Ehrenfried Stuhlpfarrer

Objektorientierte Programmierung und Java
- interaktiv -
für Handelsakademien

DIPLOMARBEIT

HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

Informationstechnik & Elektrotechnik

Mittweida, 2010

Ehrenfried Stuhlpfarrer

Objektorientierte Programmierung und Java
- interaktiv -
für Handelsakademien

eingereicht als

DIPLOMARBEIT

an der

HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

Informationstechnik & Elektrotechnik

Mittweida, 2010

Erstprüfer: Prof. Dr.-Ing. Volker Delport (Hochschule)

Zweitprüfer: Ostr. Dr. Mag. Siegfried Ledolter (betrieblich)

Vorgelegte Arbeit wurde verteidigt am: _____

Bibliographische Beschreibung

Stuhlpfarrer, Ehrenfried:
Objektorientierte Programmierung und Java - interaktiv - für Handelsakademien. - 2010. - 130 Seiten.
Mittweida, Hochschule Mittweida,
Fachbereich Informationstechnik und Elektrotechnik, Diplomarbeit, 2010

Referat

Nachdem an der Bundeshandelsakademie und Bundeshandelsschule in Judenburg der erste Jahrgang des Zweiges für Informationstechnologie und Informationsmanagement erfolgreich abgeschlossen wurde und es einen Mangel an Lehrunterlagen im Bereich Softwareentwicklung, insbesonders in dem Teilgebiet der objektorientierten Programmierung gibt, habe ich mich entschlossen, mein Diplomprojekt in das Zeichen „Lehrunterlagen für den IT - Zweig“ zu stellen. Dabei habe ich mir genau überlegt, wie man die anfänglich äußerst trockene und schwierige Materie der objektorientierten Programmierung den Schülern nahe bringen könnte. Ich kam zu dem Entschluss, dass Lehrunterlagen mit interaktivem Charakter die Schüler wohl am meisten motivieren würden. Damit meine ich auf der einen Seite kleine Einheiten mit den Zielvorstellungen, sowie kurzen Theorieteilen mit interaktivem Charakter und die sofortige Überleitung zu Praxisbeispielen, an denen selbst experimentiert werden kann. Das Niveau der Beispiele soll nun speziell auf die Handelsakademien abgestimmt sein, das heißt im Klartext, dass keine Spitzensoftwareentwickler herauskommen sollen, aber Fachleute mit Überblick, die wissen wie sie etwas angehen. Diese Arbeit soll als Ergänzung für den Unterricht gedacht sein und keineswegs einen guten Vortragenden ersetzen. Die Lehrunterlagen sollen aus einem Lehr- und Arbeitsbuch bestehen, das übersichtlich gegliedert ist und verständliche Beispiele enthält und in einer verständlichen Sprache die wichtigsten Zusammenhänge erläutert. Ein Lösungsheft, sowie eine Website mit Ergänzungen sollen weitere Lernhilfen bieten. Ein Hauptziel dieser Website ist auch das Anbieten von Links, die das theoretische Angebot sinnvoll erweitern und ergänzen. Dies soll gemeinsam ein auf den aktuellen Stand gebrachtes und methodisch und didaktisch ausgereiftes Lehr- und Lernpaket darstellen, welches als Framework eingeführt werden soll. Damit meine ich, dass Erweiterungen und Veränderungen jederzeit möglich sein müssen, ohne das Grundkonzept zu zerstören.
Die Schüler und Schülerinnen sollen unter Verwendung von modernen objektorientierten Techniken Teilaufgaben sowie Programmmodule erstellen und testen können. Die Programmiersprache Java[1], von Sun[2] dient dabei als Basis. Die Lektionen sollen so aufgebaut sein, dass auch jede andere objektorientierte Programmiersprache für Trainingszwecke eingesetzt werden könnte.

1 Java ist eine objektorientierte Programmiersprache
2 Sun Microsystems ist ein in Santa Clara ansässiger Hersteller von Computern und Software

Inhaltsverzeichnis

A Abkürzungen

APL.....**A P**rogramming **L**anguage
API.....**A**pplication **P**rogramming **I**nterface
AWK.....Autoren **A**ho, **W**einberger und **K**ernighan
BHAK.....**B**undes**h**andels**ak**ademie
BHAS.....**B**undes**ha**ndels**s**chule
C.....Programmiersprache
CAD.....**C**omputer **A**ided **D**esign
CASE.....**C**omputer **A**ided **S**oftware **E**ngineering
CRC.....**C**lass **R**esponsibility **C**ollaboration
DBMS.....**D**atabase **M**anagement **S**ystem
DNS.....**D**omain **N**ame **S**ervice
DOM.....**D**ocument **O**bject **M**odel
EE5.....**J**ava **E**nterprise **E**dition **5**
GUI.....**G**raphic **U**ser **I**nterface
HTML.....**H**yper**t**ext **M**arkup **L**anguage
HTTP.....**H**yper**t**ext **T**ransfer **P**rotokoll
IDE.....**I**ntegrated **D**evelopment **E**nvironment
IP.....**I**nternet **P**rotokoll
IT.....**I**nformations**t**echnologie
JAR.....**J**ava **Ar**chiv
JDBC.....**J**ava **D**atabase **C**onnector
JDK.....**J**ava **D**evelopment **E**nvironment
JSP.....**J**ava **S**erver **P**ages
LISP.....**Lis**t **P**rocessing (Listen-Verarbeitung)
MVC.....**M**odel **V**iew **C**ontroler
OOP.....**O**bjekt**o**rientierte **P**rogrammierung
PDF.....**P**ortable **D**ocument **F**ormat
PROLOG.....**Pro**grammation en **Log**ique
SED.....**S**tream **Ed**itor
SIMULA.....Programmiersprache (**Simu**lation u. **La**nguage)
SMALLTALK.....Dynamische, untypisierte Programmiersprache
SUN.....**S**tanford **U**niversity **N**etwork
SQL.....**S**tructured **Q**uery **L**anguage
TLD.....**T**op**L**evel**D**omain
UML.....**U**nified **M**odeling **L**anguage
VB.....**V**isual **B**asic
W3C.....**W**orld **W**ide **W**eb **C**onsortium
WWW.....**W**orld **W**ide **W**eb
XML.....E**x**tensible **M**arkup **L**anguage
XP.....E**x**treme **P**rogramming

1 Eigenständigkeitserklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Projektarbeit (Diplomprojekt) vollkommen selbstständig und nur mit Unterstützung der genannten Projektbetreuer, Projektpartner und Quellen erstellt habe.

Weißkirchen, am 1. Dezember 2009

Ehrenfried Stuhlpfarrer .....

Objektorientierte Programmierung und Java
- interaktiv -
für Handelsakademien

2 Prozessdokumentation

2.1 Projektauftrag

Projektname

Objektorientierte Programmierung und Java
- interaktiv -
für Handelsakademien

Auftraggeber

Bundeshandelsakademie und Bundeshandelsschule Judenburg

Projektbearbeiter

Ehrenfried Stuhlpfarrer

Projektbetreuer

Prof. Dr.-Ing. Volker Delport (Hochschule)
Ostr. Dr. Mag. Siegfried Ledolter (betrieblich)

Projektbeginn

1. September 2009

Projektende

31. Dezember 2009

Projektziele

Das Ziel meines Diplomprojektes ist es, methodisch und didaktisch aufgebaute Lehrunterlagen für das Fach objektorientierte Programmierung zu erstellen (1 Semester ca. 30 Stunden). Die Lehrunterlagen sollen eine Hilfe sein, den Stoff zu erlernen und den Schülern eine gute Basis für das spätere Berufsleben oder Studium sein. Die Schüler sollen aber auch selbst die Verantwortung für den Lernerfolg übernehmen und die vielfältigen Möglichkeiten nutzen.
Das gesamte Lernpaket besteht aus einem Lern- und Arbeitsbuch, einer Website und einem Lösungsheft mit weiterführenden Praxistipps. Die Gliederung soll übersichtlich sein, sinnvoll aufbauende Beispiele enthalten und in einer verständlichen Sprache die wichtigsten Zusammenhänge erläutern. Die Website bietet Ergänzungen zum Lern- und Arbeitsbuch, bietet weiteres Informationsmaterial und interessante Links und Literaturhinweise an. Weiters gibt es abwechslungsreiche Übungsmöglichkeiten um das erlernte Wissen zu vertiefen. Die Lerninhalte sind auf mehreren Schritten aufgebaut. Der erste Schritt bedeutet „Lernen“ und das heißt Informationen aufnehmen, Zusammenhänge erkennen und die Theorie erfassen. Der zweite Schritt bedeutet „Üben“. Damit meine ich Routine erwerben und einmal selbst ein Beispielprogramm zum Laufen zu bringen, Zusammenhänge verstehen und Erfahrung zu sammeln. Der dritte Schritt bedeutet „Sichern“ oder Festigen. Dabei wird Gelerntes zusammengefasst, die Übersicht gewonnen und die Inhalte wiederholt. Der vierte Schritt bedeutet „Kontrolle“. Damit soll das Wissen getestet, die Kompetenz überprüft und das Können bewiesen werden. Die Beispiele sind frei erfunden und jede Ähnlichkeit mit tatsächlichen Ereignissen in Unternehmen ist nicht zufällig, sondern bewusst gewählt. Für ein leichteres Verständnis sind die Beispiele so einfach wie möglich gestaltet, aber trotzdem nicht trivial und entsprechen betrieblichen Aufgabenstellungen, wie sie tatsächlich in Teilbereichen in jedem Unternehmen vorkommen können.

Budget und Ressourcen

Für die Ausarbeitung der Diplomarbeit benötige ich vor allem Ressourcen im Bereich Entwicklungsumgebung für Java und Informationsmaterialien im Bereich objektorientierte Programmierung. Für die anspruchsvolle Aufbereitung ist ein DDP[1] - Programm erforderlich, sowie eine HTML[2] IDE[3] für die Website. Java inklusive Entwicklungsumgebung ist kostenlos. Für allen anderen Anforderungen verwende ich die Master Collection von Adobe[4], sowie Mediator von Matchware.

1 Desktop Publishing
2 Hypertext Markup Language
3 Integrated Development Environment
4 Adobe Systems ist ein US-amerikanisches Softwareunternehmen

Nutzen für den Projektbearbeiter

- selbstständiges Arbeiten
- Vertiefen von erlerntem Projektmanagement
- Vertiefen und Umsetzen des Erlernten im Bereich OOP
- gute Unterrichtsvorbereitung
- Aufbau einer professionellen Präsentation

Nutzen für den Projektpartner

- neue Unterrichtsmaterialien
- aktuelle Unterrichtsmaterialien
- Wiederverwendbarkeit für andere Kollegen
- Selbststudium möglich
- eLearning - Einsatz

Risiken des Projektes

- Ausfall von Hardwareressourcen
- nicht korrekt funktionierende Software
- nicht kalkulierbare Verzögerungen durch beruflichen Einsatz
- keine Akzeptanz der Schüler
- zeitliche Fehldimensionierung der Lektionen
- inhaltliche Fehldimensionierung

2.2 Projektstrukturplan

- ***1 Meilenstein***
  Projektideensuche

1.1 Auswahl des Projektthemas
1.2 Kontaktaufnahme und Besprechung mit dem zukünftigen Projektauftraggeber
1.3 Kontaktaufnahme und Besprechung mit dem betrieblichen Projektbetreuer
1.4 Festlegen des Projektnamens und Logo

- ***2 Meilenstein***
  Zieldefinition

2.1 Basis objektorientierte Programmierung
2.2 Die grundlegenden Prinzipien des objektorientierten Entwurfs
2.3 Objektorientierte Software und deren Struktur
2.4 Methodik
2.5 Didaktik
2.6 Style Guide

- ***3 Meilenstein***
  Unterrichtsmaterialien

3.1 Literatur
3.2 Lehr- und Arbeitsbuch
3.3 Buch mit Lösungen
3.4 Website

- ***4 Meilenstein***
  Projektdokumentation

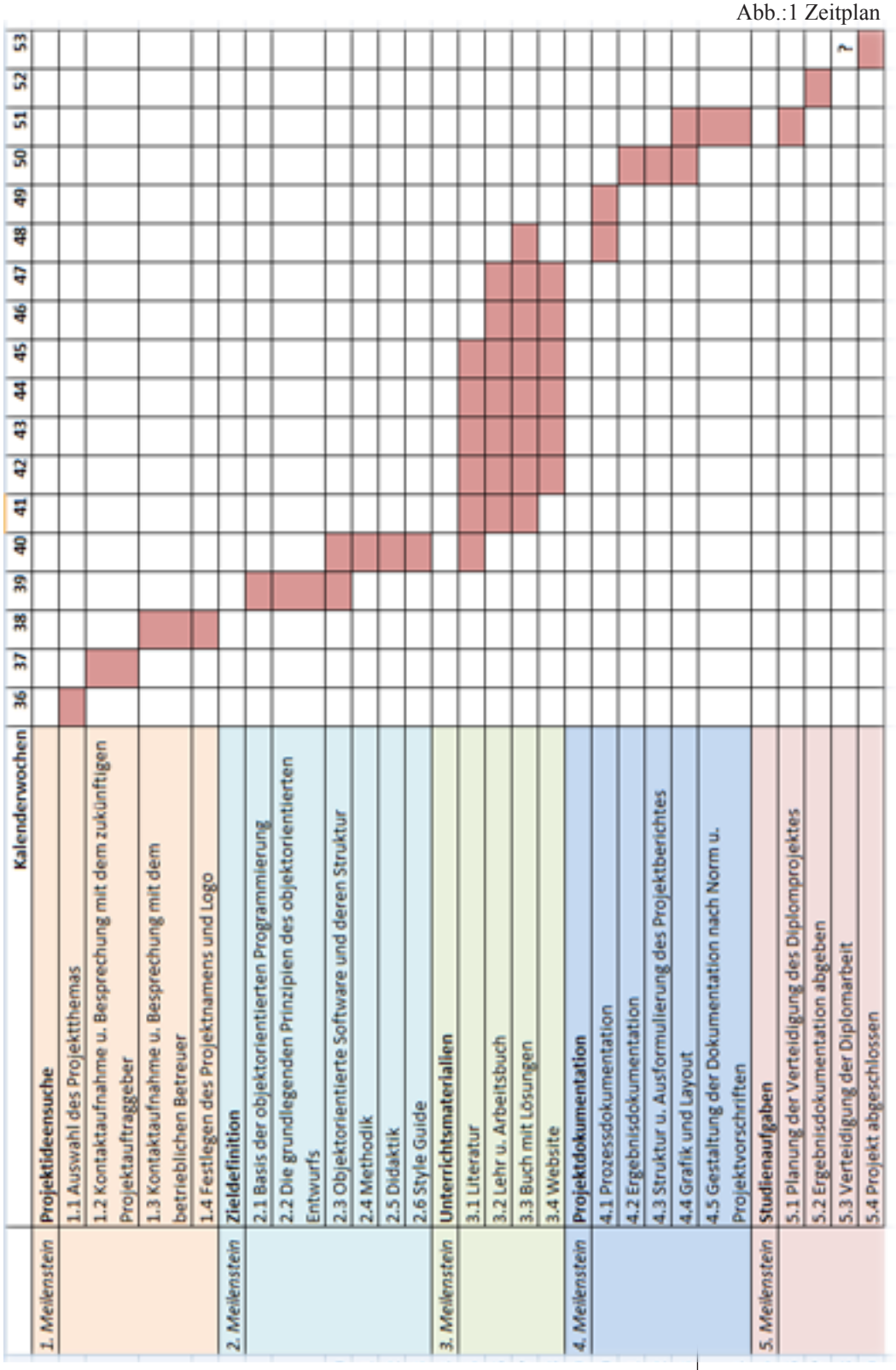
4.1 Prozessdokumentation
4.2 Ergebnisdokumentation
4.3 Struktur und Ausformulierung des Projektberichtes
4.4 Grafik und Layout
4.5 Gestaltung der Dokumentation nach Norm und Projektvorschriften

- ***5 Meilenstein***
  Studienaufgaben

5.1 Planung der Verteidigung des Diplomprojektes
5.2 Ergebnisdokumentation abgeben
5.3 Verteidigung des Diplomprojektes
5.4 Projekt abgeschlossen

2.3 Zeitplan

Abb.:1 Zeitplan



2.4 Sozialer Kontext

Abb.:2 Sozialer Kontext

2.5 Projektabgrenzung

Vorprojektphase (Ist-Zustand)

- Der Lehrplan fordert Unterricht in objektorientierter Programmierung.
- Es gibt keine Abgrenzungen und genauere Definitionen.
- Kaum Lehr- und Lernunterlagen vorhanden, die speziell die Bedürfnisse einer Handelsakademie berücksichtigen.
- Die auf den ersten Blick langweilige Thematik ist nur schwer geeignet die Schüler zu motivieren.

Projektphase

Folgende Aufgaben werden im Rahmen des Projektes durchgeführt:

- Erstellung von methodisch und didaktisch aufbereiteten Lehr- und Lernunterlagen für das Fachgebiet der objektorientierten Programmierung.
- Erstellen von handelsakademiegerechten Unterlagen mit Erklärungen, Aufgaben und Lösungen, sowie Praxistipps für besondere Fälle.
- Erstellung von Zusatzinformationen in Form einer Website mit Animationen und interaktiven Elementen.
- Evaluation von bestimmten Lektionen mit den Schülern

Nachprojektphase

- Evaluation der gesamten Lehr- und Lernunterlagen.
- Verbesserungen und Änderungen auf Grund der Evaluationsergebnisse.
- Kontaktaufnahme mit Lehrbuchverlagen zwecks weiterer Verbreitung
- Einbau der Unterlagen in die eLearningplattform Moodle[1].

1 Moodle ist ein objektbasiertes Kursmanagementsystem, eine Lernplattform auf Open-Source-Basis. Die Software bietet die Möglichkeiten zur Unterstützung kooperativer Lehr- und Lernmethoden

Objektorientierte Programmierung und Java
- interaktiv -
für Handelsakademien

3 Ergebnisdokumentation

Einleitung

Das Thema objektorientierte Programmierung hat bereits viele Abhandlungen und Pro sowie Kontras hinter sich. Viele, man muss wirklich fast sagen Glaubenskriege wurden über dieses Thema ausgefochten. Da sind auf der einen Seite die klassischen Programmierer und auf der anderen Seite Leute, welche der Hintergrund überhaupt nicht interessiert und die nur ergebnisorientiert denken. Die klassischen Programmierer, damit meine ich die sequentielle Herangehensweise, sind meistens Personen mit alter C - Schule und im Extremfall haben sie auch Erfahrungen mit Assembler gemacht. Diese Gruppe neigt schon in früher Projektphase zu einer extremen Detaillastigkeit. Die zweite Gruppe sind oft spätberufene, die vom technischen Hintergrund überhaupt keine Ahnung haben und trotzdem glauben, dass alles möglich ist. Diese Gruppe sieht schon alles wieder so übergeordnet, dass man glaubt, diese Probleme, oder besser gesagt Aufgaben, kann sowieso jeder lösen. Die Schere klafft weit auseinender.
Nun, das Gute an der Sache ist, dass beide bis zu einem gewissen Grad recht haben. Die detailverliebte Gruppe sieht Ecken und Kanten, welche die andere nicht sieht und umgekehrt. Reibungsflächen gibt es trotzdem. Die erste Gruppe verdammt den neuen Kram, weil zu modern, zuwenig durchdacht, da könnte ja jeder kommen und die zweite Gruppe hat Angst vor den versierten Technikern und macht sich über die Produktivität der Assembler und C Programmierer lustig.
Dabei liegt das Problem nur am vorhandenen Hintergrundwissen über den objektorientierten Ansatz. Ich meine damit keinesfalls die Programmierung an sich, sondern die Art objektorientiert zu denken, ja man muss es fast leben um informatiktechnisch damit klar zu kommen. Warum meine ich nicht nur die Programmierung? Nun es ist so, dass sich im Berufsleben, damit muss man ja seine Brötchen verdienen, fast alle Planungen in Projekten die objektorientierte Sichtweise bevorzugen. Zum Beispiel das Aufteilen von großen und schwierigen Aufgaben in viele kleine und überschaubare Einheiten. Auch die Eigenverantwortlichkeit und die Kommunikationsstrukturen haben objektorientierte Ansätze und Strukturen.
Zu Beginn meiner Ausführungen werde ich auf den theoretische Hintergrund und den Aufbau der Objektorientierung eingehen. Da ja auch ein Produkt entstehen soll, muss eine Programmiersprache her. Ich habe Java gewählt, da dies eine moderne und von der Industrie anerkannte Programmiersprache ist. Des weiteren folgen dann die methodisch und didaktisch aufbereiteten Angaben und Beispiele für den Unterricht. Dabei wird die Wissens- und Informationsebene dem erforderlichen Wissen für den IT-Zweig einer Handelsakademie angepasst.

Ich möchte noch einmal auf den Einsatz von Java zurückkommen. Je mehr ich mich mit dieser Programmiersprache befasste, desto mehr kam ich hinter deren grundsätzlichen Absichten, welche ziemlich unterschiedlich zu den mir bis jetzt bekannten Programmiersprachen ist. Was macht man beim Programmieren? Es geht um das Lösen von mehr oder weniger komplexen Aufgaben, wobei die Komplexität[1] der Aufgaben von der Komplexität der Maschine, auf der man diese Aufgaben zu lösen versucht, überlagert wird. Diese Komplexität, ist der Grund, warum viele Projekte, deren Kern die Programmierung ist, ganz einfach scheitern. Mir ist auch keine Programmiersprache bekannt, die geeignet ist den Entwicklungsprozess für alle komplexen Programmieraufgaben optimal zu unterstützen. So musste zum Beispiel C++ so gut als möglich abwärtskompatibel zu C sein um die Umstellung für die große Anzahl von C Programmierern zu erleichtern. Dies ist einer der Hauptgründe für die große Popularität von C++. Der Preis ist allerdings hoch, weil C++ dadurch äußerst komplex ist, was wiederrum auf Kosten der Entwicklungszeit geht und viele Projekte aus Zeitgründen sterben lässt. Als weiteres Beispiel fällt mir die Programmiersprache Visual Basic[2] ein, welche an Basic gebunden ist und von Basic kann man nicht wirklich behaupten, dass dies eine erweiterbare Sprache ist. Als weiters Beispiel möchte ist die Skriptsprache Perl[3] anführen, die abwärtskompatibel ist mit Awk[4], Sed[5], Grep[5] und vielen anden Unix[6] - Hilfsprogrammen. Perl wird vorwiegend zum Ersetzen von Text eingesetzt und das Ergebnis wird oft als „write only“ (nur schreiben) Code beschuldigt und zwar deswegen, weil man den Code dann ganz einfach nicht mehr lesen kann. Auf der anderen Seite gibt es komplexe Aufgaben, die sich mit diesen Programmiersprachen hervorragend lösen lassen.

1 Komplexität bezeichnet allgemein die Eigenschaft eines Systems oder Modells, dass sein Gesamtverhalten nicht beschrieben werden kann, selbst wenn man vollständige Informationen über seine Einzelkomponenten und ihre Wechselwirkungen besitzt.

2 Visual Basic ist eine proprietäre objektorientierte Programmiersprache, deren neuere Versionen auf dem Microsoft .NET Framework basieren. Um zwischen den alten nicht vollständig objektorientierten und den neuen auf dem .NET Framework basierenden Versionen zu unterscheiden, werden erstere bis inklusive Visual Basic 6.0 als Visual Basic Classic, die letzteren hingegen als Visual Basic .NET bezeichnet.

3 Perl ist eine freie, plattformunabhängige und interpretierte Programmiersprache (Skriptsprache), die mehrere Programmierparadigmen unterstützt.

4 Die Programmiersprache (Skriptsprache) awk dient zur Bearbeitung und Auswertung einfacher Textdaten

5 Ein Programm des Betriebssystems Unix

6 Multiuser - Betriebssystem

Je mehr Erfahrung ich mit Java gesammelt habe und je mehr ich die Programmiersprache verstanden habe, desto mehr hat mich beeindruckt, dass Sun[1] größten Wert darauf gelegt hat, den Programmierer zu unterstützen. Das klingt logisch, war aber bis dato einfach nicht der Fall. Man kann sagen, die Zeit und die Schwierigkeiten um robusten Code zu entwickeln sind drastisch gesunken. Dieser Komfort muss natürlich auch irgendwie bezahlt werden. In den Anfangszeiten von Java resultierten daraus relativ speicherhungrige und langsame Programme und es gab große Versprechungen von Sun, wie schnell Java eines Tages laufen würde. Was passierte tatsächlich? Die Entwicklungszeiten wurden drastisch verkürzt, das heißt im Vergleich zu einem äquivalenten C++ Entwicklungsprojekt, um mehr als die Hälfte, was natürlich Zeit und Geld spart. Außerdem ist Java auch hervorragend für komplexe Aufgaben, wie „multithreading[2]“ und Netzwerk-Programmierung geeignet. Eines der größten Computerprobleme wird von Java ebenfalls gelöst. Java ist plattformübergreifend und was den Code anbelangt gilt das Prinzip „write once, run anywhere“, was soviel bedeutet, dass es vollkommen egal ist, auf welcher Maschine der Code entwickelt wird. Der entwickelte Code wird in einen sogenannten Bytecode übersetzt, der dann von einer Virtuellen Maschine, die es für jede Plattform gibt, ausgeführt wird. Weiters wird das dynamische Austauschen von Objekten unterstützt, was einen immensen Vorteil für sich ändernde Bedingungen darstellt. Als letzten Aspekt möchte ich das ausgereifte Sicherheitskonzept von Java erwähnen, ein heikler Punkt, den so manche Mitbewerber von Sun nicht ganz so ernst nehmen. Für die Zukunft sehe ich großes Entwicklungspotential im WWW und in der Netzwerkprogrammierung und dies ist einer der Hauptgründe, warum meine Wahl auf Java fiel. Die IT Zukunft ist schwer vorhersehbar, aber ich möchte mit einer Sache dabei sein, von der ich überzeugt bin. Für die Entwicklung der Lehrunterlagen habe ich mir folgende Hauptziele gesetzt und werde versuchen diesem Weg, wo immer es nur geht zu folgen. Ich werde versuchen das zu lernende Material in so kleine Einheiten wie möglich aufzuteilen, aber es soll trotzdem möglich sein, den Blick für das Gesamte nicht zu verlieren. Die Beisiele sollen ebenfalls kurz und einfach und praxisorientiert sein. Weiters werde ich das erforderliche Hintergrundwissen auf das absolut nötigste reduzieren, um den Lernenden nicht die Freude zu nehmen, aber doch genug Stoff zu liefern, um dem Interessierten eine selbstständige Vertiefung des Stoffes zu ermöglichen.

1 Sun Microsystems ist ein in Santa Clara ansässiger Hersteller von Computern und Software.

2 Mehrere Bearbeitungstasks laufen parallel in einem Prozess ab

Besonders in einer Handelsakademie finde ich es auch angebracht, die Ökonomie in Bezug auf die Softwareentwicklung zu betrachten. Grundsätzlich sollte Software so konzipiert werden, dass sich die Gesamtkosten verringern. Dies hat natürlich mit objektorientierter Programmierung zu tun, da diese Philosophie stark vom Entwurf und der Planung abhängt. Die Gesamtkosten für Software setzen sich aus den „Entwicklungs- und Wartungskosten“ zusammen. Die Industrie hat nun einige Erfahrung mit der Softwareentwicklung gesammelt und kam zu der anfangs wirklich überraschenden Erkenntnis, dass die Kosten der Wartung wesentlich höher lagen, als die Kosten der anfänglichen Entwicklung. Warum ist die Wartung so teuer? Nun, es ist einfach zeitaufwendig und fehleranfällig, den vorhandenen Code zu verstehen. Normalerweise lassen sich Änderungen leicht vornehmen, wenn man genau weiß, was geändert werden muss, aufwendig ist es, zu lernen was der aktuelle Code tut, weiters muss man, nachdem man die Änderungen vorgenommen hat, diese testen und wieder bereitstellen. Dann setzen sich die Wartungskosten aus „Verstehen“ + „Ändern“ + „Testen“ + „Bereitstellen“ zusammen. Wie lassen sich die Gesamtkosten verringern? Man versucht, mehr in die anfängliche Entwicklung zu investieren, in der Hoffnung, den erforderlichen Wartungsaufwand zu verringern, oder sogar ganz zu beseitigen. Diese Vorgangsweise hat meistens nicht den erwünschten Erfolg gebracht. Wenn der Code in nicht vorhersehbarer Weise geändert werden muss, lässt er sich auch nicht auf diese Änderungen vorbereiten, egal wie weit man vorrausdenkt. Die Versuche, den Code für diese Änderungen genügend allgemein zu machen, passen oftmals nicht mit den Änderungen zusammen, die tatsächlich erforderlich werden. Es ist eine Gradwanderung, bei der die objektorientierte Programmierung, zumindest die Philosophie davon, helfen soll, sauberen Code, für eine einfache zukünftige Entwicklung zu schaffen. Diesen sauberen Code bekommt man meiner Meinung nach, wenn man sich auf die Kommunikation zwischen Programmierer und Programmierer konzentriert. Dies bedeutet verständlichern Code, leichtere gemeinsame Nutzung und zügigere Entwicklung.
Gute Softwareentwicklung hat großen ökonomischen Einfluss und ich bin mir sicher, das der objektorientierte Ansatz zur Zeit „State of the Art“ ist. Es ist wie mit der Demokratie. Sie ist sicher nicht die beste Regierungsform, aber ich kenne keine bessere[1].

