



IT-Kompaktkurs

Programmieren mit Java Skript zur Folge 9

Prof. Dr. Walter Kiel
Fachhochschule Ansbach

„Grundlagen der GUI-Programmierung (2)“

Zur Wiederholung:

Wir kennen bereits die: „Graphical User Interfaces“ – also die Grafischen Benutzeroberflächen und das AWT – das „Abstract Windowing Toolkit“, eine Art Baukastensystem, mit dem wir Fenster-Anwendungen erstellen können.

Um eine solche Oberfläche zu erzeugen, werden die gewünschten Komponenten – zum Beispiel Aktionsknöpfe, Texteingabefelder oder Menüs – in ein Rahmen-Fenster eingefügt, das dann quasi als Container dient. Genau damit kommt Leben auf den Bildschirm, denn die Elemente der grafischen Oberflächen ermöglichen eine schnelle und intuitive Interaktion. Und um diese Interaktion geht es im folgenden. Zudem werden wir verschiedene Gestaltungsmöglichkeiten für Oberflächen kennen lernen.

1. Java AWT: Verarbeitung von Ereignissen

Die Parallele zwischen einer sogenannten „**Ereigniskette**“ (z.B. **Einkauf / Verkauf**) und der **Verarbeitung von Ereignissen** unter Java sieht folgendermaßen aus:

Die Ware liegt im Schaufenster. Sie schickt mir die Botschaft: „Kaufe mich, dann kann ich dir nützlich sein“. Dies löst häufig die Aktion „Betreten des Ladens“ mit allen seinen Folgen aus. Analog funktionieren die Java-Objekte bzw. Komponenten der grafischen Oberflächen. Ein Push-Button sagt mir: „Klicke mich an!“ Ein Menü-Eintrag sagt mir: „Wähle mich aus!“ Beides soll mir nützlich sein. Es entsteht dann eine Aktion auf dem Bildschirm, die sogenannte „Interaktion“ oder auch „Mensch-Maschine-Kommunikation“. Ein **Mausklick** oder eine **Tastatur-Eingabe** löst die Reaktion des Programms aus.

Allerdings verstehen wir unter der Verarbeitung von Ereignissen mehr als nur eine Maus- oder Tastatur-Betätigung. Im Prinzip handelt es sich um ein komplexes System von Nachrichten. Java bietet dabei den Vorteil, dass man nicht alles selbst programmieren muss. Hier steht nämlich das Package „`java.awt.event`“ zur Verfügung. Man muss vor allem die Grundprinzipien der Verarbeitung von Ereignissen kennen und berücksichtigen, um ereignis-gesteuerte Anwendungen programmieren zu können.

Dieses „**Grundprinzip**“ bei der Verarbeitung von Ereignissen unter Java ist einfach:

- Die dargestellten **Komponenten** bieten die **Möglichkeit von spezifischen Ereignissen**.
- Die in Frage kommenden Ereignisse werden jeweils mit einem sogenannten **Listener** abgehört.
- Wenn ein solches Ereignis eingetreten ist, veranlasst ein **Behandler** - ein sog. „**Event Handler**“ - eine Reaktion darauf, indem er beispielsweise Methoden aufruft.

Zum Beispiel:

In der letzten Folge haben wir bereits ein Fenster programmiert, das wir schließen konnten. Der Frame-Rahmen aus der Klasse Window reagiert dabei nicht nur auf ein Schließen, sondern auch auf ein Verkleinern, ein Maximieren oder dergleichen mehr. Das Schließen mit der Maus führt zum Ereignis WindowEvent, das mit dem Listener WindowListener verbunden ist. Die Behandlung dieses Ereignisses muss dann über die Methode WindowClosing implementiert werden.

Beim Betrachten eines solchen interaktiven Fensters stellt sich zwangsläufig die Frage, wie man sich dabei die **Beziehung zwischen**

- **Komponente**,
- **Ereignis**
- und **Listener**

vorstellen kann.

Zunächst können **bestimmte Komponenten** nur **bestimmte Ereignisse** auslösen. So kann beispielsweise ein Textausgabefeld natürlich nicht auf einen Mausklick reagieren. Dagegen führt bei einer Texteingabezeile - also einem Objekt der Klasse `TextField` - die Eingabe eines Textes zu einem `TextEvent`, was über den `TextListener` abgefragt werden kann. Dabei ist zu berücksichtigen, dass es im AWT nur eine gut überschaubare Anzahl von Komponenten gibt. Diese teilen sich gerade einmal elf Ereignisarten, und zu jeder dieser Ereignisarten gehört – mit Ausnahme des `MouseEvent` – genau ein Listener (*vgl. Folie 2*).

