Java

Heutiges Programm:

- Was ist Java
- Historie Java
- Grundlagen von Java
- Java unter OpenVMS
- Fazit

Anlass und Ziel

- Java innerhalb von JSP & Servlets
- damit nur „minimal“ genutzt
- (aber) auch unter OpenVMS
- Neugier: was kann Java eigentlich alles?

- kein Java-Programmierkurs
- Erklärung von Hintergründen, Stärken, Schwächen
- Übersicht und Vergleich C/C++ (.NET/C#)

Java

- Java ist eine objektorientierte Programmiersprache
- Java ist eine Plattform: Sprache & Bibliotheken/Code & Run-Time-Environment (die wichtige Dienste bereitstellt wie Security, Portability, …)
- Java ist eine „Technologie“: Java als Basis für Lösungen, die von der Plattform profitieren (Applets, JSP, EJB, …)

Java „White Paper“

- Design-Ziele sehr klar umrissen
  → http://java.sun.com/docs/white/langenv/
- Zusammenfassung in einer Liste von Schlagwörtern (Java „White Paper“ Buzzwords)
- Sehr oft Vergleich mit C++
- (ja, Java-Entwickler haben (*hatten damals*) ein einfaches und klares Feindbild ;-))
- „man kann in jeder Sprache Mist programmieren“!
- Nicht immer werden die Ziele auch wirklich erreicht …

Java „buzzwords“

Jetzt mal die URSPRÜNGLICHEN Überlegungen beim Design von Java …

Java: „Simple“

- Java ist ein „syntax clean-up“ von C++
- viele Konstruktionen, Schlüsselwörter und Features finden sich wieder
- Vieles wurde vereinfacht (#includes, Pointer, Speicherverwaltung)
- Andere Zugänge OO: Vererbung, virtuelle Basisklassen
- Umstieg von C/C++ sehr einfach
- Umstieg von „Klick“-Sprachen (VB) schwerer

Java: „Object Oriented“

- Java ist eine OO Sprache
- OO ist „state-of-the-art“
- OO fördert effiziente Programmierung und die Verwendbarkeit von Code
- in Java ist alles in Klassen untergebracht (auch main ())
- es gibt keine „globalen“ Funktionen
- auch keine Callbacks ((anonyme innere Klassen notwendig)

Java: „Distributed“

- umfangreiche Unterstützung von Netzwerkprotokollen (TCP/IP, HTTP, FTP)
- integriert in die Sprache (Bibliotheken)
- damit gleiche Ebene, wie Dateizugriff
- RMI: remote method invocation (RPC) ebenfalls als Bestandteil von Java erlaubt „distributed objects“
- verteilte Anwendungen implementierbar

Java: „Robust“

- Java soll große, sichere und robuste Applikationen erlauben
- viel Aufwand im Compiler und Laufzeitsystem, um Fehler zu erkennen und zu behandeln
- gutes Exception Handling
- O-Ton: „Java puts a lot of emphasis … (to) eliminating situations that are error-prone … The single biggest difference between Java and C/C++ is that Java has a pointer model that eliminates the possibility of overwriting memory and corrupting data“

Java: „Secure“

- in der Sprache (Plattform) integrierte Security-Mechanismen
- Java soll Applikationen im Netzwerk und verteilte Applikationen ermöglichen
- Security unabhängig vom Betriebssystem
- gängige Mechanismen zur Abwehr von Viren und Angriffen:
  - Stack-Verwaltung
  - Memory Management
  - Steuerung und Überwachung der Dateizugriffe

Java: „Architecture Neutral“

- Compiler generiert Byte-Code in einem vom Betriebssystem unabhängigen Format
- Speicherung in vom Betriebssystem unabhängigen Format (nicht EXE, ELF, …)
- kann überall ausgeführt werden, wo Java RTE vorhanden ist (Java Interpreter, JVM)

Java: „Portable“

- Definition der Sprache unabhängig von der Architektur des Systems und ohne Kompromisse (wie C/C++ und int):
  - feste Größe der primitiven Datentypen
  - feste Größe und festes Verhalten der Arithmetik (auch FP!) als Sprachdefinition
- Bibliotheken für alle Betriebssystem gleich (Plattform-Gedanke)
- Idee: write once, run everywhere

