

Multimedia-Programmierung

Übung 4

Ludwig-Maximilians-Universität München
Sommersemester 2012

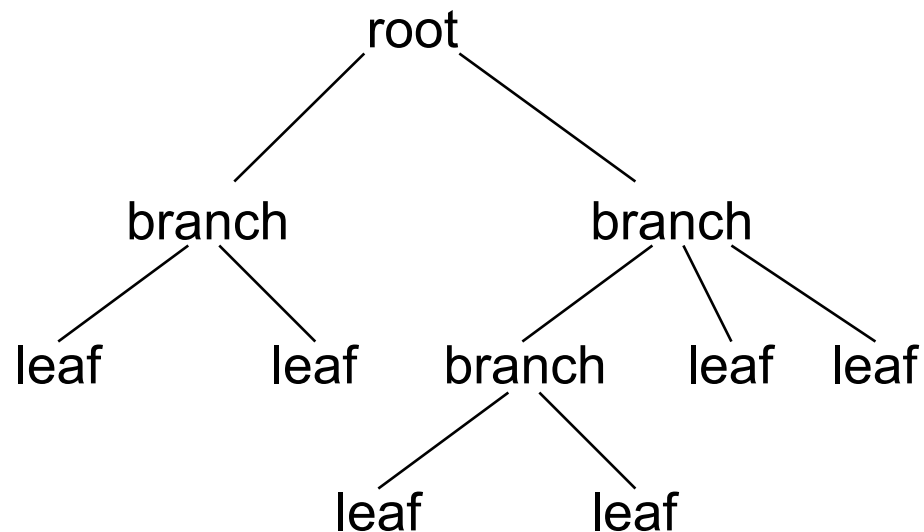


Today

- 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

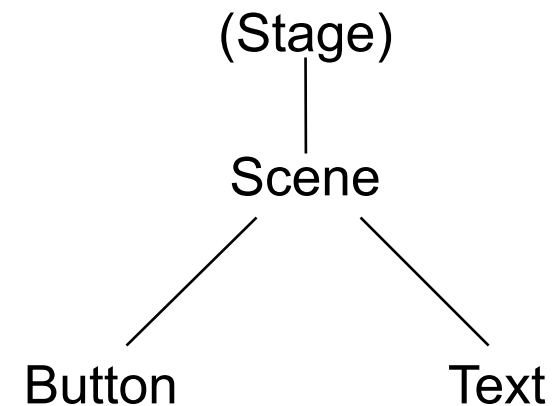


JavaFX Scene Graph 2

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

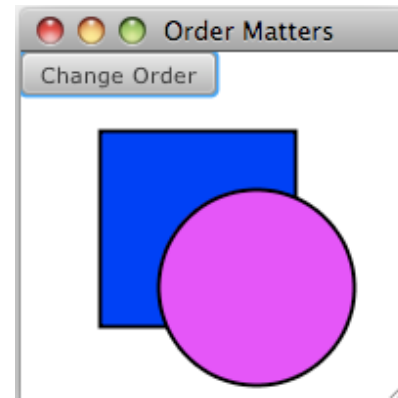
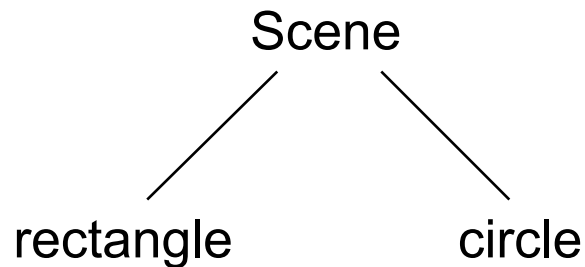
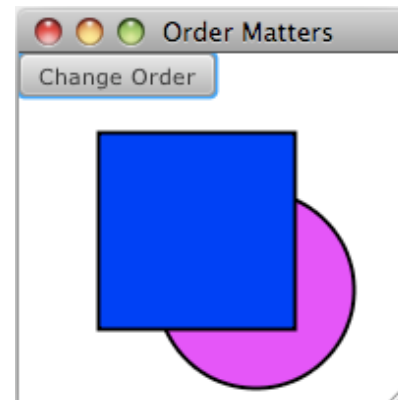
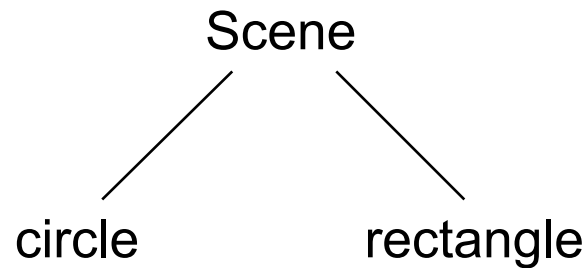
```
var counter = 0;
Stage {
  title: "My first App"
  width: 250
  height: 200

  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
      }
    ]
  }
}
```



