Multimedia-Programmierung
Übung 4

Ludwig-Maximilians-Universität München
Sommersemester 2011
Today

- Scene Graph and Layouts
- Interaction
- CustomNodes
- 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: 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
circle
rectangle
```

```
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.

Moving this group moves both children.
Layout Nodes

• Until now: layouts defined by absolute coordinates

![Diagram of layout nodes with absolute coordinates](image1)

• Now: layout nodes support relative layouts (java.scene.layout)

![Diagram of layout nodes with relative layout](image2)
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://java.sun.com/javafx/1.2/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)

```java
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

```java
Stage {
    title: "Tile Layout", width: 200, height: 200
    scene: Scene {
        content: [
            Tile {
                content: [
                    Circle {
                        ...
                    },
                    Rectangle {
                        ...
                    },
                    Rectangle {
                        ...
                    },
                    Rectangle {
                        ...
                    }
                ]
            }
        ]
    }
}
```
Tile Layout
Examples 2

- Horizontal tile layout, with width, no column count

nodes are wrapped at 200

tile with a fixed width

Stage {
  title: "Tile Layout", width: 200, height: 200
  scene: Scene {
    content: [
      Tile {
        width: 200
        content: [
          Circle {
            ...
          }
          Rectangle {
            ...
          }
          Rectangle {
            ...
          }
        ]
      }
    ]
  }
}
Tile Layout
Examples 3

- Horizontal tile layout, no width, two columns

Nodes are arranged in two columns horizontally.

```
Stage {
    title: "Tile Layout", width: 200, height: 200
    scene: Scene {
        content: [
            Tile {
                columns: 2
                content: [
                    Circle {
                        ...
                    }
                    Rectangle {
                        ...
                    }
                    Rectangle {
                        ...
                    }
                ]
            }
        ]
    }
}
```
Tile Layout
Examples 4

• Vertical tile layout, no height, no column count

Stage {
  title: "Tile Layout", width: 200, height: 200
  scene: Scene {
    content: [
      Tile {
        vertical: true
        content: [
          Circle {
            ... 
          }
          Rectangle {
            ... 
          }
          Rectangle {
            ... 
          }
        ]
      }
    ]
  }
}
Tile Layout
Examples 5

- Vertical tile layout, no width, two rows

nodes are arranged in two row vertically

Vertical layout with two rows

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 {
                        ...
                    }
                ]
            }
        ]
    }
}
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

Stage {
  title : "Transformations"
  scene: Scene {
    width: 400
    height: 400
    content: [
      Rectangle {
        x: 0, y: 0
        width: 100, height: 100
        fill: Color.BLUE
        stroke: Color.BLACK
        transforms: [
          Transform.translate(100,100),
          Transform.rotate(90, 0, 0)
        ]
      }
    ]
  }
}

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

around the center (if rectangle is 100x100px)
Transformations
some examples (2)

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

```
... transforms: [
    Transform.scale(2.0, 2.0)
]...
```

```
... transforms: [
    Transform.scale(0.50, 0.50)
]...
```
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):
  – onKeyPressed
  – 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

```plaintext
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

```java
public class MyCustomNode extends CustomNode{
    public var text:String;

    override protected function create () : Node {
        HBox {
            content: [
                TextBox {
                    text: bind text
                }
                Button {
                    text: "OK"
                    action: function() {}
                }
            ]
        }
    }
}
```

```java
Main.fx
Stage {
    title: "Stylesheets"
    scene: Scene {
        width: 280
        height: 100
        content: [
            MyCustomNode{
                text: "My Custom Node"
            }
        ]
    }
}
```
Stylesheets

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

aus JavaFX in Action (Simon Morris)
Stylesheets

`.scene{
    -fx-fill: navy;
    -fx-font: bold italic 35pt "sans-serif";
}

style.css

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

Main.fx

My styled text
Stylesheets

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

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"
      }
    ]
  }
}

Main.fx
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:MediaPlayerError) {
    println("got a MediaPlayer error : \{cause\} \{e\}\");
    mediaPlayer.stop();
    mediaPlayer.media = null;
  }
  onEndOfMedia: function() {
    println("reached end of media\"");
  }
};
```
Useful Links

• JavaFX Overview
  http://download.oracle.com/javafx/index.html

• JavaFX Getting Started
  http://download.oracle.com/javafx/1.3/tutorials/core/getStarted/

• The JavaFX GUI Tutorial
  http://download.oracle.com/javafx/1.3/tutorials/ui/index.html

• JavaFX API
  http://download.oracle.com/docs/cd/E17802_01/javafx/javafx/1.3/docs/api/