8 Programmieren mit mehreren Programmiersprachen

Zu den besonderen Kennzeichen der modernen, kollaborativer Programmierung gehört es, dass man nicht mehr in nur einer einzigen Programmiersprache entwickelt, sondern jeweils die für die Anwendung vorteilhafteste Sprache kennen muss.

»Die Grenzen deiner Sprache sind auch die Grenzen deiner Welt.«
»The limits of your language are the limits of your world.«

Ludwig Wittgenstein, deutscher Philosoph und Begründer der modernen Linguistik

8.1 Von Plattformen, Frameworks und Personalities

Beim Konkurrenzkampf der Anbieter moderner Softwarelösungen ist sehr viel die Rede von Plattformen, von Frameworks und von Personalities. Dabei versteht man unter *Plattform* die Hardware, mit *Framework* bezeichnet man die Laufzeitumgebung, unter der die Programme ausgeführt werden, und die *Personality* ist der »Charakter«, die verwendete Programmiersprache.

8.1.1 Plattformen

Für die Bedeutung des Begriffs »Plattform« gibt es zwei Definitionen:

- **Hardware-Plattform**
  Eine Plattform ist die Hardware-Umgebung, unter der eine bestimmte Applikation ausgeführt wird.
- **virtuelle Plattform**
  alternative Namen: virtuelle Maschine, Framework, Runtime, siehe auch *Framework*. Eine virtuelle Maschine, auf der eine Applikation ausgeführt wird und die sich der Applikation gegenüber wie eine reale Hardware-Plattform mit einem eigenen, abgeschlossenen Betriebssystem präsentiert.

Um die Mehrdeutigkeit hinter uns zu lassen, werden wir als »Plattform« grundsätzlich nur die Hardware-Umgebung bezeichnen. Folgende Plattformen sind im SAP-Umfeld relevant:

- **Intel-basierte Plattformen, auch Windows-NT-Plattform**
  Dies sind all die Plattformen, die mit einem Intel-kompatiblen Prozessor ausgestattet sind. In der Praxis ist das gleichbedeutend damit, dass wir einen Computer mit einer Windows-Variante als Betriebssystem haben.
- **Nicht-Intel-basierte Plattformen**
  Es geht hier um die Computer, die nicht das Betriebssystem Windows unterstützen. Dabei handelt es sich in der Mehrzahl um UNIX-Systeme, aber auch andere Betriebssysteme kommen hier durchaus in Frage. Die wichtigsten Betriebssysteme sind:
  - UNIX
    UNIX wird heutzutage von nahezu jeder Standardware unterstützt. Es ist jedoch zu beachten, dass jeder Hersteller seine Variante eines UNIX anbietet, die nicht in allen Details zueinander kompatibel sind, weder abwärts noch aufwärts.
  - Linux – die Open-Source-Variante von UNIX
    Linux ist eine weitere Variation von UNIX, hat jedoch durch die OSF eine ungewöhnlich hohe Verbreitung. Zwar gibt es auch bei Linux Dialekte, aber dadurch, dass Open-Source-Projekte Linux ihren Sourcecode grundsätzlich offen legen müssen, ist die Dialektvielfalt paradoxerweise geringer als zwischen proprietären Implementierungen des UNIX-Standards.
  - IBM AS/400
    Insbesondere in der produzierenden Industrie mit einer hoch automatisierten Maschinensteuerung finden sich noch viele AS/400-Computer. Auf diesen Computern läuft sehr häufig über viele Jahre gewachsene Software, die hoch spezialisiertes Know-how beinhaltet oder gesondert entwickelte Anpassungen an die zu steuernde Maschine enthält. Da diese Anwendungen nur schwer zu portieren sind, ist es sehr kompliziert, diese Computer zu ersetzen.
  - Mac OS – Apple Macintosh
    Der Mac ist immer noch der Exot unter den Personal Computern und wird es wohl auch bleiben. Dennoch gibt es Industriezweige, zum Beispiel die Werbe- und Druckerei-Branche, die auf diese Geräte setzen. Dank der Webtechnologie lässt sich ein Mac heutzutage auch in Enterprise-Netzwerke einbinden.
  - Mainframe
    Alles, was von den klassischen Mainframes übrig geblieben ist, sind die *S/390*-Rechner von IBM, deren Nachfolger, die *z-Series*, und

eine Anzahl von Klonen, die diese Rechner nachbauen. Auch für die Mainframes ist es heute nicht mehr opportun, den Geist aus der Flasche zu lassen und sich mit den Protokolldetails zu befassen. Stattdessen kommuniziert man auch mit Mainframes über Webservices.

8.1.2 Frameworks und Personalities

Framework = Laufzeitumgebung

Ein Framework stellt der Applikation eine Laufzeitumgebung zur Verfügung, die bereits den größten Teil der immer wiederkehrenden Funktionalitäten implementiert. Das garantiert eine einheitliche Gestaltung der Applikationen auf dem Framework, zum Beispiel ein gemeinsames Look-and-Feel der Oberfläche oder ein standardisiertes Verfahren, auf eine Datenbank oder ein Dateisystem zuzugreifen. Gleichzeitig liefert ein Framework alle Werkzeuge, um Programme für die Runtime-Umgebung aufzubereiten, zum Beispiel Direkthilfen für die Nutzung der Runtime-Bibliotheken.

Frameworks ersetzen das Betriebssystem

Frameworks sind mittlerweile zu einer strategischen Basis für Enterprise-Applikationen geworden. Durch Frameworks ist *Rapid Application Development* (RAD) erst möglich geworden. Für einen Entwickler ersetzt das Framework weitgehend das Betriebssystem.

Wahl der Programmiersprache

Es hat sich mittlerweile durchgesetzt, dass man eine Applikation nicht notwendigerweise in einer einzigen Programmiersprache programmieren muss. Vielmehr unterstützen moderne Frameworks verschiedene Programmiersprachen und Programmierkonzepte, zum Beispiel objektorientiert oder sequenziell, imperativ oder abfrageorientiert. Damit ist die Wahl der Programmiersprache keine Design-Entscheidung mehr, sondern kann sich nach den Fähigkeiten oder Vorlieben des Entwicklers richten. Je nachdem, in welcher Programmiersprache oder nach welchem Programmiermodell man entwickelt, spricht man von einer *Personality* des Frameworks.

Hier sind Beispiele für Frameworks und die enthaltenen Personalities:

- **Microsoft.NET-Framework**
  Das Microsoft.NET-Framework ist eine stabile, auf Transaktionen basierende Laufzeitumgebung für NT und Windows 2000 und die konsequente Weiterentwicklung von Microsofts COM+-Architektur, die wiederum der Zusammenschluss von COM und MTS, dem *Microsoft Transaction Server,* ist. Microsoft.NET unterstützt eine Anzahl von Programmiersprachen, die jede für sich genutzt, aber auch gemischt verwendet werden können. Der Kern von Microsoft.NET ist die Common Language Runtime (CLR), welche die eigentliche *Virtual Machine* von Microsoft.NET darstellt und in etwa der Java VM entspricht.

- **SAP Web Application Server (ABAP Personality)**
  Der SAP Web Application Server (Web AS) ist der um einen Webserver erweiterte Kernel von SAP R/3 und auch bekannt als SAP-Basis Release 6.x. Anders, als der Name Web AS suggeriert, handelt es sich nicht um einen reinen webbasierten Applikationsserver, sondern um die verbesserte, vollständige Entwicklungs- und Laufzeitumgebung von ABAP.
- **Java Virtual Machine (JVM)**
  Bei Java muss man etwas weiter ausholen, denn Java machte die virtuellen Maschinen wieder hoffähig (auch wenn ABAP noch nie anders funktionierte). Die virtuellen Maschinen waren immer das hässliche Entlein der Entwicklerwelt, immer geschmäht, weil sie angeblich zu groß, zu langsam und zu unflexibel waren, aber wenn sie, wie im Falle von ABAP, auf der richtigen Plattform liefen, zeigte sich ihre wahre Überlegenheit.