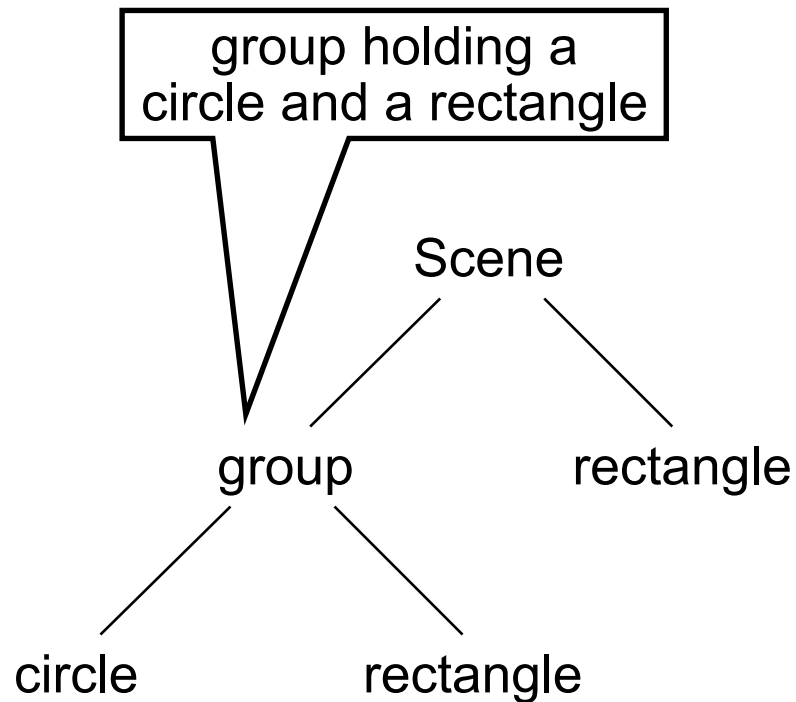
Order Matters

- 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

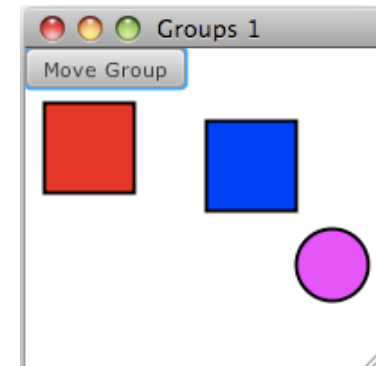
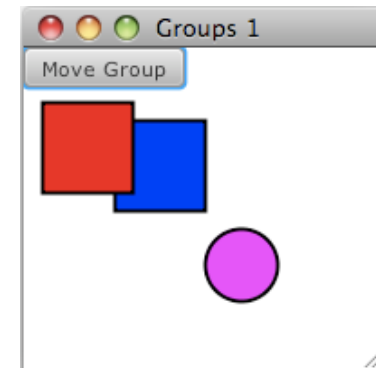
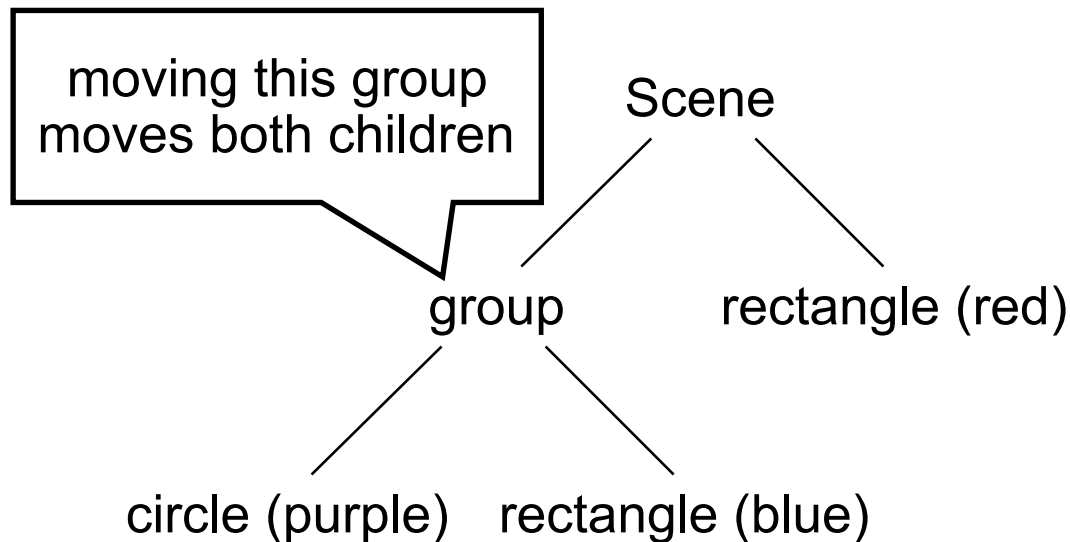


```

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
      }
    ]
  }
}
  
```

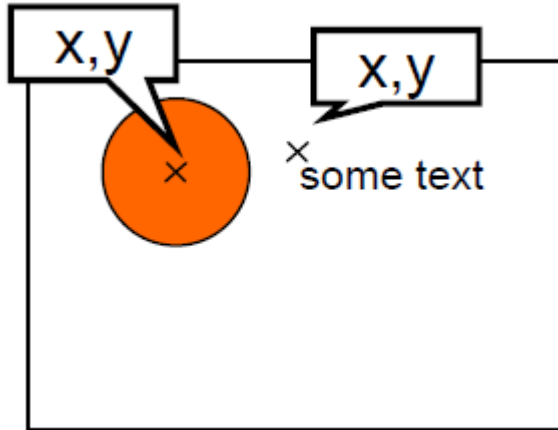
Changing Nodes

- Changes on a node (e.g. transformations) affect the node's children in the same way

