


Multimedia-Programmierung

Übung 3

Ludwig-Maximilians-Universität München
Sommersemester 2010

Today

- Introduction to

A photograph of a man with glasses and a beard, wearing a black t-shirt with the word "Google" in its multi-colored font. He is holding a glass of beer. A speech bubble points to him from the right, containing the text "No more Python :'-(" data-bbox="10 655 320 940"/>

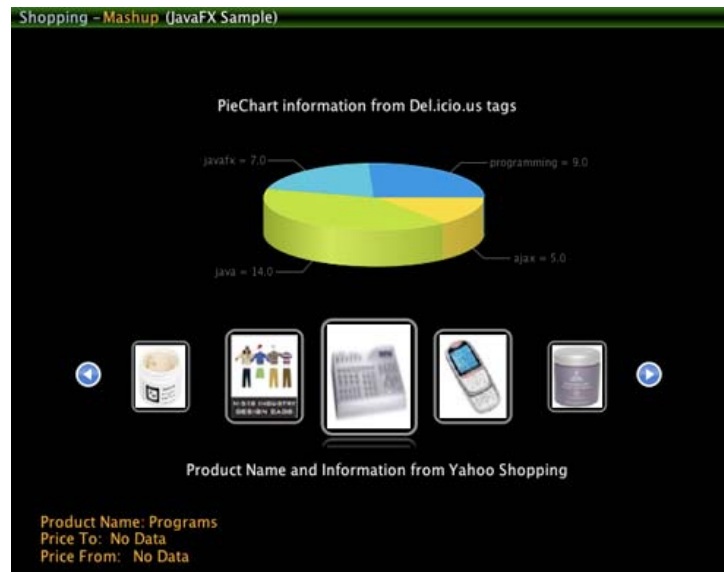
“No more
Python :'-("

For this lecture

- Netbeans IDE 6.8 with JavaFX
- Recommended IDE:
 - Netbeans due to enhanced Python and JavaFX support
<http://www.netbeans.org/>
- Installation:
 - Install Netbeans (e.g. with JavaFX)

What is JavaFX?

- A software platform to develop rich internet applications
- Competitor of Adobe Flash, Microsoft Silverlight, Open Laszlo etc.
- Runs on every device that runs JRE or JavaME (desktop computers, mobile phones, PDAs ...)



© javafx.com



JavaFX Goodies

- Drag-to-install enables quick installation of JavaFX applications and launching them from the hard drive
- Graphic import from common tools (e.g. Photoshop and Illustrator) as well as direct import of SVG graphics
- Runs on Windows, Mac OS, Linux, Solaris, ...
- Advanced development support in Netbeans

JavaFX Overview

- Basics (Variables, Types, Sequences, Classes, Functions)
- Object Literals
- Binding

- UI Programming
 - ⇒ Scene, Stage, Node

- Example

Applications are written in JavaFX Script

- Scripting language for interactive graphical applications
- Object oriented
- Static typing
- Declarative

Define:

```
var a = "world";  
println("hello {a}");
```

Output:

```
hello world
```

Variables

defining variables

- There are two types of variables in JavaFX Script
- Immutable variables defined with the “def” keyword
- Mutable variables defined with the “var” keyword

```
def a = 1;
```

```
var b = 2;
```

```
def a = 4;  
a = 3;
```