1 Winston Churchill - Indeed, it has been said that democracy is the worst form of Government except all those other forms that have been tried from time to time. (Es heißt ja, Demokratie ist die schlechteste Regierungsform - mit Ausnahme all der anderen Formen, die von Zeit zu Zeit ausprobiert worden sind.) – In einer Rede im Unterhaus am 11. November 1947.

Überblick

Kapitel 3.1
Einführung in die objektorientierten Konzepte - dieses Kapitel gibt einen Überblick, worum es bei der objektorientierten Programmierung eigentlich geht. Ich versuche auch die grundlegende Frage: „Was ist ein Objekt?“ zu beantworten. Auch werde ich der Frage nachgehen, warum Java so mächtig und erfolgreich ist.

Ist wirklich alles ein Objekt? - hier geht es darum, Objekte zu erzeugen, zu erklären, was Referenzen auf Objekte sind, wie Methoden und deren Rückgabewerte funktionieren und wie Java den Speicher verwaltet.

- Inheritance - Die Wiederverwendung von Klassen, oder auch Vererbung bildet den Schwerpunkt dieses Kapitels.

- Polymorphie - In die Königsklasse gehört wohl die Polymorphie, oder auch Vielgestaltigkeit genannt. In diesem Kapitel erörtere ich das Prinzip und den mit Bedacht zu wählenden, sinnvollen Einsatz.

- Interfaces - Schnittstellen und innere Klassen sind weitere wichtige Konzepte in der objektorientierten Programmierung.

Kapitel 3.2
Analyse und Design - ohne Grundüberlegungen und Basistechniken in Analyse und Design kann man keine vernünftige objektorientierte Software implementieren. In diesem Kapitel werde ich einige wichtige Techniken, zum Beispiel den Einsatz von UML Diagrammen, aufgreifen.

Kapitel 3.3
Moderne Programmiertechniken - Extreme Programming sowie Strategien für die Umsetzung von erfolgreichen OOP Projekten

Die Implementierung - welche Richtlinien sollte man für sich selbst dokumentierenden Code einhalten. Es geht um Werte und Prinzipien, die dem Programmierstil zugrunde liegen sollten.

Kapitel 3.4
Die Lehrunterlagen - die Lösungen - die interaktiven Elemente

3.1 Einführung in die objektorientierten Konzepte

In diesem Kapitel werde ich auf die grundlegenden Konzepte der objektorientierten Programmierung eingehen. Um dies in einen sinnvollen Rahmen zu kleiden ist es unabdingbar auch auf die wesentlichsten Methoden der Softwareentwicklung einzugehen und ich werde auch hierzu einen Überblick geben. Als Basis setzte ich dabei ein Grundwissen in prozeduraler Programmierung, möglicherweise aber nicht zwingend in C, voraus.

3.1.1 Der Abstraktionsprozess

Da gibt es einmal die Nullen und Einsen, die ein Prozessor versteht. Damit kann man nicht besonders produktiv arbeiten. Es entstehen die Assemblersprachen, die bereits einen gewissen Abstraktionslevel gegenüber den Nullen und Einsen besitzen. Doch es geht weiter, C[1] und andere sogenannte imperative Sprachen, wie FORTRAN[1] erscheinen mit noch höherem Abstraktionspotential auf der Bildfläche. Damit meine ich, dass diese Programmiersprachen mit wenigen Befehlen eine Unzahl von Assembleranweisungen nachbilden. Dies erhöht die Produktivität unheimlich und die Fehleranfälligkeit sinkt ebenfalls. Trotzdem wird noch sehr nahe an der Maschine gearbeitet und dies nimmt die ganze Kapazität des Programmierers in Anspruch und er kann sich nicht in entsprechendem Ausmaß dem gesamten Projekt widmen. Damit meine ich, der Aufbau dieser Programmiersprachen erfordert noch ein sehr hohes Maß an Verständnis für die Maschine. Daraus ergibt sich, dass die Programme schwierig zu schreiben und teuer zu warten sind. Es kam eine richtige Industrie auf, die sich nur mit dem Programmieren von Methoden beschäftigte.
Das heißt, man kann sich vorwiegend mit der Maschine herumschlagen, oder sich als Alternative mit der eigentlichen Aufgabe beschäftigen. Da gab es schon einige Programmiersprachen, die in diesen frühen Zeiten diesen Ansatz beherzigten, zum Beispiel LISP[1], APL[1], oder PROLOG[1]. Die Programmiersprache PROLOG führte zum Beispiel alle Aufgaben auf eine Kette von Entscheidungen zurück. Das war schon ein guter Ansatz, deckte aber bei weitem nicht den größten Teil der Aufgaben ab. Der objektorientierte Ansatz geht einen Schritt weiter und bietet dem Programmierer elementare Werkzeuge, die den Aufgaben angepasst sind, an. Diese Werkzeuge sind abstrakt genug, damit sich der Programmierer nicht mit Details von Problemen herumschlagen muss.

1 Programmiersprachen

Diese Elemente im Bereich der Aufgaben und im Bereich der Lösungen werden als Elemente bezeichnet[1]. Doch damit nicht genug, es gibt auch noch andere Elemente, die benötigt werden, die jedoch mit der Problemlösung nicht in direktem Zusammenhang stehen. Der Trick dabei ist, dass sich das Programm je nach Anforderungen anpassen kann und dies durch Definition von neuen Typen von Objekten. Das Programm entwickelt sich sozusagen weiter. Ein kleiner aber nicht unbedeutender Evolutionsprozess. Dies ist der größte und flexibelste Abstraktionsprozess, den es bis dato gegeben hat. Die objektorientierte Programmierung erlaubt es, ein Problem in Unterprobleme aufzugliedern und nicht in Aufgabenbereiche, bei denen man sich nur um die Maschine kümmern muss, auf der das Problem gelöst werden soll. Aber im Endeffekt ist der Zusammenhang aber doch wieder gegeben. Jedes Objekt sieht nämlich aus wie ein kleiner Computer , es hat nämlich einen Status und ein bestimmtes Verhalten, dies sind die Funktionen, die ich anstoßen kann. Ist es nicht im wirklichen Leben auch so? Alles hat einen charakteristischen Zustand und ein bestimmtes Verhalten.
Alan Kay[2] fasste fünf charakteristische Eigenschaften von SmallTalk[3] zusammen, sozusagen der ersten objektorientierten Programmiersprache.

- Alles ist ein Objekt (Everything is an object)
- Ein Programm ist eine Ansammlung von Objekten, die sich untereinander sagen, was zu tun ist, indem sie sich Nachrichten senden.
- Jedes Objekt hat einen von anderen Objekten getrennten Speicher
- Jedes Objekt hat einen Typ - das ist die Klasse (die charakteristische Frage für eine Klasse ist, „Welche Nachricht kann ich an die Klasse senden“?)
- Alle Objekte vom selben Typ können die gleichen Nachrichten empfangen.

Die Quintessenz: Jedes Objekt hat Daten, Methoden und kann eindeutig identifiziert werden. Es hat einen Status, ein Verhalten und eine Identität.

1 vgl. (14) Thinking in Java, Bruce Eckel, S 122

2 Alan Curtis Kay (* 17. Mai 1940 in Springfield, Massachusetts) ist ein Wissenschaftler der Informatik, bekannt für seine frühen Arbeiten über objektorientierte Programmierung und über die Gestaltung von Bedienoberflächen.

3 Erste objektorientierte Programmiersprache

3.1.2 Die Schnittstellen eines Objektes

Was hat die Natur[1] zu bieten? Nun, da gibt es die Klasse der Wirbeltiere, der Säugetiere, der Insekten, der Vögel, der Fische etc.
Das heißt wohl, dass solange die Objekte Teil einer einzigartigen Klasse sind, auch so genannt werden können und alle haben ein bestimmtes Verhalten und bestimmte Eigenschaften. Die erste objektorientierte Programmiersprache Simula67[2] hat genau auf diese Art und Weise den Klassenbegriff eingeführt. Das Schlüsselwort Klasse wurde als neuer Typ in der Programmierung eingeführt. Der Name Simula entstand aus dem Einsatzbereich, der erste erfolgte nämlich vorwiegend für Simulationen im Bankwesen, zum Beispiel für komplexe Finanztransaktionen. Zusammenfassend kann man sagen dass in der objektorientierten Programmierung immer neue Datentypen erzeugt werden, welch als Klassen bezeichnet werden. So können viele Aufgaben mit richtig eingesetzten objektorientierten Techniken einer Lösung zugeführt werden. Wie schon gesagt: „Ein große, fast unlösbare Aufgabe wird in viele kleine, leicht lösbare Teilaufgaben zerlegt“. Ein Weg, der in der Technik sehr oft gegangen wird. So helfen die Fourier - und Laplace[3] - Transformationen auf elegante Weise, komplexe Differentialgleichungen zu lösen. Das Prinzip ist genauso einfach, wie genial. Lieber viele einfache Berechnungen durchführen, als eine komplizierte.

Wenn eine Klasse einmal entworfen ist, kann man daraus so viele Objekte, wie nötig erzeugen. Man denke dabei an eine Form, zum Beispiel für Waschbecken, aus der beliebig viel Waschbecken erzeugt werden können. Danach kann man diese Objekte verändern und den Bedürfnissen anpassen, um die verschiedensten Aufgaben zu lösen. Ein grundsätzlicher Gedankenansatz in der objektorientierten Programmierung ist es also, von der Rohform bis zum fertigen Produkt, die Produktion konsequent durchzudenken. Damit meine ich, dass zuerst die Form passen muss, bevor man in die Massenfertigung geht. Wie bekomme ich nun ein Objekt dazu, dass es für mich eine sinnvolle Position und Funktion einnehmen kann? Es muss für mich eine Möglichkeit der Kommunikation mit dem Objekt geben.

---

1 Aristoteles war der erste, der das Prinzip der Einteilungen und Klassifizierungen erfasste

2 Programmiersprache, die von Ole-Joan Dahl und Kristen Nygaard in den 1960er Jahren in Norwegen entwickelt wurde, um Simulationen von physikalischen Prozessen am Rechner durchführen zu können.

3 Französische Mathematiker

Als Beispiel dafür möchte ich die Kochplatte eines Elektroherdes heranziehen. Welche Zustände kann so eine Platte haben? Sie kann ein, oder ausgeschaltet sein und sie kann auf irgendeinen Wert eingestellt sein, der Einfachheit halber wähle ich die Werte „warm“, „mittel“ und „heiß“. In diesem Fall müsste ich dem Objekt sagen können: „Jetzt schalte ich dich ein und der Schalter soll auf Stellung mittel sein“. Die Sache funktioniert so, indem man Anfragen (requests) an das Objekt sendet. Das Objekt selbst führt dann die gewünschte Anweisung aus. Dies geschieht über eine sogenannte Schnittstelle (Interface), die jedes Objekt besitzt und die je nach Programmdesign gestaltet ist.

Abb.:3 Schnittstellen von Klassen

Erzeugen einer Instanz der Klasse:

```
Kochplatte kp = new Kochplatte();
```

Zugriff auf die Funktionalität / Methoden:

```
kp.ein();
kp.aus();
kp.warm();
kp.mittel();
kp.heiss();
```

Um einen Request (Message) an das Objekt zu senden, genügt es auf die Referenz, in diesem Fall kp und auf die entsprechende Methode, getrennt durch einen Punkt, hinzuweisen. Das war es, das ist das Prinzip um mit Objekten zu arbeiten. Das zuvor angeführte Diagramm folgt den Spezifikationen von UML[1]. Ein Rechteck repräsentiert die Klasse, in dem der Name ganz oben steht. Die Daten, oder im Fachjargon als Membervariablen bezeichnet, stehen in der Mitte und ganz unten sind die Methoden aufgeführt.

1 Unified Modeling Language ist eine standardisierte Sprache für die Modellierung von Software und anderen Systemen

Um Programmdesign zu verstehen, kann man sich vorstellen, dass es viele Objekte gibt, die alle gewisse Teile einer speziellen Aufgabe lösen können, dies wäre zum Beispiel eine Bibliothek von Objekten. Gibt es kein passendes Objekt, oder eine Kombination von mehreren, dann muss ein entsprechendes neues Objekt erzeugt werden. Als Grundprinzip von gutem objektorientiertem Design gilt dass ein Objekt eine Sache perfekt machen soll, aber keinesfalls zuviele Dinge können soll. Es macht Sinn, ein Objekt als Dienstleister zu betrachten. Ich will etwas und die Dienstleistungsfirma versucht die Aufgabe so gut wie möglich zu lösen. Wenn jemand anders das Programm lesen und verstehen soll, hat dies ebenfalls große Vorteile, denn man kann sich mit der zuvor beschriebenen Technik viel leichter in den Code eines anderen Entwicklers hineinversetzen.

Man kann die Spielwiese der objektorientierten Programmierung in zwei große Gruppen einteilen. In einer Gruppe sind diejenigen, welche die Klassen benutzen und die Datentypen in ihren Anwendungen einsetzen, in der anderen sind diejenigen, welche neue Datentypen erzeugen. Es interessiert den Benutzer der Klassen überhaupt nicht was im Inneren der Klasse passiert, diese muss nur die geforderte Funktionalität haben. Derjenige der diese Klassen erzeugt sorgt dafür, dass nur das Wesentliche, also das was der Kunde will, der Klasse sichtbar ist. Diese restriktive Vorgangsweise vermindert ungemein die Fehleranfälligkeit, da die Verantwortungen delegiert werden und die Aufgaben in kleinen und damit exakt testbaren Häppchen erfolgt. Java verfügt über ein strenges Sicherheitskonzept, welches es ermöglicht die teilweise versteckten Implementierungen umzusetzen. Dies wird durch den Einsatz der Schlüsselwörter „public“, „private“ und „protected“1. „Public“ bedeutet, dass jeder Zugriff auf das Objekt hat. „Private“ bedeutet, dass außer dem Besitzer, oder Erzeuger, niemand Zugriff auf das Objekt hat. Das „protected“ Schlüsselwort hat die gleiche Bedeutung wie „private“, außer dass es auch für vererbte Klassen gilt.

1 Zugriffsmodifizierer

3.1.3 Die Wiederverwendung von Klassen

Wenn eine Klasse einmal implementiert und erfolgreich getestet wurde, dann stellt sie ein brauchbares Stück Code dar und man meint, dieses Juwel könnte man so ohne weiteres wo anders einsetzen. Dies stimmt jedoch nur bedingt, da es sehr großer Erfahrung und Übersicht des Programmierers bedarf, wiederverwendbare Klassen zu erzeugen. Sollte es aber gelungen sein eine Klasse so zu entwerfen, dann bettelt diese förmlich darum, wiederverwendet zu werden und genau diesen Wiederverwendbarkeit von Klassen ist einer der größten Vorzüge, den objektorientierte Programmiersprachen bieten.
Am einfachsten ist es, wenn man ein Objekt einer Klasse direkt erzeugt, es ist aber auch möglich ein Objekt dieser Klasse innerhalb einer anderen Klasse zu implementieren. In diesem Sinne hat man die Funktionalität einer bestehenden Klasse mit der Funktionalität der eingebetteten Klasse erweitert. Dieses Prinzip wird als Komposition[1] bezeichnet. Wenn dieser Vorgang dynamisch, also zur Laufzeit des Programmes passiert, dann bezeichnet man diesen Vorgang als Vereinigung[2]. Diese Beziehung wird oft mit „hat ein[3] ...“ verdeutlicht, wie zum Beispiel: „Ein Auto hat eine Maschine“. Es folgt das zugehörigen UML Diagramm, mit der Raute als charakteristischem Symbol für die Aussage, dass das Auto einen Motor hat. Eine weitere elegante Möglichkeit für die Wiederverwendung von Klassen ist die Vererbung, der ich das nächste Kapitel widmen werde.

Abb.:4 Komposition

Komposition ist eine der wichtigsten objektorientierten Programmiertechniken. Üblicherweise sind die inneren Klassen „private“, dass heißt, jene Programmierer, die diese Klassen verwenden haben keinen Zugriff darauf, auch nicht zur Laufzeit. Sollte bei der Klasse etwas geändert werden, aus welchen Gründen auch immer, dann muss der Programmierer der Klasse keine Sorge haben, dass die Methoden der inneren Klasse auch woanders gebraucht werden. Dies erleichtert die Wartung bei komplexeren Klassenstrukturen.

1 Composition engl. - Fachbegriff
2 Aggregation
3 ... has a

3.1.4 Die Vererbung von Klassen

Die Vererbung ist trotz ihrer Wichtigkeit ein sehr heikles Thema und wird auf Grund der Bekanntheit sehr oft mit übertriebenen Lorbeeren überhäuft. Deswegen glauben viele junge und unerfahrene Programmierer dass die Vererbung so oft wie möglich eingesetzt werden soll. Das Ergebnis resultiert in oft hässlichem und überkompliziertem Softwaredesign. Wenn man eine neue Klasse entwirft ist es zweifelsohne besser, sich zuerst mit „Composition“ zu befassen. Diese Vorgangsweise ist flexibler und einfacher und führt damit zu klarerem Design und mit etwas Erfahrung wird es dann bald klar ersichtlich, wann man „Inheritance[1]“ einsetzen soll.

Die Idee der Objektorientierung ist an sich ein ausgezeichneter Ansatz und führt, wie bereits ausgeführt, prinzipiell zur Erzeugung von Klassen und dafür wird das Schlüsselwort „class“ eingesetzt. Wie kam es nun zur Einführung der Vererbung? Es gibt Klassen, in denen viel Hirnarbeit steckt und es gibt Aufgaben, bei denen man diese Fähigkeiten der Klassen nutzen und erweiteren möchte. Jene Klasse, von der geerbt wird, heißt üblicherweise „Elternklasse[2]“ und jene Klasse, die von dieser Klasse geerbt hat, nennt man „Kinderklasse[3]“. Die folgende Abbildung zeigt die UML konforme Darstellung für die Vererbung. Dabei ist zu beachten, dass der Pfeil immer von der abgeleiteten Klasse zu Elternklasse zeigt.

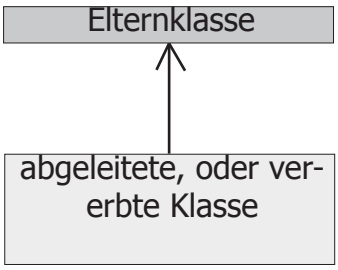


Abb.: 5 UML Diagramm für Vererbung

Die Theorie der objektorientierten Programmierung erlaubt es grundsätzlich, dass man von mehreren Klassen ableiten kann. Diese Technik wird als „Mehrfachvererbung[4]“ bezeichnet und die Gefahr besteht darin, bald die Übersicht zu verlieren. Um die Programme transparenter und weniger fehleranfällig zu machen, ist in Java gleichzeitiges Ableiten, oder Vererben von mehreren Klassen nicht möglich. Sollte es jedoch trotzdem sinnvoll erscheinen, so wird dies, wie ich später zeigen werde, über Schnittstellen umgesetzt.

1 Vererbung
2 Parent - class
3 Child - class
4 Multiple Inheritance

Wenn man von einer existierenden Klasse ableitet wird ein neuer Datentyp, oder besser eine absolut neue Klasse erzeugt und es ist wichtig, zu wissen, dass nicht nur alle Instanzvariablen der Basisklasse durch die Vererbung erhalten bleiben, sondern auch die Schnittstelle der Klasse vererbt wird. Dies bedeutet, das alle Nachrichten[1], die an die Basisklasse geschickt werden konnten, auch für die abgeleitete Klasse möglich sind. Das Verständnis dieser Tatsachen über die Vererbung ist der geistige Eintritt zur objektorientierten Programmierung und gehört damit zu den grundlegenten Prinzipien um mit der Materie der OOP zu arbeiten. Und es geht noch weiter. Man kann in der geerbten Klasse auch auf alle Methoden der Elternklasse zugreifen. Damit kommen wir zu der Erkenntnis, dass die vererbte Klasse, nicht nur den gleichen Status[2] wie die Elternklasse hat, sondern auch dieselben Eigenschaften[3].

Welche Möglichkeiten gibt es nun die vererbte Klasse um bestimmte Funktionalitäten zu erweitern. Zum einen kann man einfach neue Methoden implementieren, die das gewünschte neue, zusätzliche Verhalten umsetzen, zum Anderen gibt es die Möglichkeit des Überschreibens[4] von Methoden. Dabei hat die Methode die gleiche Bezeichnung wie in der Elternklasse, im Detail erfolgt jedoch eine andere Funktionalität. Auch die Signatur, die Anzahl und der Typ der Parameter kann variieren. Will man wieder auf die ursprüngliche Methode der Elternklasse zurückgreifen, geschieht dies mit dem Schlüsselwort „super“ und dem Methodennamen. Das wichtigste Schlüsselwort für die Vererbung ist „extends“ welches auf die Elternklasse hinweist. In der Implementierung lautet die Syntax beispielsweise: *class A extends B*

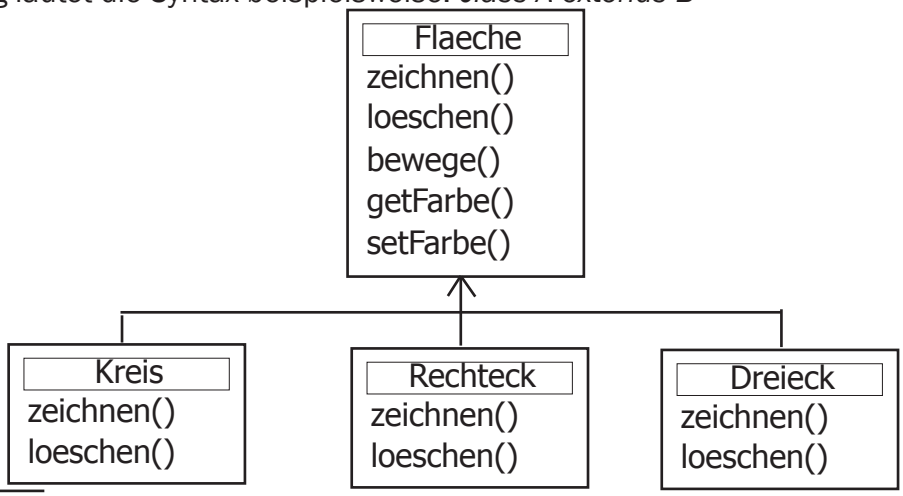


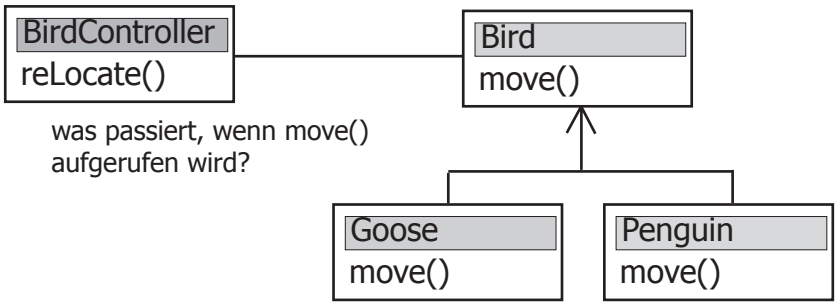
Abb.: 6 UML Diagramm für Vererbung und Überschreiben

1 Messages, Requests
2 State
3 Behavior
4 Overwriting

3.1.5 Untereinander austauschbare Objekte - Polymorphie

Beim Arbeiten mit Klassenhierachen kann es vorkommen, dass man nicht mit der spezifischen Klasse arbeiten möchten sondern mit der Basisklasse. Dies ermöglicht es, Klassen zu schreiben, die nicht an einen spezifischen Typ gebunden sind. In dem zuvor genannten Beispiel wurden die Flächen ganz allgemein[1] behandelt, unabhängig davon, ob es sich um einen Kreis, ein Quadrat, oder um ein Dreieck handelt, oder sogar um irgendeine Fläche, die noch nicht einmal definiert wurde. Alle Flächen können gezeichnet, bewegt und gelöscht werden, indem diese Methoden einfach eine Nachricht an das Flächenobjekt senden und niemand macht sich darüber Gedanken, wie das Objekt mit der Nachricht umgeht. Man kann ohne weiteres eine neue Fläche, zum Beispiel ein Fünfeck einführen, ohne die darunterliegenden generischen Methoden ändern zu müssen. Dies ist eine äußerst elegante Methode, um Änderungen zu kapseln und erweitert die Designmöglichkeiten bei gleichzeitiger Reduktion der Entwicklungskosten.

Es gibt jedoch ein Problem mit den generischen Methoden. Zur Kompilierzeit weiß der Kompiler nämlich nicht, welches Objekt er mit dieser Methode angreifen soll. Erst zur Laufzeit wird die entsprechende Entscheidung getroffen. Um dies zu veranschaulichen werde ich das Beispiel mit dem BirdController von Bruce Eckel[2] unter die Lupe nehmen. Dieses BirdController Objekt arbeitet mit generischen Bird - Objekten, von denen es nicht genau weiß, von welchem Typ diese sind. Dies hängt ganz einfach von der Perspektive des BirdControllers ab. Was passiert nun wenn die generische Methode move() aufgerufen wird? Es wird genau das Richtige passieren. Die Gans läuft, fliegt, oder schwimmt und der Pinguin läuft oder schwimmt.



1 Generisch
2 (14) Thinking in Java Seite 220

Dies ist die wichtigste Verflechtung in der objektorientierten Programmierung. Ein nicht OOP Kompiler würde auf die folgende Art und Weise funktioniern: Der Fachbegriff, wie der Kompiler das anstellt, heißt „early binding[1]“. Dies bedeutet, dass der Kompiler einen speziellen Aufruf zu einer bestimmten Funktion generiert und der Linker löst diesen Aufruf zu der absoluten Adresse des Codes, der ausgeführt werden soll, auf.