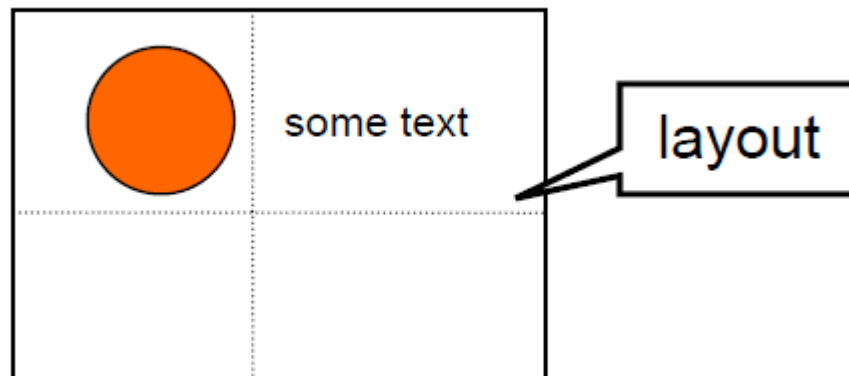


Layout Nodes

- Until now: layouts defined by absolute coordinates

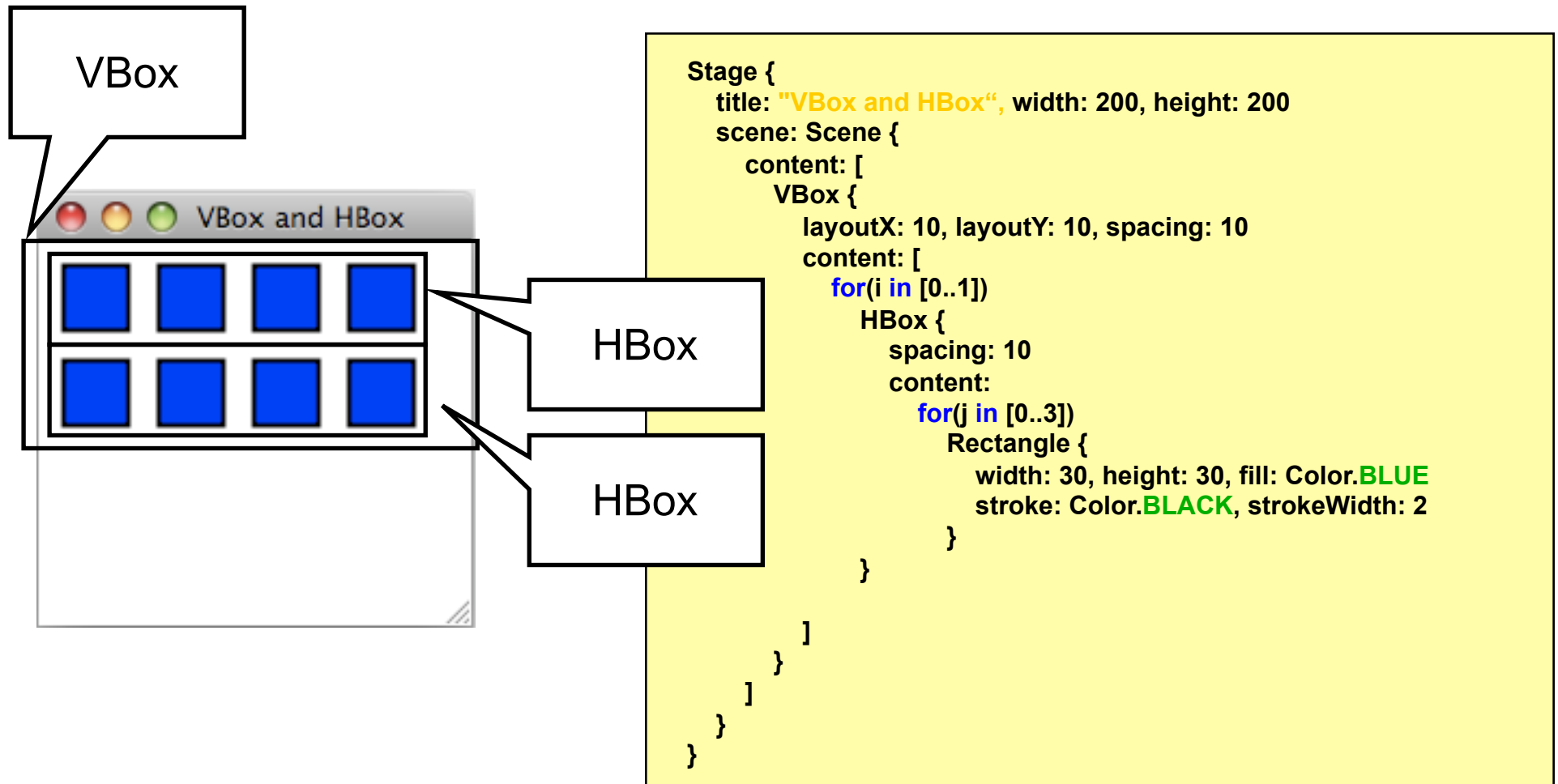


- Now: layout nodes support relative layouts ([javafx.scene.layout](https://docs.oracle.com/javafx/2/scene/layout/scene_layout_classes_tutorial.html))




VBox and HBox Layouts

- Nodes are laid out horizontally (HBox) or vertically (VBox)



HBox and VBox variables

- HBox

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

<http://java.sun.com/javafx/1.2/docs/api/javafx.scene.layout/javafx.scene.layout.HBox.html>

- VBox:
 - same variables
 - nodeHPos instead of nodeVPos

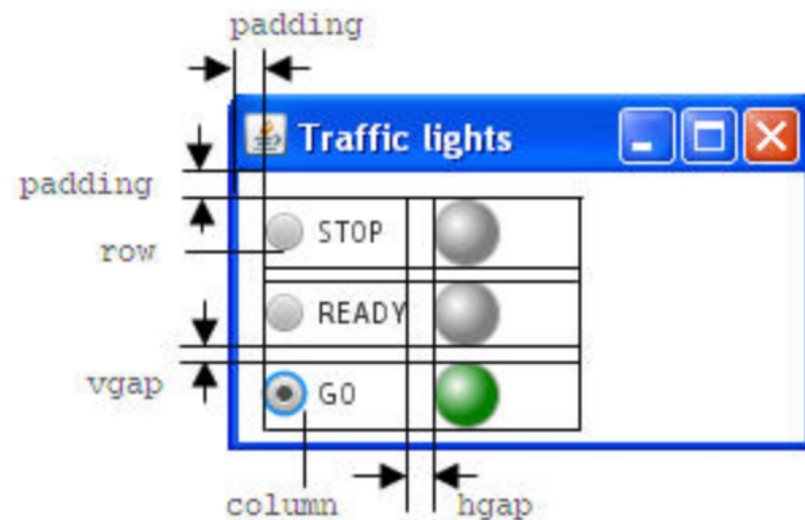
Tile Layout

- 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)

```

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

