Annotations, Combining scala and Java, GUI Programming

Alexander Rausch

Workshop “The scala programming language„

Department MNI
Professor Dr. Renz
University of applied sciences Gießen-Friedberg

24.01.2010

Overview

Overview

Overview

Overview

Overview

Annotations

What are annotations?

- structured, processable constructs
- add meta informations and other instructions to the code
- control meta programming tools (e.g. code documentation generator, transformer libraries, pretty printer...)
- in the most cases: the compiler knows the annotation syntax, but nothing about their semantic meaning

Annotations

- annotations can be placed on any declarations or definitions: vars, vals, defs, classes, objects, traits and types...
- they always references to the following element in the code

```
1 @deprecated("old stuff: will be kicked in next releases")
2 def calcSomething() =  //...
```

Syntax

@annotClass(exp$_1$, exp$_2$, ...) {val name$_1$=const$_1$, ..., val name$_n$=const$_n$}

Standard annotations

@deprecated(message: String)

- marks a method or class as 'deprecated' / old / not supported in further versions of the code
- the compiler emit a warning whenever code access the 'deprecated' code
- message is optional

Standard annotations

@volatile

- marks a variable as used by multiple threads for shared access
- same behaviour as the Java volatile implementation
- synchronizes all cached copies of variables in the threads with the main copy in the memory

Standard annotations

@serializable

- marks a class for serialization support
- scala doesn't have its own serialization framework (except XML serialization)
- must be implemented by the underlying platform

@SerialVersionUID(uid: Long)

- defines the version of a serializable class

@transient

- marks fields which will not be serialized

Standard Annotations

@BeanProperty

- the compiler automatically generates getter/setter methods for a property
- methods are only available after compiling
- useful for platform frameworks which explicit need setter/getter methods (e.g. Apache BeanUtils)

Standard Annotations

@unchecked

- tells the compiler to ignore the situation that match expressions leave out some cases
- hides the warning: match is not exhaustive! missing combination Orange

```
1 sealed abstract class Fruit
2 case class Apple() extends Fruit
3 case class Orange() extends Fruit
4
5 def foo(f: Fruit): String = (f: @unchecked) match {
6   case Apple() => "a apple"
7   //other cases ignored
8 }
```

Writing own annotations

RetentionPolicy.SOURCE

- the compiler drops the annotation information
- only available before and until the compilation
- inherit from scala.StaticAnnotation
- implement and compile the annotation in Java

RetentionPolicy.RUNTIME

- annotations will be stored in the class file
- implement and compile the annotation in Java
- access the annotation at runtime by using the Java reflection API
- maybe available in next scala versions

Annotations available at runtime

```java
//In Java
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface Author {
   String name();
}
```

Annotations available at compile time

```java
//In Java
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.TYPE)
public @interface Author {
   String name();
}
```

```scala
//In Scala
class Author(val name:String) extends StaticAnnotation
```

Example: Creating own annotations

Example in eclipse

Combining scala and Java

compatibility

- scala is highly compatible with Java
- scala is compiled to Java bytecode
- most features are directly mapped to their Java counterpart
- most framework will work with your scala code!

Differences between scala and Java

- not all language features can be directly mapped into Java!
- what about singleton objects?

```
1 object MyObject {
2   def printNum(number : Int) {
3     println(number)
4   }
5
6   def main(args: Array[String]) {
7     printNum(42)
8   }
9 }
```

Example: scala objects in Java

- $ scalac MyObject
  generate: MyObject$.class and MyObject.class
- output of the tool: javap (The Java Class File Disassembler)

```
1 $ javap MyObject$
2 Compiled from "MyObject.scala"
3 public final class MyObject$ extends java.lang.Object
      implements scala.ScalaObject{
4     public static final MyObject$ MODULE$;
5     public static {};
6     public void printNum(int);
7     public void main(java.lang.String[]);
8 }
```

Example: scala objects in Java

a „pseudo“class with static proxy methods will be created when no class is defined in scala

```
1 $ javap MyObject
2 Compiled from "MyObject.scala"
3 public final class MyObject extends java.lang.Object{
4     public static final void main(java.lang.String[]);
5     public static final void printNum(int);
6 }
```

Example: scala objects in Java

call MyObject from Java

```
1 public class ScalaObjectCall {
2   public static void main(String[] args) {
3     MyObject$.MODULE$.printNum(42);
4    //same as
5     MyObject.printNum(42);
6   }
7 }
```

Traits in Java

- Java has no types for traits
- traits are implemented as abstract Java classes and interfaces

now it's time for decompiling!

