Orientation in Objects

Einbindung dynamischer Sprachen in Java

The Beast in Me?

Steffen Schluff

Papick Garcia Taboada

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Version: 1.0

Gliederung

Orientation in Objects

- Einleitung
- Skriptsprachen in a Nutshell
- Technische Anbindung in Java
- Einsatzszenarien
- Zusammenfassung

Gliederung

Orientation in Objects

- Einleitung
- Skriptsprachen in a Nutshell
- Technische Anbindung in Java
- Einsatzszenarien
- Zusammenfassung

Kennen Sie Java?

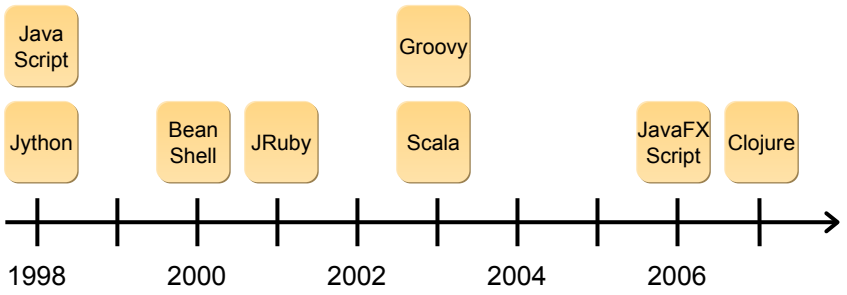
Orientation in Objects

[..] there are four completely different things that go by the name Java

- Language
- Enormous class library
- Virtual machine
- Security model

(Jamie Zawinski, „java sucks.“, 2000)

JVM Sprachen (Auszug)



Klappe, die erste!

- JVM Sprachen werden häufig als Skriptsprachen bezeichnet
  - Beispiel: JSR 223 - Scripting for the Java Platform
- „Skript“ ermöglicht Steuerung und Kontrolle eines „Programms“
  - Häufig interpretiert oder mit vernachlässigbarem Kompilierschritt
- Begriffe Skript und Programm heute schwer unterscheidbar
  - Eigene Benutzer sagen nicht Skriptsprache
- Skriptsprachen werden mit bestimmten Eigenschaften assoziiert
  - Vor allem Speicherverwaltung und dynamische Typisierung

Du bist einfach nicht mein Typ...

Orientation in Objects

- Statisch typisierte Sprachen
  - Überprüfung von Typinformationen bereits zur Compilezeit
  - Ermöglicht frühes Finden von Fehlern im Typsystem
- Dynamisch typisierte Sprachen
  - Überprüfung von Typinformationen nur zur Laufzeit
  - Ermöglicht größere Flexibilität
- Beide Varianten haben Vor- und Nachteile
  - Gegenstand zahlreicher, teils akademischer, Diskussionen

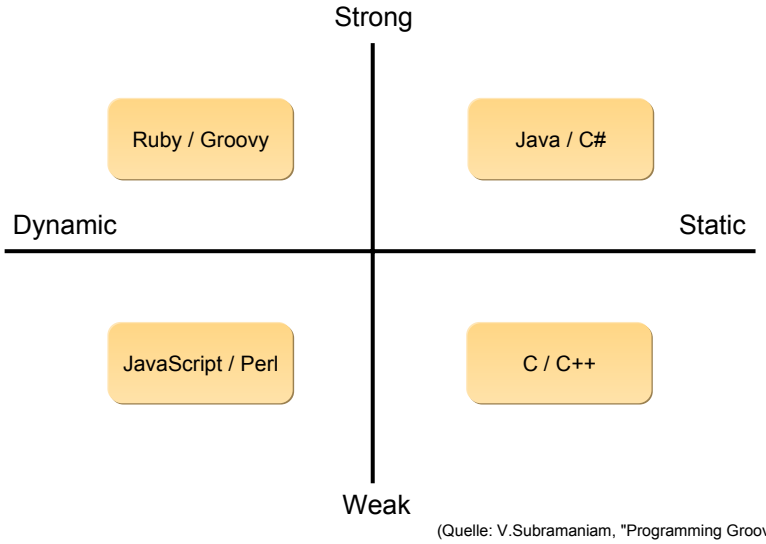
Dynamic Typing != Weak Typing (1)