Java: „Interpreted“

- Jedes System, welches eine Java RTE (Java „byte code interpreter“, JVM – Java Virtual Machine) bereitstellt, unterstützt Java
- JVM → auch als „Sandbox“ (Security, Robustheit)
- Idee nicht neu: Wirth (ETH Zürich) → Pascal
- Performance-Problem? Wirklich? (JIT, HotSpot JVM)
- außerdem: Reduzierung der Linker Vorgänge → Zugriff auf Module (Laden) zur Laufzeit
- dadurch schnellere Entwicklungszyklen (theoretisch → Zeit wird durch andere Probleme in den IDE „aufgefressen“ → Dateizugriffe beim Suchen und Laden der Klassen)

Java: „High Performance“

- Ausführung von Java Byte-Code ist schnell
- zusätzliche Möglichkeiten von JIT (just-in-time Compiler, HotSpot JVM)
- in der Ausführung kaum Unterschiede zu „native“ Code
- gleiche Diskussion wie Assembler vs. C/C++

Java: „Multithreaded“

- Multithreading als Bestandteil der Plattform
- Bibliotheken/Klassen zum Support
- ein wohl definiertes Interface, klar und einfach
- Unterstützung in der RTE

Java: „Dynamic“

- Dynamisch im Sinne von Erweiterbarkeit: Klassen und Bibliotheken (Trennung Sprache und Klassen)
- Dynamisch im Sinne von „zur Laufzeit etwas herausfinden“ (und so eine dynamische Verarbeitung von Objekten ermöglichen): Analyse und Erkennung von Objekten zur Laufzeit über Reflektion und eine Vererbung, welche immer von einer Basisklasse ausgeht → Basis für GUI/IDE

Historie & Erfolgsstory

- Hype durch Internet: Applets
- Applets als HTML Plug-In → Erweiterung der Funktionen der HTML-Seiten (Interaktion System – Nutzer)
- Download des Byte Codes und Ausführung in jedem „Java enabled“ Browser
- Probleme:
  - Unterschiedliche Browser- & Java-Implementierungen
  - Größe des Byte Codes (Modems)
  - Andere Technologien (Javascript, DHTML, HTML forms, animated GIF‘s)
- Applets für „Rich Clients“ im Intranet
- aber Java inzwischen als Plattform für andere Technologien im Umfeld WWW (Server- und Client-seitig)

Historie & Erfolgsstory

- 1991 Projekt „Green“ bei Sun → Sprache für Geräte im Heimbereich (Settop-Boxen, …)
  - langsame und vielfältige Prozessoren, kleiner Speicher → daher viele Designziele!
  - Herkunft des Teams: Unix und C/C++ (Feindbild!)
  - Sprache wurde „Oak“ genannt
  - weil Oak schon in Verwendung war: JAVA
  - 1992 erstes Produkt: „*7“ – eine intelligente Fernbedienung
  - Bis 1994 keine wirkliche Vermarktung bzw. Interesse

Historie & Erfolgsstory

- 1994: HotJava Internet Browser
  - Alternative zu Mosaic
  - in Java geschrieben
  - unabhängig vom Betriebssystem
  - erste Version 1996
  - als Demonstration von Java und Applets als Basistechnologien im Internet
  - aktuell Java 2 5.0 (1.5)
- Java als Technologie wirkt sich „inspirierend“ auf andere Entwicklungen aus, wie z.B. C# und .NET

Historie & Erfolgsstory

Version	Bemerkung	Klassen
1.0		211
1.1	Inner Classes	477
1.2	Marketing-Name „Java 2 SE“	1524
1.3	Performance Update	1840
1.4	Assertions	2723
5.0	Spracherweiterungen	3270

Java & Sun

- Java gehört Sun, kein Open Source (noch nicht), kein Standard (IEEE, …)
- deswegen „stabile“ Version und Änderungen an der Sprache erst in Version 5.0
- riesige Java Entwicklergemeinde (JCP)
- Liefern viel Input, gerade im Bereich Java-Technologie
- JSR: Java Specification Request → Standardisierung innerhalb der Java-Gruppe
- viele Entscheidungen von Sun im Interesse des Marktes und nicht der Entwickler getroffen