Exception handling between scala and Java

- scala doesn't check that thrown exceptions are caught
- there is no **throws** keyword in scala
- the cause is: Java developers often drop and forget to add the throws keyword in their code
- scala automatically throws exceptions until they are caught or the top of the stack is reached

Signaling Java which exception should be caught

```
1 @throws(classOf[Exception])
2 def div (number : Double, divisor : Double) : Double = {
3   if(divisor <= 0.0) throw new Exception("i less than or
        equal 0")
4
5   number / divisor
6 }
```

Working with the Java wildcard operator

- scala doesn't have the Java wildcard **?** operator Iterator<?>
- what should scala do with raw types like List, Iterator?

Existential types

- scala handles wildcard types and raw types with the **existential type**
- they are a full supported part of the language but only useful when working with Java generics

Syntax

type **forSome** {declarations}

Examples

List<?> in scala:
List[T] **forSome** {type T} (this is a List of T's for some type T)
short form: List[_]

GUI Programming

- the **scala.swing** package contains a GUI library which wraps the Java Swing framework
- the goal is to keep the GUI code short and hide complexity

A simple „hello world“GUI

```
1 object SimpleHelloWorldGUI extends SimpleSwingApplication {
2   def top = new MainFrame() {
3     title = "Hello World GUI"
4
5     contents = new Button("hello world")
6
7     size = new Dimension(250,250)
8   }
9 }
```

scala's class SwingApplication

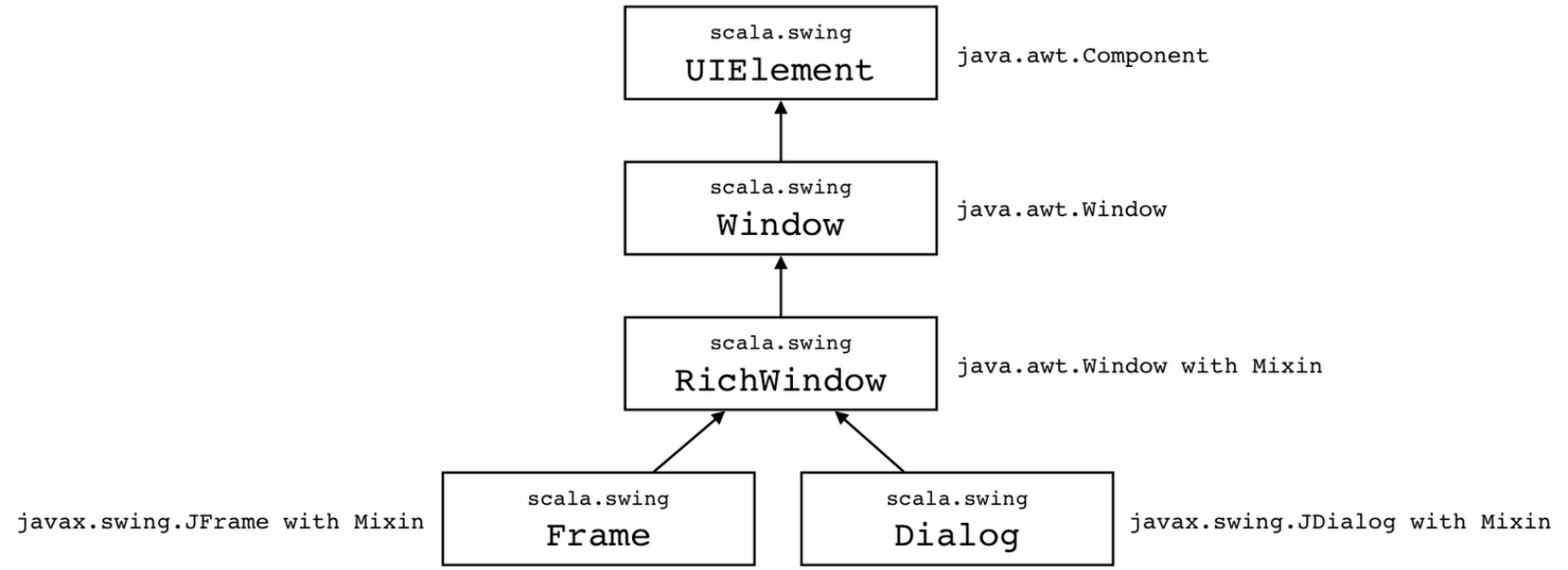
```
1 abstract class SwingApplication extends Reactor {
2   def main(args: Array[String]) = Swing.onEDT { startup(args
        ) }
3
4   def startup(args: Array[String])
5   def quit() { shutdown(); System.exit(0) }
6   def shutdown() {}
7 }
```