Orientation in Objects

- Stichwort „Weak Typing vs. Strong Typing“
  - Dynamisch typisiert heißt nicht Typinformation zu ignorieren
- „[]... usage [...] is so various as to render them almost useless.“
  - Benjamin Pierce, Autor „Types and Programming Languages“

```
~ $python
[...]
>>> i = 5
>>> s = "Hello"
>>> i + s
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Dynamic Typing != Weak Typing (2)



(Quelle: V.Subramaniam, "Programming Groovy“)

Ente gut, alles gut

- Ducktyping und dynamische Typisierung
  - „If it walks like a duck and quacks like a duck, I would call it a duck.“
  - Semantik eines Objekts durch Methoden und nicht durch Typ
- Dynamische Sprachen
  - Besitzen Laufzeitfähigkeiten, die sonst typisch für Compilezeit sind
  - Ändern von Objekten zur Laufzeit, Evaluierung von Quellcode, ...
- Vorsicht, Falle!
  - „Dynamic languages and dynamic typing are not identical concepts...“
  - Vorgestellte Begriffe ungenau in Bedeutung oder Verwendung

„We got both kinds. We got Country and Western.“

Orientation in Objects

- Viele dynamisch typisierte Sprachen sind älter als Java...
  - ... und die JVM Portierungen sind auch nicht neu
- Renaissance durch Web Frameworks als „Killer Apps“
  - Ruby on Rails, Grails, Django, ...
- Wiederentdecken für andere isolierte Zwecke
  - Shell Skripte, Buildmanagement, Testen, ...
- Einbindung in bestehende Java Anwendungen
  - Neues Steinchen im Java Architektur Baukasten
  - Zum Steuern und Kontrollieren, d.h. Scripting

Wo tut sich was?

Orientation in Objects

- Proprietäre Mechanismen der einzelnen Sprachen
- Apache Bean Scripting Framework (BSF)
- Scripting Support in Spring
- Java Community Process Scripting JSRs
  - JSR 223 - Scripting for the Java Platform
  - JSR 241 - Groovy Programming Language
  - JSR 274 - BeanShell Scripting Language
  - JSR 292 - Dynamically Typed Language Support
- Hauptentwickler von Jython und JRuby arbeiten bei Sun
  - Gleiches gilt für IronPython und Microsoft

Gliederung

Orientation in Objects

- Einleitung
- Skriptsprachen in a Nutshell
- Technische Anbindung in Java
- Einsatzszenarien
- Zusammenfassung

Kurzer Rundflug

Orientation in Objects

- Keine Einführung
- Jede Sprache ein bisschen anders
- Hier nur ein Paar Highlights

Javascript

- Sprachkern als ECMAScript (ECMA 262) standardisiert
- Dank Einsatz in Browsern sehr weit verbreitet
  - Zugriff auf DOM und Eigenschaften des Browsers
- Wie Java ist auch JavaScript in der Syntax „C“-Ähnlich“
- Objektorientierung nicht mit Klassen sondern mit Prototypen realisiert

```
var object1={a:1,b:2};
function child(){
  this.c=3;
  this.d=4
};
child.prototype=object1;
var object2=new child();
alert(object2.a); // zeigt den Wert 1 an
alert(object2.c); // zeigt den Wert 3 an
```

Wikipedia: http://de.wikipedia.org/wiki/JavaScript

Groovy

- Dynamische Programmiersprache für die JVM
- Java-Syntax und Ruby-Konzepte Mashup
- Features:
  - Closures,
  - native Syntax für Maps,
  - Listen und Reguläre Ausdrücke,
  - einfaches Templatesystem
    - **HTML und SQL-Code Generierung**
  - XQuery-ähnliche Syntax zum Ablaufen von Objektbäumen,
  - Operatorüberladung
  - native Darstellung für BigDecimal und BigInteger.

Wikipedia: http://de.wikipedia.org/wiki/Groovy

Groovy...

Orientation in Objects

