Project Mustang - Neues in Java SE 6

Daniel Adelhardt
Java Architekt
Sun Microsystems GmbH

Agenda

- Generelles zum neuen Java SE Release
- Ausgwewählte Themen
  - Desktop Java
  - Scripting
  - XML/Web Services
  - Monitoring und Management
  - JVM Internas
- Demos
- Zusammenfassung

Java SE 6 - Mustang

- Java SE 6 FCS Release: 10/2006
  - Mustang enthält keine neuen Spracherweiterungen
  - Inhaltliche Schwerpunkte
    - Desktop
    - XML/Web Services
    - Ease of Development
    - Performance
  - Erstes Release mit komplett öffentlicher Entwicklung
    - Wöchentliche Source & Binary Snapshots
    - Zahlreiche Bugfixes aus der Community!

Einige ausgewählte Features...

Java SE 6 Themen
mit eigenständigen JSR Spezifikationen

Mustang Umbrella Spec – JSR 176

XML & Web Services

JAX-B 2.0 (JSR-222)
JAX-WS 2.0 (JSR-224)
WS Metadata (JSR-181)
StAX API (JSR-173)
XML Signature (JSR-105)

Ease of Development

Common Annotations (JSR-250)
Scripting API (JSR-223)
JDBC 4 (JSR-221)
Plug. Annotations (JSR-269)

Miscellaneous

Java Compiler API (JSR-199), Class File Update (JSR-202)

Mustang - Java für den Desktop

- Mustang und Dolphin verstärken den Fokus auf Desktop Java
  - Integration von Komponenten aus den SwingLabs Projekten
  - Erweiterung der Deployment Features
  - Performance, Stabilität, Look & Feel
  - Mehr Feautures sind für das Dolphin Release geplant
    - Beans Binding Framework (JSR-295)
    - Swing Application Framework (JSR-296)
    - Deployment Modules: Java Module System (JSR-277)
    - Development Modules: “Super Packages” (JSR-294)

Desktop Java - Highlights

Neue APIs: Tray Icons, Splash Screens, Desktop Helper API, Group Layout Manager, Swing Worker, JTable Sorting

Performance:

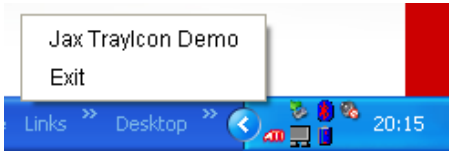
- Echtes Double Buffering
- Mehr HW Accelleration
- Single Threaded Rendering für OpenGL

Look & Feel:

- Windows Vista Support
- Native Rendering für GTK/Win
- LCD Text

Desktop Java – Tray Icons

- TrayIcons sind “Mini Applikationen” die in einem speziellen Bereich durch den Window Manager verwaltet werden
  - Haben kein eigenes Root-Window
  - Bestehen aus Icon und Kontextmenü
  - Beispiel: Windows Start Leiste



- Neue Klassen java.awt.TrayIcon/SystemTray
  - TrayIcon Applikationen registrieren ein Image und ein Menü beim SystemTray
  - SystemTray interagiert mit dem jeweiligen WindowManager

Desktop Java – Tray Icons

```
TrayIcon trayIcon = null;
if (SystemTray.isSupported()) {
    SystemTray tray = SystemTray.getSystemTray();
    Image image = Toolkit.getDefaultToolkit().getImage("...");
    PopupMenu popup = new PopupMenu();
    MenuItem defaultItem = new MenuItem("Jax TrayIcon Demo");
    ActionListener listener = new ActionListener() {
        public void actionPerformed(ActionEvent e) {.....}
    };
    defaultItem.addActionListener(listener);
    popup.add(defaultItem);
    trayIcon = new TrayIcon(image, "JFS Tray Demo", popup);
    trayIcon.addActionListener(listener);
    try {
        tray.add(trayIcon);
      } catch (AWTException e) { ...}
} //....
```

Desktop Java – Splash Screens

- Mustang bietet Support für Splash Screens die bei Start der Applikation eingeblendet werden
  - Steigerung der “perceived Performance”
  - Support für GIF,JPEG,PNG, Animated Gif
  - JVM Commandline: java -splash:myimage.gif ...
  - Manifest Eintrag