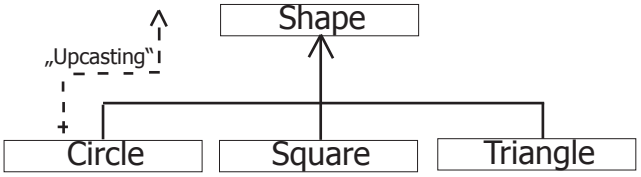
In OOP kann das Programm die Adresse des Codes, der ausgeführt werden soll nur zur Laufzeit erkennen, deswegen ist ein anderes Schema notwendig, um eine Nachricht an ein generisches Objekt zu senden. Diese Konzept der OOP-Sprachen heißt „late binding[2]“. Beim Senden einer Nachricht an eine Objekt ist der auszuführende Code bis zu Laufzeit des Programms nicht festgelegt. Der Kompiler stellt nur sicher, dass die Methode existiert und führt eine Typkontrolle bei den Argumenten und den Rückgabewerten durch, aber er kennt nicht genau den auszuführenden Code. Man bezeichnet jene Programmiersprachen, welche diese Typkontrolle nicht durchführen als „weakly typed[3]“. Um dieses Problem zu lösen, benutzt Java einen speziellen Code, anstelle des absoluten Aufrufes. Unter Verwendung der Informationen, die in diesem Objekt gespeichert sind, berechnet dieser Code die Adresse des Rumpfes der Methode. Dadurch hat jedes Objekt spezielle Vereinbarungen zu diesem speziellen Teil von Code. Wenn eine Nachricht an das Objekt gesendet wird, erkennt das Objekt sofort, was es mit dieser Nachricht machen soll. (Ein Vergleich mit C++ sei erlaubt, dort muss ein spezielles Schlüsselwort, nämlich „virtual“ eingegeben werden, um das „late binding“, anzustoßen). In diesen Sprachen gibt es standardmäßig keine dynamische Bindung, in Java schon und man benötigt keine eigenen Schlüsselwörter um „Polymorphie[4]“ zu codieren.

```
void doStuff(Shape s) {
  s.erase();
  // ...
  s.draw();
}
```

```
Circle c = new Circle();
Triangle t = new Triangle();
Line l = new Line();
doStuff(c);
doStuff(t);
doStuff(l);
```

1 Frei übersetzt - frühe Bindung
2 Frei übersetzt - späte Bindung
3 schwach typisiert
4 vielgestaltigkeit

Abb.: 7 Polymorphie



Was passiert hier?

Ein Kreis (Circle) wird an eine Methode übergeben, die eine Form (Shape) erwartet. Weil der Kreis eine Form ist, kann er von der Methode doStuff() als solche behandelt werden. Ein Kreis kann jede Nachricht akzeptieren, welche die Methode doStuff() an die Form sendet. Auf diese Art und Weise ist das ganze absolut sicher und logisch. Wenn eine abgeleitete Klasse behandelt wird wie die Basisklasse, so bezeichnet man diesen Vorgang als „upcasting[1]". Das Prinzip bedeutet nichts weiteres, als dass man nicht mehr sagen muss, „wenn du ein Kreis bist mache dies" und „wenn du eine Form bist mache jenes". Wenn man einen Code schreiben muss, der alle Eventualitäten abdeckt, wird die Sache mühselig und außerdem noch unwartbar, da man bei einer Änderung auch alle Methoden ändern müsste. In objektorientierter Sichtweise heißt es einfach, du bist eine Form und damit weiß ich, dass du dich zeichnen und löschen kannst, also bitte kümmere dich selbst darum, diese Dinge richtig zu machen.
Das Eindrucksvolle in der Methode doStuff() ist, dass immer das Richtige gemacht wird, egal was passiert. Für einen Kreis bedeutet der draw() Aufruf etwas anderes als der draw() Aufruf für ein Quadrat oder eine Line, aber wenn der draw() Aufruf an eine anonyme Form gesendet wird, wird das richtige mit der richtigen Form gemacht. Das ist eine aufregende Eigenschaft, da der Java Kompiler nicht wissen kann, welche Form gemeint ist, zu jener Zeit, wenn der Code kompiliert wird. Der Kompiler und das Runtime System bewerkstelligen diese korrekte Arbeitsweise.

Die Quintessenz: Wenn ich eine Nachricht an ein Objekt sende, weiß das System damit richtig umzugehen, auch wenn dabei ein Upcasting erforderlich ist.

1 Eine Form der Typumwandlung

3.1.6 Schnittstellen und abstrakte Basisklassen

In einem Softwaredesign kann es manchmal vorkommen, dass die Basisklasse nur eine Schnittstelle für die abgeleitete Klasse sein soll. Der Sinn dabei ist, dass nicht jeder so ohne weiteres eine Instanz der Basisklasse anlegen können soll, um diese Schnittstellen über „upcasting“ zu nutzen. Die wird dadurch ermöglicht, dass man die Klasse unter Verwendung des Schlüsselwortes „abstract“ als abstrakt deklariert. Der Kompiler verbietet alle Versuche, ein Objekt, besser eine Instanz, dieser Klasse zu erzeugen. Dies ist eine elegante Möglichkeit, ein spezielles Design zu entwerfen.
Es gibt auch die Möglichkeit, eine Methode, die noch nicht implementiert ist, als abstrakt zu deklarieren. Dies geschieht dann, wenn man zeigen will, dass es eine Schnittstellenmethode für alle abgeleiteten Klassen gibt, zu diesem Zeitpunkt aber noch keine Implementierung möglich ist. Es ist jedoch zu beachten, dass eine abstrakte Methode nur in einer abstrakten Klasse deklariert werden kann und außerdem muss die Methode in der abgeleiteten Klasse unbedingt implementiert werden, auch wenn nur der Rumpf da steht. Dies bedeutet meistens einigen Mehraufwand und aus diesem Grund ist die Schnittstelle - Interface[1] eine logische Erweiterung der abstrakten Klassen. In diesem Fall muss nämlich in der abgeleiteten Klasse nicht jede Methode des Interfaces implementiert werden, sondern nur die wirklich benötigten und das erspart unsinnige Coderümpfe. Eingesetzt wird das Schlüsselwort „interface“ um das Konzept der abstrakten Klassen weiterzuführen. Es ist auch möglich, mehrere Interfaces miteinander zu verbinden, im Gegensatz zur Mehrfachvererbung[2], die in Java nicht erlaubt ist.

1 Schnittstelle
2 Multiple Inheritance

3.1.7 Die Lebensdauer von Objekten

Eigentlich behandelt OOP nur Klassen, abstrakte Klassen, Vererbung, Schnittstellen und Polymorphie, aber ich meine, es gibt noch einige andere Dinge, die es wert sind, ins Auge gefasst zu werden. Eines davon ist die Frage, wie Objekte erzeugt werden und natürlich auch, wie werden die Objekte wieder zerstört? Oder, „wo sind die Daten des Objektes?“ und „wie wird die Lebensdauer der Objekte kontrolliert“? Es gibt da verschiedene Philosophien und Ansätze. C++ setzt darauf, dass die Effizienz eines Programmes das wichtigste ist und gibt dem Programmierer verschiedene Auswahlmöglichkeiten. Beim Schreiben des Programmes kann über den Speicher und den Lebenszyklus entschieden werden. Zum Beispiel werden die Objekte für maximale Programmlaufgeschwindigkeit am Stack[1], oder im statischen Speicherbereich abgelegt. Die Prioritäten für Speicherbelegung und Freigabe haben somit höchste Priorität und diese Kontrolle über die Zugriffsgeschwindigkeiten kann in manchen Situationen unheimlich wertvoll sein. Es gibt natürlich auch einen Nachteil und der liegt in der geopferten Flexibilität, da man, während man das Programm schreibt, exakt den Wert, die Lebensdauer und den Typ des Objektes kennen muss. Für die meisten gebräuchlichen Programme, wie CAD[2], Lagerlogistik oder zum Beispiel Flugverkehrskontrollen ist dies zu restriktiv und zu wenig produktiv.
Ein weiterer Ansatz sieht so aus, dass die Objekte dynamisch in einem Speicherpool, welcher als „Heap[3]“ bezeichnet wird, angelegt werden, mit dem Nachteil, dass man eben zur Laufzeit noch nicht weiß, wie viele Objekte man braucht, wie es mit der Lebenszeit aussieht und welchen Datentyp diese haben. Dies ergibt sich erst spontan zur Laufzeit des Programmes. Wird ein neues Objekt benötigt, wird es zu dem Zeitpunkt, an dem man es benötigt, am Heap erzeugt. Auf Grund der dynamischen Erzeugung zur Laufzeit, dauert dies etwas länger, als die Objekte auf den Stack zu legen. (Einfache Assemblerinstruktionen um den Stackpointer hinauf oder hinunter zu setzen - die Zeit um Heapspeicher zu adressieren hängt vom Speichermechanismus ab) Der dynamische Ansatz macht die generelle logische Betrachtung, dass Objekte grundsätzlich dazu neigen, kompliziert zu sein, so dass der Mehraufwand, die Objekte zu adressieren und Speicher wieder freizugeben, nicht so sehr ins Gewicht fällt und damit die Erzeugung eines Objektes nicht allzusehr beeinträchtigt wird. Um aufwendige Programmieraufgaben zu lösen, erscheint in den meisten Fällen die größere Flexibilität wichtiger.

1 Speicher der nahe am Prozessor liegt
2 Computer Aided Design
3 Datenstruktur

Java verwendet ausschließlich den zweiten Ansatz. Bei jeder Erzeugung eines Objektes wird das Schlüsselwort „new“ verwendet, um eine dynamische Instanz des Objektes zu instanzieren. Die Lebensdauer eines Objektes ist wiederrum ein anderes Problem. Bei Sprachen, die es erlauben, Objekte auf dem Stack zu erzeugen, entscheidet der Kompiler wie lange er das Objekt benötigt und zerstört es dann automatisch. Passiert dies jedoch am Heap, hat der Kompiler keine Ahnung über die Lebensdauer eines Objektes. In C++ muss dies explizit programmiert werden, was auch sehr oft zu „memory leaks[1]“ führen kann, wenn nicht gewissenhaft programmiert wird.
Java verwendet zu diesem Zwecke den „Garbage Collector“, ein Programm, das den Speicher wieder freigibt. Dies geht natürlich auch auf Kosten der Performance, was jedoch durch die Vorteile wieder aufgehoben wird. Die Vorteile gegenüber C++ sind die Vermeidung von „memory leaks[1]“ und da man sich unter anderem darum nicht kümmern muss, wesentlich schnellere Entwicklungszeiten.

1 Fragmentierte Speicherbereiche

3.1.8 Die Hierarchie von Objekten

Java ist eine „single rootet“ Programmiersprache. Dies bedeutet, dass alle Objekte von einem Urobjekt, oder besser von einer Basisklasse, namens „Object“ abgeleitet werden. Alle Objekte einer „single rootet hierarchy“ haben eine gemeinsame Schnittstelle, weil sie vom gleichen fundamentalen Typ sind und man kann auch damit rechnen, dass eine Basisfunktionalität vorhanden ist. Damit weiß man auch dass man diese Funktionalität für alle Objekte einsetzen kann. Ein weiterer Vorteil ist, dass erst diese Hierarchie es ermöglicht, einen Garbage - Collector einzusetzen. Die grundsätzliche Unterstützung wird nämlich in die Basisklasse integriert und damit kann der Garbage - Collector mit jedem Objekt im System kommunizieren. Ohne diese Hierarchie müsste man den Garbage - Collector über Referenzen implementieren, was ungleich schwieriger wäre. Es ist garantiert, dass alle Objekte die richtigen Laufzeitinformationen beinhalten, was auch das „Exception handling[1]“, wesentlich erleichtert.

1 Ausnahmebehandlung

3.2 Die Softwareentwicklung - Analyse & Design

Als Anfänger hat man große Probleme, sich mit einem objektorientierten Projekt zurechtzufinden. Man muss unbedingt die grundlegenden Konzepte verstehen um gute „Designs“ zu entwerfen und die Vorteile der OOP zu nutzen. Analyse und Design sind die wichtigsten Faktoren in der Softwareentwicklung. Wenn diese beiden Faktoren nicht genügend Aufmerksamkeit bekommen, ist das Projekt unweigerlich zum Scheitern verurteilt.

3.2.1 Mögliche Methoden

Dies ist eine Anzahl von Prozessen und Entscheidungsregeln, um die Komplexität von Aufgaben in kleine, verdaubare Stücke, herunterzubrechen. Seit die objektorientierte Programmierung sozusagen auf dem Markt ist, wurden schon viele Methoden formuliert und veröffentlicht, aber es ist ratsam, mit Bedacht damit umzugehen, da die Versprechungen und damit die Erwartungen sehr oft viel zu hoch sind. Am wichtigsten im Entwicklungsprozess ist es, dass man sich nicht im Detail verliert und nicht nur eine Methode einsetzt, da diese vermutlich nicht genau passt, sondern eine Kombination von Methoden verwendet. Die wichtigste Frage die es zu klären gibt ist immer: „Wie sehen die Objekte aus und was sind ihre Schnittstellen“? Das bedeutet die Aufteilung des Projektes in einzelne Klassen und zu untersuchen, welche Nachrichten die Objekte austauschen müssen. Dies reicht im Prinzip um ein Programm zu schreiben. Natürlich braucht man noch mehr um die gestellten Aufgaben zu lösen, aber die zwei vorhin genannten Punkte sind die Mindestanforderung. Analyse und Design können in mehreren Stufen ablaufen. Zu Beginn, oder auf der untersten Stufe ist es erforderlich einen Plan zu entwerfen. Einen Schritt weiter geht es darum, was zu tun ist, gefolgt von dem Schritt, wie man es machen sollte. Danach sollte das Kernstück, oder das Herz des Programmes eintwickelt werden, gefolgt von den Iterationen der Fallstudien. Als letzter Schritt sollte die Weiterentwicklung betrachtet werden.

3.2.2 Schritt A - Der Plan

Es klingt einfach, aber es ist entscheidend zu wissen, welche Schritte man im Entwicklungsprozess gehen wird. Wenn eine Aufgabe bis ins kleinste Detail klar ist, kann man sich ruhig hinsetzen und mit dem Codieren beginnen, meistes wird man aber für den schnellen Griff zur Tastatur fürchterlich bestraft. Eine Hilfe ist es, das Projekt in Meilensteine[1] aufzugliedern.

1 Milestones - Begriff aus dem Projektmanagement

Ich weiß schon, jetzt kommt die Frage: „Wie finde ich in meinem Projekt die Meilensteine heraus“? Nun, eine Faustregel aus dem Projektmanagement ist, dass, wenn ein Milestone nicht erreicht werden kann, eine Weiterführung des Projektes nicht möglich ist. Damit wird es sicher leichter, die richtigen Arbeitspakete zu finden und zu strukturieren. Ein sehr guter Startpunkt ist es auch, wenn man die Aufgabe nur in ein paar kurzen Sätzen formuliert, also absolut das Wesentliche.

3.2.3 Schritt B - Was sollen wir eigentlich tun?

In der sequentiellen Programmierung war die Sache klar. Formuliere die Systemanforderungen, mache ein Pflichtenheft und kümmere dich um die Systemspezifikationen. Dies ist eine Arbeitsweise, bei der man sich im Dokumentendschungel leicht verirren kann und das Projekt eine eigene, ungewollte Dynamik bekommen kann. Im Anforderungskatalog, auch Pflichten- oder Lastenheft genannt, werden alle Punkte aufgelistet die das Produkt können soll, wenn das Projekt beendet ist. Es steht keinesfalls drinnen wie die Aufgaben gelöste werden sollen und man kann sagen, dieses Dokument ist einfach ein Vertrag zwischen dem Auftraggeber und dem Auftragsnehmer, also in diesem Fall das Entwicklerteam. Die Systemspezifikationen bewegen sich schon wieder auf einem anderen Level, es geht hier bereits sehr ins Detail und es ist schon einiges an Teamarbeit erforderlich um das Pflichtenheft und die Systemspezifikation unter einen Hut zu bringen. Noch dazu sollte man bedenken dass es gewisse Personalfluktuationen gibt und nicht immer die selben Personen am Projekt arbeiten. Es ist sehr ratsam das Entwicklerteam nicht mit zuviel Text zu überfordern, sondern mit Listen und Diagrammen zu arbeiten und das Wesentlichste ist: „Nie das Ziel aus den Augen verlieren!“ Wenn man auch eine noch so große Aufgabe in ein paar kurzen Sätzen erklären kann, ist die Motivation für das Team viel leichter und sollte auch zu einer gewissen Begeisterung führen. Motivation ist einer der wichtigsten Projekterfolgsfaktoren. Als Hilfe in dieser Phase gibt es eine Möglichkeit aus dem UML - Fundus, nämlich die „Use Cases[1]“, oder beim „Extreme Programming[2]“ die „User - Stories“. Use Cases identifizieren die Schlüsselpunkte in einem System, was beim Implementieren dann meistens die Klassen sind.

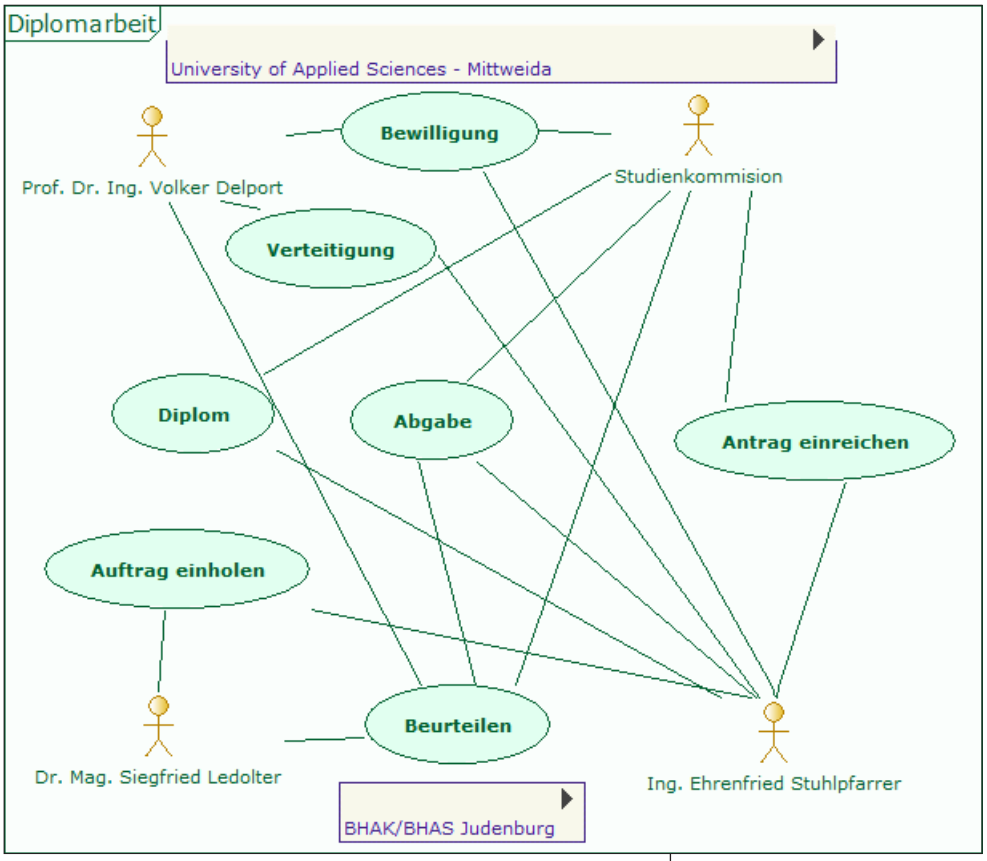
1 UML Diagramm - Use Cases
2 Ansatz für effizientes und fehlerminimiertes Programmieren

Um Use Cases zu erstellen sollte man sich fünf entscheidende Fragen stellen:

1. *Wer wird das System verwenden ?*
2. *Was können die Benutzer mit dem System anfangen ?*
3. *Wie kann der Benutzer mit dem System arbeiten ?*
4. *Wer kann sonst noch etwas mit dem System anfangen ?*
5. *Welche Problem können während der Arbeit auftreten ?*

Der Vorteil von Use Cases ist, dass sie intuitiv und einfach sind und vor Detailverliebtheit schützen. Diese Diagramme schärfen den Blick für das Wesentliche. In Use Case Diagrammen wird immer das „Was“ dargestellt und niemals das „Wie“. Deswegen sollte man auch beachten, das Use Case Diagramme im Gegensatz zu anderen UML - Diagrammen kaum Informationen enthalten. Sie sind nur nützlich, um die Übersicht zwischen den Zusammenhängen von mehreren Use Cases zu fördern.
Als Beispiel folgt ein Use Case Diagramm über mein Diplomprojekt.

Abbildung 8 Use Case Diagramm - Diplomprojekt



3.2.4 Schritt C - Wie wollen wir es machen ?

Zu Beginn dieses Schrittes muss es bereits klar sein, wie die Klassen aussehen sollen und wie sie interagieren. Meine Lieblingstechnik um die Klassen genau zu beschreiben funktioniert mit den sogenannten CRC[1] - Karten. Technisch gesehen ist das System sehr einfach. Man nimmt einfach Kärtchen mit einer ungefähren Abmessung von 8 x 10 Zentimetern und schreibt folgende Informationen darauf:

1. *Den Namen der Klasse (dabei ist es wichtig, das der Name bereits genug Aussagekraft über die Funktion hat).*
2. *Die Aufgaben der Klasse, was soll sie können. Man kann bereits die Namen der Methoden hinschreiben und diese sollen wiederrum so aussagekräftig sein, dass diese bereits die Funktionalität beschreiben, aber ansonsten keine weiteren Informationen enthalten.*
3. *Das Zusammenwirken der Klassen - welche andere Klassen arbeiten mit der eingetragenen Klasse zusammen?*

Ich habe gemerkt, dass die Karten immer zu klein waren und sie trotzdem nicht größer gemacht, da in diesem Fall zu viele Detailinformationen den Blick auf das Wesentliche getrübt hätten. Mit diesen Karten habe ich dann immer eine verbale Simulation gestartet, das heißt sobald eine Klasse instanziert wurde, musste sie ihre Funktion verbal abarbeiten. Das entsrechende Teammitglied musste die Ideen liefern. Der Vorteil liegt im systematischen Aufbau der Klassen, ohne die Sicht auf das Ganze zu verlieren. Ist man auf diese Art und Weise die Use Cases einmal durchgegangen, dann kann man schon auf ein sehr gutes Basisdesign hinweisen. Diese Form des Softwaredesigns ist nicht nur im Team sinnvoll, sondern auch wenn man alleine auf der Strecke bleibt. Es ist nur mühevoller und man muss sich sehr bemühen um konsequent vorzugehen. Was hat man erreicht? Bei den meisten Klassen sind deren Interaktion und Funktionalität festgelegt und wenn etwas fehlt ist das auch kein Problem. Der objektorientierte Ansatz bietet genug Flexibilität um Ergänzungen und Erweiterungen einzuführen.

1 Class Responsibility Collaboration

Gutes Objektdesign passiert immer in mehreren Schritten und es ist auch noch nicht beendet, wenn man bereits am Codieren ist. Da gibt es einmal die Möglichkeit „Design Patterns“ einzusetzen, die dem Programmierer für viele Aufgaben bereits vorgefertigte Strukturen (Muster) anbieten. Der Programmierer kann diese variieren und sich sicher sein, nicht allzuweit daneben zu schießen. Es ist auch ratsam, klein zu beginnen und die Klasse wachsen zu lassen, diese jedoch nicht mit Funktionalität zu überladen. Man sollte sich auch gar nichts daraus machen, zu Beginn nicht alles zu wissen. Man lernt auch, wenn man einen bestimmten Weg geht. Dies funktioniert immer, aber das „Große Ziel“ muss immer in Sichtweite bleiben.

Dazu passt ein Zitat von Albert Einstein[1]: *„Mache die Dinge so einfach wie möglich, aber nicht einfacher „!*

1 Albert Einstein (* 14. März 1879 in Ulm; † 18. April 1955 in Princeton, USA) war ein US-schweizerischer Physiker deutsch-jüdischer Abstammung

3.2.5 Schritt D - Den Kern der Anwendung implementieren ?

In dieser Phase wird die Basisimplementierung durchgeführt und man sollte sich über die Unvollständigkeit in dieser Phase keine Sorgen machen. Diese Basis sollte man als erweiterbares Framework programmieren um viele Möglichkeiten für Änderungen und Erweiterungen zu haben. Ebenfalls wichtig sind in dieser Phase die Systemintegration und umfangreiche Tests. Alle wichtigen Punkte sollte man mit den Verantwortlichen besprechen, diese in die Tests einweihen und entsprechendes Feedback einholen. Hauptzweck der Einbindung der Verantwortlichen ist der, dass sie später nicht sagen können, dass sie nichts gewusst haben und sich das ganze System möglicherweise komplett anders vorgestellt haben. In dieser Phase ist es oft auch erforderlich in der Architektur Änderungen vorzunehmen, oder diese anzupassen. Dies betrifft vor allem Dinge, die man ohne Implementierung nicht wissen konnte. Was die Tests betrifft, so ist genau zu überprüfen, ob sie den Use Cases entsprechen. Wenn das Kernsystem stabil läuft, kann man darangehen, weitere Funktionalitäten zu implementieren.

3.2.6 Schritt E - Use Cases wiederholen

Wenn das Kernsystem einmal läuft, dann ist jede zusätzliche Funktionalität wie ein kleines Projekt zu betrachten. Die Funktionalitäten werden durch konsequentes Durcharbeiten der Use Cases sichtbar. Die Iteration der Use Cases ist beendet, wenn die Zielvorstellungen per Definition erfüllt sind. Im Gegensatz dazu sind Open Source Projekte noch iterativer und von permanentem Feedback abhängig. Der iterative Entwicklungsprozess ist insofern wichtig, da man riskante Teile im Programm früh erkennt und auch der Kunde noch die Chance hat, Anpassungen vorzunehmen und eventuell bei manchen Dingen die Meinung zu ändern. Der Programmierer ist zufrieden und das Projekt kann zielsicherer fortgesetzt werden.

3.2.7 Schritt F - Die Weiterentwicklung

Jetzt kommt die Phase in der das Produkt gefertigt ist und dafür gesorgt werden muss, dass es , im übertragenen Sinn nicht rostet und der Motor immer gut geölt ist. Die Weiterentwicklung eines Projektes hat auch in gewissem Sinne mit Evolution zu tun. Man kann in der ersten Version nicht alles perfekt machen und jede weitere Version dient nicht nur der Fehlerbeseitigung, sondern auch der logischen Weiterentwicklung. Dies ist dann der Punkt, wo ein gutes Projekt zu einem großartigen wird und Dinge die man zuvor nicht verstehen konnte, klar werden und dies führt weiter dazu, dass die Klassen, die für sich im Projekt stehen zu wiederverwendbaren Ressourcen werden. Man muss also nicht nur wissen „was man baut“, sondern auch die Richtung erkennen, in die es weitergehen soll. Objektorientierte Sprachen sind bewusst so aufgebaut, dass das Projekt irgendwie umgesetzt wird, also auch bei Designfehlern und nicht komplett unbrauchbar wird, allerdings leidet unter einem schlechten Design die Wiederverwendbarkeit von Teilen der Implementierung und dies ist einfach schade, da so oder so, viel Hirnarbeit dahintersteckt und die Entwicklungskosten, die ohnehin enorm sind, bei ähnlichen Projekten nicht gesenkt werden können. So wie man kein Haus ohne Plan bauen kann, so ist es auch in der Softwareentwicklung[1]. Ein guter Plan zahlt sich immer aus. Aus meiner Erfahrung und Gesprächen mit anderen Softwareentwicklungsprojektleitern hat sich ergeben, dass ca. 50% aller Projekte scheitern. Der Hauptgrund dafür ist der sofortige Griff zur Tastatur und die mangelnde Planung.

1 Softwaredevelopment

3.3 Moderne Programmiertechniken

Es gibt viele Möglichkeiten Programmcode zu implementieren. Zwei davon erscheinen mir besonders interessant und es wert, näher betrachtet zu werden. Zum einen handelt es sich um „Extreme Programming[1]“ und zum anderen um „Pair Programming[2]“.

1 Extreme Programming (XP), auch Extremprogrammierung, ist eine Methode, die das Lösen einer Programmieraufgabe in den Vordergrund der Softwareentwicklung stellt und dabei einem formalisierten Vorgehen geringere Bedeutung zumisst. Quelle: Wikipedia 5.11.2009

2 Bei Paarprogrammierung (auch Pair Programming genannt) handelt es sich um eine Arbeitstechnik, die sich häufig bei agilen Vorgehensweisen zur Softwareentwicklung findet. So ist Paarprogrammierung beispielsweise ein wichtiger Bestandteil von Extreme Programming (XP)

3.3.1 Extreme Programming

Der Ansatz ist sehr radikal, aber dennoch hochinteressant. Der Begriff und die Definitionen wurden erstmals von Kent Beck, einer anerkannten Persönlichkeit im Bereich der objektorientierten Programmierung und Softwareanalyse, eingeführt. Zwei Aspekte des „Extreme Programming“ habe ich als besonders wichtig eingestuft, nämlich das die Tests zuerst geschrieben werden sollen und das paarweise Programmieren. Das Gebiet an sich ist viel umfangreicher, aber Beck bringt es auf den Punkt und behauptet, dass alleine diese beiden Techniken die Produktivität und die Zuverlässigkeit wesentlich erhöhen können.