```
class Greet {
  def name
  Greet(who) { name = who[0].toUpperCase() +
                      who[1..-1] }
  def salute() { println "Hello $name!" }
}

g = new Greet('world')  // create object
g.salute()              // Output "Hello World!"
```

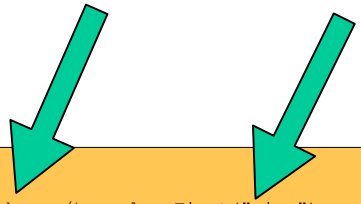
```
import static org.apache.commons.lang.WordUtils.*

class Greeter extends Greet {
  Greeter(who) { name = capitalize(who) }
}

new Greeter('world').salute()
```


Groovy RegExp...

Orientation in Objects



```
// lets do some replacement

def cheese = ("cheesecheese" =~ /cheese/).replaceFirst("nice")

assert cheese == "nicecheese"
```

Java und keine Closures

Orientation in Objects

```
package example.math;

public class MyMath {
   public static int square(int numberToSquare){
        return numberToSquare * numberToSquare;
   }
}
```

```
import example.math.MyMath;

int x, y;
x = 2;
y = MyMath.square(x); // y will equal 4.
```


Groovy Closures

Orientation in Objects

```
import example.math.MyMath;

int x, y;
x = 2;
y = MyMathdef x = 2

// define closure and assign it to variable 'c'
def c = { numberToSquare -> numberToSquare * numberToSquare }

// using 'c' as the identifer for the closure,
// make a call on that closure

def y = c(x)          // shorthand form for applying closure,
                       // y will equal 4
```

Jython

Orientation in Objects

- The language formerly known as JPython ;-)
- Python Implementierung
  - Paradigmenschlacht: Objektorientiert, Aspektorientiert und Funktional
- Strukturierung durch Einrücken

```
def fakultaet(x):
    if x > 1:
        return x * fakultaet(x - 1)
    else:
        return 1
```

- Große Bibliothek
  - Auf Internetanwendungen zugeschnitten

Wikipedia: http://de.wikipedia.org/wiki/Jython

Jython und Listen...

Orientation in Objects

```
zahlen = [1, 2, 3, 4, 5]

zweierpotenzen = [2 ** n for n in zahlen]
```

Wikipedia: http://de.wikipedia.org/wiki/Jython

Was mögen wir bei Python

Orientation in Objects

- http://wiki.python.de/Python
  - es macht Spaß, damit zu programmieren - weil es so einfach und direkt ist
  - klare, lesbare Syntax
  - sehr umfangreiche Standard-Bibliothek
  - hohe Produktivität: wenig Code notwendig durch die Mächtigkeit der Sprache/Bibliothek
  - man kann sich auf das Problem konzentrieren, statt sich mit Eigenheiten/Defiziten der Sprache rumzuschlagen

Von dict erben....

Orientation in Objects

```
-*- coding: utf-8 -*-

class MeinDict(dict):
    def __init__(self, *args, **kwargs):
        """
        Die ursprüngliche Methode kann man über super() ansprechen. Allerdings
        hat das den Nachteil, dass man den verwendeten Klassennamen einfügen
        muss. Wenn man die Klasse umbenennt muss man alle super()-Aufrufe
        abändern.
        """
        print "init..."
        super(MeinDict, self).__init__(*args, **kwargs)

    def __setitem__(self, key, value):
        """
        Ohne super() kann man die geerbte Method über das 'dict'-Objekt
        ansprechen.
        """
        print "Neuer Eintrag %s = %s" % (key, value)
        dict.__setitem__(self, key, value)
```

JRuby

Orientation in Objects

- Ruby implementierung für die JVM
- Sehr bekannt geworden durch
  - Ruby on Rails
  - JRuby on Rails
- Flexibel
- Sichtbarkeit
  - var, @var, $var
- Objektorientiert