    ```
    SplashScreen-Image: filename
    ```
  - Splash Image wird noch vom JVM Bootstrap Code geladen
- **`java.awt.SplashScreen`** für Kontrolle des SplashScreens durch die Applikation

Desktop Java – Splash Screens

```
/* Start der Applikation mit java -splash:file.gif
 * oder über META-INF/MANIFEST.MF Eintrag
 */

public static void main(String[] args) {
    //SplashScreen anfordern
        SplashScreen screen = SplashScreen.getSplashScreen();
    //G2D Objekt zur Manipulation
        Graphics2D g = screen.createGraphics();
        Font defaultFont = g.getFont();
        g.setFont(new Font(defaultFont.getName(), Font.BOLD, 20));
        g.setColor(Color.RED);
        g.drawString("Application ready!", 30, 30);
        screen.update();
    screen.close()
}
```

Desktop Java – Desktop APIs

- Über die **`java.awt.Desktop`** Klasse können Java Applikationen assoziierte native Applikationen für Filetypen starten
  - Start des Default Browser
  - Start des E-Mail Clients
  - Start der Default Applikation um Files anzuzeigen, zu drucken oder zu editieren

Desktop Java – Desktop API

```
/* Desktop API ist nicht auf allen OSen bzw.
 * Window Managern supported
 */


 if (Desktop.isDesktopSupported()){
    Desktop dt = Desktop.getDesktop();
    try {
        dt.open(new File ("C:\\duke.gif"));
        dt.edit(new File ("C:\\test.pdf"));
        dt.print(new File ("C:\\doc1.txt"));
        if (dt.isSupported(Desktop.Action.BROWSE))
            dt.browse(new URI("file:///C:/index.html"));

         } catch (Exception ex){
        ex.printStackTrace();
       }
```

Scripting in Java

- Script Sprachen und Java rücken stärker zusammen!
  - Scripting Engine Integration über Bridge in Java SE 6
  - Neuer Bytecode “invokedynamic” in Java SE 7 für dynamisch getypte Sprachen
  - Erweiterungen in Java EE geplant bzgl. Co-Packaging
- Das Sun JDK 6 enthält die Mozilla Rhino JavaScript Engine
- Weitere Scripting Sprachen/Engines integrierbar
  - Siehe http://Scripting.dev.java.net
    - Groovy, Jython, BeanShell, TCL, Ruby, Scheme, PHP,...

Scripting in Java

- Scripting Integration ist bidirektional
  - Java Objekte rufen externe Scripts auf
  - Scripts rufen Java Objekte auf
- Neues Commandline Tool **`jrunscript`** als einfache Scripting Shell
- Mögliche Scripting Usecases
  - Applikationstests
  - Extern beinflussbare Parameter (Formeln, Kontrolllogik/Workflow usw.)

Scripting in Java - jrunscript

- jrunscript erlaubt interaktives Scripting mit Java
  - Default: JavaScript, aber auch Groovy,BeanShell, JPython möglich

```
js> importPackage(Packages.javax.swing)
js> var frame = new JFrame(„JFS2006-Demo“)
js> var bt1 = new JButton(„ClickMe“)
js> frame.add(bt1,“North“)
js> frame.pack()
js> frame.show()
```

Scripts in Java Code

- Einfacher Aufruf eines Scripts aus Java

```
ScriptEngineManager sem = new ScriptEngineManager();
ScriptEngine engine = sem.getEngineByName("js");
try {
        engine.eval(new FileReader("hello.js"));
       } catch (Exception ex) { ...}
```

- Aufruf eines Scripts mit Bindings

```
ScriptEngineManager sem = new ScriptEngineManager();
ScriptEngine engine = sem.getEngineByName("js");
SimpleBindings bind = new SimpleBindings();
//Binding der Variable datum zur Verwendung im Script
bind.put("datum", new Date());
engine.setBindings(bind, ScriptContext.ENGINE_SCOPE);
engine.eval(new FileReader("hello.js"));
```

Scripts in Java Code

- Aufruf einer Script Funktion in Java

```
ScriptEngineManager sem = new ScriptEngineManager();
ScriptEngine engine = sem.getEngineByName("js");
engine.eval(new FileReader("mycalulator.js"));
//Nicht von allen Engines supported
Invocable inv = (Invocable)engine;
//Aufruf liefert immer Typ Object
Object result = inv.invoke("calc",10,10);
//Typ Mapping abhängig von Engine
Double d = (Double)result;
```