„Schreibe die Tests zuerst“

Üblicherweise gehören die Tests zum letzten Teil eines Projektes und die Prioritäten dafür sind meisten auch nicht besonders hoch. XP revolutioniert das Konzept des Testens, indem dieser Arbeit eine höhere Priorität zugewiesen wird als dem Schreiben des Codes selbst. Es ist auch so. Die Tests werden vorher geschreiben, also bevor die eigentliche Software entwickelt wird und diese Tests bleiben von Anfang bis zum Ende mit dem Code verbunden. Immer wenn ein neuer Teil des Projektes entsteht, müssen die Tests gewissenhaft durchgeführt werden. Das Schreiben von Tests zu Beginn der Entwicklung hat zwei extrem wichtige Effekte. Zum Ersten zwingt es dazu, eine klare Definition für die Schnittstellen der Klassen zu finden und zum Zweiten können die Tests bei allen Neuimplementierungen von Klassen eingesetzt werden.

Ich habe in Softwareprojekten sehr oft beobachtet, dass Programmierer idealerweise versuchen, für eine bestimmte Aufgabe die perfekte Klasse zu entwickeln. Die XP Strategie geht etwas weiter und definiert exakt wie die Klasse aus Sicht des Kunden funktionieren muss. Man kann Diagramme zeichnen und Beschreibungen erstellen, so viele man will, aber nichts ist so aussagekräftig, als eine Serie von Tests. Es gibt keine exaktere Beschreibung für die Klassen als die Funktionalitäten der Tests.

Paarweises Programmieren - Pair Programming

Zwei Personen pro Arbeitsplatz sollen den Code implementieren. Eine Person kümmert sich um die Codierung und die zweite Person denkt darüber nach. Der „Denker“ darf dabei nie das Gesamte aus den Augen verlieren und sich nicht in Details verlieren. Einer soll die Tests schreiben und der andere die Klassen für die Anwendung implementieren und beide sollten sich dabei abwechseln. Bei einem gut ausgewählten Zweierteam kann es zu weiteren positiven Effekten, wie zum Beispiel Motivation und Ehrgeiz, sowie zu einer Erweiterung des Programmierhorizontes führen.

3.3.2 Strategien für die Umsetzung

Das Schwierigste ist wie immer die Umsetzung. Das Gebiet der objektorientierten Programmierung ist komplex und gute Programmierer müssen nicht unbedingt teamfähig sein. Also was ist zu tun um für ein anspruchsvolles Projekt ein gutes Team aufzubauen? Am Anfang steht das Training. Gute Firmen haben üblicherweise Interesse daran, dass ihre Mitarbeiter auf dem letzten Stand der Technik sind. Danach wäre es ratsam ein Projekt mit wenig Risiko zu bearbeiten und auch bewusst Fehler zuzulassen und diese ausgiebig zu diskutieren. Das Projekt sollte einfach, selbstbeschreibend und selbsterklärend sein. Weiters ist es hilfreich, gute Beispiele aus dem Bereich der objektorientierten Programmierung zu präsentieren. Es gibt viele Lösungen, die mit einigen Änderungen zu der eigenen Aufgabe passen. Auch der Einsatz von Entwurfsmustern[1] ist vorzusehen. Bereits vorhandene Klassenbibliotheken sind zu durchforsten und wenn sinnvoll, dann auch einzusetzen. Bereits vorhandenen Code in anderen Programmiersprachen sollte man nicht umschreiben, dies ist meisten reine Zeitverschwendung, In diesem Fall ist es besser die Java Native Schnittstelle oder XML[2] zu verwenden.

1 Design Patterns
2 Extensible Markup Language

3.3.3 Richtlinien für die Codierung

Ein wichtiges Prinzip von OOP ist die Wiederverwendbarkeit[1] von Code. Kommentare sind dabei unerlässlich, um auch anderen Programmierern die Chance zu geben, sich in die Arbeitsweise der jeweiligen Klassen hineinzudenken, aber noch besser ist es, den Code so zu schreiben, dass er nahezu selbsterklärend ist. Dies ist meine Motivation für dieses Kapitel, obwohl es auf den ersten Blick so scheint, als habe die Codierung an sich nichts mit OOP zu tun. Es ist kein Geheimnis, wie man Code schreibt, den auch andere Menschen lesen können. Es ist wie das Schreiben im Allgemeinen. Kennen muss man das Publikum, die Gesamtstruktur muss man im Kopf haben und die Details so formulieren, dass sie zur Geschichte passen. So gibt es auch bestimmte Programmiergepflogenheiten, aus denen verständlicher Code resultiert. In der Softwareentwicklung wird viel Geld nur dafür verwendet, vorhandenen Code zu verstehen. Ein Grund mehr, dieses Thema näher zu beleuchten.

Für die meisten Programme gilt, dass sie öfter gelesen als geschrieben werden, weiters gibt es so etwas wie erledigt nicht und in die Modifikation von Programmen wird mehr investiert, als in die ursprüngliche Entwicklung. Wie versteht nun ein Leser ein Programm? Entweder man geht vom Detail zum Konzept über, oder umgekehrt. Entwurfsmuster[2] basieren auf diesen Gemeinsamkeiten. Aber keine Liste von Patterns und sei sie noch so umfangreich kann jede Programmiersituation abdecken. Irgendwann kommt es zu einer Situation, die in keine Schablone passt und dann spricht nur mehr der Code und der wird dann leichter verständlich, wenn man sich an bestimmte Prinzipien hält. Die wichtigsten Werte für gute Programmierung sind Kommunikation, Einfachheit und Flexibilität. Die besten Programme bieten viele Optionen für zukünftige Erweiterungen, enthalten keine irrelevanten Elemente und sind leicht zu erfassen und zu verstehen. Knuth[3] prägte das Paradigma „Literate Programming[4]“. „Ein Programm sollte sich wie ein Buch lesen lassen. Es sollte eine Handlung, einen Rhythmus und reizvolle, kleine Formulierungen enthalten“! Dies ist ein exzellenter Ansatz, eigentlich unverständlichen Code in ein verständliches Dokument umzuwandeln.

1 vgl. (5) Kent, Beck: Implementation Patterns, München 2008, Seite 75
2 Patterns
3 Donald Ervin Knuth (* 10. Januar 1938 in Milwaukee, Wisconsin) ist emeritierter Professor für Informatik an der Stanford University und Urvater des Textsatzsystems TeX.
4 Literarisches Programmieren

Es gibt eine solide wirtschaftliche Grundlage dafür, sich während der Programmierung auf die Kommunikation zu konzentrieren, der Großteil der Softwarekosten entsteht nämlich, nachdem die Software erstmalig eingesetzt wurde. In meiner Zeit als Softwareentwickler habe ich wesentlich mehr Zeit damit verbracht, vorhandenen Code zu lesen, als neuen Code zu schreiben. Möchte ich meinen Code billiger machen (nicht im Sinne der Qualität, sondern Punkto Zeitaufwand und damit verbundenen Kosten), dann sollte ich ihn verständlicher formulieren, indem man sich immer darauf konzentriert, die Ziele des Codes zu vermitteln, dies verbessert den Denkprozess und der Code wird realistischer.

Die Einfachheit

In „The Visual Display of Quantitative Information“ zeigt Edward Tufte[1] eine Übung, in der er in einem Graphen nacheinander alle Markierungen löscht, die keine Informationen beisteuern, mit dem Effekt, dass der resultierende Graph komplett neu ist und außerdem wesentlich leichter zu verstehen. Übermäßige Komplexität sollte man ganz einfach vermeiden, dann können alle, die die Programme erweitern und modifizieren, diese schneller verstehen. Vieles ist auch gewachsen, meistens nur deshalb, damit man das Programm überhaupt zum Laufen bringt. In dieser Zeit häuft sich viel Müll an, der beseitigt werden muss. Zur Programmierung gehört auch, dass man zurückblickt und die Spreu vom Weizen trennt. Man sollte die Einfachheit auf allen Ebenen anwenden und den Code so aufbereiten, dass sich nichts löschen lässt, ohne Informationen zu verlieren.

Die Flexibilität

Ich habe sehr oft erlebt, dass Flexibilität als Rechtfertigung für ineffizienten Code herhalten muss. Als Beispiel kenne ich einen Fall, indem eine Umgebungsvariable abgesucht wird, um den Namen eines Verzeichnisses abzurufen, in dem der konstante Wert zu finden ist. Wozu soll das gut sein? Programme sollten schon flexibel sein, aber nur an Stellen die einer Veränderung unterliegen. Sofern sich diese Konstante nie ändert, bringt die Komplexität nur Kosten und keinen Nutzen. Da die meisten Kosten eines Programmes nach seiner ersten Bereitstellung entstehen, sollten sich Programme leicht ändern lassen, deshalb sollten Entwurfsmuster gewählt werden, die die Flexibilität fördern und unmittelbaren Nutzen bringen. In den Lektionen werde ich noch näher auf dieses Thema eingehen.

1 Edward Rolf Tufte (* 14. März 1942 in Kansas City, Missouri) ist ein US-amerikanischer Informationswissenschaftler und Grafikdesigner

3.4 Lehrunterlagen - Lösungen - interaktive Elemente

Die Zielgruppe der Lehr- und Lernunterlagen sind Schüler, die bereits ein Grundwissen in Java mitbringen. Dieses Grundwissen ist jedoch viel zu gering, um bei Softwaredesign und objektorientierter Programmierung mitreden zu können. Diese Dinge sind jedoch für die Berufe, welche die Absolventen später ergreifen werden wesentlich wichtiger, als perfekt Codieren zu können. Deswegen lege ich in diesen Unterlagen besonderen Wert auf die Analyse, das Design und die Herangehensweise an eine Aufgabe. Nach einer Einführung in die objektorientierten Techniken werde ich an Hand von Praxisbeispielen diesen Weg konsequent verfolgen. Der Zeitrahmen für das Durcharbeiten dieser Lehr- und Lernunterlagen beträgt ca. 60 Stunden, wobei es natürlich immer von der Eigeninitiative des Betreffenden abhängt, mehr wissen zu wollen und die angegebenen Quellen zu studieren, oder sich selbst diesbezüglich umzusehen. Die Unterlagen bestehen aus einem Lern- und Arbeitsbuch mit praxisbezogenen Beispielen und einer übersichtlichen Gliederung. Ich werde alle Zusammenhänge kompakt und in einer verständlichen Sprache erläutern. Eine Website beinhaltet Ergänzungen, Wissensüberprüfungen und multimediale Informationen. Die gesamten Lehr- und Lernunterlagen, sowie die Webdateien stehen als Download zur Verfügung und können auch problemlos in eine der vielen Lernplattformen integriert werden.
Die didaktische und methodische Aufbereitung folgt nach einem vier Schritte Modell, das von Manz[1] eintwickelt wurde und auf folgenden Prinzipien beruht:

- *Lernen (Input), Informationen aufnehmen, Zusammenhänge erkennen, Theorie erfassen.*
- *Üben (Anwendung), Routine erwerben, Zusammenhänge verstehen, Erfahrungen sammeln.*
- *Sichern (Festigung), Gelerntes zusammenfassen, Übersicht gewinnen, Inhalte wiederholen.*
- *Wissen (Kontrolle), Wissen testen, Kompetenz überprüfen, Können beweisen*

1 www.wissenistmanz.at

3.4.1 Vorwort zu den Lehrunterlagen

Es gibt viele Möglichkeiten sich Dinge anzueignen und es gilt als hilfreich, wenn der Stoff interessant aufbereitet und wenn die entsprechende Eigenmotivation vorhanden ist. Diesem Ziel versuchen die Lehrunterlagen zu entsprechen, indem die einzelnen Lektionen zum Durcharbeiten und zum näheren Hinsehen verführen sollen.

Der Aufbau der einzelnen Lektionen erfolgt zu Beginn mit einer Einführung und nötigenfalls mit Erklärungen für den jeweiligen Teil des Stoffgebietes, es werden unter anderem auch die Ziele der Teilbereiche festgelegt. Danach folgt ein Hinweis auf das jeweilige multimedial aufgebaute Basiswissen. Die entsprechenden Links befinden sich in einer html[1] - Datei. Nach dem Durcharbeiten dieses Webdokumentes kommt es zu einer Nachbetrachtung, die meist in Form einer Wissensüberprüfung passiert und danach folgen weitere Beispiele in den Lehrunterlagen und weitere Aufgaben. Die möglichen Lösungen und Lösungswege stehen in einem eigenen Lösungsheft zur Verfügung. In diesem finden sich zusätzlich weitere Anregungen um den Stoff zu studieren.

Kennzeichnungen in den Arbeitsunterlagen:

Die Arbeitsunterlagen sind in verschiedene Bereiche unterteilt, um so einen guten Überblick zu geben, was den Inhalt anbelangt und zudem das Festlegen der Lernschwerpunkte erleichtert.

- Kennzeichnet jeweils am Anfang eines Kapitels die Lernziele.
- Allgemeine Erklärungen und Hinweise zu ergänzenden Informationen.
- Mit diesem Zeichen werden besonders wichtige Hinweise versehen.
- Damit erfolgt die Kennzeichnung von Praxistips.
- Verweist auf die Übungsaufgaben und Lösungen.

Vorraussetzungen:

Man sollte ein Java Basiswissen mitbringen und auf dem Rechner sollte die Java Netbeans IDE[2] ab Version 5 mit dem entsprechenden JDK[3] installiert sein. Alle weiteren Informationen für die Vorgangsweise sind in den einzelnen Lektionen enthalten.

Damit steht dem Wissensdrang nichts mehr im Wege. Nun etwas Zeit wird schon benötigt, aber die sollte ja durch die interessante und motivierende Aufbereitung wie im Fluge vergehen und was noch wichtiger ist, „es sollte auch etwas hängenbleiben“, damit auch eigene Ideen in die große Schatztruhe der Softwareentwicklung eingebracht werden können.

Viel Spaß beim Durcharbeiten der Unterlagen.

1 Hypertext Markup Language
2 Integrated Development Environment
3 Java Development Kit

3.4.2 Lektion 1 - Was versteht man unter OOP?


Ziel

Ziel dieser Lektion ist es, mit Fachbegriffen vertraut zu werden und sich grundsätzlich mit den Begriffen prozedural und objektorientiert auseinanderzusetzen.


Erklärung

Softwareprojekte können sehr komplex werden. Die Ansätze und Techniken der objektorientierten Programmierung sind mittlerweile sehr bewährt, um diese Komplexität in den Griff zu bekommen. Auch die Entwicklung der Programmiersprachen folgte und tut dies noch immer, dem objektorientierten Ansatz.
Dabei werden Sie die Begriffe Vererbung, Datenkapselung und Polymorphie kennenlernen. OOP geht aber noch weit über diese Grundbegriffe hinaus und es ist eigentlich kein Ende der Entwicklung auf dieser Basis abzusehen.


Übungen

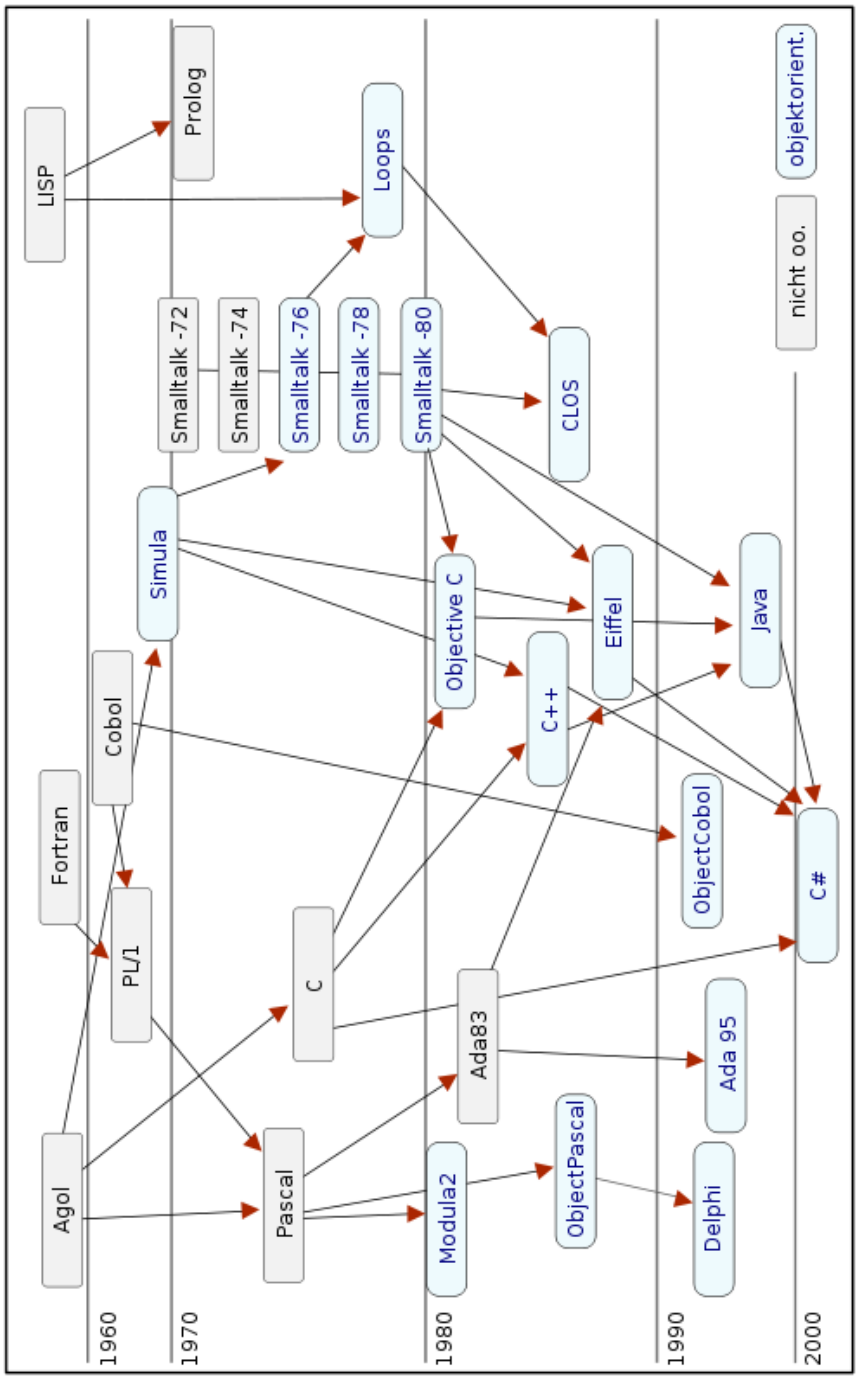
Im Verzeichnis Lektionen befindet sich eine Datei mit dem Namen links.html. Öffnen Sie diese Datei mit einem Browser Ihrer Wahl und öffnen Sie den Link Lektion 1. Auf dieser Seite sehen Sie ein Bild mit der Historie objektorientierter Programmiersprachen. Sehen Sie sich die Grafiken an und lesen Sie in Ruhe den Text durch um zu sehen, welche Vielfalt an Programmiersprachen es gibt und um einige Namen und Bezeichnungen einmal gehört zu haben.

Aufgabe 1.1: Finden Sie heraus, was mit dem Begriff „Hybride Programmiersprache“ gemeint sein könnte. (Eine mögliche Antwort finden Sie im Lösungsheft unter der jeweiligen Lektion - dies gilt auch für alle weiteren Aufgaben.

Lösung 1.1: Die hybriden Programmiersprachen arbeiten sowohl nach dem prozeduralen Prinzip, als auch nach dem objektorientierten. Als Beispiel sei dafür C++ genannt. Um die Abwärtskompatibilität mit C zu gewährleisten ist es möglich mit dieser Programmiersprache auf beide Arten zu arbeiten. Der Nachteil ist, dass sich durch die Vermischung von prozedural und objektorientiert die Fehleranfälligkeit erhöht.

Website - Info:

Abb.: 9 Historie der Programmiersprachen
Quelle: Wikipedia - 23:29, 29. Jul. 2009 Alauda - Objektorientierte Programmierung



3.4.3 Lektion 2 - Was macht gute Software aus?

Ziel

Ziel dieser Lektion ist es, die unterschiedlichen Blickwinkel zur Beantwortung dieser Frage zu betrachten. Man soll ein Gefühl dafür bekommen, worauf es im wesentlichen ankommt.

Erklärung

Um diese Frage zu beantworten muss man wissen, dass es darauf ankommt, wem diese Frage gestellt wird. Eine kleine Animation soll einige Möglichkeiten vor Augen führen. Lesen Sie sich die Antworten auf die jeweiligen Fragen aufmerksam durch. In maximal 5 Minuten sollten Sie sich ein Bild über den Typ des Fragenden und dem Beruf entsprechenden Antworten gegeben haben. Klicken Sie dazu in der Links - Website auf den Link „Lektion 2“.

Übungen

Aufgabe 2.1: Es gibt zweifelsohne unterschiedliche Sichtweisen. Trotzdem sind zentrale Kriterien zu erkennen, die eine gute Software ausmachen.
Versuchen Sie, mindestens vier wichtige Kriterien aus den gegebenen Antworten herauszufinden.

Was ist gute Software?

Die Antwort wird unterschiedlich ausfallen, je nach dem, wen wir fragen.

Wir fragen zuerst den Anwender:

Software muss korrekt sein. Sie soll das machen, was ich von ihr erwarte.
Software muss benutzerfreundlich sein und es mir einfach machen, meine Aufgaben zu erledigen.
Software muss effizient und performant sein. Ich möchte meine Arbeit möglichst schnell und effizient erledigen. Ich möchte keine Wartezeiten haben.

Wir fragen den, der die Software u. Hardware bezahlt:

Ich möchte keine neue Hardware kaufen müssen, wenn ich die Software einsetze: Software muss effizient mit Ressourcen umgehen.
Ich möchte, dass die Software möglichst günstig erstellt werden kann.
Mein Schulungsaufwand sollte gering sein.

Wir fragen den Projektmanager, der die Entwicklung der Software vorantreiben soll

Meine Auftraggeber kommen oft mit neuen Ideen. Es darf nicht aufwändig sein, diese neuen Ideen auch später noch umzusetzen: Software muss sich anpassen lassen.
Ich habe feste Zieltermine, das Management sitzt mir im Nacken. Deshalb muss ich diese Software in möglichst kurzer Zeit fertigstellen.
Das Programm soll über Jahre hinweg verwendet werden. Ich muss Änderungen auch dann noch

Abb.: 10 Was ist gute Software?

Lösung 2.1:
Korrektheit - Software soll genau das tun, was von ihr erwartet wird.

Benutzerfreundlichkeit - Software soll einfach und intuitiv zu benutzen sein.

Effizienz - Software soll mit wenigen Ressourcen auskommen und gute Antwortzeiten für Anwender haben.

Wartbarkeit - Software soll mit wenig Aufwand erweiterbar und änderbar sein.

Diese Kriterien haben einen Hauptfeind und das ist die Komplexität. Diese baut sich in der Regel bei Softwaresystemen mit zunehmender Größe auf.

3.4.4 Lektion 3 - Was war vor OOP?

Ziel

Um die Motivation für die objektorientierte Programmierung verstehen sollte man Grundkenntnisse über die Mechanismen der strukturierten Programmierung besitzen. Auch auf deren Grenzen soll in dieser Lektion hingewiesen werden.

Erklärung

Für das Grundverständnis der strukturierten Programmierung ist ein Flußdiagramm, oder auch Ablaufdiagramm genannt, sehr hilfreich. Die Grundelemente sind Eingabe, Ausgabe und Verzweigungen.
Als einfaches Beispiel möchte ich Ihnen das Aufbrühen von Kaffee mit einer Kaffeemaschine zeigen.

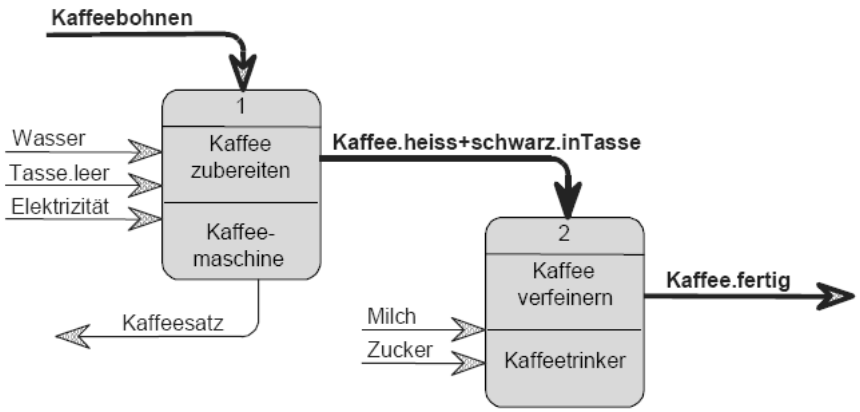


Abb.: 11 Flußdiagramm

Übungen

Aufgabe 3.1: Versuchen Sie den Ablauf des Telefonierens mit Hilfe eines Ablaufdiagramms zu beschreiben. Sehen Sie sich zuerst den Link für Lektion 3 an. Dieser enthält die Erklärungen für die Basiselemente eines Flußdiagramms.

3.4.5 Lektion 4 - Einführung in die Kapselung?

Ziel

Ziel dieser Lektion ist es, Sie mit den unterschiedlichen Blickwinkeln für die Beantwortung dieser Frage vertraut zu machen. Sie sollen ein Gefühl dafür bekommen, worauf es im wesentlichen ankommt.

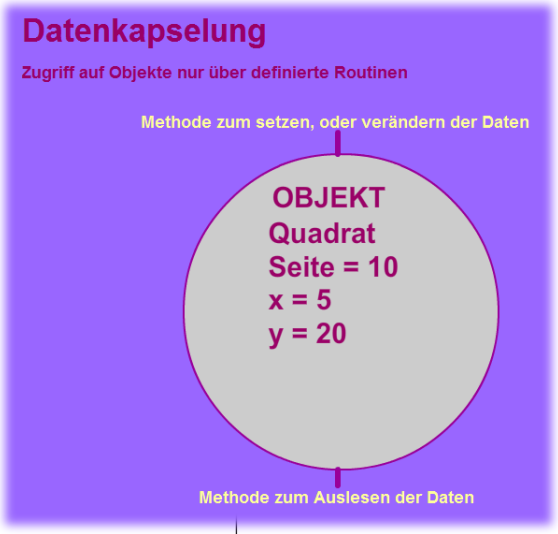
Erklärung

Um diese Frage zu beantworten müssen Sie wissen, dass es darauf ankommt, wem diese Frage gestellt wird. Eine kleine Animation soll Ihnen einige Möglichkeiten vor Augen führen. Lesen Sie sich die Antworten auf die jeweiligen Fragen aufmerksam durch. In maximal 5 Minuten sollten Sie sich ein Bild über das System Typ des Fragenden und dem Beruf entsprechenden Antworten gegeben haben. Klicken Sie dazu in der Links - Website auf den Link „Lektion 4“.

Übungen

Aufgabe 2.1: Es gibt zweifelsohne unterschiedliche Sichtweisen. Trotzdem sind zentrale Kriterien zu erkennen, die eine gute Software ausmachen.
Versuchen Sie, mindestens vier wichtige Kriterien aus den gegebenen Antworten herauszufinden.

Abb.: 11 Datenkapselung Beispiel in der Website



Lösung 4.1: Ein mögliches Beispiel wäre das Anlegen eines Kontos bei der Bank. Als gekapselte Daten und Objekte wären da einmal die Kundendaten. Einzahlung und Auszahlung funktionieren über Methoden und der jeweilige Kontostand ergibt sich daraus.

3.4.6 Lektion 5 - Einführung in die Polymorphie?

Ziel

Mit dem Begriff Polymorphie beispielhaft umgehen können und den Nutzen verstehen.