```
5.times { print "Wir *lieben* Ruby -- es ist
                          ungeheuerlich!" }
```

Wikipedia: http://de.wikipedia.org/wiki/JRuby

Gliederung

Orientation in Objects

- Einleitung
- Skriptsprachen in a Nutshell
- Technische Anbindung in Java
- Einsatzszenarien
- Zusammenfassung

Proprietäre Ansätze

Orientation in Objects

- Jede Implementierung einer Programmiersprache bietet unterschiedliche Arten der Nutzung
  - z. Bsp. Einbindung über API
  - z. Bsp. Einbindung über Compiler
- Ansätze sind Proprietär -> Sprachengines nicht einfach austauschbar

Spring und Dynamic Languages

Orientation in Objects

- Dynamic Language Support
  - Seit Version 2.0
  - Es werden wenige Sprachen unterstützt
    - **Jruby**
    - **Groovy**
    - **BeanShell**
  - Eine Anbindung über das neue JSR ist nicht vorhanden
    - **Keine Nachfrage?**

<lang:language />

Orientation in Objects

- Bean-Implementierung...
  - ... liegt in eigene Datei
    - **„Refreshable Bean support“**
  - ... wird „inline“ in der Bean-Definition eingebettet
- Bean-Definition deklarativ in XML
  - Namespace für dynamic language support
    - **<lang:jruby/> (JRuby)**
    - **<lang:groovy/> (Groovy)**
    - **<lang:bsh/> (BeanShell)**
- Achtung: Fehler in Quelltext führen zu Laufzeitfehler...

Spring DynaLang Support (1/5)

Orientation in Objects

- Definiere das Interface in Java

```
package org.springframework.scripting;

public interface Messenger {

    String getMessage();
}
```

Spring DynaLang Support (2/5)

Orientation in Objects

- Andere Beans nutzen das Interface

```
package org.springframework.scripting;

public class DefaultBookingService implements BookingService {

    private Messenger messenger;

    public void setMessenger(Messenger messenger) {
        this.messenger = messenger;
    }

    public void processBooking() {
        // use the injected Messenger object...
    }
}
```


Spring DynaLang Support (3/5)

Orientation in Objects

- Do the dynamic language magic

```
package org.springframework.scripting

class GroovyMessenger implements Messenger {

    private String message = "Bingo"

    public String getMessage() {
        // change the implementation to surround the message in quotes
        return "'" + this.message + "'"
    }

    public void setMessage(String message) {
        this.message = message
    }
}
```

Spring DynaLang Support (4/5)

Orientation in Objects

- Do the XML stuff

```
<beans>

    <!--  this bean is now 'refreshable' due to the
            presence of the 'refresh-check-delay' attribute -->
    <lang:groovy id="messenger"
          refresh-check-delay="5000"
                  <!-- switches refreshing on with 5 seconds between checks -->
          script-source="classpath:Messenger.groovy">
        <lang:property name="message" value="I Can Do The Frug" />
    </lang:groovy>

    <bean id="bookingService" class="x.y.DefaultBookingService">
        <property name="messenger" ref="messenger" />
    </bean>

</beans>
```


Spring DynaLang Support (5/5)

Orientation in Objects

- There is no 5

Inline DynaLang Bean

Orientation in Objects

```
<lang:groovy id="messenger">
    <lang:inline-script>


package org.springframework.scripting.groovy;

