

SCALA: CLASSES & TRAITS

Fortgeschrittene Anwendungen funktionaler Programmierung
Sommersemester 2010

Fynn Feldpausch

Universität Bremen, Fachbereich 3

11. Mai 2010

Classes

- Klasse sind in Scala wie in Java, nur kürzer :)

- **Java:**

```
class MyClass {  
    private int index;  
    private String name;  
  
    public MyClass(int index, String name) {  
        this.index = index;  
        this.name = name;  
    }  
}
```

- **Scala:**

```
class MyClass(index: Int, name: String)
```

- Bedeutung von **var** und **val** beachten

Traits

Definition

- **traits** können Methoden (konkret & abstrakt) und Attribute enthalten
- leiten (wenn nicht anders angegeben) von **AnyRef** ab
- Definition durch Schlüsselwort **trait**

```
trait SayHello {  
  def sayHello() {  
    println("Hello World!")  
  }  
}
```

- nach der Definition können sie in Klassen *eingemischt* werden

Traits

Einmischen

- **traits** werden mittels **extends** oder **with** in bestehende Klassen eingemischt
- mit **extends** leitet die Klasse von der Superklasse des **traits** ab
- mit **with** lassen sich beliebig viele **traits** in eine Klasse einmischen

```
class MyClass {  
  \\...  
}  
class MyClass2 extends SayHello {  
  \\...  
}  
  
val myClass = new MyClass with SayHello  
val myClass2 = new MyClass2  
  
myClass.sayHello() \\\"Hello World!\"
```



Traits

Thin vs. rich interfaces

- **Thin interface:** nur wenige Methoden
 - + *implementer* muss weniger Code schreiben
 - *client* muss mehr Code schreiben, da nur wenige (und damit oft nicht optimale) Methoden bereit stehen
- **Rich interface:** sehr viele Methoden
 - + *client* kann sich die für ihn passende Methode aussuchen
 - *implementer* muss viel Code schreiben
- **traits** beheben dieses Problem und bieten eine Kombination der Vorteile
- schließlich können neben abstrakten Methoden auch konkrete Methoden implementiert werden

Traits

Stackable modifications

- **traits** können bestehende Methoden einer Klasse modifizieren
- dabei können verschiedene **traits** gestapelt werden
- alle eingemischten **traits** kommen dann zur Ausführung

- Konzept ähnelt dem der multiplen Vererbung
- entscheidender Unterschied: linearization
- **traits** werden linearisiert, so dass sie eine feste Ableitungsreihenfolge besitzen

```
class Animal
trait Furry extends Animal
trait HasLegs extends Animal
trait FourLegged extends HasLegs
class Cat extends Animal with Furry with FourLegged
```

⇒ Cat → FourLegged → HasLegs → Furry → Animal → AnyRef → Any

BEISPIEL 1: Rectangular

■ Klasse Point:

```
class Point(val x: Int, val y: Int)
```

■ Trait Rectangular:

```
trait Rectangular {  
  def topLeft: Point  
  def bottomRight: Point  
  
  def left = topLeft.x  
  def right = bottomRight.x  
  def width = right - left  
  // and many more geometric methods...  
}
```

BEISPIEL 1: Rectangular

■ Klasse Rectangle:

```
class Rectangle(  
  val topLeft: Point,  
  val bottomRight: Point)  
extends Rectangular
```

- für `new Rectangle(new Point(0,0), new Point(10,10))` existieren nun Attribute...

```
... rect.left = 0
```

```
... rect.right = 10
```

EXAMPLE 2: Stackable

■ Klasse IntQueue:

```
abstract class IntQueue {  
  def get(): Int  
  def put(x: Int)  
}
```

■ Klasse BasicIntQueue:

```
import scala.collection.mutable.ArrayBuffer  
  
class BasicIntQueue extends IntQueue {  
  private val buf = new ArrayBuffer[Int]  
  def get() = buf.remove(0)  
  def put(x: Int) {  
    buf += x;  
  }  
}
```

EXAMPLE 2: Stackable

■ Trait Doubling:

```
trait Doubling extends IntQueue {  
  abstract override def put(x: Int) {  
    super.put(2*x)  
  }  
}
```

■ Trait Incrementing:

```
trait Incrementing extends IntQueue {  
  abstract override def put(x: Int) {  
    super.put(x + 1)  
  }  
}
```

EXAMPLE 2: Stackable

- Testprogramm liefert 21 & 41:

```
val queue = new BasicIntQueue
  with Incrementing with Doubling
queue.put(10)
queue.put(20)
println(queue.get())
println(queue.get())
```

- **Vorsicht:** Die Reihenfolge des Einmischens ist relevant!
- `new BasicIntQueue with Doubling with Incrementing` liefert die Werte 22 & 42

Übung

1. Welche Aspekte von **traits** könnten für unser Projekt besonders interessant sein? Wo kommen sie zum Einsatz?
2. **trait**-Übungen mit Bezug auf Monaden in Scala:
<http://blog.tmorris.net/monad-exercises-in-scala/>
 - » ListMonad...
 - » OptionMonad...
 - » IdentityMonad...