```

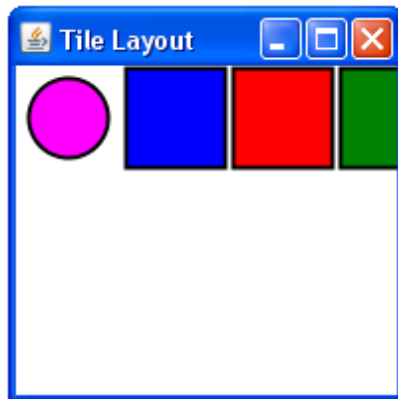


<http://java.sun.com/javafx/1/tutorials/ui/layout/>

Tile Layout

Examples 1

- Horizontal tile layout, no width, no column count



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

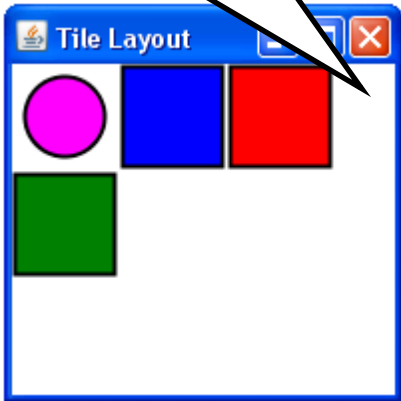
tile without
any parameters

Tile Layout

Examples 2

- Horizontal tile layout, with width, no column count

nodes are wrapped
at 200



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

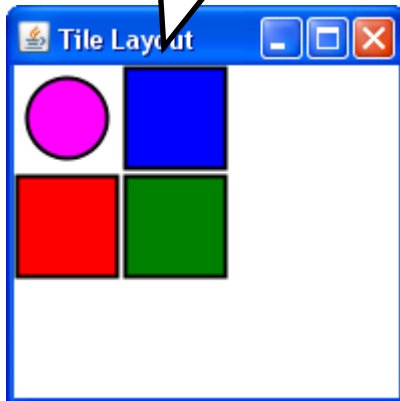
tile with
a fixed width

Tile Layout

Examples 3

- Horizontal tile layout, no width, two columns

nodes are arranged
in two columns
horizontally



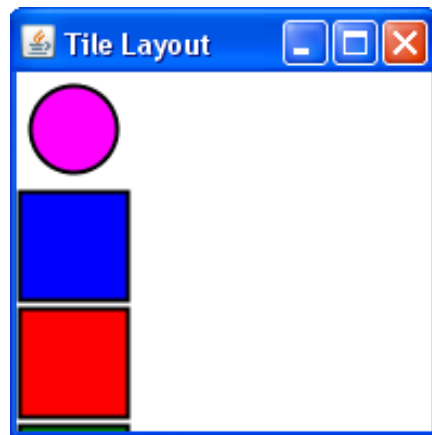
```
Stage {  
  title: "Tile Layout", width: 200, height: 200  
  scene: Scene {  
    content: [  
      Tile {  
        columns: 2  
        content: [  
          Circle {  
            ...  
          }  
          Rectangle {  
            ...  
          }  
          Rectangle {  
          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

layout with
two columns

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 {  
            ...  
          }  
          Rectangle {  
            ...  
          }  
        ]  
      }  
    ]  
  }  
}
```