import org.springframework.scripting.Messenger

class GroovyMessenger implements Messenger {

    String message
}


    </lang:inline-script>
    <lang:property name="message" value="I Can Do The Frug" />
</lang:groovy>
```


Java Scripting API

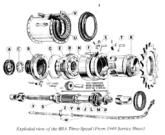
Orientation in Objects

- „Java Scripting API“ erlaubt Java Einbindung von Skriptsprachen
  - Framework zur „Script Engine“ Anbindung von Drittanbietern
  - Einbindendes Programm muss konkrete Sprache nicht kennen
- Basiert auf JSR-223 „Scripting for the Java Platform“
  - „[...] background to bridge the scripting and the Java community.“
- Seit Java 6 fester JDK Bestandteil
- Java 6 beinhaltet „Mozilla Rhino“
  - JavaScript Implementierung für Java (Rhino 1.6R2)
  - Weitere Sprachen per „scripting project“ (http://scripting.dev.java.net)

Java Scripting API Bestandteile

Orientation in Objects

- Überschaubare API
  - 6 Interfaces, 5 Klassen, 1 Exception
- Enthalten in Package javax.script
  - Weitere Tools und Beispiele im JDK (jrunscript, jconsole-plugin, ...)
- Zwei grobe Aufgabenbereiche
  - Scripting Funktionalität für die jeweiligen Sprachen
  - Kontext in dem ein Script ausgeführt wird



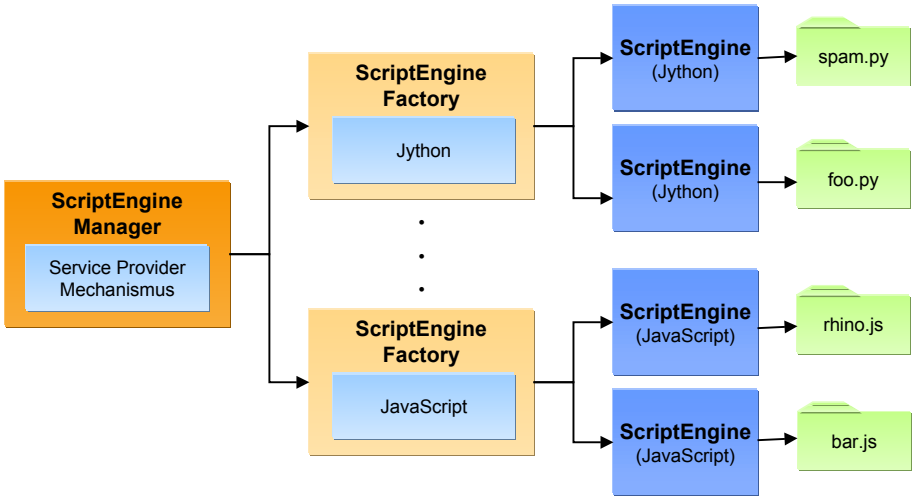
Motoren, Fabriken und Manager (1)

Orientation in Objects

- Interface ScriptEngine
  - Übersetzung und Evaluierung (Ausführung) von Skripten
  - Überladene Methode eval() für verschiedene Quellen (String, Reader)
- Interface ScriptEngineFactory
  - Erzeugt ScriptEngine Instanzen
  - Enthält ScriptEngine Metadaten (Name, Version, File Extension, ...)
  - Enthält Methoden zur Skripterzeugung in der jeweiligen Sprache
- Klasse ScriptEngineManager
  - Sucht vorhandene ScriptEngineFactory Implementierungen
  - Verwendet Service Provider Mechanismus aus Jar Spezifikation
  - Konkrete Klasse als programmatischer Einstiegspunkt

Motoren, Fabriken und Manager (2)

Orientation in Objects



Hello World in JavaScript

Orientation in Objects

```
ScriptEngineManager mgr = new ScriptEngineManager();

ScriptEngine engine = mgr.getEngineByName("JavaScript");

String sayHello = "println('Hello World')";