  Es macht zwar den Anschein, als ob mittlerweile jeder Programmierer Java beherrsche, aber das ist wohl ein Trugschluss. Deshalb hier eine kleine Einführung auch in die Konzepte von Java.

  Wenn man von Java sprechen, meinen man damit in Wirklichkeit drei verschiedene Dinge:

  - die Programmiersprache Java
    eine an die Syntax von C++ angelehnte Programmiersprache
  - Die *Java Virtual Machine*, JVM
    die Runtime-Umgebung, auf der Java ausgeführt wird
  - Java Applets
    eine besondere Version von Java-Runtime-Code, die dafür gedacht ist, über einen Browser geladen und auf der Workstation ausgeführt zu werden

Java ist ein P-Code-Compiler

Das Konzept von Java sieht vor, dass in Java geschriebene Programme (Dateisuffix: *.java*) durch einen Compiler in einen optimierten Zwischencode übersetzt werden (Dateisuffix: *.class*). Der Zwischencode wird dann beim Aufrufen der Klasse durch die Runtime interpretiert und ausgeführt.

Kein Compile-on-Demand

Grundsätzlich ist das das gleiche Prinzip wie bei ABAP. Allerdings merkt Java selbst nicht, wenn sich der Sourcecode geändert hat, demnach compiliert Java nicht automatisch den Zwischencode nach jeder Änderung neu.

Sprache und Virtual Machine sind unabhängig

Es ist wichtig, sich dieses Unterschiedes bewusst zu sein, denn viele Missverständnisse bei Diskussionen um Java entstehen dadurch, dass man die beiden Entitäten durcheinander bringt. Es wäre zum Beispiel völlig problemlos, Compiler für andere Programmiersprachen zu schreiben, die als Ergebnis einen JVM-kompatiblen Zwischencode erzeugen. Andererseits ist auch ein Compiler denkbar, der aus Java-Code eine ausführbare Datei oder den Zwischencode für eine andere Umgebung erzeugt. Microsoft.NET macht gerade dies mit der Sprache J#, hinter der sich Java-ähnliche Konzepte verbergen.

Java gehört in die Gruppe der reinen objektorientierten Sprachen und wurde als solche von SUN Microsystems entwickelt. SUN hat sich den Namen Java und Design der Sprache Java mit einem Copyright schützen lassen, so dass SUN die volle Kontrolle über die weitere Entwicklung und den Einsatz von Java behalten kann. Insbesondere möchte man damit die zu starke Dialektbildung der Sprache verhindern, aber auch sicherstellen, dass niemand mit dem Namen Java werben darf, der nicht die Zertifizierung von SUN besitzt.

8.2 Programmiersprachen und Entwicklungsumgebung

Die Auswahl der richtigen Programmiersprache gehört zu den kritischen Entscheidungen innerhalb eines Projekts. Von der Wahl der Programmiersprache hängt oft ab, auf welcher Plattform die Applikation ausgeführt werden kann. Andererseits kann, wie im Falle von Microsoft.NET, auch die Plattform die Wahl der Programmiersprache diktieren.

Ein zu großes Gewicht wird jedoch immer dem Punkt der Portabilität beigemessen. Java ist portabel, weil es grundsätzlich auf fast allen Computerplattformen läuft. Die Portabilität spielt jedoch im Enterprise-Computing eine sehr geringe Rolle, denn wenn sich ein Unternehmen für eine Plattform entschieden hat, ist dies eine Entscheidung für Jahrzehnte. Bereits Upgrades und Release-Wechsel von Applikationen sind normalerweise mit so vielen Risiken verbunden, dass man die Plattform beziehungsweise das Framework, auf dem eine Applikation einmal läuft, nicht schnell wechselt. Viel bedeutender sind folgende Aspekte:

- Erfahrung der Entwickler mit der Programmiersprache
- Kompatibilität mit vorhandenen Diensten
- Lesbarkeit der Programme und Nachvollziehbarkeit der Programmlogik
- Zuverlässige Entwicklungsumgebung mit sehr gutem Debugger

ABAP entscheidend für Erfolg von R/3

Es steht außer Zweifel, dass der Erfolg von SAP R/2 und von SAP R/3 entscheidend durch die Programmierumgebung ABAP geprägt wurde. Dabei gab weniger die Programmiersprache ABAP an sich den Ausschlag als die ausgereifte und am Zweck orientierte Entwicklungsumgebung.

8.2.1 Die Entwicklungsumgebung

Volle Integration des Data Dictionary

Die ABAP-Entwicklungsumgebung basiert auf einem vollständigen Data Dictionary, das schon zu R/2-Zeiten vom Programmeditor aus zugänglich war, in dem alle wesentlichen Entwicklungselemente durch vernetzte Verwendungsnachweise rasch erreichbar sind. Kaum eine IDE kann durch einfachen Doppelklick auf einen Tabellennamen in deren Definition springen oder durch Doppelklick auf den Namen eines Unterprogramms sofort alle Aufrufstellen anzeigen. Der Produktivitätsgewinn ist dadurch enorm, insbesondere bei der Fehlersuche, ganz zu schweigen davon, dass man Fehler wie inkompatible oder fehlende Parameter schon im Vorfeld erkennt.

8.2.2 Die Compile-on-Demand-Runtime

ABAP und Turbo-Pascal haben etwas Entscheidendes gemeinsam. Beide haben bewiesen, dass die theoretischen Ansätze im Compilerbau wenig praxistauglich sind. Borlands Philipp Kahn gelang der Beweis, dass nicht Pascal als Sprache langsam ist, sondern nur die Compiler schlecht. ABAP bringt den Beweis, dass nicht Runtime-Umgebungen langsam sind, sondern nur deren Implementierungen. Die Performance von SAP R/3 wird nicht durch die Runtime-Umgebung ausgebremst.

Runtime garantiert Unabhängigkeit von der Plattform

Dafür garantiert die Runtime-Umgebung, dass SAP R/3 unabhängig von der Hardware-Plattform und dem Betriebssystem bleibt und dass Programme sofort produktiv zur Verfügung stehen, wenn sich der Sourcecode oder eines von dessen Elementen ändert. Dies nennt man heute *Compile-on-Demand*, auch wenn es den Begriff noch gar nicht gab, als ABAP dies schon konnte.

8.2.3 Der Debugger

Debugger sind Schlüssel zum Erfolg

Ein hervorragender Debugger ist ein wichtiger Teil einer Entwicklungsumgebung. Er dient vornehmlich der Fehlersuche, aber auch der Qualitätskontrolle. Das Nachvollziehen jedes einzelnen Programmschrittes unter Beobachtung ausgewählter Variablen (*Watches*) erlaubt Einsichten in den Programmablauf, die im bildlich analogen Sinne den Unterschied zwischen Tag und Nacht machen: Ohne Debugger läuft Ihr Programm im Dunkeln ab, während mit Debugger Sie das Programm beobachten können.

8.2.4 Einrichten einer Entwicklungsumgebung

Bevor Sie anfangen zu entwickeln, sollten Sie sich zunächst eine einigermaßen stabile Entwicklungsumgebung installieren. Vor allem sollten Sie sich das SAP GUI komplett auf Ihrer Arbeitsstation einrichten, da dieses fast alle RFC-relevanten Tools enthält.