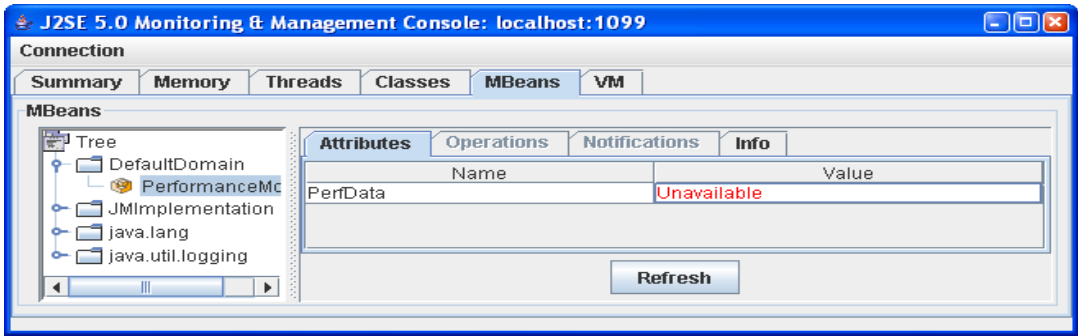
```
//JS
function calc(x,y){
    return x+y
}
```

Management und Servicability

- Management und bessere Servicability waren bereits Fokus für JDK 5.0 gewesen
  - JMX Integration und JVM Instrumentierung, Tooling (jps, jstat, jmap, JConsole)
- Mustang erweitert Management Features
  - Neuer MBean Typ: MXBeans
  - Deskriptor Support über Annotations
  - Neue Meta Annotation `@DescriptorKey` um Bean Properties mit Metainformationen zu versehen
  - Generification von einigen Methoden z.B. MBeanServer.queryNames
  - Helper Klasse JMX mit static Methoden um Proxies für MXBeans zu erzeugen

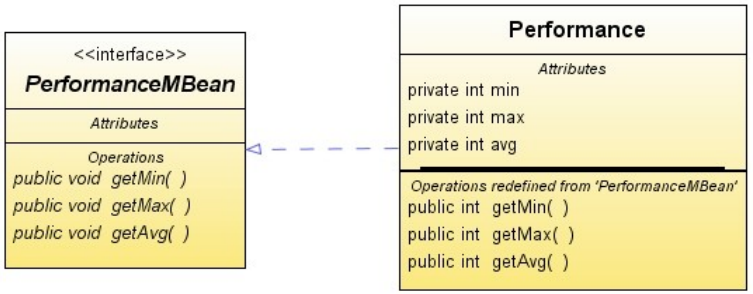
Management: MXBeans

- MXBeans sind ein neuer Typ von JMX MBeans
  - Aufbau: Interface mit Prefix <Name>MXBean, Klasse <Name>
  - JMX Infrastruktur mapped Parameter/Return Types auf JMX OpenTypes wie CompositeDataSupport
  - Type Mapping beschrieben in MXBean Spec
- Weshalb MXBeans???
  - MBeans liefern oft zusammenhängende Werte
    - z.B. min/max/avg Werte für Requeststatistiken oder Pools
    - Werte sind meist primitive Typen
  - Einführung von Containerobjekten problematisch!
    - Generische JMX Konsolen müssen Containerobjekte im Classpath haben