// Throws ScriptException
engine.eval(sayHello);
```

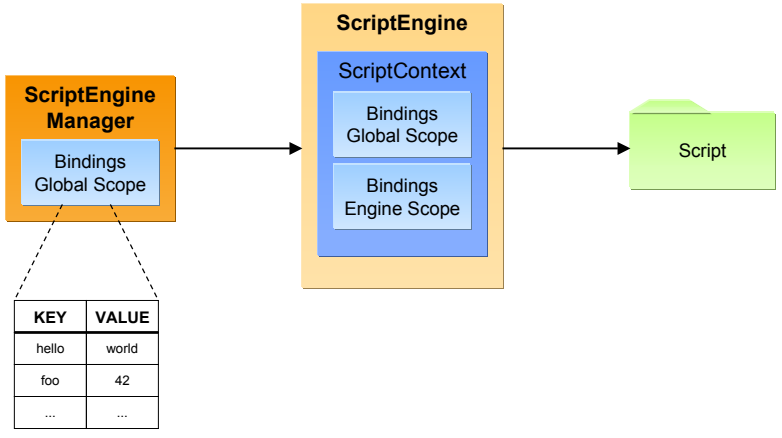
Gebunden im Kontext (1)

Orientation in Objects

- Interface ScriptContext
  - Verbindet ScriptEngine und ausführende Applikation
  - Hat Reader und Writer für ScriptEngine Ein-/Ausgabe
  - Hat ein oder mehrere Bindings
- Interface Bindings
  - Subtyp von Map<String, Object>
  - Für Werteaustausch von ScriptEngine und Applikation
- Scope ordnet die Bindings eines ScriptContext
  - Global Scope - Sichtbar in allen Engines
  - Engine Scope - Sichtbar in einer bestimmten Engine Instanz
  - Suche nach Werten vom kleinsten Scope aufwärts

Gebunden im Kontext (2)

Orientation in Objects



Kann man machen, muss man aber nicht

Orientation in Objects

- Manche Features von ScriptEngine sind optional
  - Wenn vorhanden, implementiert ScriptEngine zusätzliche Interfaces
- Interface Compilable
  - Skripte sind mehrfach ausführbar ohne Neucompilierung
  - Verwendung eines Abstract Syntax Tree (AST) oder Java Byte Code
  - ScriptEngine Instanz wird umgewandelt in CompiledScript Objekt
- Interface Invocable
  - Einzelne Methoden eines Skripts sind gezielt ausführbar
  - Methoden eines Skript können Java Interfaces implementieren

jrunscipt - command line script shell (1)

Orientation in Objects

- Skriptsprachen unabhängige Kommandozeilen Shell
  - Sprachanbindung entspricht „Java Scripting API“
- Experimenteller Bestandteil des JDK
  - siehe %JAVADOC%\technotes\tools\share\jrunscript.html
- Unterstützt „interactive“ und „batch“ Verwendung
  - Ausführung von „one liner“ Skripten möglich
- Defaultsprache JavaScript hat praktische eingebaute Funktionen
  - siehe %JAVADOC%\technotes\tools\share\jsdocs\index.html

jrunscipt - command line script shell (2)

Orientation in Objects

```
~ $jrunscript -q
Language ECMAScript 1.6 implemention "Mozilla Rhino" 1.6 release 2
Language python 2.2.1 implemention "jython" 2.2.1
Language groovy 1.5.1 implemention "groovy" 1.5.4