Erklärung

Damit Sie sich unter Polymorphie etwas vorstellen können, will ich diese an einem Beispiel aus dem Alltag erläutern, nämlich das Auswechseln einer Glühbirne.
Als Animation öffnen Sie dazu den Link Lektion 5.

Abb.: 12 Polymorphie (Website)

In objektorientierten Systemen möchte man diese Möglichkeiten auch nutzen. Ist ein Verfahren nicht mehr schnell genug, so wird einfach ein neues in die alte Fassung gedreht. Das Problem im Softwareentwurf liegt darin, zu erkennen, wo man Fassungen vorsehen sollte und zu bestimmen, welche Objekte in die Fassung geschraubt werden können.

Übungen

Aufgabe 5.1: Versuchen Sie ein ähnliches Beispiel aus der realen Welt für Polymorphie zu finden.

3.4.7 Lektion 6 - Einführung in die Vererbung?

Ziel

In der OOP tritt die Vererbung in zwei unterschiedlichen Formen auf. In dieser Lektion sollen Sie Beispiele aus dem Alltag an diese beiden Arten der Vererbung heranführen.

Erklärung

Denken Sie noch einmal an die Glühbirnen aus der Lektion5. Prinzipiell können sich die Glühbirnen im Energieverbrauch und in der Leuchtkraft unterscheiden, aber es gibt eine Festlegung, die es ermöglicht, die Glühbirne in eine genormte Fassung zu drehen und mit einer dafür vorgesehenen Spannung zu betreiben. In der OOP kann man diese Spezifikationen vererben. Ein weiteres Beispiel aus der realen Welt wären Elektrogeräte, die alle an derselben Steckdose angeschlossen werden können. Diese Art der Vererbung hängt sehr stark mit der Polymorphie zusammen. Die zweite Art basiert auf hierarchischen Regeln. Öffnen Sie für die näheren Erklärungen den Link zu Lektion 6.

Hierarchische Regeln

Die Regelungen sind hierarchisch organisiert, wobei die weiter oben liegenden Regeln jeweils weiter unten liegende überschreiben. Dies sollte man sich in etwa wie in dem Stapel Bücher aus der folgenden Abbildung vorstellen: Man prüft zunächst im obersten Buch im Stapel, ob eine Regelung für einen Bereich vorliegt. Findet man sie dort nicht, geht man zum nächsten Buch im Stapel weiter. Und so fort, bis eine Regelung gefunden wird.

Durch dieses Vorgehen wird viel bedrucktes Papier (und damit auch Redundanz) vermieden, denn wenn alle Kommunen die bereits existierenden Regeln erneut selbst auflegen müssten, würde extrem viel Papier unnütz bedruckt.

Abb.: 13 Vererbung (Website)

Übungen

Aufgabe 6.1: Überlegen Sie sich wie eine Vererbungshierarchie für Fahrräder aussehen könnte? Versuchen Sie, die Abhängigkeiten an Hand einer Skizze darzustellen.

3.4.8 Lektion 7 - Beispiel Kapselung?

Ziel

Ein Beispiel kennenlernen, in dem der Zugriff auf die Daten über sogenannte „Setter und Getter“ Methoden durchgeführt wird.

Erklärung

Sinn und Zweck der Datenkapselung ist es, auf die Variablen nur mit speziellen Methoden zuzugreifen. Mit den Setter und Getter Methoden werden einzelne Eigenschaften eines Objektes abgefragt oder geändert.

Das folgende Beispiel lässt sich auch als Video über den Weblink Lektion 7 demonstrieren.

Übungen

Codieren Sie das folgende Beispiel, überlegen Sie sich, was die einzelnen Codezeilen bedeuten und achten Sie vor allem auf die Kommentare.

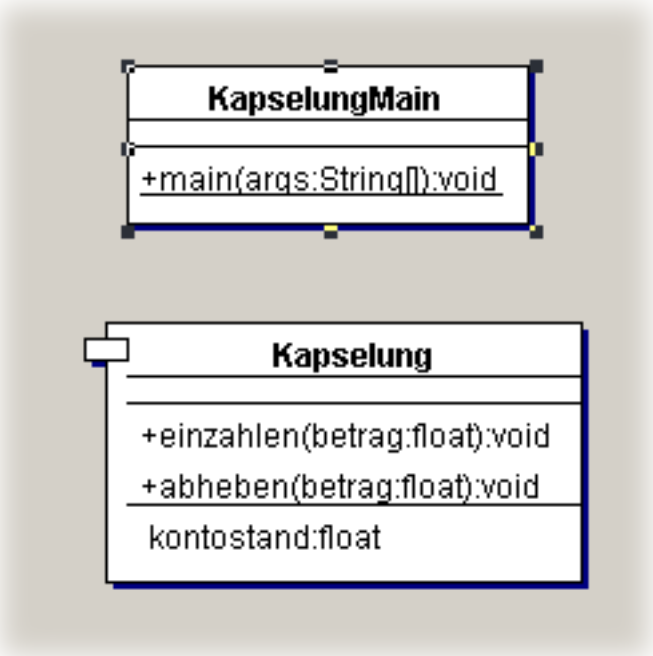


Abb.: 14
UML[1] Klassendiagramm für das Beispiel der Kapselung.

1 Unified Modeling Language

```java
public class KapselungMain {


    public static void main(String[] args) {
        //Instanz der Klasse Kapselung erstellen
        Kapselung kapselung = new Kapselung();
        //Derzeitigen Kontostand ausgeben
        System.out.println("Kontostand: ");
        System.out.println(kapselung.getKontostand());
        //Betrag einzahlen
        kapselung.einzahlen(200.34f);
        System.out.println("Kontostand: ");
        //Derzeitigen Kontostand ausgeben
        System.out.println(kapselung.getKontostand());
        //Geld vom Konto abheben
        kapselung.abheben(120.50f);
        //Derzeitigen Kontostand ausgeben
        System.out.println(kapselung.getKontostand());
    }
```

```java
public class Kapselung {

    //Instanzvariable anlegen
    float Kontostand = 0;

    //Geld einzahlen
    public void einzahlen(float betrag){
        Kontostand = Kontostand + betrag;
    }
    //Geld abheben
    public void abheben(float betrag){
        Kontostand = Kontostand - betrag;
    }
    //aktuellen Kontostand abrufen
    public float getKontostand(){
        return Kontostand;
    }
```

Für die Implementierung empfiehlt es sich, zuerst die Main Klasse anzulegen und sich dann schrittweise vorzuarbeiten. Siehe Video - Lektion 7.

Übungen

Aufgabe 7.1: Analysieren Sie die Angaben im UML Diagramm. Verwenden Sie dazu den Quellcode der beiden Klassen.

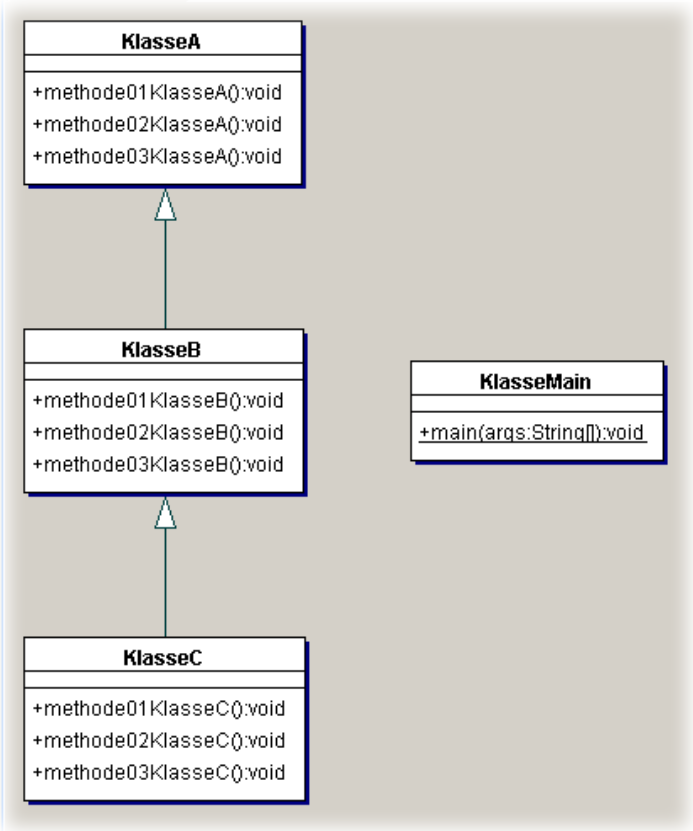
3.4.9 Lektion 8 - Beispiel Vererbung?

Ziel

Im folgenden Beispiel werden die grundlegenden Mechanismen der Vererbung gezeigt. Das Beispiel ist allgemein gehalten und soll dem allgemeinen Verständnis der Techniken dienen.

Erklärung

Das folgende Klassendiagramm zeigt den Aufbau des Beispiels. Der zeigt immer in Richtung zu der Klasse, von der geerbt wird. Mit dem Schlüsselwort extends wird die Vererbung festgelegt. Damit kann man dann auf alle Klassenmethoden der geerbten Klasse zugreifen.



Übungen

Codieren Sie die folgenden Klassen und führen Sie die Klasse mit der main - Methode aus. Vergleichen Sie dabei immer das UML Klassendiagramm und den Quellcode.

```java
package vererbungBasis;

/**
 * Klasse zur Demonstrationszwecken für die Vererbung
 * @author Stuhlpfarrer Ehrenfried
 */
public class KlasseA {

    public void methode01KlasseA(){
        System.out.println("Aufruf: Methode01KlasseA");
    }

    public void methode02KlasseA(){
        System.out.println("Aufruf: Methode02KlasseA");
    }

    public void methode03KlasseA(){
        System.out.println("Aufruf: Methode03KlasseA");
    }

}
```

```java
package vererbungBasis;

/**
 *
 * @author Stuhlpfarrer Ehrenfried
 */
public class KlasseB extends KlasseA {

    public void methode01KlasseB(){
        System.out.println("Aufruf: Methode01KlasseB");
    }

    public void methode02KlasseB(){
        System.out.println("Aufruf: Methode02KlasseB");
    }

    public void methode03KlasseB(){
        System.out.println("Aufruf: Methode03KlasseB");
    }

}
```

Die Reihenfolge der Klassenimplementierung sehen Sie in einer Flash Animation zu Lektion 8.

```
package vererbungBasis;
/**
 *
 * @author Stuhlpfarrer Ehrenfried
 */
public class KlasseC extends KlasseB {

    public void methode01KlasseC(){
        System.out.println("Aufruf: Methode01KlasseC");
    }

    public void methode02KlasseC(){
        System.out.println("Aufruf: Methode02KlasseC");
    }

    public void methode03KlasseC(){
        System.out.println("Aufruf: Methode03KlasseC");
    }

}
```

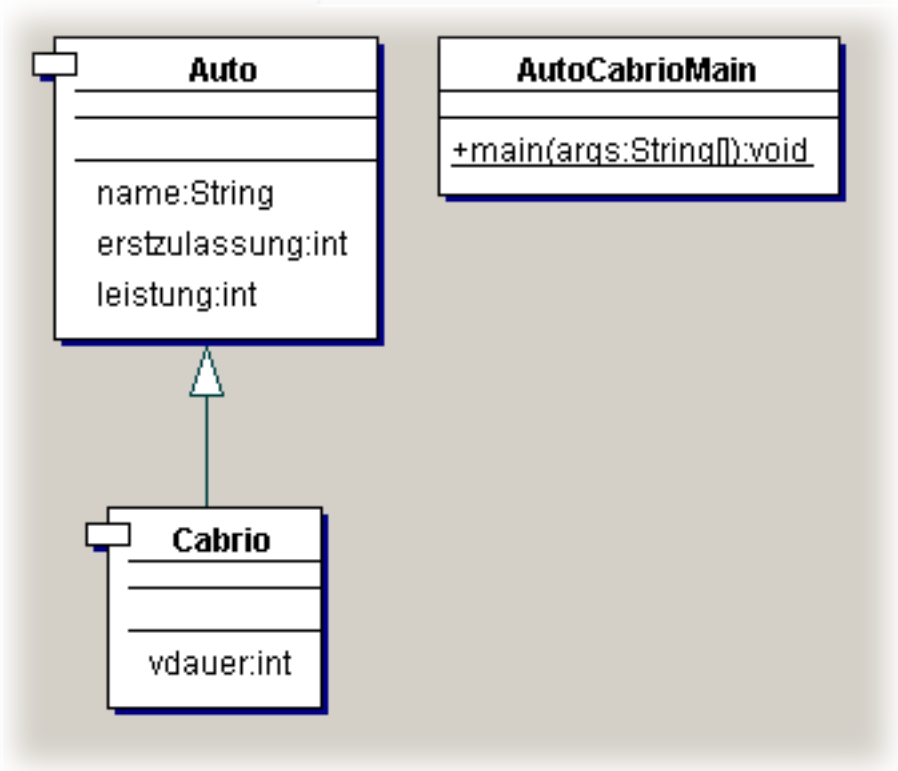
```
package vererbungBasis;
/**
 *
 * @author Stuhlpfarrer Ehrenfried
 */
public class KlasseMain {

    public static void main(String[] args){
        System.out.println("KlasseMain");
        KlasseC klassec = new KlasseC();
        klassec.methode01KlasseA();
        klassec.methode01KlasseB();
        klassec.methode01KlasseC();
        klassec.methode02KlasseA();
        klassec.methode02KlasseB();
        klassec.methode02KlasseC();
        klassec.methode03KlasseA();
        klassec.methode03KlasseB();
        klassec.methode03KlasseC();
    }

}
```

Übungen

Aufgabe 8.1: Entwerfen Sie eine Klasse Cabrio, die sich von der Klasse Auto nur dadurch unterscheidet, dass Zeit für das Öffnen des Verdecks benötigt wird. (Variablen: name, erstzulassung, leistung, vdauer[1]

1 Verdeckköffnungsdauer

Lösung Aufgabe 8.1:



Übungen

Aufgabe 8.2: Welche Auswirkungen hat es, wenn die Klassenvariablen nicht als private deklariert werden?

Lösung 8.2: Man könnte direkt auf die Klassenvariablen zugreifen. Davor ist jedoch aus Sicherheitsgründen abzuraten, denn das Prinzip der Datenkapselung wird dabei verletzt und die Set und Get Methoden werden damit ausgehebelt.

3.4.10 Lektion 9 - Lohn u. Gehaltsabrechnung - Analyse - Design - Implementierung (Schwerpunkte: Kalkulation u. Kapselung)

Ziel

Implementierung einer Anwendung mit Benutzeroberfläche (GUI). Datenkapselung und Kalkulation sollen im Vordergrund stehen. Als Annahme gilt, dass die Software als Teil einer kommerziellen Software eingesetzt werden soll, aber nach den Grundprinzipien der Objektorientierten Entwicklung auch eigenständig lauffähig sein muss.
Es geht um eine Lohn- und Gehaltsabrechnung in ihrer einfachsten Form. Die Abrechnung von Gehältern und Löhnen ist im Zeitalter der EDV sehr schnell erledigt. Darum wollen wir uns einmal den Hintergrund dieser „so schnell“ erledigten Arbeit ansehen. Dieses Beispiel gilt natürlich auch für alle ähnlich gestalteten Aufgaben

Erklärung

Analyse

Die Berechnung der Nettobezüge, des Weihnachtsgeldes, der Urlaubsbeihilfe, der Abfertigungen etc. ist durch Gesetze bzw. Kollektivverträge genau geregelt. So besteht in Österreich Sozialversicherungspflicht für jeden Arbeitnehmer, das heißt es wird vom Bruttobezug ein bestimmter Prozentsatz für Kranken-, Arbeitslosen-, Unfall- und Pensionsversicherung abgezogen. Für die Abrechnung kann das folgende Schema herangezogen werden.

Abrechnung: Angestellter		*Sozialversicherung: 18,07%*			
Brutto:					
	Bruttobezug				€ 1.500,00
Sozialversicherung:					
	SV Gehalt	€ 1.500,00	x	18,07	€ 271,05 -
Lohnsteuer:					
	LSt-Gehalt	brutto		€ 1.500,00	
		-SV		-271,05	
		-FB			
		- Gewerksch.			
		LSt-Bemessungsgrundlage		€ 1.228,95	-€ 116,77
Netto:					
	- Gewerlschaftsbeitrag				€ 0,00 -
	- Akontozahlung				€ 0,00 -
	- Betriebsratsumlage				€ 0,00 -
	Auszahlung				**€ 1.112,18**

Abb.: 15, Quelle: (17) PV08, Seite 02

Woher kommen die 18,07%? Dieser Wert wird bis zu einem Bruttogehalt von 3.930.- € berechnet. Dies ist der höchste Wert für die Sozialversicherungsbeitragssätze. Darüber hinaus erfolgt kein Abzug mehr.

Design

Das GUI könnte so ähnlich aussehen wie das Schema aus der Analyse. Für den Sozialversicherungsbeitrag wären Konfigurationsmöglichkeiten interessant, denn die Werte könnten sich, je nach Regierung ja auch einmal verändern. Stammdaten des Angestellten sollten auch integriert werden. In dieser Basisform genügt einmal eine fiktive Angestelltennummer. Aus Wartungs- und Änderungsgründen sollte die Logik in eine eigene Klasse gepackt werden. Der Zugriff auf die Variablen wird sinnvollerweise über set und get Methoden vollzogen.
Als Basis für das Design werde ich mich an ein bewährtes Grundprinzip halten, nämlich die Trennung der Benutzerschnittstelle und der Logik. Dieses „Design Pattern“ wird auch „MVC[1] Pattern“ genannt. Als Model dient üblicherweise ein Datenbank - Management System[2]. In der folgenden Skizze stelle ich einen möglichen Entwurf für die Lösung der Aufgabe vor. Die Lösung der Aufgabe kann auch als eigenständiges Objekt in einem größeren aufgefasst werden. Als persistentes Datenformat ist es sinnvoll XML[3] zu wählen, da hierfür mittlerweile für fast alle Programme Schnittstellen existieren.

1 Model View Controller
2 DBMS
3 Extensible Markup Language

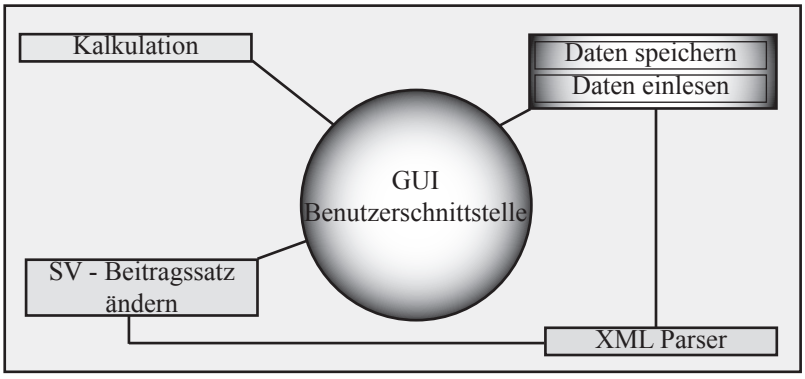
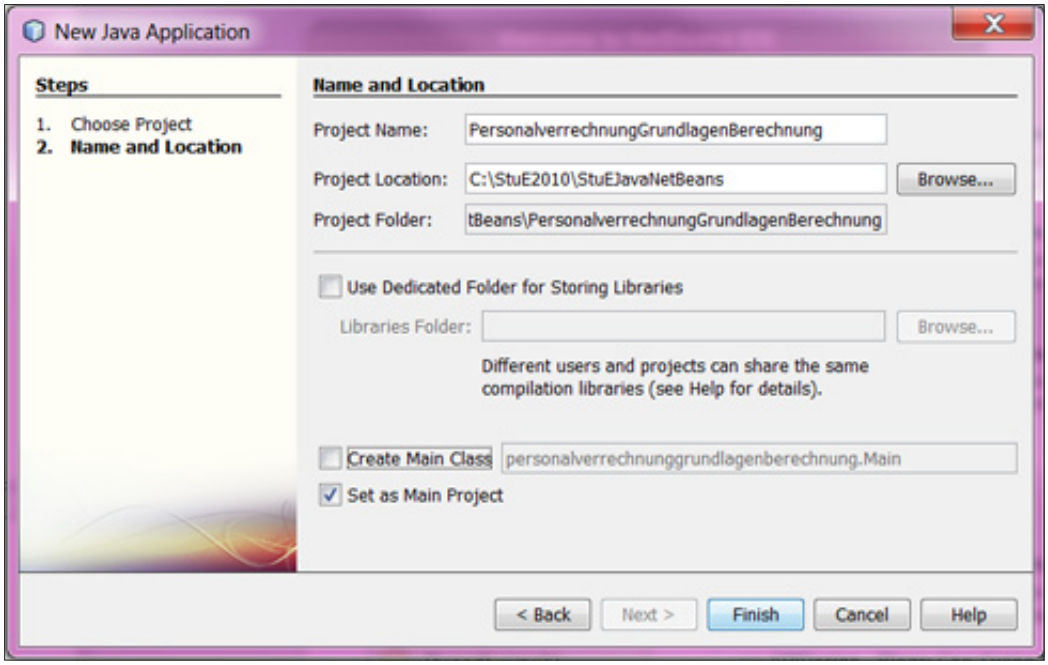


Abb.: 16 Funktionalitätsskizze

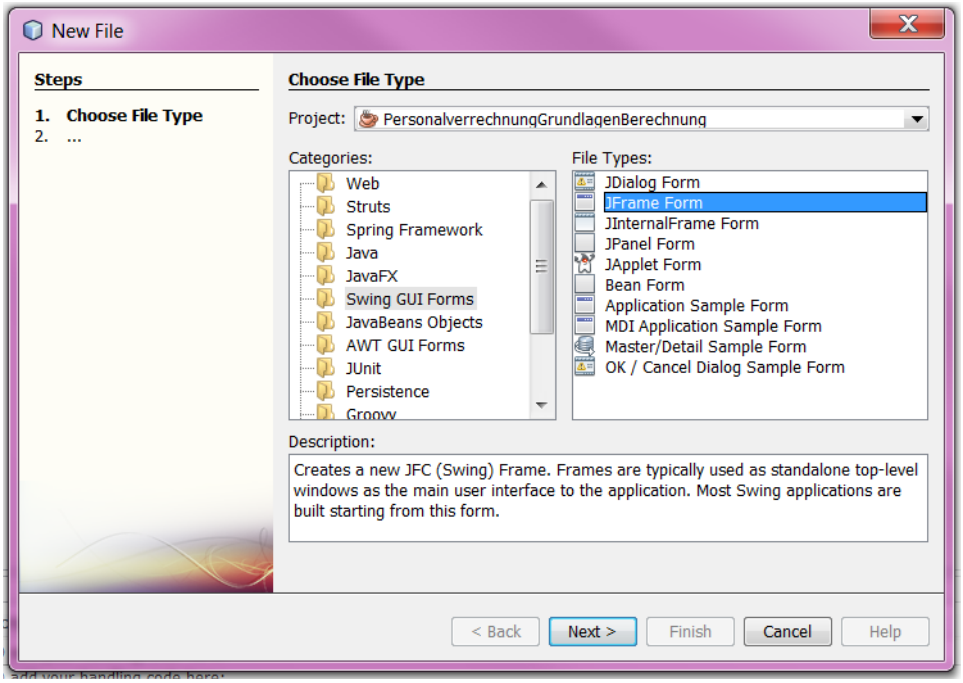
Umsetzung

Netbeans IDE 6.8 Beta - Entwicklungsumgebung

- Anlegen eines neuen Projektes. (Personalverrechnung-GrundlagenBerechnung)



- Maus - Rechtsklick im Projektfenster u. „New file“

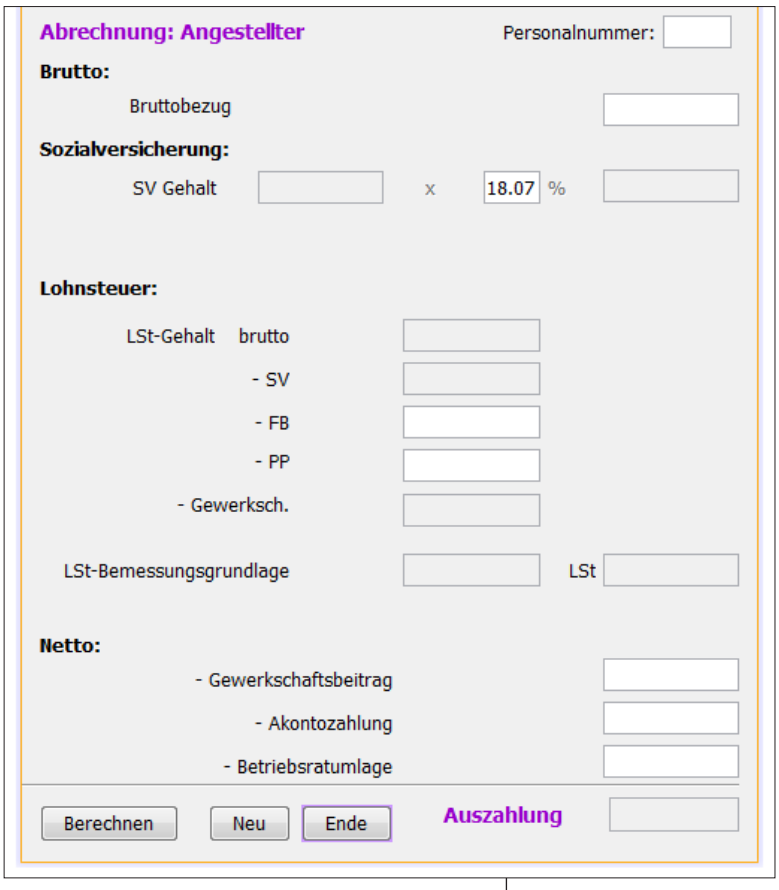


- Klassenname „Gui“

Implementierung der Benutzeroberfläche

(dieses GUI dient als Basis - Erweiterungen werden noch eingeführt)
Variablennamen:

Personalnummer - *jTextPersonalnummer*
Bruttobezug - *jTextFieldBruttobezug*
Sozialversicherung - *jTextFieldSVProzentsatz*
Kopie der Sozialversicherung - *jTextFieldSVCopyBruttogehalt*
Berechneter SV Betrag - *jTextFieldSozialversicherungBerechnet*
LSt-Gehalt Brutto - *jTextFieldSVCopyBruttogehalt1*
SV - *jTextFieldSV*
Gewerkschaftsbeitrag - *jTextFieldGewerksch*
LSt-Bemessungsgrundlage - *jTextFieldLohnsteuerBemessungs-grundlage*
Lohnsteuer berechnet - *jTextFieldLSt*
Kopie des Gewerkschaftsbeitrages - *jTextFieldGewerksch1*
Akontozahlung - *jTextFieldAkontozahlung*
Betriebsratsumlage - *jTextFieldBetriebsratsumlage*
Auszahlung - *jTextFieldAuszahlung*
Berechnen - *jButtonBerechnen*
Neu - *jButtonNeu*
Auszahlung - *jTextFieldAuszahlung*
Ende - *jButtonEnde*



Ergänzende Informationen zur Lohnsteuer

Eine der wichtigsten Steuern in jedem Steuersystem eines Staates ist die Einkommensteuer bzw. die Lohnsteuer. Diese Steuer ist abhängig von der Höhe des Einkommens. Wenn bestimmte Einkünfte gegeben sind, ist jeder Bürger eines Staates, unabhängig von Alter, Beruf und Geschlecht, einkommensteuerpflichtig.

Mithilfe der Effektiv-Tarif-Tabelle kann von jedem Betrag die Lohnsteuer berechnet werden. Für dieses Beispiel genügt es, ohne den Abzug mit Alleinverdiener- oder Erzieherabsetzbetrag zu arbeiten.

Effektiv - Tariftabelle Arbeitnehmer

von	bis	keine Steuer bis 924, 34	Steuersatz	Abzug
0,00	900,42	Klasse 1	0,00000%	9,167
900, 43	2099,33	Klasse 2	38,33333%	354,328
2099,34	4266,00	Klasse 3	43,59615%	464,812
4266,01	-	Klasse 4	50,00000%	738,000

Abb.: 17 Effektiftariftabelle - Quelle PV08

Ergänzende Informationen zu den Sozialversicherungs - Beitragssätzen für Angestellte

Höchstbeitragsgrundlage: € 3.930.-
Monatliche Bruttoeinkünfte über diesem Betrag sind sozialversicherungsfrei.