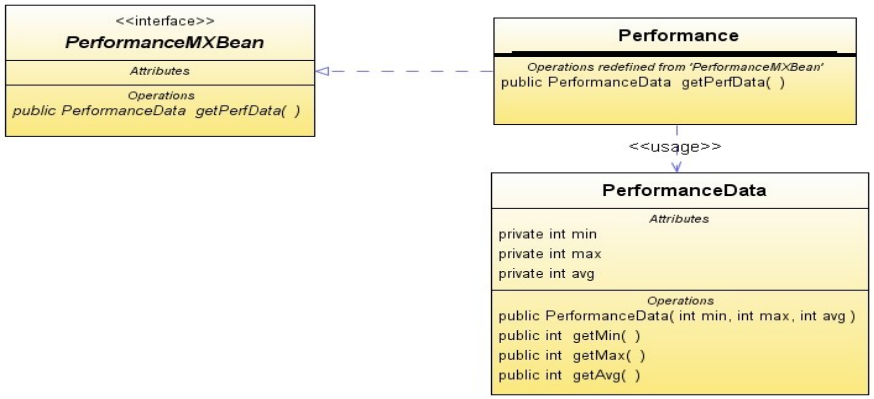


Management - MXBeans

- Beispiel (Standard MBean)
  - > MBean hat einzelne Properties für Performance Daten

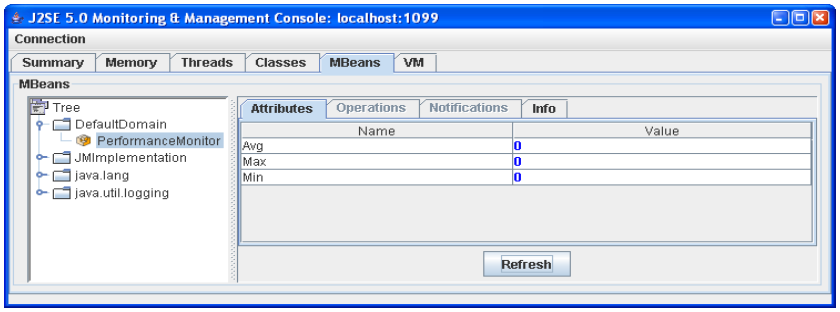


- Beispiel mit MXBean
  - > Zusammenhängende Daten werden in Helper Objekt aggregiert

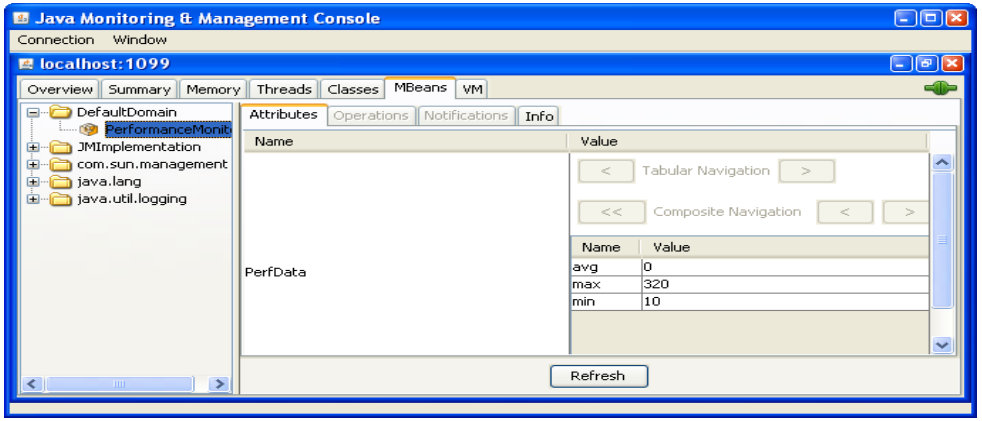


Management - MXBeans

- Beispiel (Standard MBean)
  - > JConsole für Standard MBean



- Beispiel mit MXBean
  - > JConsole mit MXBean und Containerobjekt



Mustang - JVM internals

- Classpath Wildcards für Commandline Launcher
  - Matched jar Files – nicht rekursiv
  - **`java -cp lib/* <Main>`**
- Split Verifyer - Performance
- Standard Compiler API – JSR 199
  - Schafft Java API zum Steuern des Compilers
- JVM Attach on Demand für JConsole
  - Ermöglicht dynamischers Monitoring jeder Applikation ohne `-Dcom.sun.management.jmxremote` Flag

Mustang - JVM internals

- SIGQUIT Verbesserungen (CTRL-BREAK)
  - Liefert nun Thread States und Heap Information
  - Lock Informationen für java.util.concurrent
- Verbessertes OutOfMemory Errorhandling
  - `Exception in thread "main" java.lang.OutOfMemoryError: Java heap space`
- Neue JVM Parameter
  - `-XX:OnOutOfMemoryError=<script>`
  - `-XX:+HeapDumpOnOutOfMemoryError`
- Solaris Only: DTrace Probes in der JVM
  - Ermöglicht Analyse von GC, Threads, Monitoren...