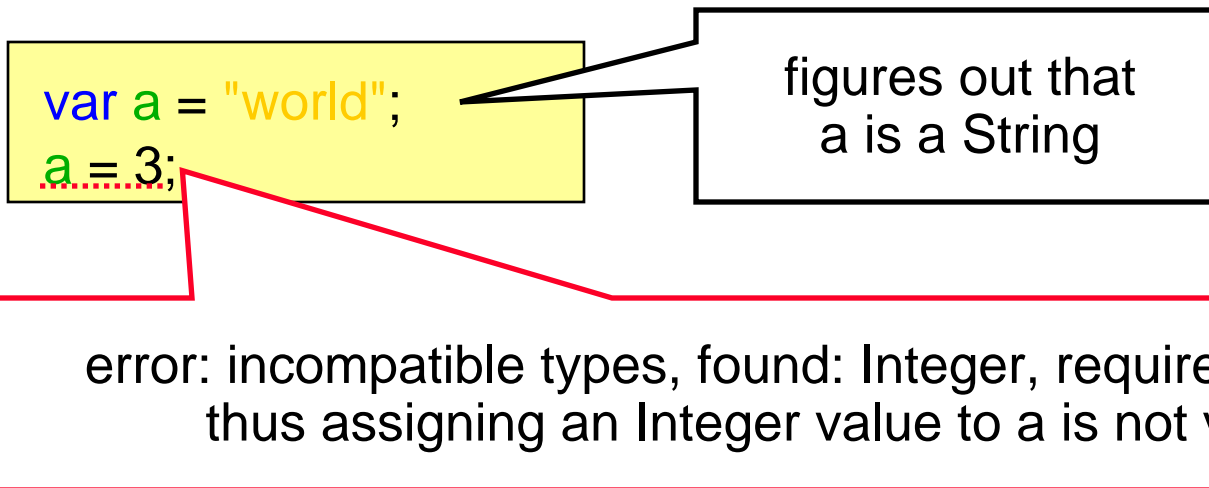
a is defined
immutable

trying to change its value results in an error

Variables

type inference

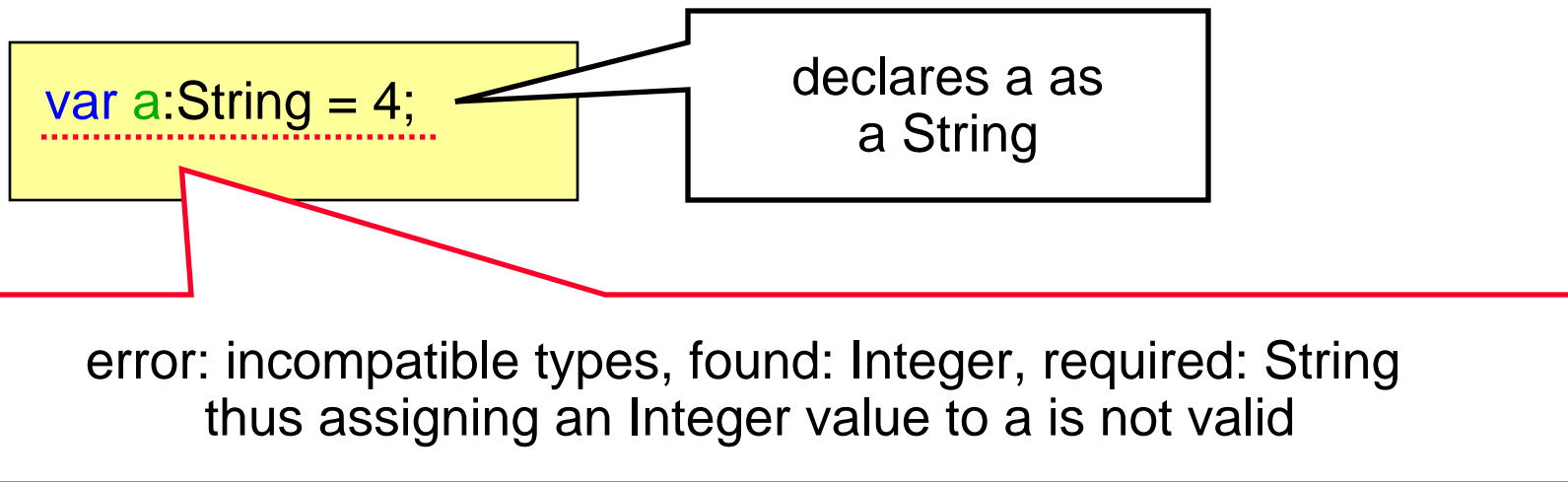
- Compiler “figures out” the type of the variable



Variables

naming types

- But you can always manually define the type if you like



Variables

on replace clause

- Executes any code when a variable is changed
- Optional parameters (e.g. to get the old value)

Define:

```
var a = 0 on replace { println("a is now {a}") }  
  
a = 2;
```

Output:

```
a is now 0  
a is now 2
```

Define:

```
var a = 0 on replace oldA {  
    println("a was {oldA} and is now {a}")  
}  
  
a = 2;
```

Output:

```
a was 0 and is now 0  
a was 0 and is now 2
```

Pseudo-Variables

- Predefined variables for each script
- Immutable (def)
- **__PROFILE__**: either mobile, desktop or browser
- **__FILE__**: URL of the script file
- **__DIR__**: URL of the folder from which the script was loaded

Define:

```
println(__PROFILE__);
```

Output:

```
desktop
```

Types

- Functions, variables and any expression in JavaFX always have a type
- Types have default values (the value it has before it is set)
 - null for user defined classes
 - 0 for Integer
 - 0.0 for Number
 - false for Boolean
 - "" for String
 - Etc.