Geringfügigkeitsgrenze: € 349,01.- (monatlich)
Bei Einkünften bis zu diesem Betrag fällt für den Dienstnehmer keine Sozialversicherung an.

Die Klasse Basisdaten

Die Klasse soll als Zustandsvariablen enthalten: Bruttobezug, Freibetrag, Pendlerpauschale, Gewerkschaftsbeitrag, Akontozahlungen, Betriebsratsumlage.

Setter und Gettermethoden sollen die Zugriffsmethoden sein. Um die Klasse zu testen, wird sie auch als eigenständige Konsolenanwendung implementiert. Für die weitere Verwendung aus anderen Klassen heraus ist der Zugriff nur mehr über die entsprechenden Methoden möglich.

UML konformes Klassendiagramm mit den Membervariablen und einer MainMethode.

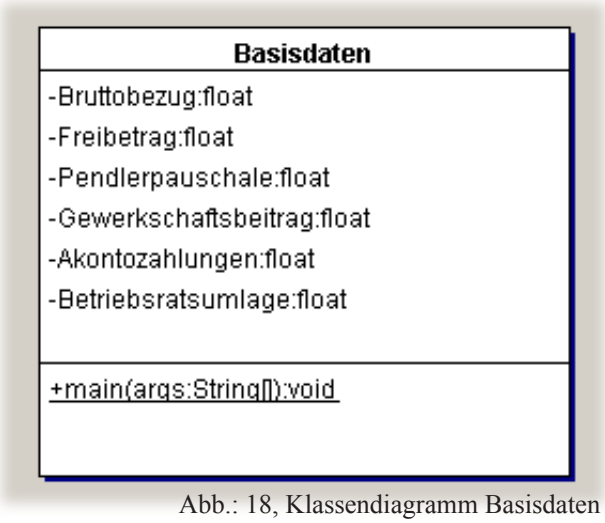


Abb.: 18, Klassendiagramm Basisdaten

Die Implementierung der Klasse Basisdaten

1 Neue Klasse anlegen: Basisdaten

2 Variablendeklarationen

```
private float Bruttobezug, Freibetrag, Pendlerpauschale,
            Gewerkschaftsbeitrag, Akontozahlungen, Be-
triebsratsumlage;
```

3 Implementierung der set Methoden

```
public void setBruttobezug(float bruttobezug){
        this.Bruttobezug = bruttobezug;
    }
```

Übergabeparameter

```
public void setFreibetrag(float freibetrag){
        this.Freibetrag = freibetrag;
    }

public void setPendlerpauschale(float pendlerpauschale){
        this.Pendlerpauschale = pendlerpauschale;
    }

public void setGewerkschaftsbeitrag(float gewerkschaftsbeitrag){
        this.Gewerkschaftsbeitrag = gewerkschaftsbeitrag;
    }

public void setAkontozahlungen(float akontozahlungen){
        this.Akontozahlungen = akontozahlungen;
    }

public void setBetriebsratsumlage(float betriebsratsumlage){
        this.Betriebsratsumlage = betriebsratsumlage;
    }
```

Implementierung der get Methoden

```
public float getBruttobezug(){
   return Bruttobezug;
}

public float getFreibetrag(){
   return Freibetrag;
}

  public float getPendlerpauschale(){
   return Pendlerpauschale;
}

   public float getGewerkschaftsbeitrag(){
    return Gewerkschaftsbeitrag;
}

    public float getAkontozahlungen(){
    return Akontozahlungen;
}

     public float getBetriebsratsumlage(){
    return Betriebsratsumlage;
}
```

Die Methode für den Sozialversicherungsbeitrag

In der ersten Phase wird die Methode für den Sozialversicherungsbeitrag nicht generisch aufgebaut. Die erfolgt aus Übersichtlichkeitsgründen erst in einer späteren Phase.

```
// Sozialversicherung - Beitragssätze
          // Höchstbeitragsgrundlage 3.930.- (18.07%)
          // unter 349.01.- keine SV
          // die Methode ist noch nicht generisch, da Prozentsatz
          // Hoechstbeitrag und mindestbeitrag noch nicht als
          // Parameter uebergeben werden
          // In Vorbereitung: public float svBetrag(float abzugProzent,
          // float hoechstbeitrag, float mindestbeitrag)
          public float svBetrag(){
              // Prozentsatz für den Abzug vom Bruttogehalt
              float abzugProzent = 18.07f;
              // Hoechstbeitragsgrundlage
              float hoechstbeitrag = 3.930f;
              // Mindestbeitragsgrundlage
              float mindestbeitrag = 349.01f;
              // Variable fuer den Sozialversicherungsbeitrag
              float svAbzug = 0.00f;
              // Bruttobezug
              float brutto = this.getBruttobezug();
              // die Bedingungen
              if ((brutto <= hoechstbeitrag) || (brutto >= mindestbeitrag)
){
                  brutto = (brutto / 100) * abzugProzent;
                  svAbzug = brutto;
              }
              else svAbzug = 0;
              // Abzuziehender Sozialversicherungsbeitrag
              return svAbzug;
       }
```

Set und Get Methoden für die Personalnummer implementieren

Membervariable deklarieren; üblicherweise am Anfang der Klasse;

```
private String Personalnummer;

public void setPersonalnummer(String personalnummer){
          this.Personalnummer = personalnummer;
     }

public String getPersonalnummer(){
               return Personalnummer;
          }
```

Die Berechnung der Lohnsteuer

Diese Implementierung dient als Basis. Der generische Teil mittels variabler Tabelle kann noch nachgezogen werden.

```
// Berechnung der Lohnsteuer ------------------------------------
    // zuerst die Sozialversicherung abziehen
    // keine Steuer bis 924, 34
    // 0 - 900.42 --> 0% mit Abzug = 9.167
    // 900.43 - 2099.33 --> 38.33333% mit Abzug = 354.328
    // 2099.34 - 4266.00 --> 43.59615% mit Abzug = 464.812
    // 4266.01 - ~ --> 50.00% mit Abzug = 738.000
    public float LSt(){
        // Variable für die berechnete Lohnsteuer
        float lohnsteuer = 0.00f;
        // temporaere Variable für die Sozialversicherung
        float sv = this.svBetrag();
        // Bruttobezug minus Sozialversicherung
        float brutto = this.getBruttobezug() - sv;
        // die entsprechenden Bedingungen
        if(brutto <= 900.42){
            lohnsteuer = 9.167f;
        }
        if((brutto > 900.43) && (brutto <= 2099.33)){
            lohnsteuer = ((brutto / 100) *  38.33333f) -
            354.328f;
        }
        if((brutto > 2099.34) && (brutto <= 4266.00)){
            lohnsteuer = ((brutto / 100) *  43.59615f) -
            464.812f;
        }
        if(brutto > 4266.01){
            lohnsteuer = ((brutto / 100) *  50.00f) - 738.000f;
        }


        return lohnsteuer;
    }
```

Die Klasse TestBasisdaten

Diese Klasse dient zum Testen der Controller - Komponente aus dem MVC[1] - Pattern.

1 Model View Controller

```
/**
 * Testklasse um auf Konsolenebene die Klassen zu testen
 * @author Stuhlpfarrer
 */
public class TestBasisdaten {

    public static void main(String[] args) {
        Basisdaten bd = new Basisdaten();
        bd.setAkontozahlungen(450.50f);
        bd.setBetriebsratsumlage(34.25f);
        bd.setBruttobezug(1500.00f);
        //bd.setBruttobezug(1000.00f);
        bd.setFreibetrag(12.20f);
        bd.setGewerkschaftsbeitrag(20.00f);
        bd.setPendlerpauschale(30.90f);
        System.out.println(„Akontozahlungn: „ +
bd.getAkontozahlungen());
        System.out.println(„Betriebsratsumlage: „ +
bd.getBetriebsratsumlage());
        System.out.println(„Bruttobezug: „ + bd.getBruttobezug());
        System.out.println(„Freibetrag: „ + bd.getFreibetrag());
        System.out.println(„Gewerlschaftsbeitrag: „
                + bd.getGewerkschaftsbeitrag());
        System.out.println(„Pendlerpauschale: „ +
bd.getPendlerpauschale());

        System.out.println(„SV Betrag: „ + bd.svBetrag());

        System.out.println(bd.LSt());
    }
}
```

Die Kalkulation

Die Berechnung erfolgt ebenfalls als eigene Methode in der Klasse Basisdaten. Man könnte aus Gründen der Übersichtlichkeit dafür eine eigene Klasse einführen, dies ist jedoch auf Grund des geringen Umfanges hier nicht vorgesehen.

```
// Berechnung des Auszahlungsbetrages
   public float auszahlung() {
       // temp. Variable fuer den Auszahlungsbetrag
       float aBetrag = 0.00f;
       // Kalkulation
       aBetrag = this.getBruttobezug()
               - this.svBetrag()
               - this.getFreibetrag()
               - this.getPendlerpauschale()
               - this.getGewerkschaftsbeitrag()
               - this.LSt()
               - this.getGewerkschaftsbeitrag()
               - this.getAkontozahlungen()
               - this.getBetriebsratsumlage();
       return aBetrag;
   }
```

Ergänzung der Klasse TestBasisdaten

Die Klasse muss um den Aufruf der Auszahlungsmethode ergänzt werden.

```
System.out.println(bd.auszahlung());
```

Die Funktionalität der „Ende" Buttons

Der Ende - Button muss mit der Funktionalität zur Beendigung des Programmes ausgestattet werden.
Vorgang: Markieren des „Ende" Buttons - Kontextmenü mit rechter Maustaste aufrufen >> events >> Action >> actionPerformed
Danach erfolgt die automatische Navigation zur Ereignismethode des „Ende" Buttons. Auf die Abfrage, ob man das wirklich will und möglicherweise, ob man ganz sicher ist, werde ich in diesem Fall verzichten. Der zu implementierende Code könnte folgendermassen aussehen. (Nicht die Kommentare und die Prüfung der Funktionalität vergessen!)

```
// Beim Klicken das Programm verlassen
    private void jButtonEndeActionPerformed(java.awt.
event.ActionEvent evt) {
        System.exit(0);
    }
```

Die Funktionalität der „Neu" Buttons

In dieser Prototyp - Version soll der „Neu" Button alle Eingabefelder zurücksetzen und den Cursor im ersten Eingabefeld aktivieren.

```
// Beim Klicken den Inhalt aller Eingabefelder löschen
   private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
       // löschen der Inhaltes der editierbaren Felder
       this.jTextPersonalnummer.setText(„");
       this.jTextFieldBruttobezug.setText(„");
       this.jTextFieldFB.setText(„");
       this.jTextFieldPP.setText(„");
       // loeschen des Inhaltes der nicht editierberen Felder
       this.jTextFieldSVCopyBruttogehalt1.setText(„");
       this.jTextFieldSozialversicherungBerechnet.setText(„");
       this.jTextFieldSVCopyBruttogehalt1.setText(„");
       this.jTextFieldSV.setText(„");
       this.jTextFieldGewerksch.setText(„");
       this.jTextFieldLohnsteuerBemessungsgrundlage.setText(„");
       this.jTextFieldLSt.setText(„");
       this.jTextFieldAuszahlungsbetrag.setText(„");
       this.jTextFieldGewerksch1.setText(„");
       this.jTextFieldAkonto.setText(„");
       this.jTextFieldBetriebsratsumlage.setText(„");
   }
```

Die Berechnung - Kalkulation

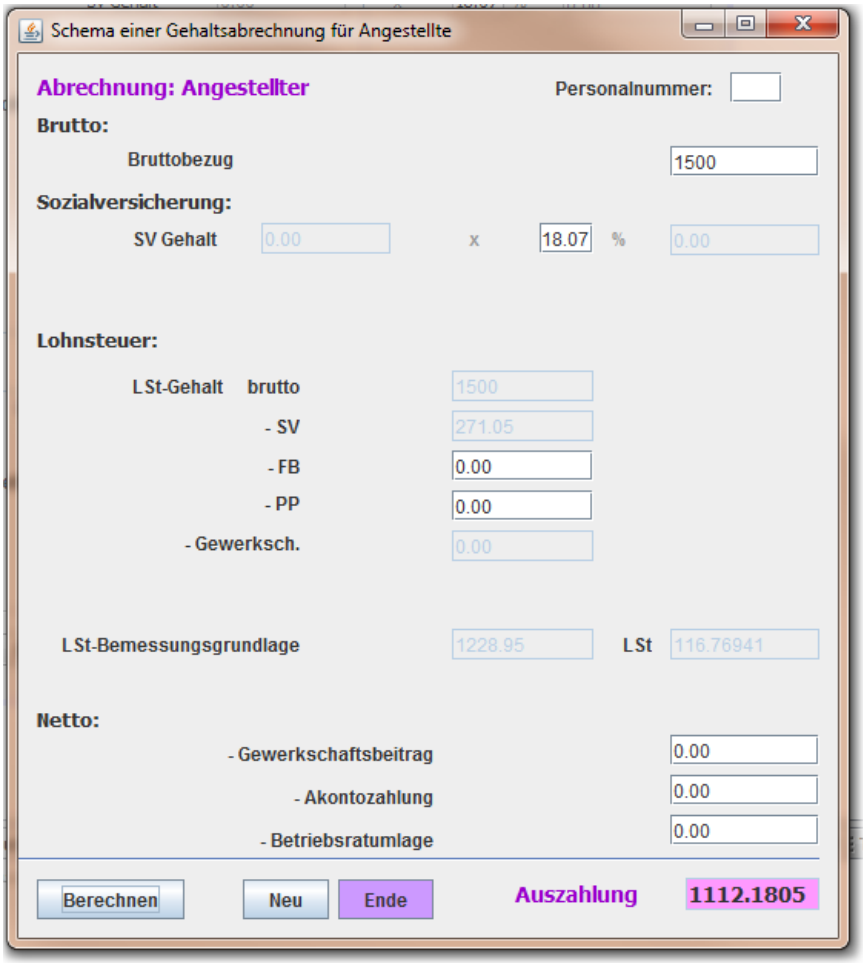
Der Button „Berechnen" ruft alle erforderlichen Funktionen aus der Basisklasse auf und trägt die entsprechenden Werte in die dafür vorgesehenen Textobjekte ein. Da Java eine typsichere Sprache ist, muss von String in float und umgekehrt ein „Casting[1]" durchgeführt werden.

```
// Funktionalitaet des Berechnen - Buttons
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
```

1 Typumwandlung

```
// Funktionalitaet des Berechnen - Buttons
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        Basisdaten basisdaten = new Basisdaten();
        // Bruttobetrag setzen
        String bruttoStr = this.jTextFieldBruttobezug.getText();
        float bruttoFl = Float.valueOf(bruttoStr);
        basisdaten.setBruttobezug(bruttoFl);
        // Freibetrag setzten
        String freibetragStr = this.jTextFieldFB.getText();
        float freibetragFl = Float.valueOf(freibetragStr);
        basisdaten.setFreibetrag(freibetragFl);
        // Pendlerpauschale setzten
        String ppStr = this.jTextFieldPP.getText();
        float ppFl = Float.valueOf(ppStr);
        basisdaten.setFreibetrag(ppFl);
        // Gewerkschaftsbeitrag setzten
        String gwStr = this.jTextFieldGewerksch1.getText();
        float gwFl = Float.valueOf(gwStr);
        basisdaten.setGewerkschaftsbeitrag(gwFl);
        // Akontozahlungen setzten
        String akStr = this.jTextFieldAkonto.getText();
        float akFl = Float.valueOf(akStr);
        basisdaten.setFreibetrag(akFl);
        // Betriebsratsumlage setzten
        String bruStr = this.jTextFieldBetriebsratsumlage.getText();
        float bruFl = Float.valueOf(bruStr);
        basisdaten.setFreibetrag(bruFl);
        // Kalkulation
        float ausFl = basisdaten.auszahlung();
        String ausStr = String.valueOf(ausFl);
        this.jTextFieldAuszahlungsbetrag.setText(ausStr);

        // Ausfüllen der Kontrollfelder
        this.jTextFieldSVCopyBruttogehalt1.setText(this.jTextFieldBruttobe-
zug.getText());
        Float svBerechnetFl = basisdaten.svBetrag();
        String svBerechnetStr = String.valueOf(svBerechnetFl);
        this.jTextFieldSV.setText(svBerechnetStr);
        this.jTextFieldGewerksch.setText(this.jTextFieldGewerksch1.get-
Text());
        Float lstFl = basisdaten.LSt();
        String lstStr = String.valueOf(lstFl);
        this.jTextFieldLSt.setText(lstStr);

        float lstBmFl = basisdaten.lstBemessung();
        String lstBmStr = String.valueOf(lstBmFl);
        this.jTextFieldLohnsteuerBemessungsgrundlage.setText(lstBmStr);

    }
```

Der fertige Prototyp



Nachbetrachtung

Es erscheint sicher auf den ersten Blick sehr aufwendig, ein Anwendung auf diese Art zu implementieren, aber in Hinblick auf die Wartbarkeit lohnt sich die Arbeit allemal. Der Grund ist der, dass man die Logikklassen jederzeit austauschen kann, ohne das Benutzerinterface anzugreifen. Der Prototyp wurde auf seine Funktionalität getestet und nach weiteren Beispielen, die einen weiteren Einsatz von objektorientierten Technologien aufzeichen sollen, erfolgt jeweils eine Erweiterung der Basisversion.
Der Source - Code, der Klassen bis zu diesem Zeitpunkt befindet sich in der Website unter Lektion 9.

3.4.11 Lektion 10 - Listen und Sammlungen (Lists, Collections)

Diese Lektion beschäftigt sich mit den Listen und Sammlungen, diese erleichtern den Umgang mit Gruppen von Daten, oder mit Mengen. Ziel diese Lektion ist es, den prinzipiellen Umgang mit den wichtigsten Vertretern kennenzulernen, um diese in späteren Aufgaben sinnvoll einsetzen zu können. Der Einsatz dieser Klassen wird ganz allgemein eingeführt, was für spätere, praxisbezogene Adaptierungen genügen sollte.

String Array

Das String Array stellt die einfachste Form einer Liste dar. Die Instantiierung erfolgt über „new“, der Angabe des Datentyps und der Größe.

```
String[] einfacheListe = new String[3];
```

Der Index für die Wertezuweisung beginnt bei 0, das heißt für die Deklaration der einfachen Liste werden die Werte von 0 bis 2 belegt.

```
einfacheListe[0] = „A“;
einfacheListe[1] = „B“;
einfacheListe[2] = „C“;
```

Ein Fehler bei der Indizierung würde zu einer „Index out of Bounds Exception“ führen.
Generell kann man nicht nur String - Objekte erstellen, sondern es sind jegliche Datentypen und Objekte möglich. Arrays können iterativ bearbeitet und auch ausgegeben werden. Die einfachste Form des Durchlaufens eines Arrays ist die „for“ Schleife.

```
for(int i = 0; i < einfacheListe.length; i++) {
      System.out.println(einfacheListe[i]);
}
```

Von dem Wert i = 0 bis zur maximalen Länge des Arrays wird der Inhalt des Wertes mit dem entsprechenden Index ausgegeben. Als Ergebnis wir A, B, C ausgegeben.

Die for-each Schleife erspart die Angabe eines Integer-Wertes und einer Zählvariablen.

```
for(String listenwerte : einfacheListe) {
      System.out.println(listenwerte);
}
```

Die Ausgabe ist dieselbe, wie beim vorigen Beispiel.

Arrays können auch mehrdimensional sein und werden wie folgt deklariert:

```
String[][] zweidimArray = new String[3][2];
```

Dies ist ein zweidimensionales Array mit drei Spalten und zwei Zeilen. Das Durchlaufen erfolgt mit einer for-Schleife, der Einsatz der for-each Schleife ist hier nicht möglich. Man benötigt zwei verschachtelte for-Schleifen mit zwei Zählvariablen.

```
zweidimArray[0][0] = „A";
zweidimArray[1][0] = „B";
zweidimArray[2][0] = „C";
zweidimArray[0][1] = „D";
zweidimArray[1][1] = „E";
zweidimArray[2][1] = „F";

for (int i = 0; i < zweidimArray.length; i++){
      for(int y = 0; y < zweidimArray.length; y++){
            System.out.println(zweidimArray[i][y]);
      }
}
```

Die Einfachheit der Verwendung und die universelle Einsetzbarkeit beinhalten aber auch einige Nachteile. So ist zum Beispiel das Vergrößern und Verkleinern eines Arrays nicht ohne weiteres möglich, weiters gibt es keinen Schlüssel, der einen gezielten Zugriff erlaubt und es ist nicht bekannt, ob eine Element bereits in einem Array vorhanden ist. Eine elegantere Implementierungsmöglichkeit bietet die Array-List.

Array List

Dazu ist der Import von zwei Java - Packages erforderlich. Diese Deklaration erfolgt im Code vor der Klassendeklaration.

```
import java.util.ArrayList;
import java.util.Arrays;
```

Die Deklaration wird folgendermaßen durchgeführt:
Erzeugen einer Array - Liste mit 100 Elementen;

```
ArrayList liste = new ArrayList(100);
```

Es ist auch möglich, das Array während der Laufzeit zu vergrößern, oder zu verkleinern.
Die ArrayList besitzt mehrere Konstruktoren, unter anderem auch einen, in dem der Wert für die Größe direkt übergeben werden kann.
Zunächst ist es sinnvoll ein String Array zu erzeugen und dieses an die ArrayList zu übergeben.

```
String[] daten = new String[]{„A", „B", „C", „D", „E"};
ArrayList <String> liste = new ArrayList <String> ();
liste.addAll(Arrays.asList(daten));
```

Um die ArrayList typsicher zu machen, muss der Datentyp mit angegeben werden. Diese ArrayList darf nur Strings annehmen.
Die Zuweisung eines einzelnen Wertes erfolgt mit:

```
liste.add(„f");
```

Die Ausgabe der Werte erfolgt mit einer einfachen for-each Schleife.

```
for(String felder : liste) {
      System.out.println(felder);
}
```

Bei der ArrayList ist auch der Einsatz eines Iterators möglich. Dazu muss das Package Iterator importiert werden.

```
import java.util.Iterator;

Iterator <String> itera = liste.iterator();
      while (itera.hasNext()) {
            String feld = itera.next();
            System.out.println(feld);
      }
```

Die ArrayList enthält einen Index, über den man die Werte ansteuern kann.

```
liste.set(1, „neuerWert“);
```

In der Liste wird an Stelle 1 der Wert „neuerWert“ angelegt.
Das Löschen eines Objektes erfolgt mit der Methode „remove“.

```
liste.remove(„neuerWert“);
```

Es ist auch möglich, einen Index-Wert zu löschen.

```
liste.remove(1);
```

Die Überprüfung, ob eine Element vorhanden ist, erfolgt mit der Methode „contains“.

```
liste.contains(„A“);
```

Als Return-Wert wird ein boolean zurückgegeben.

Diese Möglichkeiten und noch viele Methoden, die in der Java-Doc[1] nachzulesen sind, zeigen die vielfältigen Methoden der ArrayList und warum diese in den meisten Fällen dem Array vorzuziehen ist. Der Hauptgrund ist die flexible Einsetzbarkeit.

1 Java Dokumentation - java.sun.com

Hash Maps

Hash Maps bieten eine einfache und performante Möglichkeit, Daten in Form von Schlüssel- und Wertepaaren abzulegen.
Die Deklaration erfolgt folgendermaßen:

```
HashMap map = new HashMap();
HashMap <String, Buch> buecher = new HashMap <String,
Buch>();
buecher.put(„Buch A“, new Buch(„Die Zwei“, 200);
buecher.put(„Buch B“, new Buch(„Die Drei“, 250);
buecher.put(„Buch C“, new Buch(„Die Vier“, 100);
```

Die Kennzeichnung erfolgt immer durch Wertepaare, die als Key und Value Bezeichnet werden. Es folgt ein Beispiel, welches eine Kombination von Text und dazugehörigen Werten betrachtet.

Übungen

Beispiel: HashMap für die Steuerfreibeträge der Pendlerpauschale

Die Pendlerpauschale ist ein Steuerfreibetrag für Personen, die einen langen und/oder unzumutbaren Arbeitsweg haben. Es gibt ein „kleines Pendlerpauschale“ - dabei beträgt der tägliche Weg zur Arbeitsstätte 20 km oder mehr, für eine Fahrtstrecke und ist mit öffentlichen Verkehrsmitteln möglich. Beim so genannten „großes Pendlerpauschale“ ist das Erreichen der Arbeitsstätte durch die Benutzung eines öffentlichen Verkehrsmittels unzumutbar, oder nicht möglich.
Die Freibeträge sollen programmtechnisch mit HashMaps verfügbar gemacht werden.
Als Bibliothek muss das Package HashMap importiert werden.

```
import java.util.HashMap;
```

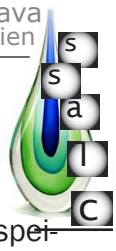
Danach erfolgt das Anlegen der HashMap mit den erforderlichen Datentypen.
HashMap<String, Float> pp = new HashMap<String, Float> ();

Die Zuweisung der Werte erfolgt mit put.

```
pp.put(„Ab 2 km bis 20 km“, 0.00f);
```

Die Ausgabe erfolgt mit get - Methoden.

```
import java.util.HashMap;

public class Main {

    public void pendler() {
        HashMap<String, String> pp = new HashMap<String,
String>();

        pp.put(„Ab 2 km bis 20 km“, „0.00“);
        pp.put(„20 bis 40 km“, „45.50“);
        pp.put(„40 bis 60 km“, „90.00“);
        pp.put(„Über 60 km“, „134.50“);

        System.out.println(pp.get(„20 bis 40 km“));
    }

    public static void main(String[] args) {

        Main m = new Main();
        m.pendler();
    }
}
```



Properties

Properties werden häufig verwendet um Informationen zu speichern und weiterzugeben. Sie dienen dazu, dass Werte und Daten auch nach Beendigung des Programmes wiederverwendet werden können. Die Werte müssen also in irgend einer Form gespeichert werden. Properties werden auf die übliche Art und Weise deklariert und über put werden Werte zugewiesen. Properties sind von HashMap abgeleitet, können aber nur Strings aufnehmen, allerdings ist ansonsten die komplette Funktionalität vorhanden.

```
Properties props = new Properties();

props.put(„KeyA", „ValueA");

props.getProperty(„KeyA");
```

Sinnvollerweise kann man diese Daten in einer Datei abspeichern und beim Öffnen des Programmes wieder einlesen.

3.4.12 Lektion 11 - Properties als Basis für die Überstundenberechnung für Name und Normalbezug

Als Basis dient eine fiktive Tabelle, in der die Namen der Mitarbeiter/innen und deren Normalbezug angegeben wird. Diese Tabelle soll als Stammdatenbasis fungieren, bei Bedarf geändert werden können und immer in aktueller Form als Datei vorliegen. Um mögliche Datenbankexporte durchführen zu können ist auch die Ausgabe im XML - Format wünschenswert.
Zunächst muss das Package java.util.*; importiert werden.
Danach werden mit put die erforderlichen Daten eingegeben.

```
public void abrechnungBasis(){
    Properties pros = new Properties();
    pros.put(„Stein Uwe", „2100");
    pros.put(„Zernig Karl", „4755");
    pros.put(„Hofer Erna", „2600");
    }
```

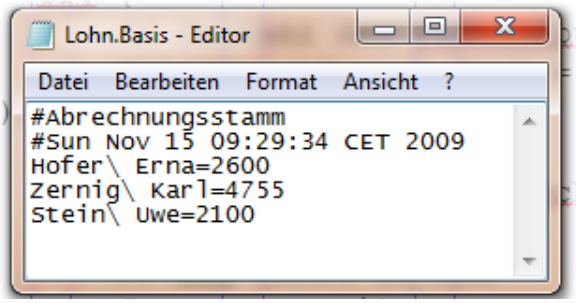
Die Ausgabe erfolgt über die Methode getProperty.