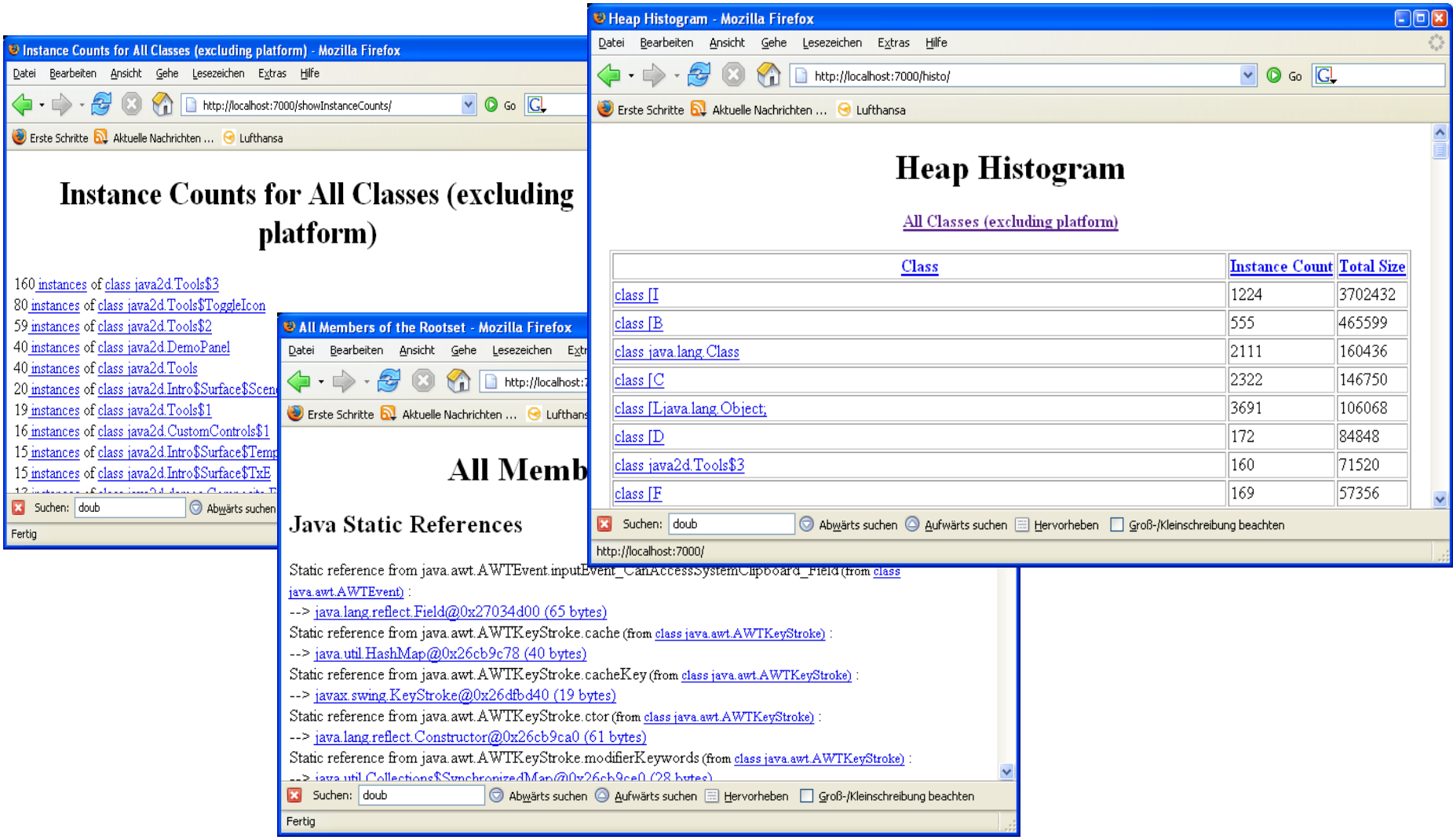
Mustang - JVM internals

- Heap Dump Analyse (nun auch für Windows)
  - jmap -histo <pid> liefert ein Heap Histogramm

```
num   #instances    #bytes  class name
----------------------------------------
  1:      1298     5571944        [I
  2:       555      469824        [B
 12:       160       72960        java2d.Tools$3
 15:      2298       55152        java.lang.String
```