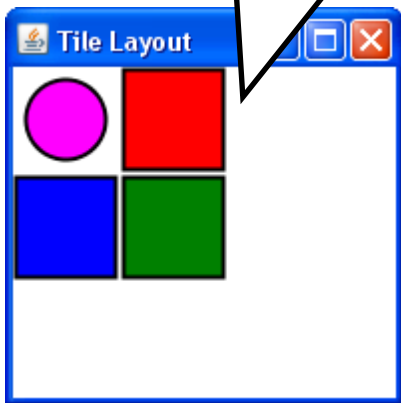
vertical layout

Tile Layout

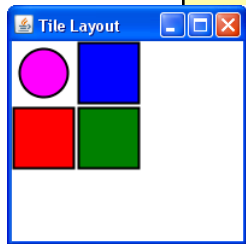
Examples 5

- Vertical tile layout, no width, two rows

nodes are arranged in two row vertically



compare to two columns horizontally

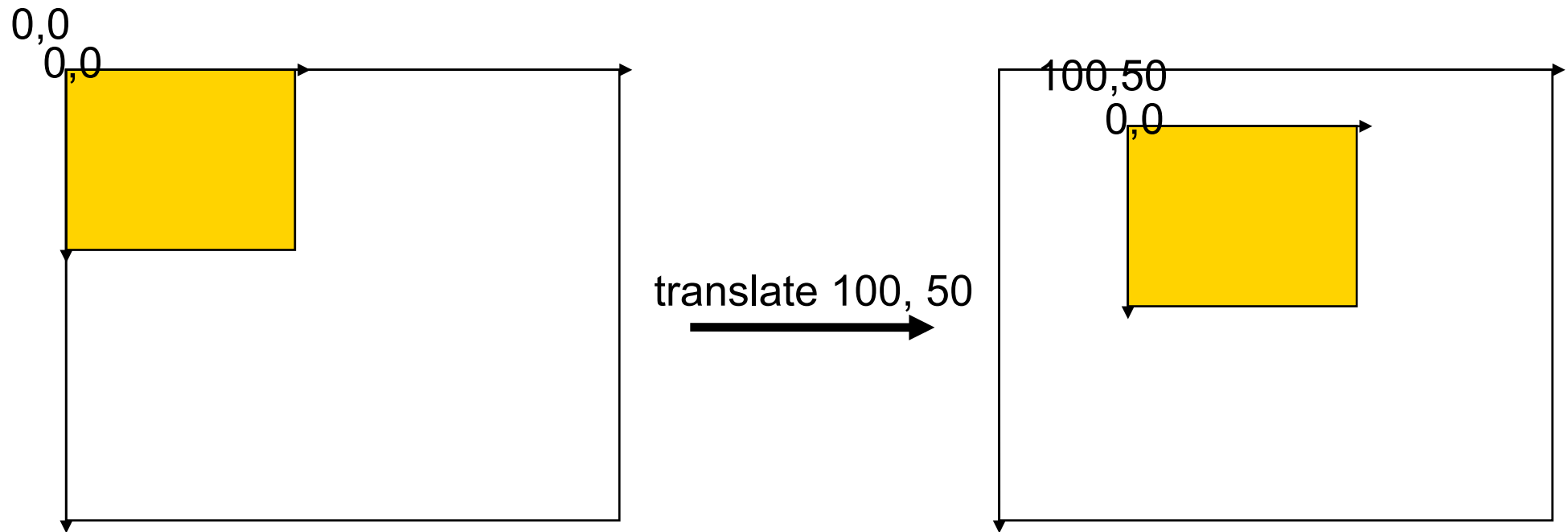


```
Stage {  
  title: "Tile Layout", width: 200, height: 200  
  scene: Scene {  
    content: [  
      Tile {  
        vertical: true  
        rows: 2  
        content: [  
          Circle {  
            ...  
          }  
          Rectangle {  
            ...  
          }  
          Rectangle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

vertical layout with two rows

Transformations

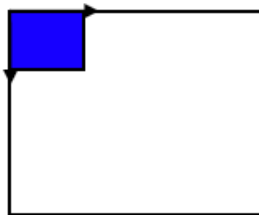
- Nodes can be transformed (rotation, translation, scaling, skew)
- Transforming a node does not change its size, height, width, x, y, etc. but its coordinate system



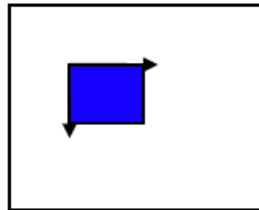
Transformations

the transform variable

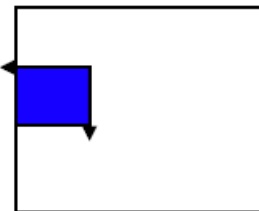
- Transformations are applied in order of their appearance within the **transform** sequence



1. translate(100,100)



2. rotate(90,0,0)

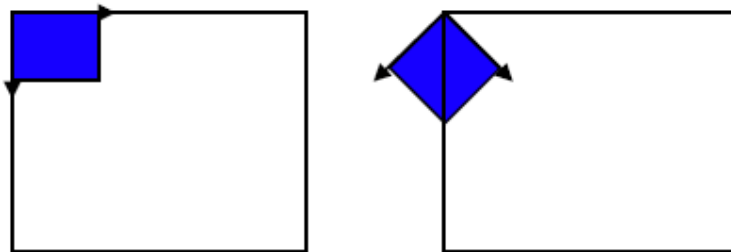


```
Stage {  
  title : "Transformations"  
  scene: Scene {  
    width: 400  
    height: 400  
    content: [  
      Rectangle {  
        x: 0, y: 0  
        width: 100, height: 100  
        fill: Color.BLUE  
        stroke: Color.BLACK  
        transforms: [  
          Transform.translate(100,100),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

Transformations

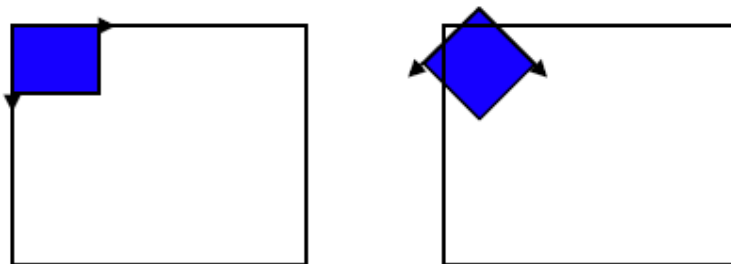
some examples

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



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

rotate 45° clockwise
around 0,0



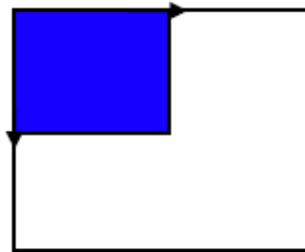
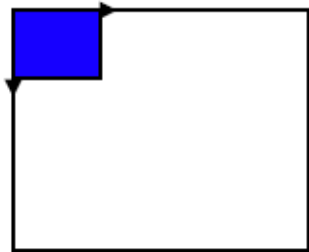
```
...  
transforms: [  
    Transform.rotate(45, 50, 50)  
]  
...
```

around the center
(if rectangle is 100x100px)

Transformations

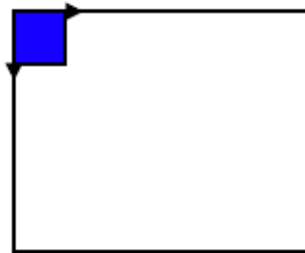
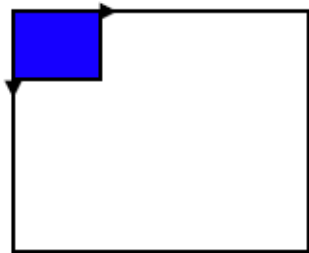
some examples (2)