Komplette SAP GUI-Installation

Es ist für einen Entwickler grundsätzlich empfehlenwert, das SAP GUI komplett mit allen Zusätzen zu installieren. Es kann auch nicht schaden, sich die neueste Version des SAP GUI noch einmal über einen bestehenden SAP GUI zu installieren, da dann sichergestellt ist, dass auch keine der SAP GUI-Komponenten von anderen Installern zwischenzeitlich überschrieben wurde. Wenn Sie bisher noch nicht mit RFC von Fremdsystemen gearbeitet haben, ist es auf alle Fälle empfehlenswert, die ersten und auch zweiten Schritte mit einem Windows-System zu machen. Zwar geht auch fast alles von UNIX aus, aber dort sind Sie weitgehend auf Java beschränkt und viele Dinge sind deutlich schwieriger zu realisieren.

Installation eines Webservers

Sie müssen einen Webserver installiert haben, der Ihre Testbeispiele auch verwalten und ausführen kann. Dabei haben Sie die Wahl zwischen Microsoft Internet Information Server, der Bestandteil jedes Windows 2000 Professional-Systems ist. Für ältere NT-Systeme gibt es eine Version von IIS im NT-Options-Pack.

Für Entwicklungen in Java können Sie entweder auf Apache Jakarta Tomcat (*www.apache.org*, Projekt Jakarta) oder auf einen anderen der kommerziellen Webserver, zum Beispiel IBM WebSphere, zurückgreifen. Tomact 4.0 basiert übrigens auf der Version von IBM WebSphere 4.0.

IDE installieren

Wenn Sie ASP-Seiten für den IIS entwickeln wollen, müssen Sie sicherstellen, dass Sie eine aktuelle Version des *Windows Scripting Host* (WSH) zum Ausführen der Visual-Basic-Script-Befehle (VBS) verwenden. Alle Scripting-Befehle für VBS und Jscript innerhalb einer Active Server Page werden an den WSH weitergereicht. WSH wiederum kann dann eine beliebige Windows-Applikation oder ein DCOM/ActiveX-Objekt aufrufen.

Microsoft.NET

Um richtige, transaktionsfähige Applikationen zu entwickeln, bietet sich dann das Microsoft.NET-Framework an. Dieses ist die Zusammenführung der alten COM-Technologie (auch *OLE/ActiveX* genannt) und des *Microsoft Transaction Server*, MTS, mit der *Common Language Runtime*, CLR.

8.2.5 Plattformentscheidung: UNIX versus Windows NT

Wie bereits erwähnt, ist die Entscheidung für eine Plattform eine Entscheidung für ein Jahrzehnt. Einmal getroffen, werden Sie die nächsten zehn Jahre damit leben müssen. Dennoch ist die Entscheidung für die eine oder andere Umgebung weniger dramatisch, als es gerne dargestellt wird. Sie sollten sich zur Entscheidungsfindung folgende Fragen stellen:

- **Passt der neue Rechner in die alte Landschaft?**
  Die Frage ist von hoher finanzieller Bedeutung. Wenn Ihr Unternehmen zum Beispiel nur UNIX-Rechner in Betrieb hat, erfordert bereits das Hinzufügen eines einzigen Windows-Rechners, dass Sie Ihr Personal für die Hardware und die Applikationen schulen. Auch müssen die Verbindungen zwischen den Systemen funktionieren und Massen-Lizenzen werden oft getrennt nach Betriebssystem berechnet.
- **Wird es in Zukunft genug ausgebildetes Personal geben?**
  Je mehr Fachleute es für eine bestimmte Technik auf dem Markt gibt, desto schneller können Sie Personalengpässe ausgleichen und desto leichter finden Sie einen Spezialisten für gezielte Maßnahmen. Hier haben Microsoft und Linux ganz deutlich die Nase vorn, denn allen anderen Betriebssystemen, auch dem klassischen UNIX, laufen die guten Leute weg. Derzeit finden sich deutlich mehr Spezialisten, die detailliert die Microsoft-Palette beherrschen, als für Linux.
- **Wie sind die Unterhaltskosten im Vergleich?**
  Auch hier gibt es kaum einen Unterschied zwischen NT und UNIX. Durch die Annäherungen von PC und Mainframe und den Trend, lieber mehrere kleine als einen ganz großen Computer einzusetzen, ist die Hardware-Plattform weitgehend identisch, wenn auch Microsoft auf Intel-Systeme begrenzt ist.
- **Für welche Plattform gibt es die meisten Utilities?**
  Hier gibt es klare Vorteile für Windows. Da die meisten brauchbaren Tools nicht explizit für eine Server-Umgebung, sondern für die breite Masse der Desktop-Nutzer entwickelt wurde, ist das Angebot für Windows-Utilities riesenhaft im Vergleich zu UNIX.

8.2.6 Framework: Web AS/ABAP, Microsoft.NET oder J2EE

Bei der Entscheidung für das Framework erweist sich derzeit der SAP Web AS mit der ABAP Personality als der Favorit. Das kann sich noch ändern, wenn Microsoft seine Microsoft.NET-Plattform auch auf Linux-Systemen herausbringt und somit die Begrenzung auf Intel-Rechner beziehungsweise auf Macintosh überwindet.

Jede Menge Tool für Microsoft

Microsoft.NET hat eine hohe Flexibilität bei der Auswahl der Programmiersprache und die bei weitem größte Palette an Hilfsmitteln und Applikationen, die durch COM benutzt werden können. Für Textformatierungen verwendet man die Controls aus Microsoft Word, für komplexe Berechnungen greift man auf Excel zurück, und das Erzeugen von PDFs überlässt man den Destiller-Tools von Adobe oder dem *Fineprint PDFmaker*. Im Grunde lässt sich jedes moderne Windows-Programm von einem anderen Programm steuern und somit durch Microsoft.NET unmittelbar nutzen.

J2EE ist unabhängig von der Plattform

J2EE läuft auf fast allen Betriebsystemen, nicht nur auf Windows und UNIX, sondern auch auf Großrechnern. Außerdem wird J2EE von WebSphere und Tomcat unterstützt, nicht aber von IIS. Allerdings ist dieser hohe Grad an Portabilität nur nachrangig für ein einzelnes Unternehmen, das sich für eine Plattform bereits entschieden hat. Das hohe Maß an Unterstützung von Open-Source-Projekten spricht für J2EE.

WebAS hat höchste Flexibilität und Portabilität

Der SAP Web AS ist das Framework mit dem höchsten Grad an Flexibilität, da es auf dem bewährten SAP R/3-Kernel aufbaut und in ABAP entwickelt ist. Die BSP stellen den vollen Funktionsumfang von Server Pages zur Verfügung, bieten demnach die Möglichkeit, Webentwicklungen aus Java direkt zu übernehmen.[1] Gleichzeitig steht das SAP R/3-Transaktionsmanagement in vollem Umfang zur Verfügung.

Web AS nutzt das Transaktions-Management

Damit ist der WebAS das einzige Framework, das ein in der Praxis erprobtes und auf die realen Bedürfnisse zugeschnittenes Transaktionsmanagement und Applikationsserver in den Webserver integriert. Zwar bieten auch J2EE und Microsoft.NET diese Möglichkeiten, aber es gibt bis heute keine Business-Applikationen, die eine solche Verbreitung haben, dass sie in der Lage wären, einen Standard zu etablieren.

Anwendungen treiben einen Standard voran

Es darf nämlich niemals vergessen werden, dass alle Frameworks nur das Vehikel für eine Applikation bereitstellen. Um diesen zum Erfolg zu verhelfen, muss das Vehikel auch einen praktischen Nutzen haben. Der Erfolg von SAP R/3 gründet darauf, dass es nicht leere Datenbanken und Entwicklungsumgebungen ausliefert, sondern dazu auch funktionierende Anwendungen. Es müssen also Anwendungen auf den Frameworks entstehen, die eine Verbreitung in der Praxis finden. In diesem Punkt ist der SAP Web AS klar im Vorteil.