  - jmap -dump:format=b,file=heap.dmp <pid> liefert Heap Dump einer laufenden JVM
  - jhat <dumpfile> startet Heap Analyse Tool
    - Mini Web Server auf localhost:7000

Mustang - JVM internals



XML & Web Services in Mustang

- JAX-WS 2.0 API
  - SOAP basierte Web Services
- JAX-B 2.0 API
  - XML Databinding
- XML Digital Signature API
- Streaming API for XML (StAX)

JAX-WS 2.0 API in Mustang

- JAX-WS ist die Weiterentwicklung vonJAX-RCP
  - Verwendung von Code Annotations für einfachere Implementierung
  - Mustang bietet Client + Server Support für JAX-WS
  - Code Annotations sind durch JSR-181 („Web Service Metadata“) definiert
  - „Configuration by Exception“ - API arbeitet mit vielen Default Settings
    - @WebService() -> Name des Service = Klassenname
- Ausgangsbasis für Web Service Erstellung kann sein:
  - bestehendes WSDL Dokument
    - Java Skeleton Generierung über **`wsimport`** Tool
  - Java Implementierung
    - Generierung WSDL aus Java Code per **`wsgen`** Tool

JAX-WS 2.0 Beispiel

- Trivialer Echo Service

```
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
@WebService
@SOAPBinding(style=SOAPBinding.Style.RPC)
public class MyService {
    public String echo(String param){
        return "Hello " +param;
    }
}
```

- Endpoint Aktivierung
  - Benutzt eingebauten HTTP Server von Mustang

```
public static void main(String args[]){
  Endpoint.publish("http://host:8080/echo",
                   new MyService());
  }
```

JAX-WS 2.0 Beispiel

- Erzeugung der Client Artefakte

```
<jdk6>/bin/wsimport -d <dir> http://localhost:8080/echo?wsdl
```

- Client Code

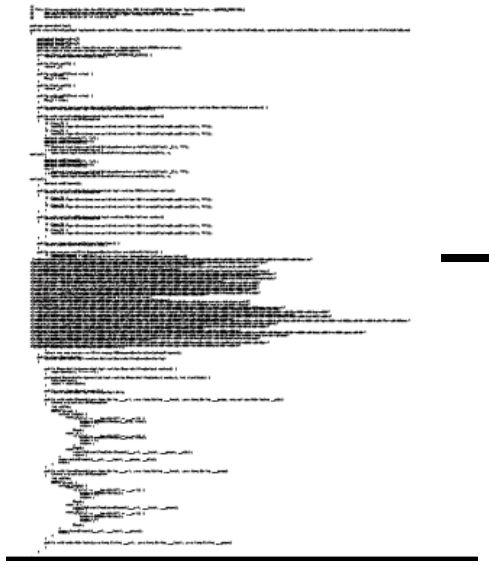
```
public static void main(String[] args) {
        MyService svc = new MyServiceService()
                    .getMyServicePort();
        String ret = service.echo("JFS2006");
        //...
 }
```

XML Databinding mit JAX-B 2.0

- JAX-B 2.0 ist eine komplette Überarbeitung
  - JAX-B 2.0 bietet nun 100% Support für XML Schemas
  - Drastische Reduktion des generierten Codes über Annotations und höhere Performance
  - JAX-B 1.0 : Schema ➞Java only
    - JAXB Schema Compiler, Generierte Code nicht änderbar
  - JAX-B 2.0 : Auch Java ➞ XML + schema compiler
    - JAXB 2.0 = XML Persistenz für POJOs
- Java <-> XML Mapping über Annotationen
  - Schema Compiler ist eher ein Skeleton Generator
  - Generierter Code kann nach Belieben modifiziert werden
    - Customization über Inline Annotations im Schema
    - Externes Binding File