scala's class SimpleSwingApplication

```
1 abstract class SimpleSwingApplication extends
      SwingApplication {
2   def top: Frame
3
4   override def startup(args: Array[String]) {
5     val t = top
6     if (t.size == new Dimension(0,0)) t.pack()
7     t.visible = true
8   }
9 }
```

- the class SimpleSwingApplication is a default implementation
- client must only implement a method **top() : Frame** to get started

scala Swing class hierarchy (excerpt)



source: http://ingomaier.blogspot.com/2010/11/scalaswing-package-in-28-and-beyond.html

scala use the proxy pattern to forward method calls to the Swing library and add additional code

MainFrame

MainFrame

- subclass of Frame - shutdown the framework and close application
- title - set the frame (window) title
- size - set the size of the frame
- contents - contains the children component

Dialog

- independent subwindow of the main window (which is in most cases a Frame)
- useful for error/information messages, input dialogs or custom wizards
- scala companion object **Dialog** gives access to the standard dialogs like JOptionPane in Java

Panels & Layouts

Panel

- structuring component on the top of a Frame or Dialog
- contains other GUI elements like Label, Text and Button...
- each Panel has a specific layout and a own class!

Layouts

- FlowPanel
- BoxPanel
- BorderPanel
- GridBagPanel
- GridPanel
- important properties: contents, border

What about the other GUI components?

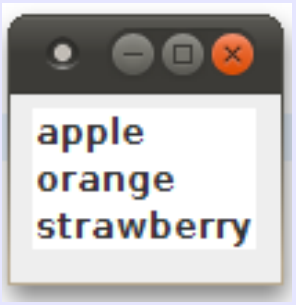
- remove the prefix **„J“** to get the scala name of the Swing class
- e.g Swing component JTextField is called TextField in scala

Example ListView

ListView

- component that shows elements of a List
- cells are not editable (that's possible in a Table)
- companion object ListView holds helper object for rendering cells