Development

- JDK: Java Development Kit, frei verfügbar
  - Compiler, Dokumentation, Klassen, Tools
- JRE: Java Run-Time-Environment
- IDE: Eclipse, Netbeans, …
- Texteditoren (mit & ohne Java Unterstützung): Textpad, Emacs, …

Java: die Sprache

- Beispiel
  - *Edit: FirstSample.java*
  - *Compile: javac FirstSample.java*
  - *Run: java FirstSample*

```
public class FirstSample
{
        public void printArgument (int i, String s) // print
        {
                System.out.println („i= „ + String.valueOf (i) + „ s= „ + (s == null) ? „<null>“ : s) ;
        }

        public static void main (String[] args)
        {
                FirstSample fs = new FirstSample () ;

                for (int i = 0; i < args.length; i++)
                        fs.printArgument (i, args[i]) ;
        }
}
```

Java: die Sprache

- Zeichensatz, Kommentare //, /* .. */
- Datentypen
  - Streng typisierte Sprache
  - Primitive Typen (int, short, byte, long, float, double, char, boolean) → Klassen (Boxing)
  - Char: 16-bit, UTF-16 → code unit (16-bit) & code point (1-2 code unit)
- Variablen: String s; int i; Integer j;
- Konstanten: final int i = 5 ;
- Operatoren: „wie C++“, extra: >>> (vs. >>)
- Mathematik in „Bibliothek“ (package) Math
- Implizite Konvertierung (byte→short→int→…) und cast
- Auswertung logischer Ausdrücke: short circuit
- Klammerung, Operator-Reihenfolge
- Flow-Control: if, for, while, do while, foreach, switch, break, continue
- Blocks: { }

Java: Strings

- Strings als Klasse, sind Konstant, Objekte
- String s = „Paul“ ; s += „ Otto“ ; (Sprache!)
- Operationen über Methoden der Klasse:
  - substring ()
  - compareTo ()
  - equals ()
  - indexOf ()
  - …
  - Definiert in java.lang.String

Java: Arrays

- int[] aNumbers = {1, 2, 3, 4, 5} ;
- Int[] aNumbers = new int[10] ;
- → andere Schreibweise als in C/C++
- Arrays von beliebigen primitiven Datentypen oder Objekten (Klassen)
- Nur Arrays, keine Strukturen oder Unions
- Strukturen = Klassen! (ist in C++ eine Erbschaft von C)

Objekte & Klassen

- Java ist eine OO Sprache!
- Objekt-Variablen: enthalten Referenzen auf Objekte:
  - Date today = new Date () ;
  - Date saveDate = today ;
  - hinter „Referenz“ verstecken sich Pointer (!)
  - aber: Objektvariablen sind keine Pointer
- Objekt-Variablen sind am Anfang leer (null)
- Objekte müssen erzeugt werden
- Automatisches Aufräumen (garbage collection)

Objekte & Klassen

- typisch OO:
  - Methoden, Members
  - Lebensdauer und Sichtbarkeit (public, private)
  - statische Methoden
  - Konstruktoren
  - KEINE DESTRUKTOREN (nur finalize)
  - Überladung von Methoden (aber NICHT von Operatoren!)
  - + für Strings in Sprache

Objekte & Klassen

- keine Mehrfachvererbung → strenge Hierarchie
- dafür Interfaces (z.B. Comparable)
  - class Employee implements Comparable<Employee>
    { public int compareTo (Employee other) {…}
    }
- immer alles von Object (die „kosmische Superklasse“) abgeleitet
- virtuelle Klassen
- Inner Classes + anonyme Inner Classes („Callbacks“)
- Standardmethoden: toString(), equals()
- virtuelle Klassen
- mit Version 5.0: type parameters
  - ArrayList<Employee> staff = new ArrayList<Employee> ();

Methoden

- nur Funktionen:
  - höchstens ein Rückgabewert, der aber ein Objekt sein kann
  - void
  - nur Input-Parameter
  - alle Parameter „call by value“
  - Objektvariablen != Objekte → Referenz wird „by value“ übergeben!
  - man kann Objekt ändern aber nicht die Referenz → Cloning notwendig (Interface Cloneable.clone ())