JAX-B 2.0 vs 1.0

- XML Dokument: **`<point><x>1</x><y>2</y></point>`**
  - JAX-B 1.0: 308 LoC, 38 Files, ~220kb Code
  - JAX-B 2.0: 62 LoC, 2 Files, 3kb Code



```
@XmlAccessorType(FIELD)
@XmlType(name = "", propOrder = {"x","y"})
@XmlRootElement(name = "point")
public class Point {

    protected float x;
    protected float y;
    public float getX() {
        return x;
    }
    public void setX(float value) {
        this.x = value;
    }
    public float getY() {
        return y;
    }
  public void setY(float value) {
        this.y = value;
    }
}
```

JAX-WS und JAXB Integration

- JAX-WS delegiert Data Binding zu JAX-B
- Zur Entwicklungszeit (wsgen/wsimport Tools)
  - JAXB generiert Java Typen von WSDL Schemas
  - JAXB generiert WSDL Schemas von Java Typen
- Zur Laufzeit
  - JAX-WS un/marshalled die Message(soap:envelope)
  - JAXB un/marshalls die Payload (soap:body child, soap:header, soapfault Elements)
  - StAX Parser

Streaming API for XML (StAX)

- Nachteile traditioneller XML Parsing Ansätze
  - DOM: Memory intensiv, langsam
  - SAX: Callback-Ansatz gibt Applikation wenig Kontrolle über das Parsing
- Streaming API ist ein Pull Parser Ansatz
  - Client hat volle Kontrolle des Parsing Vorgangs
  - Event Filter, Namespace Support
  - Geringer Footprint, hohe Performance
- StAX sieht zwei Modelle vor: Cursor vs. Iterator
  - Cursor API: Entwickler navigiert ein Cursor Objekt über den XML InputStream
    - Metainformationen müssen über XMLStreamReader Instanz erfragt werden
  - Iterator API: XML Stream wird als Menge diskreter Parsing Events zugestellt
    - Metainfos als immutable Objekte im Event gekapselt

StAX Cursor Beispiel

```
XMLInputFactory pf = XMLInputFactory.newInstance();
XMLStreamReader reader =
   pf.createXMLStreamReader(new FileInputStream(...));

   while (reader.hasNext()){
      int evt = reader.next();
      switch (evt){
         case XMLStreamConstants.START_ELEMENT:
         //z.B. Abfrage des Elements per
                  reader.getLocalName
            break;
         case XMLStreamConstants.CHARACTERS:
            System.out.println("Element:" + reader.getText());
            break;
      }
   }
```

Mustang Performance

- Java SE 6 enthält sehr viele Performance Optimierungen
  - ~ 10% Performance Verbesserung für viele Applikationen realistisch
- Biased Locking
  - Biasing eines Objektes zu dem Thread mit Monitor
  - Beinflussbar per -XX:+UseBiasedLocking (Default: On)
- Lock Coarsening
  - Vermeidung unnötiger Locks über Escape Analyse
- java.math.BigDecimal Optimierung
- Parallel Old Generation Garbage Collector
  - `-XX:+UseParallelOldGC`
- Large Pages Support nun auch für Windows/Linux
  - `-XX:+UseLargePages` - Default bei Solaris

JDBC 4.0

- Vereinfachung der Treiber Registrierung
  - > Verwendung der Services per META-INF/services/java.sql.driver

```
String url =
  „jdbc:oracle:thin:@localhost:1521/db“;
DriverManager.getConnection(url,user,pwd);
//kein ClassForName mehr!
```

- Diverse Anpassung an neuere ANSI SQL Standards
- Annotations für Result Manipulation (@Query, @Update)
- Support for SQL ROWIDs,SQL/XML Support
- Neu: Derby DB gebundelt mit JDK!

Fazit

- Mustang ist die konsequente Weiterentwicklung von Java 5
  - Sukzessive Vereinfachung von API
  - Behutsame Integration von neuen Features
  - JDK wandelt sich zur komletten Entwicklungsplattform
    - HTTP Server, Java DB (Derby), Scripting, Diagnosetools
  - Performance & Stabilität als permanenter Fokus
- Dolphin bringt mehr Revolutionäres
  - Neue Packaging Konzepte für Java
  - XML als Sprachtyp
  - Neue GUI Frameworks für Swing
- Participate: http://jdk.dev.java.net !

Mustang – Java SE 6

Daniel Adelhardt
daniel.adelhardt@sun.com