value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public</td>
<td>hpos</td>
<td>HPos</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>HPos.LEFT</td>
<td>The horizontal position of the row of nodes within this container's width.</td>
</tr>
<tr>
<td>public</td>
<td>nodeVPos</td>
<td>VPos</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>VPos.TOP</td>
<td>The vertical position of each node within the hbox's row.</td>
</tr>
<tr>
<td>public</td>
<td>spacing</td>
<td>Number</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>0</td>
<td>The amount of horizontal space between each child node in the hbox.</td>
</tr>
<tr>
<td>public</td>
<td>vpos</td>
<td>VPos</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>VPos.TOP</td>
<td>Defines the vertical position of the row of nodes within this container's height.</td>
</tr>
</tbody>
</table>

http://docs.oracle.com/cd/E17802_01/javafx/javafx/1.3/docs/api/javafx.scene.layout/javafx.scene.layout.HBox.html

## VBox:

- same variables
- nodeHPos instead of nodeVPos
Tile Layout

• Nodes are laid out in tiles
• Tiles are of equal size (by default the size of the largest node)
• Nodes can be ordered horizontally or vertically
• The layout will automatically wrap its content when the width or height of the Tile layout is reached (has to be specified manually)

```
Tile {
    columns: 2
    rows: 3
    tileWidth: 40
    nodeHPos: HPos.LEFT
    padding: Insets{top: 10 left: 10}
    vgap: 5
    hgap: 10
    content: for (i in [0..2])
        [choices[i], lights[i]] //Tile
}
```

http://java.sun.com/javafx/1/tutorials/ui/layout/
Tile Layout
Examples 1

- Horizontal tile layout, no width, no column count
Tile Layout
Examples 2

- Horizontal tile layout, with width, no column count

nodes are wrapped at 200

tile with a fixed width
Tile Layout
Examples 3

- Horizontal tile layout, no width, two columns

Nodes are arranged in two columns horizontally.
Tile Layout
Examples 4

- Vertical tile layout, no height, no column count
Tile Layout
Examples 5

- Vertical tile layout, no width, two rows

Nodes are arranged in two rows vertically.

Compare to two columns horizontally.

Stage {
  title: "Tile Layout", width: 200, height: 200
  scene: Scene {
    content: [
      Tile {
        vertical: true
        rows: 2
        content: [
          Circle {
            ...
          }
          Rectangle {
            ...
          }
          Rectangle {
            ...
          }
          Rectangle {
            ...
          }
        ]
      }
    ]
  }
}
Transformations

- Nodes can be transformed (rotation, translation, scaling, skew)
- Transforming a node does not change its size, height, width, x, y, etc. but its coordinate system
Transformations
the transform variable

- Transformations are applied in order of their appearance within the transform sequence

1. translate(100,100)
2. rotate(90,0,0)
Transformations

some examples

- `Transform.rotate(angle,x,y)` rotates clockwise around a pivot point

  rotate 45° clockwise around 0,0
  
  ```
  ... transforms: [
    Transform.rotate(45, 0, 0)
  ]
  ...
  ```

  around the center (if rectangle is 100x100px)
  
  ```
  ... transforms: [
    Transform.rotate(45, 50, 50)
  ]
  ...
  ```
Transformations
some examples (2)

- `Transform.scale(xfactor, yfactor)` scales the node’s axes
Interaction with Nodes

- Nodes can receive mouse and keyboard events
- Depending on the node, different events might be available
- Instance variables map to event related functions
- Events include (but are not limited to):
  - onKeyDownPressed
  - onKeyReleased
  - onMouseClicked
  - onMouseDragged
  - onMouseMoved
  - onMouseReleased
  - onMouseWheelMoved
  - etc.
Interaction with Nodes
example 1: clicking a node

Stage {
  title: "Clicking a Node"
  scene: Scene {
    width: 400
    height: 400
    content: [
      Circle {
        centerX: 100, centerY: 100
        radius: 40
        fill: Color.RED
        onMouseClicked: function(e: MouseEvent): Void {
          (e.node as Circle).fill = Color.BLUE; // type casting
        }
      }
    ]
  }
}

function assigned to instance variable onMouseClicked

JavaFX type casting: (object as object)
Interaction with Nodes

example 2: entering an element

Stage {
    title: "Hovering a Node"
    scene: Scene {
        width: 200
        height: 200
        content: [
            Circle {
                centerX: 100, centerY: 100
                radius: 40
                fill: Color.RED
                onMouseEntered: function( e: MouseEvent ):Void {
                    (e.node as Circle).fill = Color.BLUE;
                }
                onMouseExited: function( e: MouseEvent ):Void {
                    (e.node as Circle).fill = Color.RED;
                }
            }
        ]
    }
}
Interaction with Nodes
example3: simple node dragging

```javascript
var xOffset:Number = 0;
var yOffset:Number = 0;
Stage {
    title : "Dragging a Node"
    scene: Scene {
        width: 200
        height: 200
        content: [
            Circle {
                centerX: 100, centerY: 100
                radius: 40
                fill: Color.RED
                onMousePressed: function( e: MouseEvent ):Void {
                    def cur_circle = (e.node as Circle);
                    xOffset = e.sceneX - cur_circle.centerX;
                    yOffset = e.sceneY - cur_circle.centerY;
                }
                onMouseDragged: function( e: MouseEvent ):Void {
                    def cur_circle = (e.node as Circle);
                    cur_circle.centerX = e.sceneX - xOffset;
                    cur_circle.centerY = e.sceneY - yOffset;
                }
            }
        ]
    }
}
```

When the circle is pressed, calculate the offset.