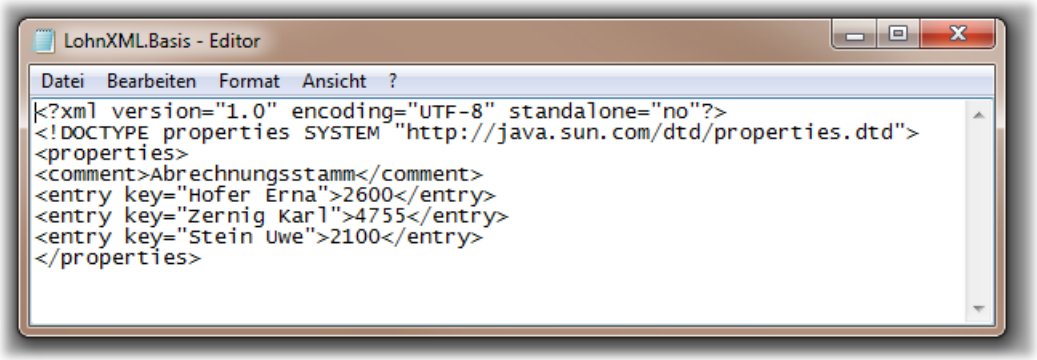
```
System.out.println(pros.getProperty(„Stein Uwe");
```

Nun erfolgt die Ausgabe in eine Datei unter Verwendung der File - Output Streams. Dazu muss das Package java.io.*; importiert werden.

```
// File schreiben
        try {
            // Instanz des Fileoutputstreams anlegen
            FileOutputStream fout = new FileOutputStream(„Lohn.
Basis“);
            // File anlegen
            pros.store(fout, „Abrechnungsstamm“);
            // File schliessen
            // das Ganze in XML
            fout = new FileOutputStream(„LohnXML.Basis“);
            pros.storeToXML(fout, „Abrechnungsstamm“);
            // File schliessen
            fout.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
```

Die beiden Files sehen folgendermaßen aus:





Da kein expliziter Pfad angegeben wurde, befinden sich die Dateien im Hauptverzeichnis des Projektes.

Um das Property - File wieder einzulesen, muss ein File - Input Stream angelegt werden.

```
//Property File einlesen
    public void propsEinlesen() {
        Properties pros = new Properties();
        try {
            // Instanz des Fileoutputstreams anlegen
            FileInputStream fin = new FileInputStream(„Lohn.Basis“);
            // File anlegen
            pros.load(fin);
            // File schliessen
            fin.close();

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

    }
```

Für die Auswertung muss natürlich auch eine Ausgabe generiert werden.

```
System.err.println(pros.getProperty(„Hofer Erna“));
```

Für die XML - Datei funktioniert das Prinzip der Properties analog. Im Prinzip wurde aus einer Textdatei ein Wert ausgelesen und zugewiesen. Einsatzbereich der Properties ist hauptsächlich die Internationalisierung, das heißt für die Mehrsprachlichkeit von Programmen.

Es gibt noch mehrere Formen von Aufzählungen und Listen, unter anderem auch den Vector, der aber mittlerweile als veraltert gilt und nicht mehr verwendet werden sollte. Es gibt auch likedLists, sowie linkesHashMaps, man sollte aber beachten, dass alle diese Container Vorteile und Nachteile besitzen und es ratsam ist, sich in der Planungsphase mit diesen zu befassen, um einen optimalen Einsatz zu gewährleisten.

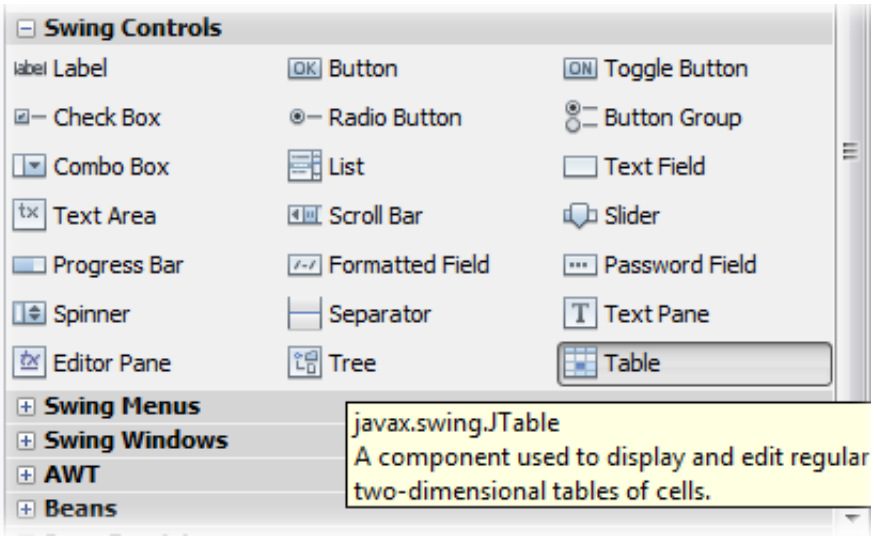
Ausnahmen - Exceptions

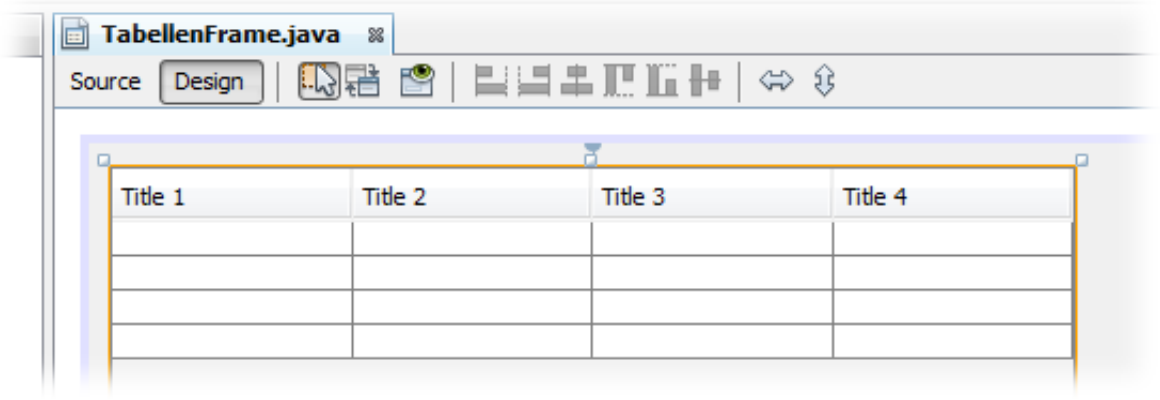
Java bringt eine mächtige Funktionalität für die Ausnahmebehandlung mit. Der Aufbau erfolgt mit einem try und catch Block. Im try - Block stehen die Anweisungen, die das Programm zu erledigen hat, diese werden im catch - Block abgefangen, wenn im try - Block Fehler auftreten. Dabei sind echte und potentielle Fehler zu unterscheiden. Ein echter Fehler wäre eine Division durch Null und ein potentieller, eine Datei, die geöffnet werden soll, aber nicht vorhanden ist. Der stackTrace bietet hierbei die umfangreichsten Informationen zum jeweiligen Fehler. Für weitere Informationen sei auf die Dokumentation verwiesen.

3.4.13 Lektion 12 - Zinsberechnung

Tabellen - als Vorbereitung für Daten aus Datenbanken

Eine sehr häufige Art und Weise, Daten strukturiert darzustellen ist eine Tabelle. Darum verdient diese auch ein besonderes Augenmerk. Mit Tabellen können gleichförmige Informationen geordnet präsentiert werden. Dies eignet sich besonders für die Präsentation von Daten aus Datenbanken. Java - Swing stellt zu diesem Zwecke die Tabelle JTable zur Verfügung. Als einführendes Beispiel soll eine einfache Tabelle mit Beträgen und Prozentsätzen erstellt





Beim Ausführen erhält man bereits eine Anwendung mit einer Tabelle, in die man Daten eingeben kann.
Von der IDE wurde außer der Tabelle auch eine ScrollPane angelegt. Dies ist immer dann sinnvoll, wenn die Datenmengen und damit der erforderliche Platz nicht bekannt sind. Bei Datenmengen, die in der Standardgröße der Tabelle nicht mehr angezeigt werden können, werden automatisch Bildlaufleisten eingefügt.

Der von der IDE erstellte Code für das Table - Model

```
jScrollPane1 = new javax.swing.JScrollPane();
        jTable1 = new javax.swing.JTable();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_
CLOSE);
        setTitle(„Basistabelle“);

        jTable1.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null},
                {null, null, null, null}
            },
            new String [] {
                „Title 1“, „Title 2“, „Title 3“, „Title 4“
            }
        ));
        jScrollPane1.setViewportView(jTable1);
```

In der ersten Zeile wird die ScrollPane erstellt und danach die Table. Zusätzlich wird ein default - Table Model angelegt, mit der Struktur „null“ für ein leeres Objekt. Auch die Spaltenköpfe werden als eindimensionales StringArray angelegt.
Um die Tabelle mit Leben zu füllen lege ich eine weitere Klasse mit dem Namen ZinsTm an.
Diese Klasse erweitert das abstractTableModel, welches von sich aus schon sehr viele Funktionalitäten für Tabellen bietet. Diese Funktionalitäten erkaufen wir uns durch Vererbung und dass wir gezwungen sind, drei Methoden zu überschreiben, getRowCount, getColumnCount und getValueAt. Um nun die Auswirkungen von Zins und Zinseszinsen zu zeigen, wurde diese Klasse mit den Teilberechnungen angelegt. Begonnen wird mit einem Startwert (StartBetrag) von 1.0000,0.- , einer Laufzeit (Laufzeit), einem Mindestzinssatz (minZ) und einem maximalen Zinssatz (maxZ). Diese Membervariablen werden im Konstruktor der Klasse übergeben. Die Reihen der Tabelle (getRowCount) werden aus dem Wert der Laufzeit berechnet. Will man zum Beispiel 15 Jahre berechnen, so muss für die Laufzeit dieser Wert angegeben werden. Die Spalten werden aus der Differenz zwischen dem maximalen und minimalen Zinssatz gezählt und eine 1 hinzugefügt. Anschließend werden alle Felder gemäß des Modells mit getValueAt berechnet und nach der Standardformel werden die Zins und Zinseszinswerte berechnet. Dazu wird die Java-Potenzfunktion benötigt und für die korrekte Ausgabe der Werte wird NumberFormat implementiert.

Der Sourcecode der Klasse ZinsTm

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import java.math.*;
import java.text.*;

public class ZinsTm extends AbstractTableModel {

    public static double StartBetrag = 10000.0;
    private int Laufzeit;
    private int minZ;
    private int maxZ;

    // Erzeugen einer neuen Instanz von ZinsTM
    public ZinsTm() {
    }

    public ZinsTm(int l, int min, int max) {
        this.Laufzeit = l;
        this.minZ = min;
        this.maxZ = max;
    }

    public int getRowCount() {
        return Laufzeit;
    }

    public int getColumnCount() {
        return maxZ - minZ + 1;
    }

    public Object getValueAt(int r, int c) {
        double zz = (c + minZ) / 100.0;
        int lauf = r;
        double eff = StartBetrag * Math.pow(1 + zz, lauf);
        return NumberFormat.getCurrencyInstance().format(eff);
    }

    public String getColumnName(int c) {
        double zz = (c + minZ) / 100.0;
        return NumberFormat.getPercentInstance().format(zz);
    }
}
```

In der Klasse TabellenFrame muss nun die Berechnungsklasse in den Konstruktor eingebunden werden. Im Konstruktor der Klasse TabellenFrame wird der Aufruf tabellenmethode() eingefügt.

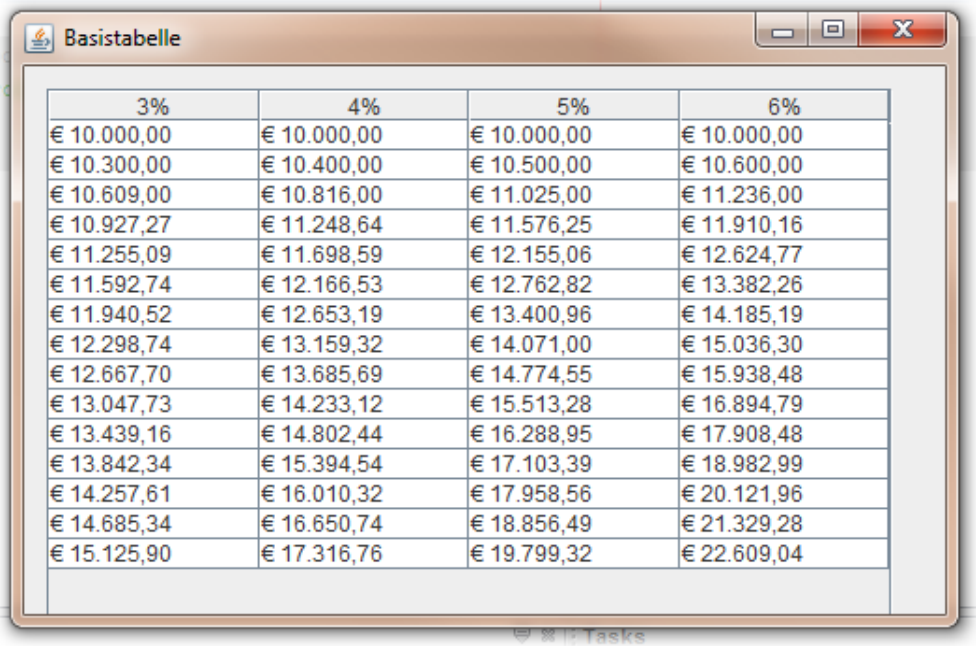
```
/** Creates new form TabellenFrame */
    public TabellenFrame() {
        initComponents();
        tabellenmethode();
    }
```

Die Implementierung der Tabellenmethode sieht wie folgt aus:

```
private void tabellenmethode(){
        ZinsTm  tM = new ZinsTm(15, 3, 6);
        jTable1.setModel(tM);
    }
```

Zunächst wird die Klasse instantiiert. Die Instanz dieser Klasse ist ein TableModel. Die neu berechneten Daten werden der Tabelle mit setModel übergeben und dadurch wird eine völlig andere Präsentation erreicht.

Das Ergebnis:



3%	4%	5%	6%
€ 10.000,00	€ 10.000,00	€ 10.000,00	€ 10.000,00
€ 10.300,00	€ 10.400,00	€ 10.500,00	€ 10.600,00
€ 10.609,00	€ 10.816,00	€ 11.025,00	€ 11.236,00
€ 10.927,27	€ 11.248,64	€ 11.576,25	€ 11.910,16
€ 11.255,09	€ 11.698,59	€ 12.155,06	€ 12.624,77
€ 11.592,74	€ 12.166,53	€ 12.762,82	€ 13.382,26
€ 11.940,52	€ 12.653,19	€ 13.400,96	€ 14.185,19
€ 12.298,74	€ 13.159,32	€ 14.071,00	€ 15.036,30
€ 12.667,70	€ 13.685,69	€ 14.774,55	€ 15.938,48
€ 13.047,73	€ 14.233,12	€ 15.513,28	€ 16.894,79
€ 13.439,16	€ 14.802,44	€ 16.288,95	€ 17.908,48
€ 13.842,34	€ 15.394,54	€ 17.103,39	€ 18.982,99
€ 14.257,61	€ 16.010,32	€ 17.958,56	€ 20.121,96
€ 14.685,34	€ 16.650,74	€ 18.856,49	€ 21.329,28
€ 15.125,90	€ 17.316,76	€ 19.799,32	€ 22.609,04

Abb.: 19 Das Ergebnis der Zinseszinsberechnng

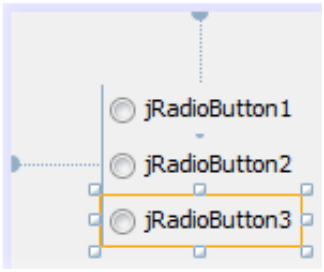
3.4.14 Lektion 13 - Bestellsystem mit Look & Feel

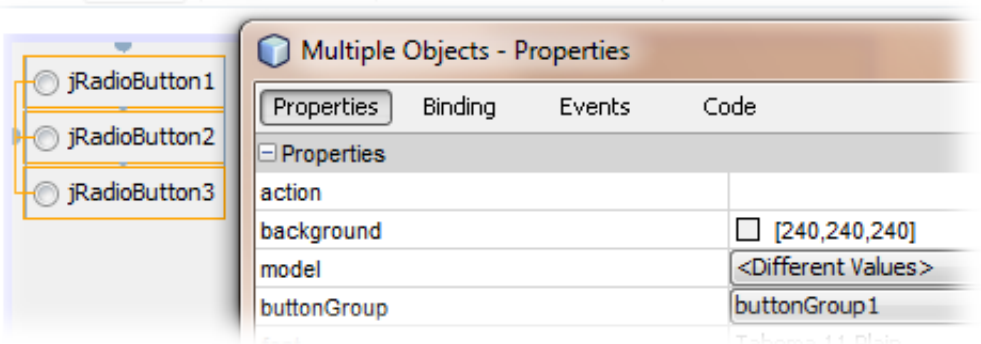
Erklärung

In Java lässt sich das Aussehen der grafischen Objekte frei bestimmen. So wird bei jedem Benutzer die Illusion erzeugt, als handle es sich um eine plattformabhängige Anwendung. Java kann nicht nur auf verschiedenen Plattformen laufen, sondern auch das Look & Feel von verschiedenen Plattformen annehmen.

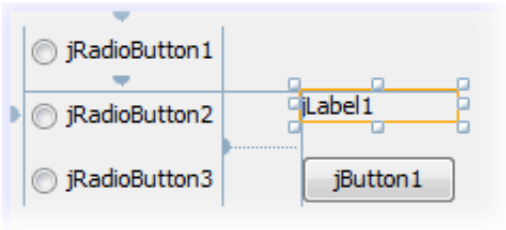
Übungen

Um das Java Look & Feel zu demonstrieren soll beispielhaft eine Benutzeroberfläche für ein Bestellservice angelegt werden und danach das Look & Fell implementiert und verändert werden.
Als erstes wird eine ButtonGroup mit RadioButtons angelegt. RadioButtons sind Schaltflächen, von denen jeweils nur eine aktiviert sein kann.
Damit die Buttons richtig funktionieren muss eine ButtonGroup hinzugefügt werden. Wenn mehrere Elemente des gleichen Datentyps aktiviert werden, kann man die Eigenschaften für diese Elemente gleichzeitig setzen.



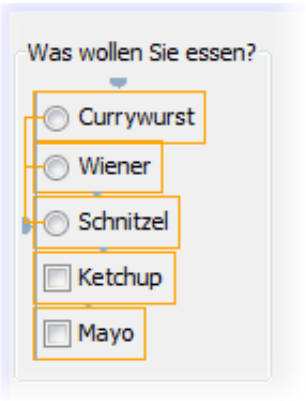


Das hinzufügen der ButtonGroup und die entsprechende Zuweisung der einzelnen Buttons bewirkt das richtige Verhalten der RadioButtons beim klicken.



Um die Anwendung mit einer Basisfunktionalität zu versehen werden nun ein Label und ein Button implementiert. Der Text des Labels soll sich ändern, wenn der Button gedrückt wird. Der Button wird mit einem ActionListener versehen, dessen Code folgendermassen aussieht. Dies ist zwar noch nicht besonders sinnvoll, dient aber als Basisfunktionalitätstest.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        this.jLabel1.setText(„Ich wurde gedrückt!“);
    }
```

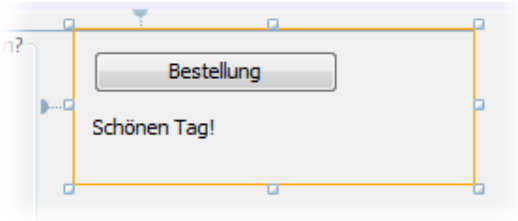


Als weitere Elemente werden zwei Checkboxes und ein Panel mit TitleBorder eingeführt. Das Textlabel soll nun insofern geändert werden, dass die Ausgewählten Felder aus den Checkboxes ausgegeben werden.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        if (this.jCheckBox1.isSelected() && this.jCheckBox2.isSelected()) {
            this.jLabel1.setText(„Nur eine Auswahl ist möglich!“);
            // deaktivieren der Checkboxes und Benutzer zu einer neuen
            // Wahl zwingen
            this.jCheckBox1.setSelected(false);
            this.jCheckBox2.setSelected(false);
        }
    }
```

Nun wird der Text des Buttons in „Bestellung“ geändert und eine weiteres Panel hinzugefügt, sowie der Text des JLabels in „Schönen Tag!“ geändert.

Als TitleBorder schlage ich vor: „Infos zur Bestellung“.



Zentrieren der Anwendung am Monitor

Ein Fenster kann natürlich sehr einfach über setLocation() pixelgenau an jeder Stelle auf dem Desktop platziert werden. Will man es allerdings mittig erscheinen lassen, benötigt man Informationen über die derzeitige Bildschirmauflösung. Über die statische Methode getDefaultToolkit() der Klasse Toolkit bekommt man eine Referenz auf das Toolkit - Objekt. Über dieses Toolkitobjekt, welches auch andere plattformabhängige Informationen kapselt, bekommt man die Auflösung des Desktops heraus. Liest man nun noch die aktuellen Abmessungen des Fensters über getWith() und getHeight(), ist es ein Leichtes, die linke obere Ecke des Fensters über setLocation() so zu platzieren, dass sein Fenstermittelpunkt im Zentrum des Bildschirms steht.
Als Package muss java.awt.event.*; importiert werden.
So sieht nun der Code für die geänderte main() Methode aus:

```
public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                // new NewJFrame().setVisible(true);

                NewJFrame njf = new NewJFrame();
                // Über DefaultToolkit kann die Bildschirmgroesse
                // bestimmt werden
                Dimension dim = Toolkit.getDefaultToolkit().ge-
tScreenSize();
                // Halbe Bildschirmabmessung ergibt den Bild-
schirmmittelpunkt
                int xCenter = dim.width / 2;
                int yCenter = dim.height / 2;
                // Die Platzierung der Fenster orientiert sich an
der linken
                // oberen Ecke. Zur Korrektur muss somit die hal-
be
                // Komponentenabmessung mit eingerechnet werden
                int xDiff = njf.getWidth() / 2;
                int yDiff = njf.getHeight() / 2;
                int xCalculated = xCenter - xDiff;
                int yCalculated = yCenter - yDiff;
                // setLocation() legt den Ort der Komponente fest
                njf.setLocation(xCalculated, yCalculated);
                // setVisible(true) macht die Komponente sichtbar
                njf.setVisible(true);

            }
        });
    }
```

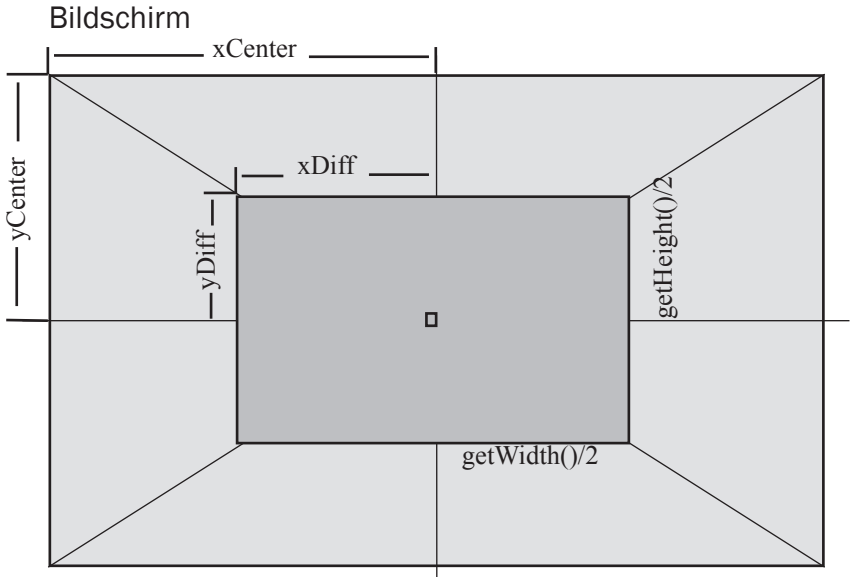
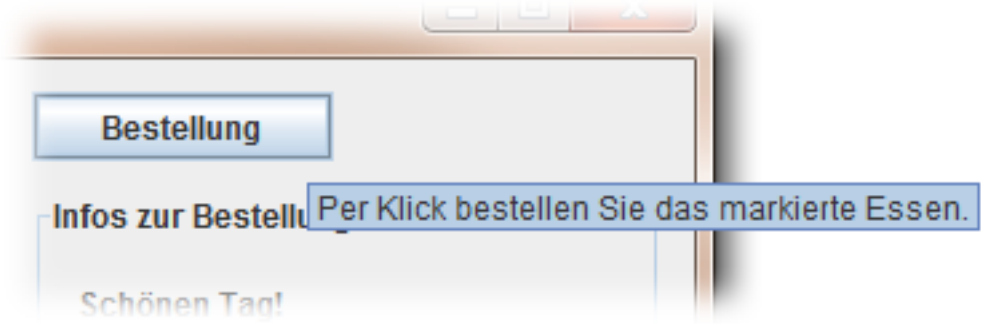


Abb.: 20, Fenster am Bildschirm zentrieren

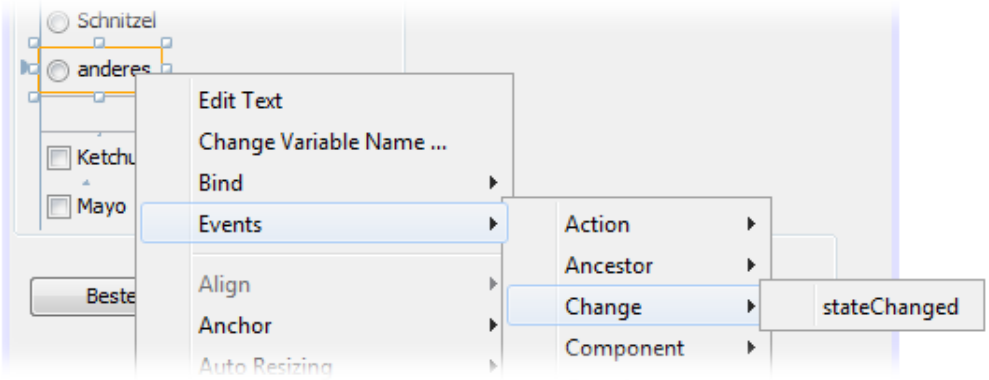
Tooltiptext

Nun wird ein Tooltiptext für den Button eingetragen. Als Tooltip soll folgender Text erscheinen: „Per Klick bestellen Sie das markierte Essen“
Vorgangsweise: Öffnen der Eigenschaften des Buttons und den Eintrag im Editorfenster durchführen.



Es wird nun ein weiterer RadioButton im Panel „Was wollen Sie essen?“ mit der Bezeichnung anderes eingeführt. Darunter wird ein Textfeld eingeführt, das gewisse Eigenschaften besitzen sollte. Wenn „anderes“ noch nicht ausgewählt ist, darf es noch nicht editierbar sein. Diese Einstellung erfolgt über die Properties. Damit dieser RadioButton nun richtig funktioniert, muss er mit einem Event versehen werden.

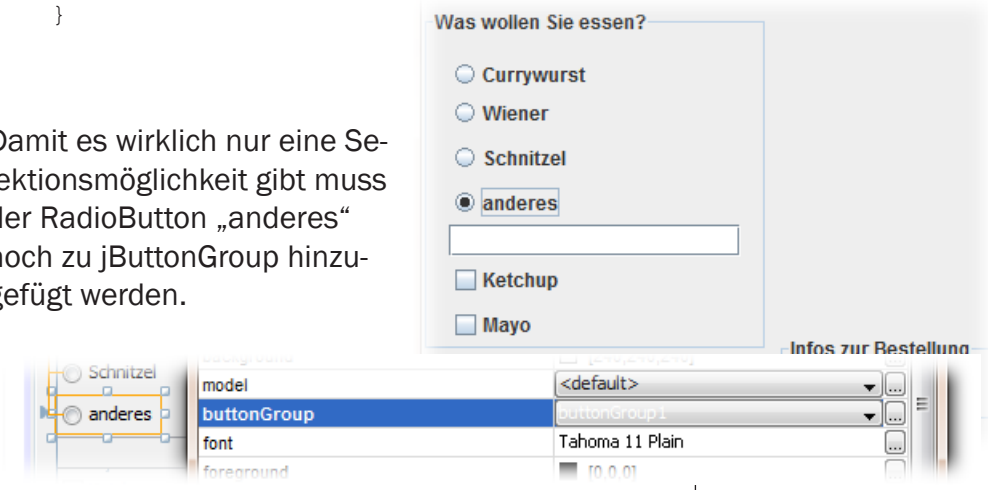
Dieses Event ist diesmal ein Change - Event. Das heißt, wenn sich der Status des RadioButtons ändert, muss etwas passieren. (Properties - Event - Change)



Wenn der RadioButton ausgewählt (selected) ist, soll das Textfeld einen aktiven (enabled) Status bekommen. Das heißt das Textfeld ist verwendbar und auch bereit für eine Eingabe, in jedem anderen Fall soll dieses Textfeld nicht verwendet werden.
Der Code der jButton4 stateChange Methode:

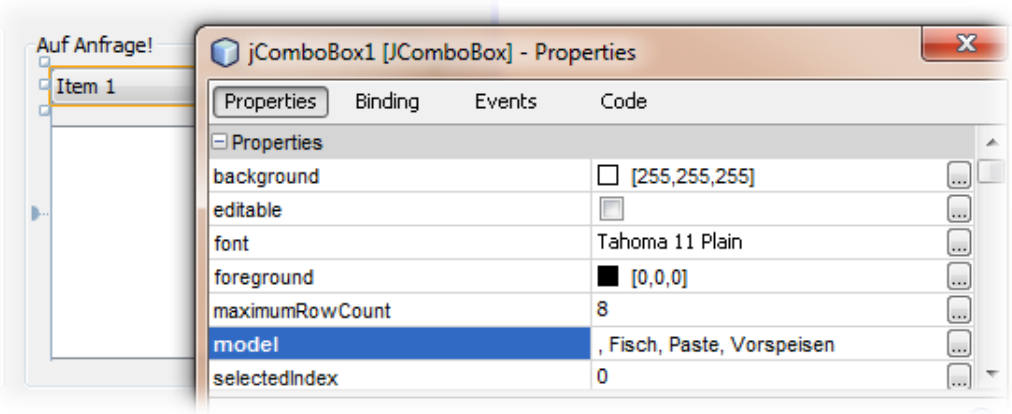
```
private void jRadioButton4StateChanged(javax.swing.event.Chan-
geEvent evt) {
        if (this.jRadioButton4.isSelected()) {
            this.jTextField1.setEnabled(true);
            this.jTextField1.setEditable(true);
        } else {
            this.jTextField1.setEnabled(false);
            this.jTextField1.setEditable(false);
        }
    }
```

Damit es wirklich nur eine Selektionsmöglichkeit gibt muss der RadioButton „anderes“ noch zu jButtonGroup hinzugefügt werden.



Combobox - Textarea

Nun wird ein neues jPanel angelegt mit dem textBorder - String „Auf Anfrage“. In dieses Panel kommt eine Combobox, welche auf Mausklick ein Pulldown - Menü erzeugt, aus dem man auswählen kann. Es wird noch ein Textfeld benötigt, welches Informationen geben soll, ob die gewünschte Anfrageauswahl überhaupt verfügbar ist.
Die Combobox muss „editable“ sein und bekommt folgendes „Model“.
Der Anfang vor dem Komma dient dazu, ein leeres Feld zu erzeugen.
(, Fisch, Pasta, Vorspeisen)



Als Event für die Combobox wird itemStateChanged gewählt.