1 Mit dem Realese 6.30 wird auch die Implementierung der J2EE-Laufzeit- und Entwicklungsumgebung vollständig abgeschlossen sein, so dass auch auf dem SAP Web AS direkt in Java programmiert werden kann.

8.2.7 Windows Scripting Host versus Visual Basic

Der *Windows Scripting Host* (WSH) ist eine Klassenbibliothek und beinhaltet auch den VBScript-Interpreter. Der WSH ist ein Add-On zu Windows und findet sich in dem EXE-File *WSCRIPT.EXE*. Es gibt auch eine Version für DOS mit dem Namen *CSCRIPT.EXE*.

Sie sollten sich vergewissern, dass Sie jeweils die neueste Version von VBS haben. Sie können Versionen bei Microsoft herunterladen unter *http://msdn.microsoft.com/scripting*.

VBS sind Textdateien

Alle Eingaben für VBS sind einfacher ASCII-Text, das heißt, die Programme können mit einem beliebigen Texteditor erfasst werden. VBS-Programme werden an der Datei-Erweiterung *.vbs* von Windows erkannt und durch WSCRIPT ausgeführt. Andere Betriebssysteme wie UNIX haben ähnliche Script-Prozessoren, für UNIX gibt es zum Beispiel RSH und Apache benutzt die Scriptsprache Perl.

Folgendes charakterisiert die Sprachen Visual Basic, Visual Basic Script und Visual Basic for Applications:

- **Visual Basic**
  Visual Basic als Compiler-Variante ist Bestandteil von Microsoft Visual Studio, der IDE von Microsoft. Mit VB können Sie eigene COM- und EXE-Objekte erzeugen.
- **Visual Basic Script**
  VBS ist eine eingeschränkte Interpreter-Version von Visual Basic. Die Sprachelemente sind auf dem Niveau von Visual Basic 3 und erlauben keine typisierten Variablen. Zum Ausführen von VBS-Scripts benötigt Windows den Windows Scripting Host, der in den EXE-Dateien *Cscript.exe* (für DOS) und *Wscript.exe* (für Windows) implementiert ist.
- **Visual Basic for Applications**
  Vollständige Interpreterversion von VB als Bestandteil von Microsoft Office. Mit VBA ist fast alles möglich, was mit VB möglich ist, außer dem Erzeugen von kompiliertem Code.

8.2.8 Windows Classic versus Microsoft.NET

Microsoft Professional Office Suite

Vielleicht stellen Sie sich die Frage, warum sich jemand noch mit klassischem Visual Basic und insbesondere dem Windows Scripting Host befassen soll, nachdem doch Microsoft.NET der neue Standard für alle Entwicklungen ist. Leider laufen noch lange nicht alle Applikationen unter

Microsoft.NET. Insbesondere hat Microsoft Office eine Variante von Visual Basic, das Visual Basic for Applications (VBA) als festen Bestandteil eingebaut.

VBA eignet sich für Rapid Prototyping

Die Entwicklungsumgebung von VBA ist sehr an die von Visual Studio angelehnt und durch die jeweiligen Kernfunktionen der Office Suite, wie Textverarbeitung und Tabellenkalkulation, eignet sich VBA hervorragend als Tool für Rapid Development. Auch lassen sich damit fast alle ASP-Entwicklungen elegant und einfach durchführen. Dies ist besonders interessant, wenn Sie von ASP aus nur auf vorgefertigte OLE-Komponenten zugreifen wollen, was zum Beispiel der Fall ist, wenn Sie mit SAP-BAPIs programmieren wollen, kurz gesagt, wenn Sie schnell etwas entwickeln wollen. Der Debugger hat die gleiche volle Funktionalität wie auch Visual Studio. Der einzige Nachteil ist, dass VBA keine kompilierten COM-Objekte erzeugen kann. Hierfür müssen Sie dann doch Visual Studio oder Borland Delphi heranziehen. Für komplexe Neuentwicklungen empfehlen wir Delphi oder Microsoft.NET.

8.3 Die Wahl der richtigen Programmiersprache

Für Webentwicklungen kommen eine ganze Reihe von Programmiersprachen in Frage. In einer modernen Client-Server-Umgebung geht es dabei nicht darum, sich für eine Sprache für die komplette Architektur zu entscheiden, sondern die Sprache für den jeweiligen Zweck auszuwählen.

8.3.1 Sprache für den Webserver

Die Programmiersprache zum Entwickeln von dynamischen Webseiten wird in der Regel durch den Webserver bestimmt. Für den IIS ist das Visual Basic Script (VBS), die auf WebSphere basierenden Server (WebSphere und Tomcat) unterstützen zunächst nur Java. Zwar sind Plug-Ins für weitere Programmiersprachen bei allen Webservern vorgesehen, aber dies ist in der Praxis unrealistisch, denn damit würde man in den Kern des Webservers eingreifen und genau das ist ein Tabu.

ASP.NET unterstützt alle .NET-Sprachen

Mit ASP.NET ist die neue Vielfalt für Sprachen eingezogen, denn der .NET-IIS unterstützt zunächst die Common Language Runtime (CLR) und damit alle Microsoft.NET-Sprachen die den Zwischencode kompatibel zur CLR erzeugen. Damit haben Sie zunächst einmal die Wahl zwischen C#, J# und VB.NET, demnächst wohl auch noch X#, einer LISP-ähnlichen Sprache zum Programmieren in XML-Dokumenten.

8.3.2 Java

Java an sich dürfte mittlerweile in der Entwicklergemeinde hinreichend bekannt sein. Die Sprache wurde vom kalifornischen Hardware-Hersteller SUN als plattformunabhängige Sprache entwickelt. Über den schon vom amerikanischen Verteidigungsministerium bei der Sprache ADA angewandten Trick, die Sprachdefinition und Sprachbezeichnung einem Copyright zu unterwerfen, versucht SUN, die Kontrolle über die Evolution von Java zu behalten.

8.3.3 J#

J# ist die Java-Implementierung von Microsoft und ein Dialekt des Originals. Wegen des erwähnten Copyrights auf Java und der damit verbundenen Dauerfehde zwischen Microsoft und SUN entstand der neue Name.

8.3.4 Visual Basic (VB) und Visual Basic for Applications (VBA)

Visual Basic basiert auf der alten Basic-Programmiersprache, ist aber mittlerweile zu einer mächtigen, prozeduralen Programmiersprache gereift. VB ab dem Release 6 unterstützt viele Ansätze der Objekt-Programmierung, allerdings nicht den Polymorphismus und das Overloading. Für Entwicklungen, die gezielt für Windows-Plattformen gemacht werden, ist VB die Programmiersprache der Wahl. Man muss aber dabei eingestehen, dass die Objektansätze vor allem in der COM-Struktur liegen.

Visual Basic for Applications

Visual Basic for Applications (VBA) ist eine fast vollständige Interpretervariante von VB und Bestandteil aller Microsoft-Office-Anwendungen, zum Beispiel von Microsoft Word, Excel, Access 2000 oder Visio. Es beweist sich, dass man mit VBA und Excel sehr effiziente Prototypen für Windows entwickeln kann. Wenn sich die Anwendung stabilisiert hat, kann man sie dann mit VB in eine Windows-DLL kompilieren.