While dragging the circle, recalculate its center.
CustomNodes

• Build own custom nodes that can be used within a scene

• Build subclass of CustomNode

• Implement `create()` function, that returns a node
Effects

**Attention**: desktop profile only

Effects are applied to nodes using the `effect` variable

Effects include:

- Blend
- Bloom
- Shadow
- Glow
- Gaussian Blur
- Reflection
- Etc.
Effects

example1: shadow

Stage {
    title: "Shadow Effect"
    scene: Scene {
        width: 400
        height: 400
        content: [
            Circle {
                centerX: 100, centerY: 100
                radius: 40
                fill: Color.RED
                effect: DropShadow {
                    offsetX: 10
                    offsetY: 10
                    color: Color.BLACK
                    radius: 10
                }
            }
        ]
    }
}
Effects

example2: Reflection

Stage {
    title: "Shadow Effect"
    scene: Scene {
        width: 400
        height: 400
        content: [
            Circle {
                centerX: 100, centerY: 100
                radius: 40
                fill: Color.RED
                effect: Reflection {
                    fraction: 0.45
                    topOffset: 0.0
                    topOpacity: 0.5
                    bottomOpacity: 0.0
                }
            }
        ]
    }
}

adding the Reflection effect to the Circle node.
Animation

JavaFX support the keyframe concept
That is, animations are defined by so called keyframes
Other values are interpolated
Animation
creating a timeline

To animate an object, a **Timeline** is needed
Within the **Timeline**, **Keyframes** are defined

```javascript
var x:Number = 0;
Timeline {
    repeatCount: Timeline.INDEFINITE
    autoReverse: true
    keyFrames : [
        KeyFrame {
            time 0s
            values x => 0.0
        }
        KeyFrame {
            time 5s
            values x => 200 tween Interpolator EASEBOTH
        }
    ]
}.play();
```

- **first keyframe at the start of the timeline** (0s <- Duration type)
- **loop is indefinite and reverses itself**
- **interpolate x**
- **second keyframe at 5s uses the EASEBOTH interpolator**
- **play the timeline**
Animation
binding to the animated value

The interpolated variable can be used like any other variable
Animation
Interpolators

Discrete: no interpolation, value “jumps” directly to the keyframe value

Linear: linear interpolation

EaseIn: interpolated values smaller at the beginning then linear

EaseOut: smaller in the end

EaseBoth: EaseIn + EaseOut
Stylesheets

- Stylesheets determine the appearance of UI elements
- Separate file *.css
- Mostly known from HTML

from JavaFX in Action (Simon Morris)
Stylesheets

```javascript
Text {
    fill: navy;
    font: bold italic 35pt "sans-serif";
}
```

Or:

```javascript
"javafx.scene.text.Text" {
    fill: navy;
    font: bold italic 35pt "sans-serif";
}
```

```
Stage {
    title: "Stylesheets"
    scene: Scene {
        stylesheets: ["{__DIR__}style.css"]
        width: 250
        height: 80
        content: [
            Text {
                x: 10
                y: 30
                content: "My styled text"
            }
        ]
    }
}
```
Stylesheets

```css
Text#Caption {
    fill: navy;
    font: bold italic 35pt "sans-serif";
}
```

```java
Stage {
    title: "Stylesheets"
    scene: Scene {
        stylesheets: ["{__DIR__}style.css"]
        width: 280
        height: 100
        content: [
            Text {
                id: "Caption"
                x: 10
                y: 30
                content: "Text with Style"
            }
            Text {
                x: 10
                y: 60
                content: "Text without Style"
            }
        ]
    }
}
```

Text with Style

Text without Style
Stylesheets

```
style.css

Text.Caption {
    fill: navy;
    font: bold italic 35pt "sans-serif";
}

Main.fx

Stage {
    title: "Stylesheets"
    scene: Scene {
        stylesheets: ["${__DIR__}style.css"]
        width: 280
        height: 100
        content: [
            Text {
                styleClass: "Caption"
                x: 10
                y: 30
                content: "Text with Style"
            }
            Text {
                x: 10
                y: 60
                content: "Text without Style"
            }
        ]
    }
}
```
Stylesheets

Style Sheets in JavaFX Version 1.3:

```css
.scene {
    -fx-font: 16pt "Amble Cn";
    -fx-base: #AEBBD2;
    -fx-accent: #385589;
    -fx-mark-color: #3E857C;
}

.text-box {
    -fx-effect: innershadow( two-pass-box, rgba(0,0,0.2), 10, 0.0, 0, 2 );
    -fx-text-fill: #385589
}
```
MediaPlayer

- `javafx.scene.media.Media` is used for storing audio or video
- `javafx.scene.media.MediaPlayer` controls the play of the media
- Common media formats (e.g. .mp3, .flv, .avi, .mov, .mp4, .wav, etc.) and audio/video codecs supported (e.g. MP3, MPEG-4, MPEG-1, MIDI, H264, H.261 etc.)

```javascript
var song1 = Media {
  onError: function(e:MediaError) {
    println("got a media error {e}"");
  }
  source: "someURLorFile"
};

var mediaPlayer:MediaPlayer = MediaPlayer {
  media: song1
  volume: 0.5
  autoPlay: false
  onError: function(e:MediaError) {
    println("got a MediaPlayer error : {e.cause} {e}"");
    mediaPlayer.stop();
    mediaPlayer.media = null;
  }
  onEndOfMedia: function() {
    println("reached end of media");
    mediaPlayer.play();
    mediaPlayer.stop();
    mediaPlayer.media = null;
  }
};
```