Mit einem Blick auf diese Komponenten können wir jetzt unmittelbar nachvollziehen, wie unser schon aus der vorherigen Sendung bekanntes Frame-Fenster mit dem „Auf Wiedersehen“-Pushbutton funktioniert.

- Die Komponente **Button** kann einen `ActionEvent` auslösen, was über den **Action-Listener** abgefragt werden kann (*vgl. Folie 3*).
- Bei der Klassendefinition muss der Button dazu lediglich über die Anweisung `implements ActionListener` und über den Methodenaufruf `addActionListener` entsprechend aktiviert werden.
- Um eine Reaktion auf ein Button-Ereignis herbeizuführen, muss schließlich nur noch der „**Event Handler**“ über die Methode `actionPerformed` implementiert werden (*vgl. Folie 4*).
- Ein Klick auf den „Auf Wiedersehen“-Button macht mein Fenster also zu und schließt somit die Anwendung (*vgl. Folie 5*).

Woher weiß Java aber nun, wie es auf **Ereignisse** reagieren soll? Je nach **Kontext** müsste beispielsweise ein **Maus-Ereignis** doch sehr **unterschiedliche Reaktionen** auslösen können!?

Wir differenzieren hier zwischen **Low-Level-Events** und den **semantischen Events**.

- Die **semantischen Events** sind unmittelbar auf ein Ereignis mit einer der genannten Interaktionskomponenten zurückzuführen. Hier steht genau eine Behandlungsmethode zur Verfügung.
- Bei den **Low-Level-Events** – zum Beispiel bei einer Aktion mit der Maus – kann die Reaktion über sogenannte Event-Adapter koordiniert werden. Damit können dann evtl. auch mehrere Methoden erfasst und gebündelt werden.

Es stellt sich nun noch die Frage ob bei der Verarbeitung von Events wieder das Problem mit den unterschiedlichen Java-Versionen auftritt ?

Diese Problematik ist hier in der Praxis nicht so gravierend. Die besondere Art der Ereignis-Behandlung vom alten Java 1.0 wird nur noch in begründeten Ausnahmen verwendet. Ab der Version 1.1 funktioniert die Behandlung von Ereignissen mit dem AWT auf die dargestellte Weise. Dies gilt übrigens grundsätzlich auch für die bereits erwähnte GUI-Programmierung mit Swing.

2. Java AWT: Management von Layouts

Es stellt sich als erstes die Frage, warum dem Layout eine so große Bedeutung zukommt.

Dies hängt mit dem Haupt-Anliegen der grafischen Oberflächen zusammen. Sie müssen in erster Linie benutzerfreundlich und selbsterklärend sein. Die Software muss deshalb ein einheitliches Look-and-Feel vermitteln. Zugleich gibt es aber auch erhebliche Freiheiten der Benutzer, mit ihren Fenstern umzugehen – Freiheiten, die möglichst wenig eingeschränkt werden sollten.

Beides wirkt sich direkt auf das Layout der Oberfläche aus: Die Arbeitsumgebung sollte zum einen beispielsweise einheitlich in der jeweiligen Landessprache gefasst sein. Hier erfordert schon allein die einfache Umsetzung eines „Bye“-Buttons in einen „Auf Wiedersehen“-Button einen wesentlich höheren Platzbedarf. Zum anderen gibt es beispielsweise Anwender, die Applikationen immer in einem maximierten, also bildschirmfüllenden Fenster laufen lassen. Andere User bevorzugen stets frei angeordnete bzw. verkleinerte Fenster.

Unter all diesen Umständen muss ein GUI immer ansprechend, zweckmäßig und selbsterklärend sein und auch bleiben.

Wesentliche Aufgaben aus den genannten Problemen werden in Java im Rahmen der Sprache selbst, hier durch den Layout-Manager gelöst. Ein Layout-Manager hat einerseits die Aufgabe, die Komponenten in einem Container zunächst einmal sinnvoll anzuordnen. Andererseits muss er dafür sorgen, dass alle Elemente – etwa nach einer Größenänderung des Fensters - noch immer sichtbar und ansprechend sind.

Der Layout-Manager wird über die Methode `setLayout` mit dem jeweiligen Container verknüpft. Beim Methodenaufwurf wird auch eine Entscheidung über die Anordnung von Komponenten in diesem Container getroffen. Es gibt hier beim AWT fünf verschiedene Grundmodelle (**siehe Folie 6**) – wir werden von diesen aber nur die drei wichtigsten behandeln. Die Komponenten werden dabei wie gewohnt erzeugt und mit der `add()`-Methode hinzugefügt - allerdings versehen mit Zusatzinformation für den Layout-Manager.