Example ListView



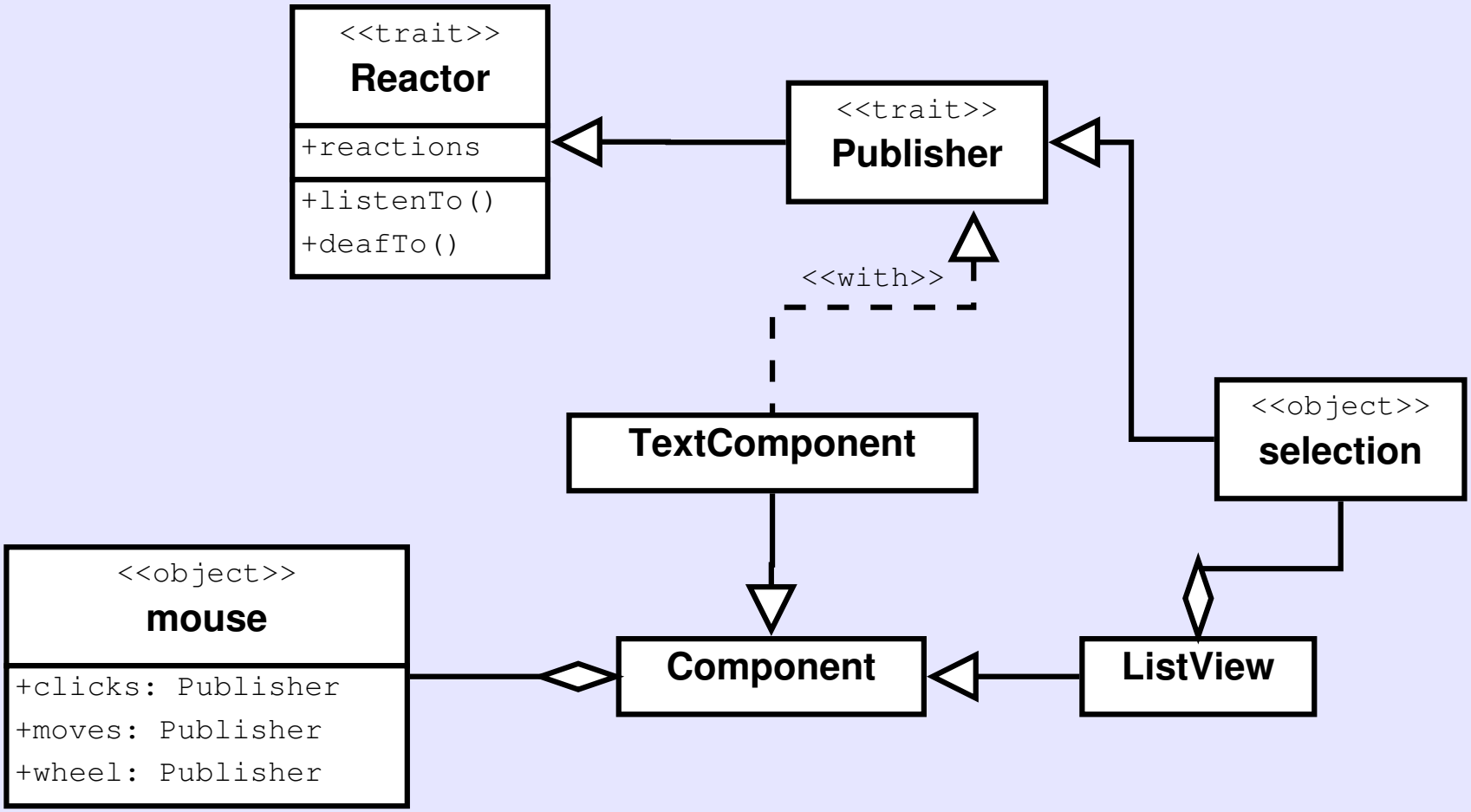
Render the name property of a case class Fruit

```
1 case class Fruit(name: String)
2
3 val items = List(Fruit("apple"), Fruit("orange"), Fruit("
      strawberry"))
4 val listView = new ListView(items) {
5     renderer = ListView.Renderer(_.name)
6 }
```

Event handling

- scala swing components implement the Reactor pattern
- a parent component can listen to events fired by their children
- listenTo(p: Publisher) - register a listener for all events
- deafTo(p: Publisher) - deregister a listener for all events
- each event is a case class:
  (e.g. case class ButtonClicked(source: Button))
- events can be caught by match the correct event class

Event handling hierarchy (excerpt)



Register listener to button events

```
1 def top = new MainFrame() {
2   val btn = new Button("some cool function")
3   listenTo(btn)
4 }
```

React on events

```
1 def top = new MainFrame() {
2   val btn = new Button("some cool function")
3   listenTo(btn)
4
5   reactions += {
6     case ButtonClicked(`btn`) =>
7       println("btn clicked!")
8   }
9 }
```

Mouse & keyboard events

- for performance reasons the mouse & keyboard events are not automatically published by the controls
- they must be registered on the mouse and keys Publisher property manually!

```
1 def top = new MainFrame() {
2   val btn = new Button("some cool function")
3
4   //register for mouse move event
5   listenTo(btn.mouse.moves)
6
7   //register for all keyboard events
8   listenTo(btn.keys)
9 }
```