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



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

scale 200%

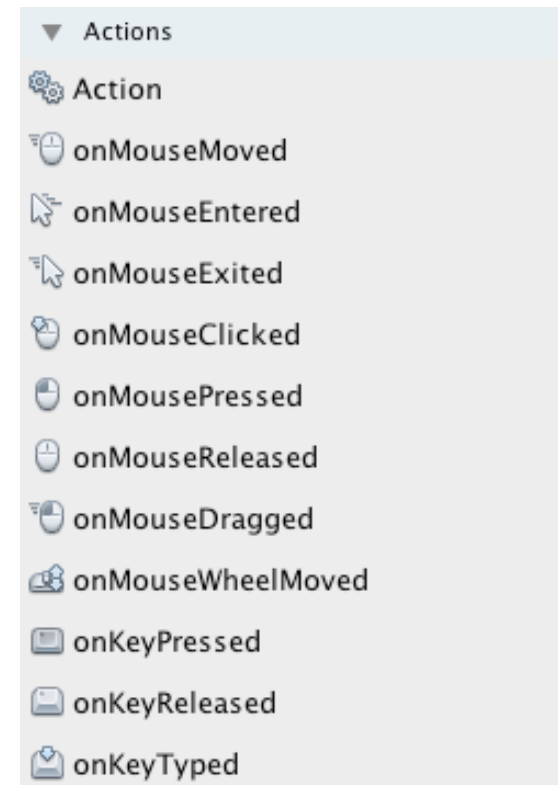


```
...  
transforms: [  
    Transform.scale(0.50, 0.50)  
]  
...
```

scale 50%

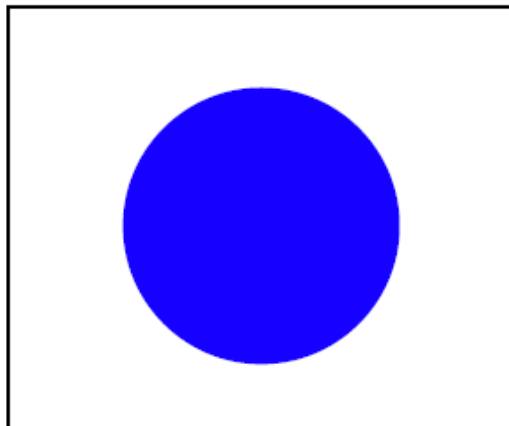
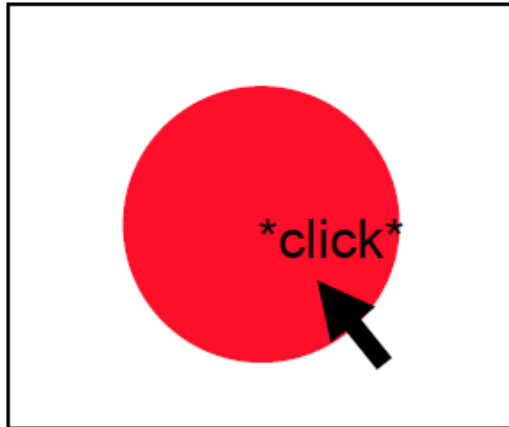
Interaction with Nodes

- 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
        }
      ]
    ]
  }
}

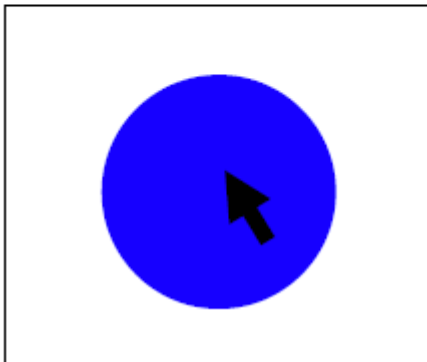
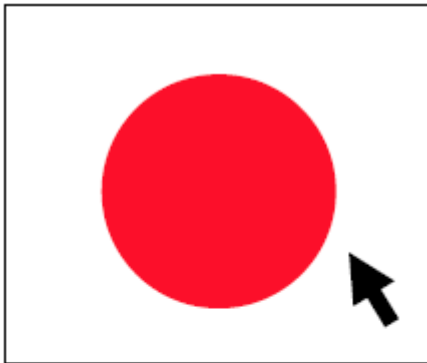
```

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

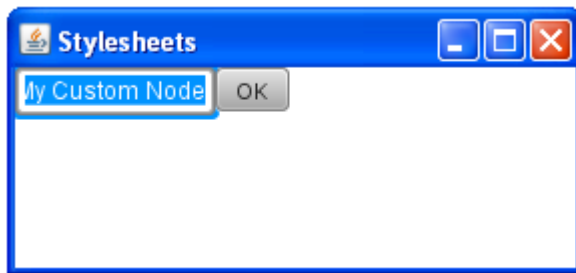
```
var xOffset:Number = 0;
var yOffset:Number = 0;
Stage {
  title : "Dragging a Node"
  scene: Scene {
    width: 200
    height: 200
    content: [
      Circle {
        centerX: 100, centerY: 100
        radius: 40
        fill: Color.RED
        onMousePressed: function( e: MouseEvent ):Void {
          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



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

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

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

Effects

Attention: desktop profile only

Effects are applied to nodes using the **effect** variable

Effects include:

Blend

Bloom

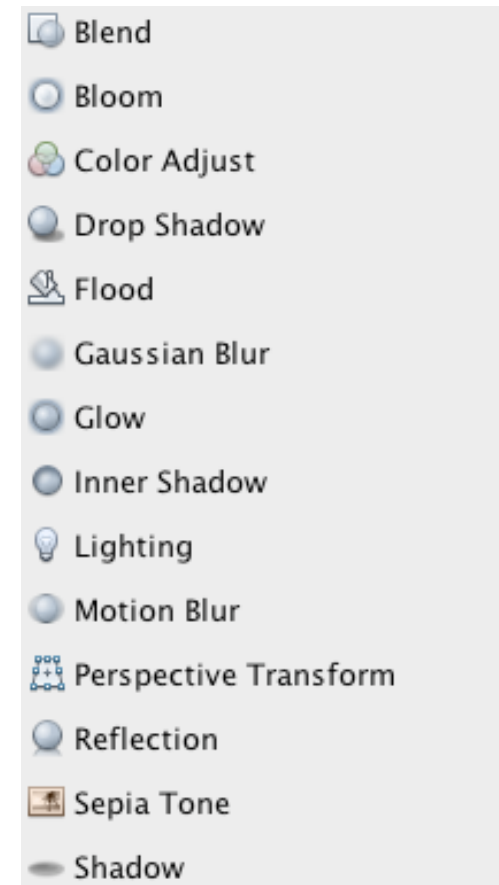
Shadow

Glow

Gaussian Blur

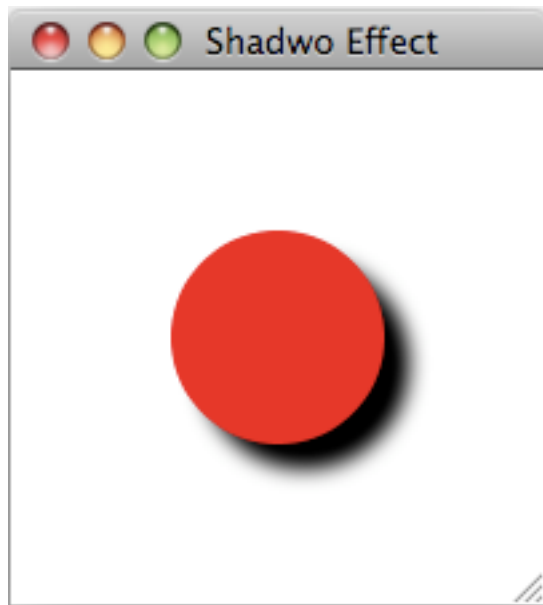
Reflection

Etc.



Effects

example1: shadow

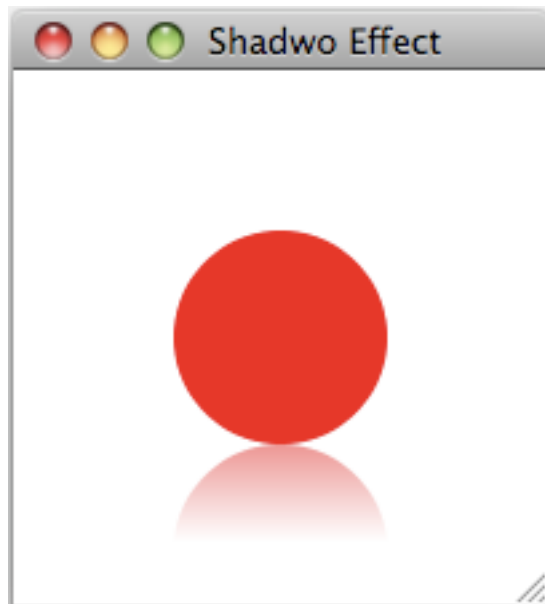


```
Stage {  
  title : "Shadow Effect"  
  scene: Scene {  
    width: 400  
    height: 400  
    content: [  
      Circle {  
        centerX: 100, centerY: 100  
        radius: 40  
        fill: Color.RED  
        effect: DropShadow {  
          offsetX: 10  
          offsetY: 10  
       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

adding the
DrowpShadow effect
to the Circle node.

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  
        }  
      }  
    ]  
  }  
}
```

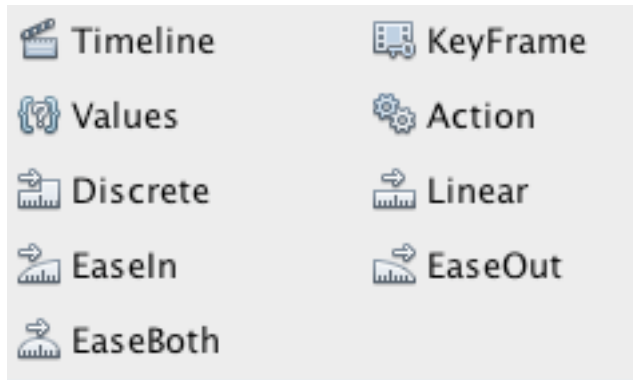
adding the
Reflection effect
to the Circle node.