Die drei wichtigsten Grundmodelle:

- **FlowLayout**

Beim FlowLayout werden die Komponenten der Reihe nach angeordnet und ggf. umgebrochen. Die Ausrichtung kann dabei – ähnlich wie beim Fließtext in der Textverarbeitung – mit den Parametern LEFT, RIGHT oder CENTER als „linksbündig“, „rechtsbündig“ oder „zentriert“ vorgegeben werden.

- **BorderLayout**

Beim BorderLayout ist der Rand das maßgebliche Element der Anordnung. Es gibt die Parameter NORTH, SOUTH, WEST und EAST für die jeweils gewünschte Positionierung. Als CENTER wird infolge dessen einfach der mittlere Bereich des Containers gesehen.

- **GridLayout**

Das GridLayout beinhaltet – wie der Name es schon sagt - die Anordnung innerhalb eines Gitters aus Zeilen und Spalten. (**vgl. Folie 7**)

Derart strenge Anordnungen in reiner Form sind *eigentlich* nie bei einer Standard-Anwendung zu sehen, denn die Grundmodelle werden nur selten in ihrer Reinform

verwendet. Meist erfolgt eine Mischung der Grundtypen. So kann der NORTH-Bereich eines BorderLayouts mit einer FlowLayout-Anordnung von Buttons versehen werden. Entscheidend ist, dass der Layout-Manager nach den Vorgaben der Programmierer für ein ansprechendes Look-and-Feel sorgt – und dies unabhängig von der Plattform, von den Bildschirm-Einstellungen, den verfügbaren Fonts, usw. Die Anordnung der Komponenten bleibt auch nach einem Resize des Fensters noch ansprechend. **(siehe Folie 8)**

Wenn man für eine spezielle Anwendung ein durchgängiges Layout braucht, ist man nicht einmal auf die verfügbaren Grund-Layouts beschränkt, denn prinzipiell gibt einem Java sogar die Möglichkeit, einen eigenen Layout-Manager zu entwickeln. Sollte dies tatsächlich einmal erforderlich sein, kann eine entsprechende individuelle Programmierung ohne weiteres anhand der einschlägigen Fachliteratur erfolgen.

3. Sun Forte for Java Community Edition, insbesondere Anwendung des GUI-Builders „Form Editor“

Für die Praxis der Programmierung gibt es mächtige Werkzeuge. Damit können auch komplexe Anwendungen mit grafischer Oberfläche software-gestützt – und nicht per Hand – entwickelt werden. Gemeint sind hier die integrierten Entwicklungs-Umgebungen, die sogenannten Integrated Developer Environments oder kurz IDEs. Forte für Java ist die aktuelle IDE-Technologie von Sun Microsystems.

Forte für Java liegt in der Community Edition zum kostenlosen Download über das Internet bereit - bei der Installation der IDE ist jedoch folgendes zu beachten:

- Forte benötigt die aktuelle Fassung von Java, also Java 2 mit einem Versionsstand 1.2 oder höher. Es werden alle Plattformen unterstützt, für die es ein entsprechendes Software Development Kit gibt. Bei den Plattformen, die besonders häufig für die Installation von Forte in Frage kommen, gibt es spezielle Installationspakete in den plattform-eigenen Formaten. Dies gilt für Windows 9x, Windows NT, Windows 2000, Solaris und Linux.
- Die betreffenden Computer brauchen aber eine schon etwas gehobene Ausstattung an Hardware, insbesondere hinsichtlich Prozessor-Leistung und Hauptspeicher. So wurde z. B. für eine Installation von Forte für Windows 1.0 in der Community Edition unter Windows ein Pentium-Prozessor mit mindestens 300MHz und ein Hauptspeicher von mindestens 128 MB empfohlen.

(Stand der Erstaussstrahlung von Folge 9. Bitte entnehmen Sie die aktuellen Voraussetzungen den im Addendum genannten Quellen.)

Die Forte for Java IDE bietet uns eine Reihe unterschiedlicher Entwicklungswerkzeuge. Dazu gehören unter anderem:

- ein Haupt-Fenster zur Steuerung,
- der Explorer zum Überblick über die verwendeten Klassen
- ein Quelltext-Editor mit Syntax-Fähigkeit,
- ein „Properties“-Fenster zur Festlegung der Eigenschaften der jeweiligen Objekte,
- der „Form Editor“ zur Gestaltung grafischer Oberflächen,
- und natürlich der Debugger.

Bei Forte für Java werden die einzelnen Werkzeuge in jeweils eigenen Fenstern angeordnet (**vgl. Folie 9 und 10**). Das heißt, man kann leicht zwischen verschiedenen programmiertypischen Arbeitsansichten (sog. workspaces) wechseln: Edition von

Quelltext, GUI-Entwurf, Browsing der Objekte, Programm-Ausführung und Debuggen.