8.3.5 Visual Basic Script (VBS) und ASP

Visual Basic Script ist eine Untermenge von VB. Es ist eine reine Interpreter-Sprache und unterscheidet sich vor allem dadurch von VB, dass es keine typisierten Variablen unterstützt. Der Typ der Variablen wird erst zur Laufzeit bestimmt und die passende Datenstruktur auf dem Stack angelegt. Folgende Deklaration geht also nicht in VBS:

```
Dim LogonControl As SAPLogonCtrl.SAPLogonControl
Dim Functions As SAPFunctionsOCX.SAPFunctions
Dim TableFactory As SAPTableFactoryCtrl.SAPTableFactory
```

Stattdessen können wir nur deklarieren:

```
Dim LogonControl
Dim Functions
Dim TableFactory
```

Entsprechend geht auch die Instanzziierung eines Objekts nur über die explizite Angabe des Verweises auf die Windows-Registry. Es geht also nicht:

```
Set LogonControl = New SAPLogonCtrl.SAPLogonControl
Set Functions = New SAPFunctionsOCX.SAPFunctions
Set TableFactory = New SAPTableFactoryCtrl.SAPTableFac-
tory
```

Stattdessen müssen wir die Objekte entsprechend den folgenden Beispielen erzeugen.

```
Set LogonControl =
CreateObject("SAP.LogonControl.1")
Set Functions = CreateObject("SAP.Functions")
Set TableFactory =
CreateObject("SAP.TableFactory.1")
```

8.3.6 C#

C# ist Microsofts Zugpferd, um die Java-Fangemeinde auch für Windows-Plattformen zu gewinnen, ohne sich dabei zu sehr von SUN abhängig machen zu müssen. Offiziell ist C# trotz der auffallenden Ähnlichkeit mit Java als Nachfolger von C++ positioniert, was aber vor allem den Sinn hat, den rechtlichen Problemen mit dem Copyright auf Java aus dem Weg zu gehen.

8.3.7 X#

X# existierte nur als Konzept, als dieses Buch in Druck ging, weshalb Beispiele hier nur von begrenzter Haltbarkeit sind. X# basiert auf XML und ist ein Ansatz, XML-Dokumente als Teil eines Programms zu sehen und nicht als manipulierbares Objekt, wie es die anderen Programmiersprachen wie Java oder VB tun. Das wird dann etwa wie folgt aussehen:

Gehen wir von dem simplen Animal-Farm-Beispiel aus:

```
<FARM></FARM>
```

Mit einer X#-Sprache könnte das Programm dann etwa so aussehen:

```
Set XMLDocument = new MSXML.Document()
For each ANIMAL in XMLDocument
For each NAME in ANIMAL
       Write NAME.text
Next
Next
```

Wie gesagt, so *könnte* X# einmal aussehen, veröffentlichte Prototypen gibt es bislang nicht.

8.3.8 Delphi

Reiches Erbe

Delphi ist von allen Programmiersprachen die bei weitem mächtigste. Entstanden aus dem Klassiker Turbo-Pascal, war Delphi zu jeder Zeit die Best-of-Breed-Sprache, die die jeweiligen Konzepte zuverlässig und sauber implementiert hat. Delphi erbt also von

- **Pascal und VB**
  Delphi ist durch die Pascal-Syntax einfach lesbar.
- **Java**
  Delphi hatte alle OOP-Konzepte schon frühzeitig implementiert, insbesondere
  - Vererbung
  - Polymorphismus
  - Overloading
  - Interfaces
- **C und Pascal**
  Delphi ist aber auch prozedural, erlaubt also auch prozedurale Programmierung, falls nötig.

Für die Entwicklung komplexer Applikationen ist Delphi zu jeder Zeit die erste Wahl.

8.3.9 ADA

Produkt des Verteidigungsministeriums

Vielleicht wäre ADA groß herausgekommen, wenn nicht ausgerechnet das US-Verteidigungsministerium der geistige Vater wäre. ADA wurde 1979 in dessen Auftrag von dem französischen Software-Giganten Honeywell-Bull entwickelt und hatte bereits bei seinem Entstehen 1979 sämtliche Konzepte des OOP realisiert. Gleichzeitig hatte man sich aber

von den kryptischen Ansätzen anderer Sprachen, allen voran C, gelöst und entscheidenden Wert auf Lesbarkeit gelegt. Auch die Idee, dass man eine Programmiersprache mit einem Copyright belegen könnte, um die Evolution in den Griff zu bekommen, stammte aus dem ADA-Projekt.

ADA fand jedoch niemals große Verbreitung in der Industrie, was vermutlich weniger dem Misstrauen dem Verteidigungsministerium gegenüber zu verdanken ist als der Tatsache, dass sich kaum ein einziger brauchbarer und gleichzeitig günstiger ADA-Compiler auf dem Markt wiederfand. Dadurch blieb ADA trotz seiner Vorzüge immer eine Sprache für eine Elite und finanzstarke Software-Schmieden, während sich die Heerscharen von jungen, kreativen und privaten Software-Entwicklern zunächst mit Turbo-Pascal zufrieden gaben und dann an dem kostenlos vertriebenen Java Gefallen fanden.

8.3.10 SmallTalk

SmallTalk ist der Urvater der OOP und realisiert den Objekt-Ansatz in radikaler Form. Die legendäre Apple Lisa besaß damals ein Betriebssystem, das vollkommen in SmallTalk geschrieben war. Dem Lisa-Anwender war das aber egal, der merkte nur, dass das Betriebssystem ziemlich langsam war.

8.3.11 LISP

LISP war eigentlich schon fast vergessen, aber die Ankündigung von X# brachte es wieder auf den Tisch. LISP ist eine Sprache, die ein Programm als hierarchisches Dokument ansieht und zwischen Daten und Programm keinerlei Unterscheidung trifft. Ursprünglich wurde LISP ausschließlich in der Künstlichen-Intelligenz-Forschung verwendet, vor allem zur Manipulation von neuronalen Netzen. Mit der zunehmenden Verbreitung von XML wird das Konzept LISP wieder für die breite Masse interessant.

Ein XML-Dokument ist eine Liste. Verbindet man dieses mit einem XSL-Stylesheet, wird in Wirklichkeit ein Programm auf diesem Dokument ausgeführt mit dem Ziel, das Dokument in eine andere Gestalt zu überführen. Genau das ist auch der Ansatz von LISP. Sobald man erlaubt, dass sich das XSLT-Stylesheet auch selbst manipuliert, hat man die Grenze zwischen Programm und Daten durchlässig gemacht.

8.4 Synopsis der Objektprogrammierung mit Java, VB und ABAP

8.4.1 Grundbegriffe

Programmieren mit Objekten

Objektorientierung ist Form des Denkens

Objektorientierung ist eine Form des Denkens. Ihr Grundgedanke ist die harmonische und gleichberechtigte Zusammenarbeit einzelner Programmteile auf der Basis von Nachrichten und Ereignissen. Im strengen Sinne werden Objekte und ihre Methoden nicht mehr aufgerufen, wie dies bei imperativen Programmiersprachen der Fall ist, sondern sie reagieren auf Ereignisse.

Sich selbst organisierende Einheiten

Das Denken in selbst organisierenden Einheiten stellt die Grundlage der objektorientierten Programmierung (OOP) dar. Das Wesen der OOP ist nicht gekennzeichnet durch Interfaces, Methoden, Properties, Instanzen, Lebenszeit, Procedure Overloading, Polymorphismus oder Datenkapselung. All diese schönen Dinge sind nichts weiter als »Features« und stellen Leistungen der verwendeten Entwicklungsumgebung und des zu Grunde liegenden Entwicklungssystems dar. Als solche waren diese Features schon immer in der einen oder anderen Form auch in den prozeduralen Sprachen vorhanden und spiegeln damit nur die Fortschritte auf dem Niveau der Entwicklungsumgebungen wider.

Messages sind das Herz der OOP

Damit kehrt die objektorientierte Programmierung auch den traditionellen Ansatz um: Anstatt eine Applikation in einem großen Programmhybrid zu erstellen, existieren nun unzählige kleine Objekte, die eine Gemeinschaft bilden und miteinander kommunizieren. OOP ist gekennzeichnet durch die Koexistenz von spezialisierten Objekten und deren Fähigkeit, auf *Messages* zu reagieren und selbst Aufgaben (*Tasks*) durch Austausch von Messages zu delegieren.