Animation

JavaFX support the keyframe concept

That is, animations are defined by so called keyframes

Other values are interpolated

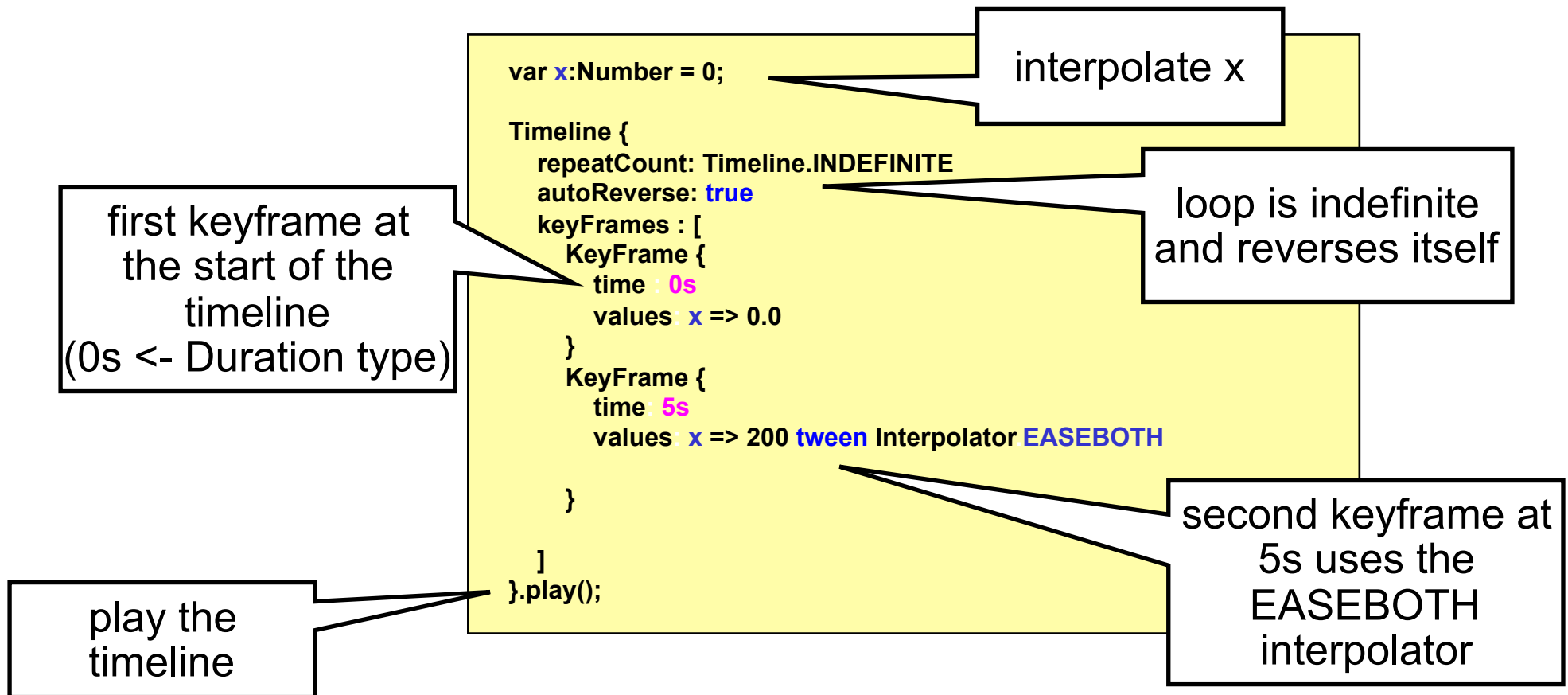


Animation

creating a timeline

To animate an object, a **Timeline** is needed

Within the **Timeline**, **Keyframes** are defined



Animation

binding to the animated value

The interpolated variable can be used like any other variable

```
Stage {
  title : "First Animation"
  scene: Scene {
    width: 200
    height: 200
    content: [
      Circle {
        centerX: bind x
        centerY: 100
        radius: 40
        fill: Color.RED
      }
    ]
  }
}
```

bind to the interpolated variable



Animation

Interpolators

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

Linear: linear interpolation

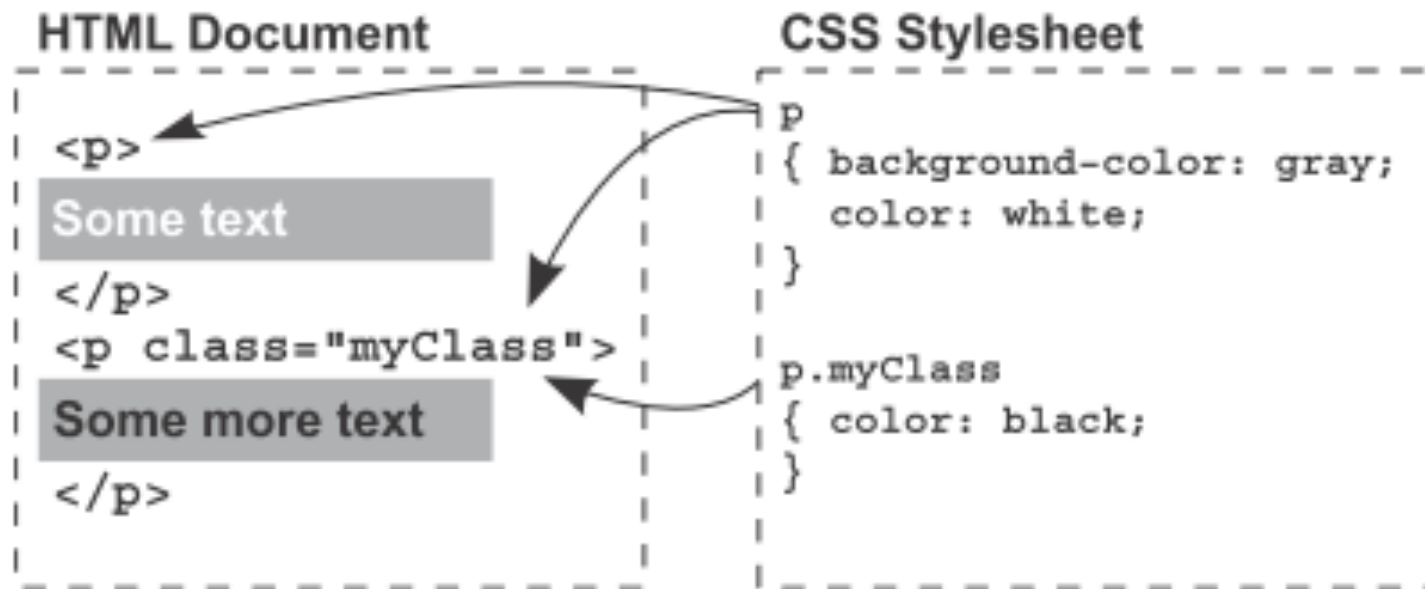
EaseIn: interpolated values smaller at the beginning than linear

EaseOut: smaller in the end

EaseBoth: EaseIn + EaseOut

Stylesheets

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



aus JavaFX in Action (Simon Morris)

Stylesheets



style.css

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

or:

style.css

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



Main.fx

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

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 {  
        id:"Caption"  
        x: 10  
        y: 30  
        content: "Text with Style"  
      }  
      Text {  
        x: 10  
        y: 60  
        content: "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:

```
.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(0,0,0,0.2), 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.)

```
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;  
  }  
};
```

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/