Pakete (packages)

- Name Klasse = Name Source-Datei!
  - javac FirstSample.java → class FirstSample {}
  - java FirstSample
  - mehrere Klassen pro Source-File
  - implizites make
- Pakete: Zusammenfassung Klassen in einem Namensraum:
  - de.equicon.decus.Sample1
  - Abbildung Verzeichnisstruktur
  - Packen: jar = Zip + Manifest
  - Suche im Classpath (jar-Files) oder in der Verzeichnisstruktur

Und vieles mehr …

- die „Sprache“ als solches ist damit (fast) abgebildet (Exceptions, Generics, …)
- es gibt viel an Funktionalität, die sich i.a. durch entsprechende Klassen darstellt:
  - Graphik-Programmierung (GUI) → systemunabhängig
  - Netzwerk
  - (File) I/O
  - Algorithmen und Collections
  - …

Java heute

- Basis für viele neue Technologien und Ansätze:
  - Aspektorientierte Programmierung
  - Model driven architecture
  - …
- Web-Technologien basieren häufig auf Java
- Insbesondere auch, weil Windows und „Unix“ durch Linux mehr denn je parallel bedient werden müssen
- Java nicht an Plattform/Hersteller gebunden

Java & OpenVMS

- JA! → relativ aktuell: 5.0.3 (AXP), 5.0.2 (IA64)
- bei mir 1.4.2.5 (ohne Probleme)
- freier Download (HP OpenVMS eBusiness Bereich)
- Installation kann auf ODS-2 erfolgen, Quellen und Projekte immer auf ODS-5 ratsam (Namenskonventionen)
- Speicherfresser!
- Ausführungsgeschwindigkeit akzeptabel, Problem: Dateizugriffe Klassen (Suche der Klassen und Pakete → RMS lässt grüßen)
- Herkunft Unix lässt sich nicht (immer ganz) verleugnen
- Netbeans 3.6 unter OpenVMS (langsam!)
- unbedingt lesen:
  - Optimizing Java™ Technology Software Performance on HP OpenVMS – Tips and Tricks for Users
  - Memory, File I/O, File name translation, …

Literatur

- beliebig viel (auch sehr schlechte)
- „Core Java™ 2“ Horstmann/Cornell → meine „Bibel“
- „The Java™ Programming Language“ Arnold/Gosling/Holmes → gute, knappe Einführung
- „The Java™ Tutorial“ Campione/Walrath/Huml → „Schwimmring“

.NET/C# (1)

- Meine „Entwicklung“: C → C++ → MFC → Java → .NET/C#
- MFC/C++ immer irgendwo Stückwerk
  - Verschiedene Ansätze bei Klassenbibliotheken und Templates, mehrere Implementierungen
  - MFC verschleiert oft, was passiert
  - Parallel zur MFC andere „Mittel“ (Schnittstellen, Modelle, Begriffe), dadurch „Vermischung“
- MFC nicht portabel

.NET/C# (2)

- Java neu und „clean“ → sauberes Modell von Klassen, zwar oft sehr „theoretisch“, aber nicht so ein „Chaos“, wie MFC oder C++ unter Windows
- Java wahr mit von Anfang an „angenehm“ ☺
- Nach erstem Kontakt mit .NET/C#:
  - Ups!, da kommt einem doch vieles bekannt vor!
  - Viele gute Ideen, die man bei Java findet (Bytecode, sauberes Klassenmodell, …) auch hier umgesetzt
  - Viele Ideen, die VMS schon sehr lange kann (Interoperabilität der Sprachen, CLR = system services?)

Fazit (?)

- Java ist „hot“
  - modern, „akademisch“, elegant, Plattform-übergreifend
- Java ist kein Allheilmittel
  - Erfolg hängt von Projektplanung, -management und der Disziplin der Programmierer ab
  - C/C++ ist „nicht tot“, C# ist wie Java
- Java Bedeutung als Technologie-Basis
- Java dort, wo man portabel sein möchte

Fragen & Diskussionen

- Fragen
- Hinweise
- Tipps