Interfaces

Das Interface eines Objekts ist eine Anzahl von Methoden, die zum Austausch von Informationen mit fremden, nicht vertrauenswürdigen (*not trusted*) Objekten dienen.

Interfaces bestimmen die Identität des Objekts

Eine wesentliche Forderung an ein Objekt-Interface ist es, dass das Interface die Objektklasse auch hinreichend charakterisieren sollte. Das bedeutet vor allem, dass das Objekt-Interface sich nicht mehr ändern darf, wenn die Klasse bereits von dritter Seite verwendet wird. Anders ausgedrückt: Ändert sich das Interface einer einmal veröffentlichten

Klasse, ist es nicht mehr dieselbe Klasse, auch wenn sie den gleichen Namen trägt. Die meisten Entwicklungsumgebungen tragen dem auch Rechnung, indem sie automatisch Versionen ziehen und die jeder Klasse eindeutig zugewiesenen GUID (*Global Universal Identifier*) neu vergeben.

Methode

Eine Methode ist die Korrespondenz zu einer Prozedur oder einer Funktion in klassischen Programmiersprachen.

Property

Eine Property oder Eigenschaft ist eine Sonderform einer Methode einer Klasse.

Instanz

Bevor wir ein Objekt verwenden können, muss es anhand einer Vorlage erzeugt werden. Die Vorlage eines Objekts heißt *Klasse* und das erzeugte Objekt ist eine *Instanz* der Klasse (Instanz: zeitweilige Existenz, englisch: *instance*). Das Erzeugen der Instanz nennt man *Instanziierung* und die Dauer der Existenz der Instanz ist ihre Lebenszeit.

Lebenszeit

Die Dauer der Existenz einer Instanz, das heißt die Zeit zwischen der Erzeugung und der Zerstörung der Objektinstanz, nennt man deren Lebenszeit (*life time*).

Overloading

Als *Overloading* bezeichnet man die Möglichkeit, eine beliebige Methode einer Klasse zu überschreiben, ohne die Klasse ändern zu müssen.

Polymorphismus

Unter *Polymorphismus* versteht man die Fähigkeit einer Laufzeitumgebung, die Identität eines Objekts nicht nur direkt durch dessen Namen, sondern auch durch eine eindeutige Zusammenstellung seiner Eigenschaften zu bestimmen.

Polymorphismus berücksichtigt Eigenschaften

Polymorphismus ist eine Dienstleistung des Compilers, die mit dem Aufkommen der objektorientierten Sprachen populär geworden ist. In den traditionellen formalen Programmiersprachen wie Pascal, C oder Basic

konnte man zwar problemlos eigene Unterprogramme und Funktionen deklarieren, aber wenn ein Name bereits einmal für eine Funktion vergeben war, konnte er an anderer Stelle nicht noch mal eingesetzt werden.

Nehmen wir als Beispiel eine typische polymorphe Prozedur, wie sie in fast allen Programmiersprachen existiert, etwa die `write`-Funktion zum Ausgeben von Variablen. Gewöhnlich kann eine solche `write`-Prozedur lediglich Zeichenfolgen ausgeben, andere Datentypen müssen zuvor in eine solche Zeichenfolge umgewandelt werden. Je nachdem, welchen Datentyp die auszugebende Variable besitzt, muss eine ganz andere Aufbereitungsroutine aufgerufen werden.

Welche Aufbereitungsroutine dann aufgerufen wird, entscheidet der Compiler oder die Laufzeitumgebung selbstständig. Sie müssen also niemals selbst unterscheiden, indem Sie etwa verschiedene Prozeduren wie `writeNum` oder `writeString` gezielt aufrufen.

Wie bereits gesagt, gab es solche Funktionen schon in vielen Programmiersprachen. Neu ist jetzt, dass man auch eigene polymorphe Objekte definieren kann. Es ist aber noch einmal festzuhalten, dass zwar der Polymorphismus von allen objektorientierten Sprachen verlangt wird, jedoch keine charakteristische Eigenschaft ist. Zum ersten Mal propagiert wurde der Polymorphismus in der 1970 vorgestellten Programmiersprache MODULA-2.

Beispiel Schauen wir uns noch einmal ein weiteres Beispiel an und nehmen folgende sinnlose Klasse .

```
public class test
{private String myName;
  public test(String initString)
  {myName = initString;}
```

Auffallend daran ist, dass der Constructor der Klasse parametrisiert ist.

```
  public test(String initString)
```

Wenn wir nun eine Instanz dieser Klasse anlegen wollen und wir vergessen, den Parameter zu übergeben,

```
myTest = new test()
```

meldet uns der Java-Compiler korrekterweise:

```
The constructor test is undefined
```

In der Regel führt er uns im ersten Moment in die Irre, denn der Constructor `test` existiert doch! Der Irrtum liegt darin, dass eine Methode in Java immer eindeutig durch den Namen, die Parameter und die Parametertypisierung bestimmt wird.

Datenkapselung

Mit der Datenkapselung gibt man dem Entwickler eines Objekts die Möglichkeit, die innerhalb der Prozeduren deklarierten Variablen und Methoden in öffentliche und private Objekte zu klassifizieren. Private Objekte können fortan nicht mehr von externen Programmen zugegriffen werden. Praktisch versteckt sich dahinter meistens der Zusatz `public` oder `private` in der Deklaration eines Objekts oder einer Variablen.

Klassen

Klassen sind Vorlagen für Objektinstanzen

Klassen sind Muster für die Anlage von Objekten. Jedes Mal, wenn ein Objekt erstellt wird, wird nach Vorlage der Klasse ein Speicherbereich reserviert. Dadurch können Objekte im Gegensatz zu klassischen Prozeduren und Funktionen in beliebig vielen Instanzen existieren. Ein Programm kann also mehrere Instanzen eines Objekts erzeugen und bekommt jedes Mal einen sauberen initialen Zustand des Objekts.

Klassen in Visual Basic

Jede COM-Bibliothek wird von Visual Basic automatisch als Klasse angesehen. Zusätzlich können Klassen jederzeit als Teil des aktuellen Projekts lokal deklariert werden.

Listing 8.1 Sample-Code einer Klasse

```
Public counter as Integer

Public Sub inc()
  counter = counter + 1
End Sub

Public Sub clear()
  counter = 0
End Sub
```

Klassen in ABAP

Klassen werden in ABAP mit dem Class Builder (SE24) angelegt und auch verwaltet. Einmal angelegt, sind die Klassen über das Repository für alle anderen ABAP-Programme sichtbar.

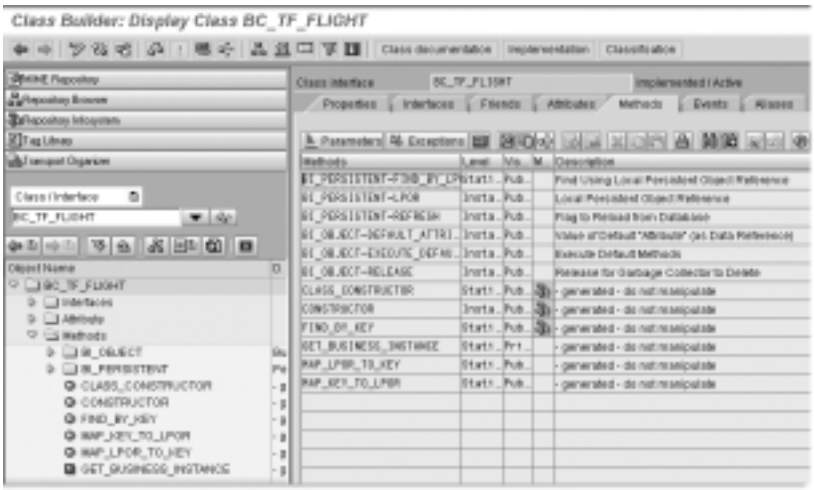

Abbildung 8.1 ABAP Class Builder

Klassen in Java

Jedes Java-Programm ist per Definition eine Klasse. Damit geht Java einen umgekehrten Weg wie andere Sprachen: Anstatt Klassen mit traditionellen Sprachmitteln zu emulieren (wie es auch C++ oder ABAP/IV mit Hilfe einer Makrobibliothek tun), erzwingt Java das Anlegen von Klassen. Klassische Konstrukte wie Unterprogramme oder Funktionen müssen als Klasseninstanz angesprochen werden.