Define:

```
var a:Integer;  
println(a);
```

Output:

```
0
```

String Type

- Any expression can be embedded into a string using { }

Define:

```
var cool = true;  
var a = "mmp is the {if(cool) "coolest" else "worst"} lecture in the world";  
  
println(a);
```

Output:

```
mmp is the coolest lecture in the world
```

Duration Type

- Built-in type of JavaFX to represent an amount of time
- Default value: 0.0ms
- Units: ms (milliseconds), s, (seconds) m (minutes), h(hours)

Define:

```
var a:Duration;  
println(a);
```

Output:

```
0.0ms
```

Define:

```
var a:Duration = 1m;  
var b:Duration = 20m;  
println(a+b);
```

Output:

```
1260000.0ms
```

Sequences

aka arrays

- Represent a sequence of objects
- Constructed explicitly...

```
var a = [1,2,3];
```

- ... as ranges

```
var a = [1..5];  
var a = [1..10 step 2];
```

...from other sequences

```
var positiveIntegers = [1,2,3,4,5];  
var evenIntegers =  
    positiveIntegers[n | n mod 2 == 0];
```

...from a for loop

```
var lineNumbers:Text[] =  
for(n in [1..100]) {  
    Text { content: "{n}" };  
};
```


Sequences

working with sequences

- Adding elements

```
insert 4 into a;
```

- Insert at a specific position

```
insert 4 before a[1];
```

- Sequence size

```
sizeof a;
```

- Deleting elements

```
delete a[1];
```

- ...

Functions

defining script functions

- Script functions defined loosely in a script (as opposite to within a class)

Define:

last line is considered as the return value

```
function test() {  
  2 + 4;  
}
```

```
println(test());
```

Output:

6

can also be explicitly stated

```
function test():Integer {  
  return 2 + 4;  
}
```

```
println(test());
```

Output:

6

Functions

arguments 1

- Can be defined with or without type
- If no type is used, the type is inferred

Define:

```
function test(a,b) {  
    println(a)  
}  
  
test("a","b");
```

Output:

a

```
function test(a:String,b:String) {  
    println(a)  
}  
  
test("a","b");
```

a

Functions

arguments 2 - an example

```
function test(a,b) {  
  a + b;  
}
```

a and b must be numbers

```
test("a","b");
```

Error: test(a:Double, b:Double) ... cannot be applied to (String,String)

Always better to
explicitly declare
types manually!

Classes

defining classes

Define:

```
class Book {  
    var name : String;  
    var pages: Integer;  
}  
  
def mmp_book = Book {  
    name: "MMP rocks"  
    pages: 1088  
}  
  
println(mmp_book.pages);
```

Output:

1088

definition of
instance variable

object literal is
used to create
an instance of
a JavaFX class

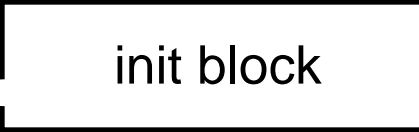
Classes

init block

- The expressions in the init block are executed immediately after the instance is created

Define:

```
class Book {  
    var name : String;  
    var pages: Integer;  
  
    init {  
        println(name);  
    }  
}
```



Classes

instance functions

Define:

```
class Book {  
  var name : String;  
  var pages: Integer;  
  function print_book() {  
    println("Title: {name}, Pages: {pages}");  
  }  
}  
  
def mmp_book = Book {  
  name: "MMP rocks"  
  pages: 1088  
}  
  
mmp_book.print_book();
```

Output:

```
Title: MMP Rocks,  
Pages: 1088
```

definition of an
instance function

Classes

inheritance

Define:

```
class YeahBook extends Book {  
  override function print_book() {  
    super.print_book();  
    println("yeah!");  
  }  
}
```

```
def mmp_book = YeahBook {  
  name: "MMP rocks"  
  pages: 1088  
}
```

```
mmp_book.print_book();
```

Output:

```
Title: MMP Rocks,  
Pages: 1088  
yeah!
```

override the
superclass
function

Modifiers

- Functions, variables and classes can have different modifiers
- **Access modifiers:**
 - **package:** access within its package
 - **protected** (does not apply to classes): within package and from subclasses
 - **public:** can be accessed anywhere
- **var modifiers:**
 - **public-read:** variable can be read anywhere
 - **public-init:** variable can be read and set anywhere
- **Function modifiers:**
 - **abstract:** defines an abstract instance function
 - **bound:** bound function (see later)
 - **override:** used to override a function of a superclass

