



Scala

Stefan Heinemann

08. November 2011

Inhalt

Einleitung

Hallo Welt

Objektorientiert

Funktionale Programmierung

Typinferenz

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar
- Scriptsprache

Einleitung

1. OO geht so weit, dass es keine primitiven gibt
2. Funktional erklären!
3. Elegantere Problemlösung

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar
- Scriptsprache

- objektorientiert und funktional
- statisch typisiert
- Cross-Platform
- Integration mit JVM
- Kompilierbar
- Scriptsprache

Einleitung

1. Variablen können nur einen bestimmten Wert annehmen

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar
- Scriptsprache

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar
- Scriptsprache

Einleitung

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar
- Scriptsprache

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar
- Scriptsprache

Einleitung

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar
- Scriptsprache

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar

Einleitung

- ▶ objektorientiert und funktional
- ▶ statisch typisiert
- ▶ Bytecode → JVM
- ▶ Integration mit Java
- ▶ Kompilierbar
- ▶ *Scriptsprache*

- objektorientiert und funktional
- statisch typisiert
- Bytecode → JVM
- Integration mit Java
- Kompilierbar
- Scriptsprache

Einleitung

- ▶ objektorientiert und funktional
- ▶ statisch typisiert
- ▶ Bytecode → JVM
- ▶ Integration mit Java
- ▶ Kompilierbar
- ▶ Scriptsprache

└─ Hallo Welt

└─ Hallo Welt

Hallo Welt

```
1 object HelloWorld {  
2   def main(args: Array[String]) {  
3     println("Hello World!")  
4   }  
5 }
```

Hallo Welt

1. Obligatorisches Hallo Welt
2. Unvermeidlich, Code zu zeigen
3. object = Singleton-Objekt, keine statischen Sachen
4. Singleton wird beim ersten Aufruf erzeugt
5. Syntax ähnlich wie Java
6. Datentypen werden hinter die Deklaration gestellt
7. Main-Methode wie in Java

```
1 object HelloWorld {  
2   def main(args: Array[String]) {  
3     println("Hello World!")  
4   }  
5 }
```

```
1 class foo(var x:Int, y:Int) extends bar(y) {  
2   if (y > 0) {  
3     throw new Exception("Invalid Argument");  
4   }  
5  
6   var member:String = "Ich bin drin!"  
7  
8   override def toString(): String =  
9     "foo says hello"  
10  
11   private def doSomething() {  
12     println("something stupid")  
13   }  
14 }
```

1. Da Objektorientierte Programmierung bekannt, nicht interessant das ganze zu erklären, darum ein Beispiel
2. Erklären was wo ist
3. Abstract classes

Klassen

```
1 class foo(var x:Int, y:Int) extends bar(y) {  
2   if (y > 0) {  
3     throw new Exception("Invalid Argument");  
4   }  
5  
6   var member:String = "Ich bin drin!"  
7  
8   override def toString(): String =  
9     "foo says hello"  
10  
11   private def doSomething() {  
12     println("something stupid")  
13   }  
14 }
```

Scala
└─ Objektorientiert
 └─ Traits

Traits

- Keine Interfaces
- Keine Mehrfachvererbung

Traits

- Keine Interfaces
- Keine Mehrfachvererbung

Scala
└─ Objektorientiert
 └─ Traits

Traits

- Keine Interfaces
- Keine Mehrfachvererbung

Traits

- Keine Interfaces
- Keine Mehrfachvererbung

Scala
└─ Objektorientiert
 └─ Traits

Traits

- ▶ Ähnlich wie Interfaces
- ▶ Können aber Implementierungen enthalten
- ▶ Können mehrfach an Klassen weitervererbt werden
- ▶ Weniger problematisch als Mehrfachvererbung

Traits

- ▶ Ähnlich wie Interfaces
- ▶ Können aber Implementierungen enthalten
- ▶ Können mehrfach an Klassen weitervererbt werden
- ▶ Weniger problematisch als Mehrfachvererbung

Scala
└─ Objektorientiert
 └─ Traits

Traits

- ▶ Ähnlich wie Interfaces
- ▶ Können aber Implementationen enthalten
- ▶ Können mehrfach an Klassen weitervererbt werden
- ▶ Weniger problematisch als Mehrfachvererbung

Traits

1. Ähnlich zu abstrakte Klassen
2. Abstrakte Klassen können nicht mehrfach vererbt werden

- ▶ Ähnlich wie Interfaces
- ▶ Können aber Implementationen enthalten
- ▶ Können mehrfach an Klassen weitervererbt werden
- ▶ Weniger problematisch als Mehrfachvererbung

Scala
└─ Objektorientiert
 └─ Traits

Traits

- ▶ Ähnlich wie Interfaces
- ▶ Können aber Implementationen enthalten
- ▶ Können mehrfach an Klassen weitervererbt werden
- ▶ Weniger problematisch als Mehrfachvererbung