~ $jrunscript -e "which('javac.exe')"
c:\java\jdk1.6.0_10\bin\javac.exe

~ $jrunscript -e "ip('localhost')"
localhost/127.0.0.1

~ $jrunscript -l groovy -e "println 'hello groovy'"
hello groovy

~ $jrunscript
js> 23 + 19
42.0
js> exit()
```


Demo

Orientation in Objects

Demo „Technische Anbindung in Java“

Gliederung

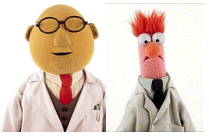
Orientation in Objects

- Einleitung
- Skriptsprachen in a Nutshell
- Technische Anbindung in Java
- Einsatzszenarien
- Zusammenfassung

Die üblichen Verdächtigen

Orientation in Objects

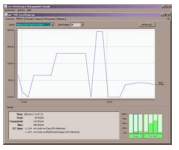
- Kommandozeilen Shell in Anwendungen
  - Graphische Konfiguration ist Standard
  - Administratoren bevorzugen häufig Kommandozeile
- Anpassung und Erweiterung von Anwendungen
  - Auslagern von Geschäftslogik und dynamische Konfiguration
  - Benutzermakros und Reporting
- Rapid Prototyping
  - Vermeidung des „edit-compile-deploy“ Zyklus
  - Ermöglicht schnelles Benutzer-Feedbacks



Kommandozeilen Shell in JConsole

Orientation in Objects

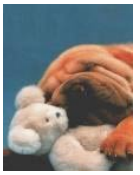
- Seit Java 5 „Monitoring and Manageability“ Unterstützung
  - Java Management Extensions API (JMX)
  - JVM Management Interface (java.lang.management)
  - Platform MBeans für Classloading, Memory, Threads, ...
- Swing basiertes Tool JConsole
  - JMX Oberfläche mit Zugriff auf eigene und Platform MBeans
  - Eigene JConsole Erweiterungen möglich
  - Seit Java 6 Standard
- JDK Demo für Scripting Admin Console
  - siehe %JAVA_HOME%\demo\scripting\jconsole-plugin



Wachs in meinen Händen

Orientation in Objects

- Komplexe Anwendungen sollen häufig „customizable“ sein
  - Geschäftslogik, Reporting, Benutzermakros
  - Benutzerschnittstelle von Computerspielen
- Keiner ist wirklich geeignet
  - Entwickler ist zu teuer und hat kein fachliches Verständnis
  - Anwender kann nicht (genug) programmieren
- Auslagern von bestimmten Anwendungsteilen
  - Zugriff durch kundengerechte Abstraktion
  - Skriptsprachen oder Domain Specific Language (DSL)
- JDK Demo Scriptpad
  - %JAVA_HOME%\sample\scripting\scriptpad



Rapid Prototyping

Orientation in Objects

- Experimentierkasten um Ideen auszuprobieren
  - Ziel ist das schnelle Evaluieren und Bewerten von Ideen
  - Lösung hat keinen Anspruch auf technische „Perfektion“
  - Gewisse Opfer werden gemacht (Architektur, Typisierung, ...)
- Dynamische Sprachen können unterstützen
  - Buildprozess schrumpft von „edit-compile-run“ auf „edit-run“
  - Dynamische Sprachen sind oft einfach und mächtig
  - Sprache ist aber nur Teil der Lösung (Infrastruktur, Frameworks, ...)
- Prototypen können gefährlich sein
  - Kunden können Prototyp und Produkt oft nicht unterscheiden
  - Nicht jedes dynamische Framework ist ein Prototyp-Framework

Demo

Orientation in Objects

Demo „Einsatzszenarien“

Gliederung

Orientation in Objects

- Einleitung
- Skriptsprachen in a Nutshell
- Technische Anbindung in Java
- Einsatzszenarien
- Zusammenfassung

If you remember one thing

Orientation in Objects

„Every sufficiently large Java program, anything beyond medium-sized, needs a scripting engine, whether the authors realize it or not.“

(Steve Yegge, Stevey‘s Blog Rants: „The Universal Design Pattern“, 2008)

Links

- Java Scripting Programmer‘s Guide
  - http://java.sun.com/javase/6/docs/technotes/guides/scripting/programmer_guide/index.html
- The scripting project
  - http://scripting.dev.java.net
- Scripting for the Java Platform
  - http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/
- The Mustang Meets the Rhino: Scripting in Java 6
  - http://www.onjava.com/pub/a/onjava/2006/04/26/mustang-meets-rhino-java-se-6-scripting.html

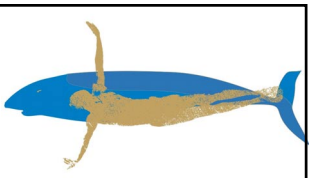
Vielen Dank für Ihre Aufmerksamkeit !

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Version: 1.0



Papick Garcia Taboada

Software Architekt

Technologie-Scout

Beratung

Projekte

Training

Orientation in Objects

Steffen Schluff

~~*Im a pogramar*~~

~~*Iam a programer*~~

~~*I'm a programor*~~

I write code

Beratung

Projekte

Training

Orientation in Objects

Fragen ?

Orientation in Objects GmbH

Weinheimer Str. 68
68309 Mannheim

www.oio.de
info@oio.de

Version: 1.0

„Orientierung“ in Objekten

Orientation in Objects

Java und XML

) Akademie)

- **Schulungen**, **Coaching**, **Weiterbildungsberatung**, **Train & Solve**-Programme

) Beratung)

- **Methoden**, **Standards** und **Tools** für die Entwicklung von **offenen, unternehmens-weiten Systemen**

) Projekte)

- **Schlüsselfertige Realisierung** von Software
- **Unterstützung laufender Projekte**
- **Pilot- und Migrationsprojekte**