Java gehört leider zu den Programmiersprachen, deren Programmaufbau alles andere als intuitiv ist. Das liegt daran, dass Java zum einen die kryptische C-Syntax übernommen hat und zum anderen die Sprache selbst auf einer Reihe von Konventionen aufbaut. Deshalb wollen wir uns das Anlegen einer Klasse etwas genauer ansehen, sozusagen als kurze Einführung in Java und als Erinnerungshilfe für die Gelegenheitsprogrammierer.

Java speichert jede Klasse in einer eigenen Datei

Das Konzept von Java sieht vor, dass jede öffentliche Klasse (*public class*) in einer eigenen Datei des Dateisystems abgespeichert wird. Mehrere Klassen lassen sich dann zu einem *Package* zusammenfassen. Die Dateien eines Packages können auch in einer komprimierten Archivdatei mit der

Endung *.jar* zusammengefasst abgespeichert werden. Grundsätzlich geht Java von der Annahme aus, dass die Verwaltung von Klassen die Sache des Betriebssystems sein muss, was eine sehr gute Auffassung ist.

»Hello World«

Als Beispiel wollen wir eine neue Klasse `HelloWorldClass` erzeugen. Zu dieser Klasse gehört eine Datei mit dem Namen *HelloWorldClass.java*, eine Textdatei, die den Code der Datei enthält. Da wir die Klasse wie ein Hauptprogramm direkt aufrufen wollen, implementieren wir eine Methode mit dem fest vorgegeben Namen `main`.

Listing 8.2 Java-Class HelloWorld

```
public class HelloWorldClass {
  public HelloWorldClass () {} /* Constructor */
  public static void main(String[] args) {
    System.out.println("Welcome to Java Hello World");
  }
}
```

```
public static void main(String[] args) {
```

Das Beispiel wird beim Aufruf aus der Kommandozeile den String `Welcome to Java Hello World` ausgeben. Die einzelnen Elemente des Beispiels lassen sich wie folgt sezieren:

- **Aufbau einer Klasse**
  Das Keyword `class` beginnt eine neue Klasse. Der Code der Klasse folgt dann in geschweiften Klammern ({}) dahinter.
- **Standard-Methode `main`**
  Die Klasse definiert eine einzige Methode mit dem Namen `main`. Der Name `main` ist ausschließlich reserviert für einen so genannten *Main Thread*, demnach ein Hauptprogramm. `main` ist der Name der Default-Methode, die die JVM versucht aufzurufen, wenn keine andere Methode explizit angegeben wurde.
- **`Void`**
  Das Keyword `void` zeigt an, dass die definierte Prozedur keine Werte zurückgibt. Demnach ist eine Prozedur mit `void` ein echtes Unterprogramm, während alle anderen einer Funktion entsprechen.
- **Kommandozeilen-Argumente**
  Im Beispiel besitzt die Methode `main` noch einen Parameter `String[] args`. In dieses Array von dynamischer Länge werden alle Parameter

übergeben, die beim Aufruf der Methode vom Aufrufer angegeben wurden. Die Anzahl der Parameter entspricht der Länge des Arrays und kann mit `args.length` ermittelt werden.

- **Java-Compiler**
  Bevor die Klasse ausgeführt werden kann, muss sie erst mit dem Java-Compiler `javac` übersetzt werden. Wenn die Klasse erfolgreich generiert wurde, kann man sie mit dem Java-Runtime-Utility `java` von einer Kommandozeile aus testen. `java` ruft dazu die JVM auf, erzeugt eine Instanz der Klasse und führt die Start-Methode aus. Mit dem `javap`-Utility erhält man Informationen über die Klasse.

Listing 8.3 Kompilieren der HelloWorld-Class mit der Java Command Line Utility

```
java -verbose HelloWorldClass
[parsing started HelloWorldClass.java]
[parsing completed 130ms]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/
Object.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/
String.class)]
[checking HelloWorldClass]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/lang/
System.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/
PrintStream.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/Fil-
terOutputStream.class)]
[loading F:\jbuilder5\jdk1.3\jre\lib\rt.jar(java/io/Out-
putStream.class)]
[wrote HelloWorldClass.class]
[total 561ms]
```

Nach dem Kompilieren hat der Java-Compiler die Runtimeversion mit der Endung *.class* erzeugt.

Listing 8.4 Files, die der Compiler von Java erzeugt hat

```
07.10.2001  19:33                 451 HelloWorld-
Class.class
07.10.2001  19:31                 302 HelloWorld-
Class.java
               2 File(s)            753 Bytes
```

Mit der Java-Utility kann eine Klasse direkt von der Kommandozeile ausgeführt werden, vorausgesetzt eine Methode mit dem Namen `main()` wurde implementiert.

Listing 8.5 Ausführen der Klasse mit der Java Runtime

```
D:\JDK> java HelloWorldClass
Welcome to Java hello World
```

8.4.2 Constructor

Ein Constructor ist eine Methode, die immer dann ausgeführt wird, wenn eine Klasse oder eine Instanz zum ersten Mal angesprochen wird. Sie eignet sich besonders zum Initialisieren von Werten innerhalb der Klasse, und zwar just-in-time, nämlich dann, wenn die Daten das erste Mal gebraucht werden.

Constructor in Java

Ein Constructor in Java ist eine Methode mit den folgenden Charakteristika:

- Ein Constructor hat immer den gleichen Namen wie seine Klasse.
- Ein Constructor hat keinen Ergebnistyp, also auch nicht die Kennzeichnung VOID.

So sieht ein Constructor in Java aus:

```
public class Test {
```

Folgendes ist der Constructor der Klasse `Test`:

```
  public Test() {
  }
```

Erzeugung der Klasse, bei der der Constructor ausgeführt wird:

```
  public static void main(String[] args) {
    Test test1 = new Test();
  }
}
```

Constructor in ABAP Objects

Ein Constructor in ABAP Objects ist eine Methode, mit dem Namen `CLASS_CONSTRUCTOR` oder `CONSTRUCTOR`. Es gibt zwei Ebenen von Constructoren, einmal beim erstmaligen Initialisieren der Klasse und einmal bei jedem Anlegen einer Instanz.

Constructor für die ganze Klasse

So sieht ein Constructorfür eine Klasse in ABAP aus:

```
method CLASS_CONSTRUCTOR .
endmethod.
```

Constructor für eine Instanz

Das Folgende ist ein Constructor für eine Instanz:

```
method CONSTRUCTOR .
endmethod.
```

Constructor in VB

Ein Constructor in Visual Basic ist eine spezielle Methode mit dem Namen `Class_Initialize` und mit folgendem Aussehen.

```
VERSION 1.0 CLASS
Attribute VB_Name = "TEST"
Private Sub Class_Initialize()

End Sub
```

8.4.3 Destructor

Ein *Destructor* ist eine spezielle Methode, die immer dann aufgerufen wird, wenn die Klasse oder eine Instanz der Klasse zerstört wird. Damit ist der Aufruf des Destructors die definitiv letzte Aktion während der Lebenszeit einer Instanz.