Hierzu ein praktische Beispiel:

Der Weg zu einer fertigen Anwendung – beispielsweise für eines der bisher gezeigten Fenster:

- Zuerst muss man sich ein passendes Template aussuchen – also z. B. einen Frame-Container auf der Basis des AWT. Dieser soll die Grundlage unseres neuen Entwicklungs-Projekts bilden.
- Danach muss man die Einzelheiten festlegen, zum Beispiel die Auswahl des Layout-Managers und die gewünschten Komponenten. Das funktioniert weitestgehend intuitiv über Maus-Bedienung. Syntax-Kenntnisse sind hierzu kaum erforderlich.
- Diese werden nur bei der Implementierung des Event-Handler-Codes benötigt. Hier muss man eben wissen, wie die Anwendung auf Ereignisse reagieren soll. Das entsprechend erzeugte Programm kann dann übersetzt und ausgeführt werden.

Forte ist insbesondere auch für komplexe Anwendungen geeignet. Der dabei entstehenden Gefahr, dass das Java-Programm dann sehr umfangreich und unübersichtlich wird, wird durch eine **Projektverwaltung** entgegnet.

Das heißt: Eine spezielle Anwendung beinhaltet in der Regel mehrere java-Dateien, unter denen eine Arbeitsteilung herrscht. Spezialaufgaben – wie etwa der Entwurf der grafischen Oberfläche - können somit prinzipiell delegiert werden. Diese Arbeitsteilung ist für heutige Entwicklungs-Projekte typisch: Komplexe Anwendungen werden im Teamwork entwickelt. Dabei sind allerdings klare Schnittstellen unerlässlich, was bei Java infolge des objektorientierten Ansatzes ohnehin charakteristisch ist.

Konkrete **Vorteile**, die das Arbeiten mit Forte für Java bietet:

- Die moderne IDE ist in der Praxis gut zu verwenden. So gibt es zum Beispiel Templates - also Muster - für häufig verwendete Entwicklungsformen.
- Auch die Offenlegung der Quellen - Stichwort: Open Source im Zusammenhang mit dem netBeans-Projekt – und der konsequente Einsatz von Pure Java für die kostenlose Entwicklungs-Software sind richtungsweisend. Außerdem ist die originale Dokumentation von Sun - insbesondere auch im Rahmen der Lehre – überaus instruktiv und gut einzusetzen.
- Überzeugend ist schließlich das modulare bzw. plug-in-Konzept: Spezial-Anforderungen können bequem über die plugin-Technik durch Einbindung und/oder Austausch von Modulen integriert werden.

Bei der Programmierung über Forte für Java ist jedoch zu beachten:

Die reine Laufzeit-Umgebung – das Runtime Environment - von Java reicht nicht aus. Es muss das Java Development Kit in Version 1.2 oder höher (*Stand: Erstausstrahlung Folge 9; siehe Addendum*) installiert sein, denn Forte arbeitet mit Java 2.

Abschließend sei noch kurz auf das Konzept der **Java Foundation Classes** hingewiesen. Dies ist ein Begriff, der im Zusammenhang mit der Programmierung von grafischen Oberflächen häufig zu hören ist. Die Java Foundation Classes stellen eine lose Sammlung von Bibliotheken zur Programmierung grafischer Oberflächen einschließlich der damit verbundenen Programmieraufgaben dar. Für Standalone-Anwendungen sind das AWT und Swing die Eckpfeiler der Foundation Classes. Bei Web-bezogenen Anwendungen sind zudem die Applets besonders wichtig. Die Applets werden in den Folgen 10 und 11 von Prof. Dr. Helmut Roderus behandelt.

Addendum:

Seit der Aufzeichnung der Folge 9 wurde auch Java stark weiterentwickelt. Das J2SE SDK liegt zum Zeitpunkt der Wiederholungssendung am 14. Mai 2002 bereits in der Version 1.4 vor. Die in Folge 9 gezeigte IDE Forte for Java 1.0 CE ist mittlerweile regulär in Version 3.0 und im Early Access sogar schon in Version 4.0 verfügbar. (**siehe Folien 13 bis 16**)

Das Java 2 (J2SE) Software Development Kit (SDK) erhalten Sie kostenlos unter dem folgenden URL:

- <http://java.sun.com>

Forte for Java CE ist kostenlos über den folgenden URL zu erhalten:

- <http://forte.sun.com/ffj>

Forte basiert auf der Netbeans-IDE („Forte is based on netBeans“). netBeans ist die zu Forte passende OpenSource IDE unter dem URL:

- <http://www.netbeans.org>

Schließlich sei auch hier noch einmal hingewiesen auf die Sun Developer Connection unter dem URL:

- <http://www.sun.de/Entwickler/SDC>