Traits

- ▶ Ähnlich wie Interfaces
- ▶ Können aber Implementationen enthalten
- ▶ Können mehrfach an Klassen weitervererbt werden
- ▶ Weniger problematisch als Mehrfachvererbung

Scala
└─ Objektorientiert
 └─ Traits

Traits

- ▶ Ähnlich wie Interfaces
- ▶ Können aber Implementationen enthalten
- ▶ Können mehrfach an Klassen weitervererbt werden
- ▶ Weniger problematisch als Mehrfachvererbung

Traits

- ▶ Ähnlich wie Interfaces
- ▶ Können aber Implementationen enthalten
- ▶ Können mehrfach an Klassen weitervererbt werden
- ▶ Weniger problematisch als Mehrfachvererbung

- ▶ Funktionen sind auch Objekte
- ▶ Speichern in Variablen
- ▶ Übergeben an Funktionen
- ▶ Rückgabewerte
- ▶ Callbacks
- ▶ Anonym möglich

Funktionsobjekte

- ▶ Funktionen sind auch Objekte
- ▶ Speichern in Variablen
- ▶ Übergeben an Funktionen
- ▶ Rückgabewerte
- ▶ Callbacks
- ▶ Anonym möglich

- Funktionen sind auch Objekte
- Speichern in Variablen
- Übergeben an Funktionen
- Rückgabewerte
- Callbacks
- Anonym möglich

Funktionsobjekte

- ▶ Funktionen sind auch Objekte
- ▶ Speichern in Variablen
- ▶ Übergeben an Funktionen
- ▶ Rückgabewerte
- ▶ Callbacks
- ▶ Anonym möglich

- Funktionen sind auch Objekte
- Speichern in Variablen
- Übergeben an Funktionen
- Rückgabewerte
- Callbacks
- Anonym möglich

Funktionsobjekte

- ▶ Funktionen sind auch Objekte
- ▶ Speichern in Variablen
- ▶ Übergeben an Funktionen
- ▶ Rückgabewerte
- ▶ Callbacks
- ▶ Anonym möglich

- Funktionen sind auch Objekte
- Speichern in Variablen
- Übergeben an Funktionen
- Rückgabewerte
- Callbacks
- Anonym möglich

Funktionsobjekte

- ▶ Funktionen sind auch Objekte
- ▶ Speichern in Variablen
- ▶ Übergeben an Funktionen
- ▶ Rückgabewerte
- ▶ Callbacks
- ▶ Anonym möglich

- Funktionen sind auch Objekte
- Speichern in Variablen
- Übergeben an Funktionen
- Rückgabewerte
- Callbacks

1. in Java ist dies nicht möglich
2. Callbacks mit anonymen Klassen

Funktionsobjekte

- ▶ Funktionen sind auch Objekte
- ▶ Speichern in Variablen
- ▶ Übergeben an Funktionen
- ▶ Rückgabewerte
- ▶ Callbacks
- ▶ Anonym möglich

- Funktionen sind auch Objekte
- Speichern in Variablen
- Übergeben an Funktionen
- Rückgabewerte
- Callbacks
- Anonym möglich

Funktionsobjekte

- ▶ Funktionen sind auch Objekte
- ▶ Speichern in Variablen
- ▶ Übergeben an Funktionen
- ▶ Rückgabewerte
- ▶ Callbacks
- ▶ Anonym möglich

```
1 def describe (x: Any) = x match {  
2   case 5      => "five"  
3   case true   => "truth"  
4   case "hello" => "hi!"  
5   case Nil    => "the empty list"  
6   case _      => "something else" // Default  
7 }
```

Pattern Matching

```
1 def describe (x: Any) = x match {  
2   case 5      => "five"  
3   case true   => "truth"  
4   case "hello" => "hi!"  
5   case Nil    => "the empty list"  
6   case _      => "something else" // Default  
7 }
```

- Typ kann explizit angegeben werden
- Typ kann manchmal vom Kontext her abgeleitet werden
- Kürzerer Code

Typinferenz

- Typ kann explizit angegeben werden
- Typ kann manchmal vom Kontext her abgeleitet werden
- Kürzerer Code

Typinferenz

- Typ kann explizit angegeben werden
- Typ kann manchmal vom Kontext her abgeleitet werden

Typinferenz

- ▶ Typ kann explizit angegeben werden
- ▶ Typ kann manchmal vom Kontext her abgeleitet werden
- ▶ Kürzerer Code

Scala
└─ Typinferenz
 └─ Typinferenz

Typinferenz

- Typ kann explizit angegeben werden
- Typ kann manchmal vom Kontext her abgeleitet werden
- Kürzerer Code

Typinferenz

- ▶ Typ kann explizit angegeben werden
- ▶ Typ kann manchmal vom Kontext her abgeleitet werden
- ▶ Kürzerer Code

```
1 var x = "hallo"
```

Typinferenz

```
1 var x = "hallo"
```

2011-11-08

Scala
└─ Fragen
 └─ Fragen

Fragen?

Fragen

Fragen

Fragen?