Destructor in Java

Java kennt keinen Destructor

Das Konzept von Java sieht wie eine ständige und versteckte Garbage-Collection von nicht mehr benötigtem Speicherplatz aus. Ausgehend von der irrigen Annahme, dass der Destructor einer Objekt-Klasse nur das Freigeben von Speicherplatz und andere Aufräumarbeiten vornehmen sollte, verzichteten die Designer von SUN auf die Destructor-Methode. Das ist jedoch eher ärgerlich, da der Destructor oft auch zum Anstoßen anderer Tasks verwendet wird, und wenn es nur das Aussenden eines globalen *Goodbye* an den Rest der Welt ist.

Destructor in ABAP Objects

Auch in ABAP gibt es keine Destructoren.

Destructor in VB

Ein Destructor in Visual Basic ist eine spezielle Methode mit dem Namen `Class_Terminate`. Folgendermaßen sieht er aus:

```
VERSION 1.0 CLASS


Private Sub Class_Initialize()

End Sub


Private Sub Class_Terminate()

End Sub
```

Constructor für die Klasse Test

Destructor für die Klasse Test

8.4.4 Property-Deklarationen

Eine Property in einer Klasse entspricht in etwa einer öffentlich zugreifbaren Variablen. Tatsächlich werden in VB alle Public-Variablen automatisch als Properties behandelt. Das Kennzeichen einer Property ist, dass sie einen Wert hat. Dieser Wert lässt sich im Normalfall lesen und setzen wie bei einer Variablen.

Properties sind öffentliche Variablen

Properties in Java

Properties in Java werden durch eine bestimmte Namenskonvention angelegt. Diese Namenskonvention wird in der Sprachbeschreibung als *Design Pattern* ausgelegt, was aber etwas hoch gegriffen ist, nur um zu rechtfertigen, warum man semantische Inhalte an den Aufbau eines Bezeichners hängt.[2]

Properties werden durch Namenskonventionen bestimmt

Eine Property in Java besteht aus drei Elementen

- eine innerhalb der Klasse deklarierte Variable `xxx`, wobei das erste Zeichen der Variablen ein kleiner Buchstabe sein muss
- eine Methode zum Setzen des Wertes der Variablen `xxx` mit dem Namen `setXxx`, wobei das erste Zeichen des Namens der Property ein Großbuchstabe sein muss

2 Mehr zu diesem Problem: *Thinking in Patterns With Java* auf *www.BruceEckel.com*.

- eine Methode zum Auslesen des Wertes der Variablen `xxx` mit dem Namen `getXxx`, wobei das erste Zeichen des Namens der Property ein Großbuchstabe sein muss

Die Syntax sieht folgendermaßen aus:

Get Properties in Java

```
private int temperature;
public int getTemperature() { return temperature; };
```

Set Properties in Java

```
public void setTemperature(int newTemperature)
{ temperature = newTemperature; };
```

Properties in VB

Properties in VB werden durch das Keyword `property` eingeleitet und ansonsten wie Unterprogramme behandelt.

Get Properties in VB

```
Private myTemperature as Integer
public property Get Temperature()
       Temperature = myTemperature;
End Property
```

Set Properties in VB

```
Private myTemperature as Integer
public property Get Temperature(newTemperature as Inte-
ger)
       myTemperature = Temperature;
End Property
```

Properties in ABAP

ABAP Objects vermeidet den direkten Umgang mit Properties. Es gibt das Konstrukt der Attribute, das alle global definierten Variablen innerhalb einer Klasse bezeichnet. Properties im Sinne von VB oder Java müssen als Methoden realisiert werden.

8.4.5 Methoden

Methoden in Java

Methoden in Java werden durch das Keyword `proc` definiert. Falls die `proc` mit dem Vorsatz `void` angelegt wird, handelt es sich um eine echte Prozedur, die keinen Wert zurückgibt, während eine Methode, die einen Wert zurückgibt, einer Funktion entspricht.

Methoden in VB

Eine Methode in VB ist identisch mit einer `Sub` oder einer `Function`:

```
Function AddInt(a as Integer, b as Integer) As Integer
       AddInt = a + b
End Function
```

Methoden in ABAP Objects

Folgendermaßen sieht eine Methode in ABAP aus:

```
method FIND_BY_KEY .
RESULT = IMPL_AGENT->FIND_BY_KEY(
KEYATT_4 = KEYATT_4
KEYATT_5 = KEYATT_5
FLDATE = FLDATE
CONNID = CONNID
CARRID = CARRID ).
endmethod.
```

8.4.6 String-Deklarationen

Strings in Java

Strings in Java werden durch den eigenen Datentyp `String` angelegt. Dieser unterscheidet sich von einem `Array of Char` dadurch, dass Strings eine dynamische Länge haben können.

```
String aText[100];
```

Die Länge des Strings wird durch die Methode `Length` bestimmt:

```
System.out.println(aText.length());
```

Verkettung von Strings in Java kann wie folgt aussehen:

```
String first = new String("sure");
String second = "plea";
String combined = second + first;
System.out.println(combined);
System.out.println(combined.length());
```

Strings in VB

Strings in VB werden durch den eigenen Datentyp `String` angelegt. Die maximale Länge eines Strings kann in Klammern angegeben werden, muss aber nicht.

```
Dim aText As String
Dim aText80 As String[80]
```

Die Länge des Strings wird durch die Funktion `Len` bestimmt.

```
Debug.Print len(aText)
```

Strings in ABAP Objects

Strings im Sinne von variablen Zeichenfolgen existieren in ABAP nicht. Zum Arbeiten mit Strings muss immer eine Variable mit einer fest vorgegebenen Länge deklariert werden:

```
DATA: mystring(80) TYPE C. "String der Länge 80
```

Zur Manipulation von Strings in ABAP existieren ein paar wenige Funktionen.

ABAP	Bedeutung
Strlen(mystring)	Bestimmt die Länge des Strings, nachdem die Leerzeichen am Ende abgeschnitten wurden
CONCATENATE astr bstr INTO cstr	Zusammenführen zweier Zeichenfolgen
SPLIT	Teilen eines Strings an einer bestimmten Stelle
Mystring+10(25)	Eine Art Substring-Bildung in ABAP, ab Stelle 10, 25 Zeichen; für variable Substrings benötigen Sie ABAP-FILED_SYMBOLS und den Befehl ASSIGN.

Tabelle 8.1 String-Manipulation in ABAP

8.4.7 Array-Deklarationen

Arrays in Java

Arrays in Java werden durch das Klammernpaar `[]` deklariert. Sie können eine feste Länge haben. Wird keine Länge angegeben, wird die Länge des Arrays während der Laufzeit bestimmt.

```
String args[];
```

Arrays in VB

Arrays in VB werden durch das Klammernpaar `[]` deklariert. Sie können eine feste Länge haben. Wird keine Länge angegeben, wird die Länge des Arrays während der Laufzeit bestimmt. Durch die Funktion `Redim()` kann ein Array jederzeit umdimensioniert werden.

```
DIM: anArray[10] Of Integer
```

Arrays in ABAP Objects

ABAP kennt keine Arrays, sondern nur so genannte Interne Tabellen, die in etwa den Recordsets von Visual Basic entsprechen.

```
DATA: itab TYPE Standard Table Of Integer.
```

Teil 3
Programmieren in verteilten Systemen

Das Programmieren in einem Netzwerk von miteinander kooperierenden Computern und Computerdiensten erfordert auch eine besondere Ausbildung der Entwickler. Anstatt sich auf ein Betriebssystem, eine Programmiersprache und eine Oberflächengestaltung zu konzentrieren, werden heutzutage Leute gebraucht, die ausdrücklich mehrere Systeme und Sprachen beherrschen. Die Frage ist demnach nicht mehr, ob SAP, UNIX oder Microsoft, sondern ob SAP, UNIX und Microsoft. Zwingend sind fortan Kenntnisse in Frontend-Entwicklung, Workflow und Middleware sowie Programmierung in mehreren Sprachen nach Bedarf.