The run Function

- The run function is the entrance point to a script
- Arguments are passed to a script by the run function

```
function run(args:String[]) {  
    // do something with args[0]  
}
```

- Argument parameter can have any name
- Compiler creates a no-argument run function for scripts that do not provide it and puts all the code in it
- Loose expressions are not allowed in scripts with a run function

Data Binding

or: who needs the observer pattern

- Variables can be bound to expressions
- That is, whenever the expression changes, the variable will be updated accordingly
- Example:

Define:

```
var a = 1;  
var b = 2;  
  
var c = bind a + b;  
println(c);  
a = 2;  
println(c);
```

Output:

```
3  
4
```

Data Binding

binding to function calls

- Binding to a non-bound function only updates the variable if one of the arguments is changed

Define:

```
var b = 3;  
  
function test(a) {  
  return a * b;  
}
```

```
var a = 4;  
var c = bind test(a);  
println(c);  
a = 1;  
println(c);  
b = 1;  
println(c);
```

parameter change
causes update

changing b (used
within the function)
does not cause
an update

Output:

```
12  
3  
3
```

Data Binding

binding to bound functions

- Binding to a non-bound function only updates the variable if one of the arguments is changed

Define:

```
var b = 3;  
  
bound function test(a) {  
  return a * b;  
}
```

```
var a = 4;  
var c = bind test(a);  
println(c);  
a = 1;  
println(c);  
b = 1;  
println(c);
```

bound function

changing b causes
an update now

Output:

```
12  
3  
1
```

Data Binding

binding object literal arguments

Define:

```
var text = "Test";  
class Book {  
  var name : String;  
  function print_book() {  
    println("Title: {name}");  
  }  
}  
  
def mmp_book = Book {  
  name: bind text  
}  
mmp_book.print_book();  
text = "Test2";  
mmp_book.print_book();
```

Output:

```
Title: Test  
Title: Test2
```

UI Programming

- **Attention:** this is only a rough introduction. For more information please refer to the JavaFX tutorial and the API.
- Most UI elements are available in the desktop AND the mobile profile (e.g. user interface elements, charts, colors, text, transformations)
- Effects and Cursors are only available in the desktop profile
- Remember SVG? Defining UIs in JavaFX works similar!

Scene Graph

- JavaFX UIs are based on the scene graph concept
- Defines a hierarchy of graphical objects in a scene
- Tree-like data structure that consists out of nodes
- Nodes in JavaFX can be shapes, ui components, texts etc.
- Some nodes can have child-nodes while others can only be leafs

```
Stage {  
  title: "I am the mighty window."  
  width: 200  
  height: 200  
  scene:Scene {  
    content: [  
      Button { text: "press me" }  
    ]  
  }  
}
```

a boring one but
a scene graph

UI Programming

Theater metaphor:

Stage – Scene - Node



Quelle: Wikimedia Commons – Author:Stephen Moorer

UI Programming

creating a stage

- The class Stage represents a screen (a window for desktop applications)
- Creating a simple window is quite easy

Define:

```
Stage {  
    title: "I am the mighty window."  
}
```

Output:



- <http://java.sun.com/javafx/1.2/docs/api/javafx.stage/javafx.stage.Stage.html>

UI Programming

modifying the stage

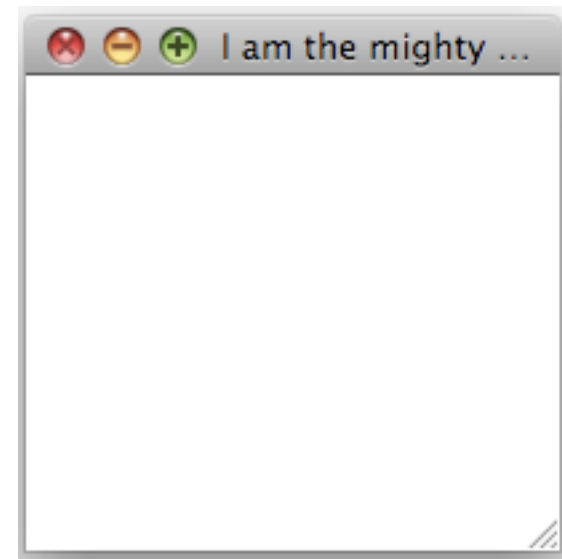
- Stage allows to easily manipulate the stage using the manifold instance variables
- E.g. adding width and height

Define:

```
Stage {  
  title: "I am the mighty window."  
  width: 200  
  height: 200  
}
```

changing the size

Output:



UI Programming

adding a Scene

- A scene holds the different graphical nodes of the UI
- Nodes are places in the **content** sequence
- Example scene holding a button:

Define:

```
Stage {  
  title: "I am the mighty window."  
  width: 200  
  height: 200  
  scene:Scene {  
    content: [  
      Button { text: "press me" }  
    ]  
  }  
}
```

Output:



UI Programming

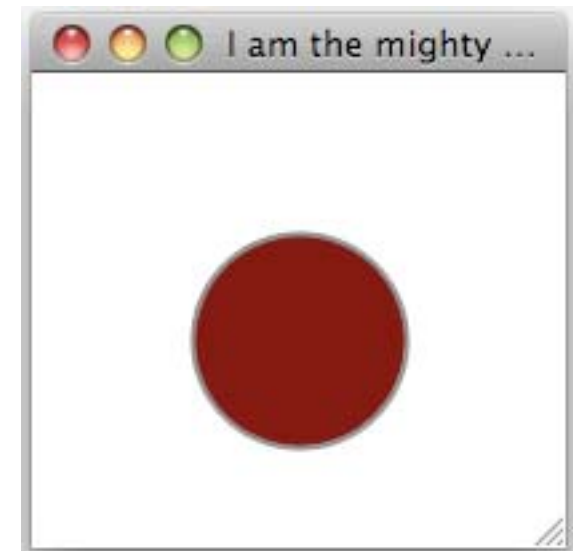
Forget the button, I want a circle!!

- Second example: Adding a circle to the scene.

Define:

```
Stage {  
  ...  
  content: [  
    Circle {  
      centerX: 100  
      centerY: 100  
      radius: 40  
      fill: Color.MAROON  
      stroke: Color.GREY  
      strokeWidth: 2.0  
    }  
  ]  
  ...  
}
```

Output:



UI Programming

loading an image

- Third example: Adding an image to the scene.

Define:

```
Stage {  
  ...  
  content: [  
    ImageView {  
      image:Image {  
        url: "{__DIR__}head.png"  
        width: 100  
        preserveRatio: true  
      }  
    }  
  ]  
  ...  
}
```

Output:

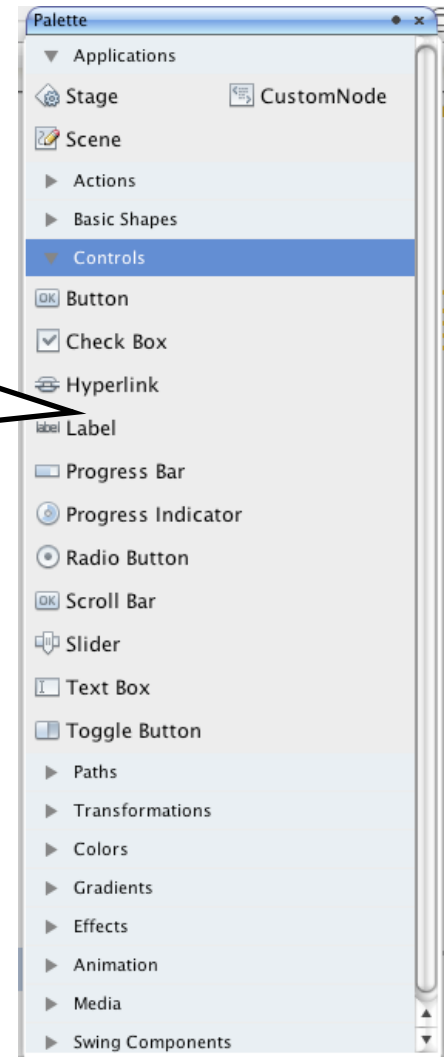


UI Programming

Netbeans your friendly helper

- Netbeans supports a simple drag&drop mechanism to add nodes to the scene graph

just drag the nodes
to wherever you
want to place them
in your code



UI & Data Binding Example

observer for dummies

```
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 "{if(counter<10) "0{counter}" else counter}"
      }
    ]
  }
}
```

if button is pressed,
increase the value of
counter

the content of the text
field is bound to
counter which
automatically updates
the display

UI & Data Binding Example 2

result



Useful Links

- JavaFX Language References
<http://openjfx.java.sun.com/current-build/doc/reference/JavaFXReference.html>
- JavaFX Getting Started
<http://java.sun.com/javafx/1/tutorials/core/getStarted/>
- The JavaFX Tutorial
<http://java.sun.com/javafx/1/tutorials/ui/index.html>
- JavaFX API
<http://java.sun.com/javafx/1.2/docs/api/>