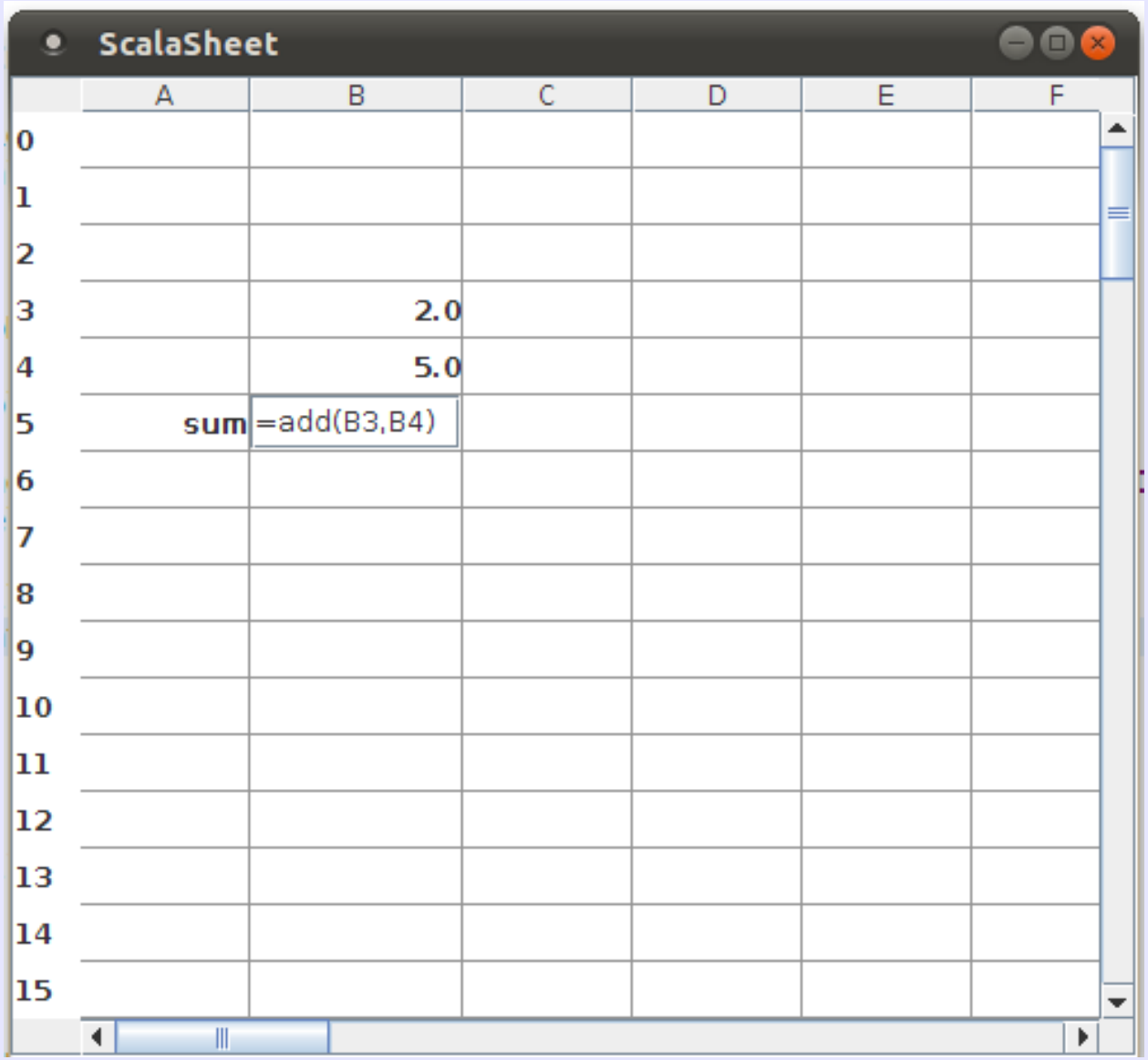
Additional

- at the moment there is no scala Swing GUI designer available
- a SWT library is available:
  ScalaSWT `https://github.com/rodant/ScalaSWT`
- but last commit was in 2009 :(

Spreadsheet Application

- SCells is a spreadsheet application with less than 200 lines of code
- rows are identified by a number, columns by chars from A to Z
- one cell can contain a number, formula or text
- a formula has the syntax: =add(1,1) or =add(A1,B1)

Example: SCells



Thank you for your attention!

Are there any questions?