```
private void jComboBox1ItemStateChanged(java.awt.event.ItemEvent
evt) {
        String f1 = „Forelle“;
        String f2 = „Hecht“;
        String f3 = „Spaghetti Bolognese“;
        String f4 = „Pasta Mista“;
        String f5 = „Vorspeisenplatte“;
        String f6 = „Entenleber“;

        // ausgewählten Index der Combobox herausfinden
        int i = this.jComboBox1.getSelectedIndex();
        //Text in der Textarea löschen
            this.jTextArea1.setText(„“);
        switch (i) {
            case 1:
                this.jTextArea1.append(f1 + „\n“);
                this.jTextArea1.append(f2 + „\n“);
                break;
            case 2:
                this.jTextArea1.append(f3 + „\n“);
                this.jTextArea1.append(f4 + „\n“);
                break;
            case 3:
                this.jTextArea1.append(f5 + „\n“);
                this.jTextArea1.append(f6 + „\n“);
                break;
        }
}
```

Swing Komponenten besitzen in der Regel eine ungeheure Vielfalt an Funktionalitäten. Erweitern wird nun die Combobox um die Funktionalität der Editierbarkeit. Man kann nun Text eingeben, aber es passiert nicht viel, da programmtechnisch noch keine Reaktion darauf implementiert wurde, zum Beispiel anfügen an die Liste und weiterverwenden. Man sieht zumindest beispielhaft, wie vielseitig Komponenten in Swing sein können.

ProgressBar und JSlider

Ein weiteres interessantes Element ist die jProgressBar. Dieses Element wird nun über dem Button Bestellung hinzugefügt. Beim Betrachten der Properties sieht man, dass als Standardwert der Bereich von 0 bis 100 gilt. Ändern wir nun den Maximumwert auf 50 und es muss noch ein wenig Code hinzugefügt werden. Grundsätzlich ist es erforderlich einen ActionListener zu erweitern und es muss ein Timer kreiert werden.
Ergänzung der Import Anweisung in der Klasse:

```
import javax.swing.*;
import java.awt.event.ActionListener;
```

Die Codeergänzungen vor und im Konstruktor:

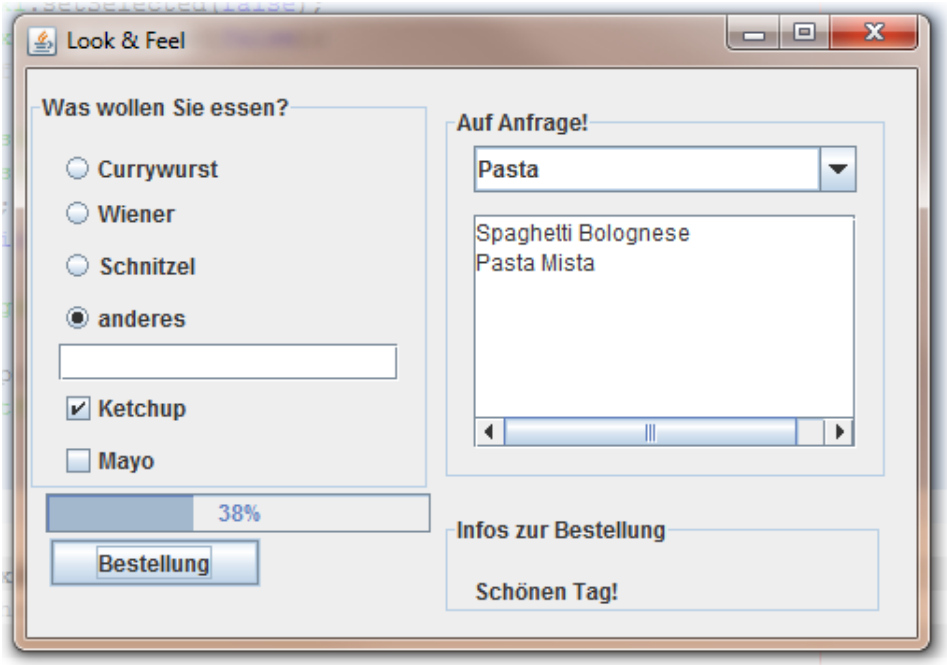
```
public class NewJFrame extends javax.swing.JFrame {

    /** Creates new form NewJFrame */
    Timer timer;
    // Startwert initiiern
    int wert = 0;

    public NewJFrame() {
        initComponents();
        // Action Listener des Buttons
        ActionListener[] al = jButton1.getActionListeners();
        timer = new Timer(100, al[0]);

    }
```

Der Timer wird nun dem Button hinzugefügt. In der jProgressBar soll der Rand gezeichnet werden, sie soll gefüllt werden. Danach wird der Timer gestartet und jedes mal, wenn eine Zehntelsekunde vergangen ist, wird der anfangs initialisierte Wert um eins erhöht und in der jProgressBar angezeigt. Ist der maximale Wert erreicht, wird der Timer gestoppt und im Textfeld erscheint „Bestellung eingegangen!“

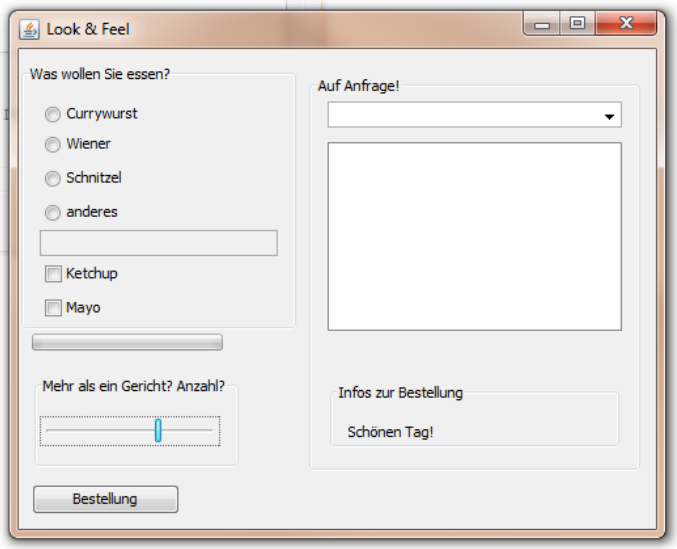
Code der Actions auf dem jButton1:

Damit em Ende der Text „Bestellung eingegangen!“ wirklich erscheint, muss das jTextfield1 auf editierbar gesetzt werden.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        if (this.jCheckBox1.isSelected() && this.jCheckBox2.isSelected()) {
            this.jLabel1.setText(„Nur eine Auswahl ist möglich!“);
            // deaktivieren der Checkboxes und Benutzer zu einer neuen
            // Wahl zwingen
            this.jCheckBox1.setSelected(false);
            this.jCheckBox2.setSelected(false);
        } // Ergänzungen für die ProgressBar
        else {
            this.jProgressBar1.setBorderPainted(true);
            this.jProgressBar1.setStringPainted(true);
            timer.start();
            if (wert < this.jProgressBar1.getMaximum()) {
                wert++;
                this.jProgressBar1.setValue(wert);
            } else {
                timer.stop();
                this.jTextField1.setText(„Bestellung eingegangen!“);
                  // this.jLabel1.setText(„Bestellung eingegangen!“);
            }
        }
    }
```



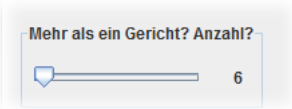
Ein weiteres zu implementierendes Element ist der jSlider. Ein jPanel wird hinzugefügt, mit einem TitledBorder und dem Text „Mehr als ein Gericht? Anzahl?“. In dieses Panel wird nun der jSlider gepackt und das Formulat hat nun folgendes Aussehen.



Die Properties paintLabels, paintticks und snapToTicks sollten aktiviert werden. Als Startwert wird 1 eingegeben. Danach wird eine neuer label mit einem Leerstring hinzugefügt. Als Ereignistyp für den jSlider wird stateChanged implementiert.

```
 private void jSlider1StateChanged(javax.swing.event.ChangeEvent
evt) {
        this.jLabel2.setText(String.valueOf(jSlider1.getVa-
lue()));
```

Um zu vermeiden, dass der Kunde auf Null Gerichte zurückgeht, wird bei den Properties der Wert „minimum“ auf 1 gesetzt.



Um den gewünschten UI - Manager zu auszuwählen ist in der run() Methode das „setLookAndFeel“ einzustellen.

```
try {
                    UIManager.setLookAndFeel(„javax.swing.plaf.metal.Metal-
LookAndFeel“);
                    //UIManager.setLookAndFeel(„com.sun.java.swing.plaf.
motiv.MotivLookAndFeel“);
                    //UIManager.setLookAndFeel(„com.sun.java.swing.plaf.
windows.WindowsLookAndFeel“);
                    //UIManager.setLookAndFeel(„javax.swing.plaf.multi.
MultLookAndFeel“);
                } catch (Exception ex) {
                }
```

3.4.15 Lektion 14 - Datenbanken, XML, Reflection

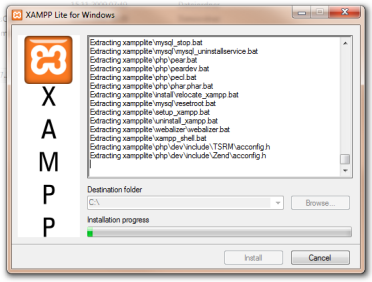
Datenbankzugriff mit JDBC

Erklärung

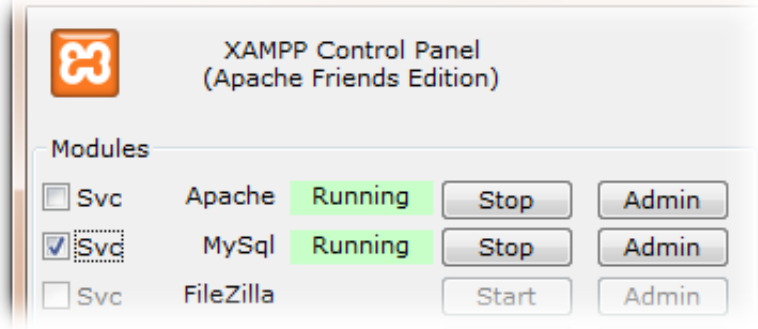
In diesem Kapitel geht es um drei Themenkreise, die für jeden Programmierer wichtig sind. Es geht um Datenbanken, sowie Datenbankzugriff, das andere, XML und Reflection, was hier im Speziellen im Zusammenhang mit Datenbanken behandelt wird.

Übungen

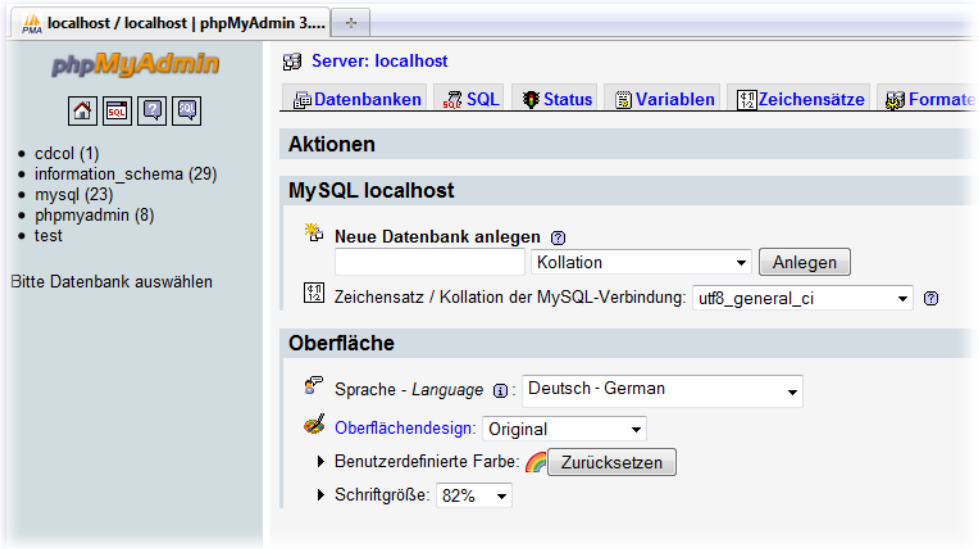
Neues Projekt als Java - Applikation anlegen.
Als Datenbank - Management - System wird das Open Source Produkt MySQL verwendet. Das Grundprinzip ist immer gleich und es kann prinzipiell auch jedes andere DBMS verwendet werden. Der Kontakt zu einer Datenbank erfolgt über einen so genannten Connector, die für Java „JDBC“ genannt werden. Als Einstieg gibt es einige Grundlagen zu Datenbanken. Um schnell eine Datenbankentwicklungsumgebung bereit zu haben, fiel die Entscheidung auf XAMPP. Dies ist eine Distribution von Apache, MySQL, PHP und Perl, die es ermöglicht diese Programme auf sehr einfache Weise zu installieren.



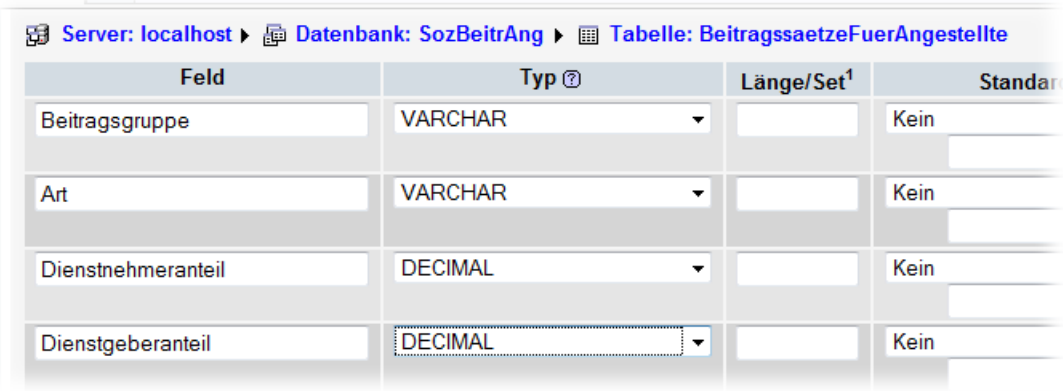
Nach erfolgreicher Installation kann über den Apache Webserver und dem Administrationstool MyPHPAdmin auf das MySql Database - Management - System zugegriffen werden.



Für Testimplementierungen habe ich für den Zugriff zum Datenbankserver den Benutzernamen „root“, ohne Passwort gewählt. Sollte man sonst keinesfalls machen, aber für Trainingszwecke ist es ausreichend.



Nun wird eine Datenbank mit dem Namen SozBeitrAng (Sozialversicherungsbeitragssätze für Angestellte) angelegt.



Beitragsgruppe für alle D1; Arbeitslosenversicherung 3.00, 3.00; Krankenversicherung 3.82, 3.83; Unfallversicherung 0.00, 1.40; Pensionsversicherung 10.25, 12.55; Kammerumlage 0.50, 0.00; Wohnbauförderungsbeitrag 0.50, 0.50; IESG Zuschlag 0.00, 0.55; MV Beitrag 0.00, 1.53;

Beispielhafte SQL Anweisung zum erstellen der Tabelle:

```
CREATE TABLE `SozBeitrAng`.`BeitragssaetzeFuerAngestellte` (
`Beitragsgruppe` VARCHAR( 20 ) NOT NULL ,
`Art` VARCHAR( 20 ) NOT NULL ,
`Dienstnehmeranteil` DECIMAL NOT NULL ,
`Dienstgeberanteil` DECIMAL NOT NULL
) ENGINE = MYISAM ;
```

Die SQL Insert Anweisung für den ersten Datensatz lautet:

```
INSERT INTO `SozBeitrAng`.`beitragssaetzefuerangestellte` (`Bei-
tragsgruppe`, `Art`, `Dienstnehmeranteil`, `Dienstgeberanteil`)
VALUES (‚D1', ‚Arbeitslosenversicherung', ‚3.00', ‚3.00');
```

Die SQL Select Anweisung, um alle Datensätze anzuzeigen:

```
SELECT *
FROM `beitragssaetzefuerangestellte`
```

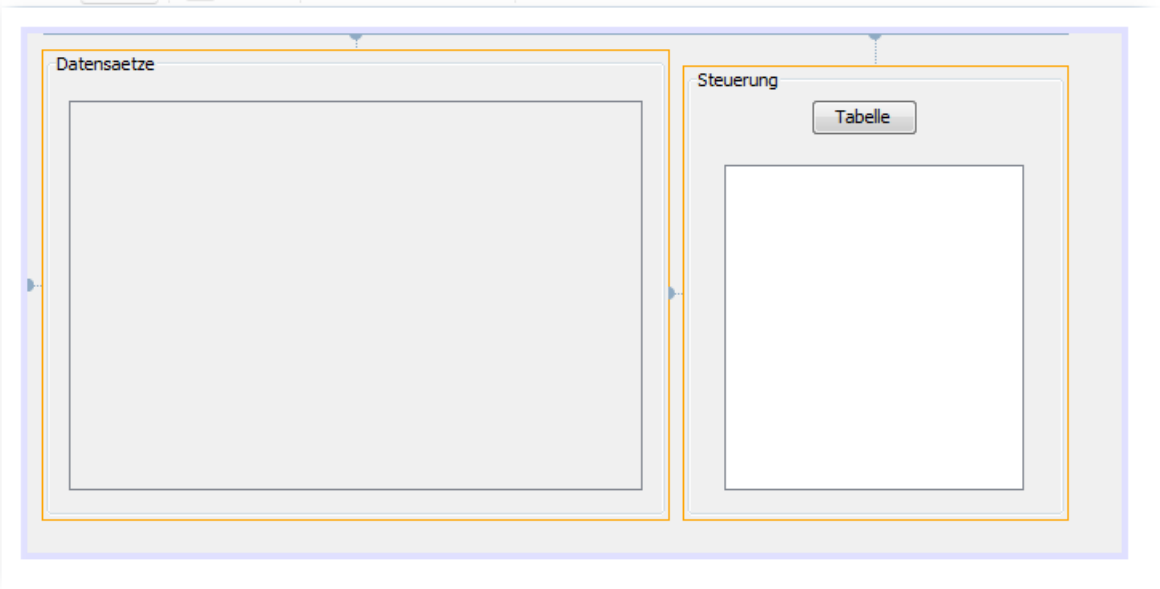
Als nächstes werden die Klassen Main, Verbindung, Kommunikation und Datensattzeiger angelegt. Dies ist nicht zwingend notwendig, aber im Sinne der Datenkapselung sehr übersichtlich.

Die Klasse Main:

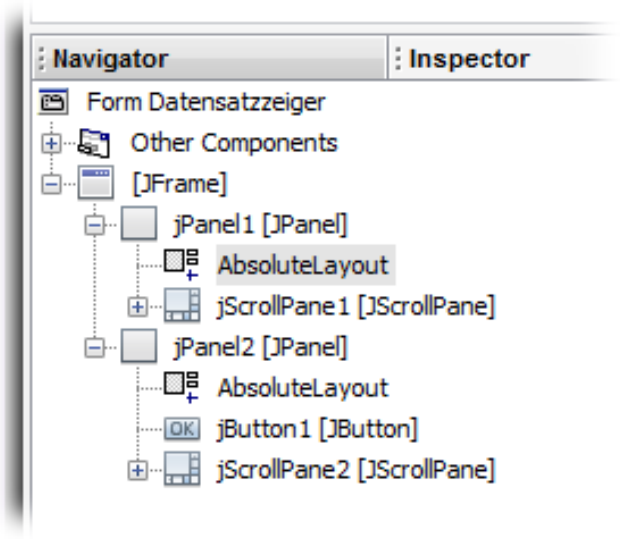
```
public class Main {

    public static void main(String[] args) {

        // Einstig und Aufruf des jFrames
        Datensatzzeiger dz = new Datensatzzeiger();

    }
}
```

Hier wird eine Instanz der Klasse Datensatzzeiger erzeugt und die Klasse dient als Programmeinstieg.
Die Klasse beinhaltet ein jPanel mit dem TextborderText „Datensaetze“ und einer ScrollPane. Ein weiteres jPanel hat im Textborder „Steuerung“, eine Button mit dem Namen „Tabelle“ und eine jTextArea.

Layout der Klassen Datensatzzeiger



Die Einträge im Inspector



Es ist auch ratsam, alle Container mit einem Layoutmanager zu versehen, um später nicht unliebsame Überraschungen zu erleben. In diesem Datenbankbeispiel ist dies jedoch nicht unbedingt erforderlich, ich habe jedenfalls das AbsolutLayout gewählt.

Grundsätzlich muss eine Kommunikation, oder Verbindung zur Datenbank aufgebaut werden. Dies geschieht mit Hilfe der Klasse Verbindung. Zwei import Anweisungen ermöglichen die Kommunikation mit der Datenbank und das Abfangen von Ereignissen.

```
import java.sql.*;
import java.awt.*;
```

Nicht vergessen, den MySQL Connector für Java, am besten von der MySQL Website herunterladen. Das jar File muss im jdk Ordner unter jre lib ext gespeichert werden.

```
import java.sql.*;
import java.awt.*;

public class Verbindung {

    // Verbindungsaufbau mit der statischen
    // Klasse Connection
    // statisch bedeutet, die Klasse muss
    // nicht instantiiert werden
    public static Connection driver() {
        java.sql.Connection conn = null;
        try {
            Class.forName(„com.mysql.jdbc.Driver“).newInstance();
            // String url = „jdbc:mysql://localhost/SozBeitrAng?us
er=root&password=“;
            String url = „jdbc:mysql://localhost:3306/SozBei-
trAng“;
            // conn = DriverManager.getConnection(url);
            conn = DriverManager.getConnection(url, „root“, „“);

        } catch (ClassNotFoundException cex) {
            cex.printStackTrace();
        } catch (SQLException sqlex) {
            sqlex.printStackTrace();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return conn;
    }
}
```

Um nun mit dieser Connection etwas anfangen zu können, wird ein Statement benötigt, dieses wird in der Klasse Kommunikation implementiert.

Die Klasse Kommunikation:

```
import java.sql.*;
/**
 *
 * @author Ehrenfried Stuhlpfarrer
 */
public class Kommunikation {

    /** Creates a new instance of Kommunikation */
    public Kommunikation() {
    }

    public static Statement stm(Connection conn)
    {
        Statement stm = null;

        try{
            stm = conn.createStatement();
        }
        catch (Exception ex){ex.printStackTrace();}
        return stm;
    }

    public static ResultSet rs(String query,Statement stm)
    {
        ResultSet rs= null;
       try{
            rs = stm.executeQuery(query);

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

Um mit der Connection etwas anfangen zu können, benötigt man ein Statement. Diesem Statement stm wird eine „query“ zugeordnet und mit „executeQuery“ wird diese Methode ausgeführt. Als Rückgabe erhalten wir ein „Result Set“.

Als nächstes wird im Datensatzzeiger in der ScrollPane eine Tabelle eingebaut. In der TextArea sollen SQL Statements eingegeben werden.
Zunächst muss der Frame nach initComponents() mit setVisible(true) sichtbar gemacht werden. Als Importanweisungen werden swing und table benötigt.

```
import javax.swing.*;
import javax.swing.table.*;
```

```
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.util.*;
import java.util.Vector.*;
import java.sql.*;
```

Der Text, welcher in der TextArea eingegeben wird, soll abgefragt werden, wenn der Button „Tabelle“ gedrückt wird.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
       String query = jTextArea1.getText();
       query.replaceAll(„\n“, „ „);
       query.replaceAll(„\r“, „ „);
       //Abfragen(query);
    }
```

Nun wird die Methode Abfragen implementiert. Diese Methode baut zunächst einmal eine Verbindung auf. Auf die statische Methode driver kann direkt über den Klassennamen zugegriffen werden und liefert uns eine Connection zurück. Das gleiche passiert mit der Kommunikation, wir verwenden die Connection und holen uns ein Statement. Mit dem Statement und dem String „was“ erhalten wir über die Funktion rs das ResultSet. Nun kommen wir zur Verknüpfung zwischen relationalen Datenbanken und Swing-Tabellen. Die Methode tabellen(rs) nimmt das ResultSet entgegen.

```
public void Abfragen(String was) {
        Connection conn = Verbindung.driver();
        Statement stm = Kommunikation.stm(conn);
        tabellen(rs);
    }
```

Nun erfolgt die Abarbeitung der ResultSets. Zunächst werden die ColumnIdentifiers (die Köpfe) belegt. Die Strings für die Header werden erstellt und fügen die Strings als Spalten hinzu. Die einzelnen Datensätze werden zeilenweise mit getRow abgegriffen.

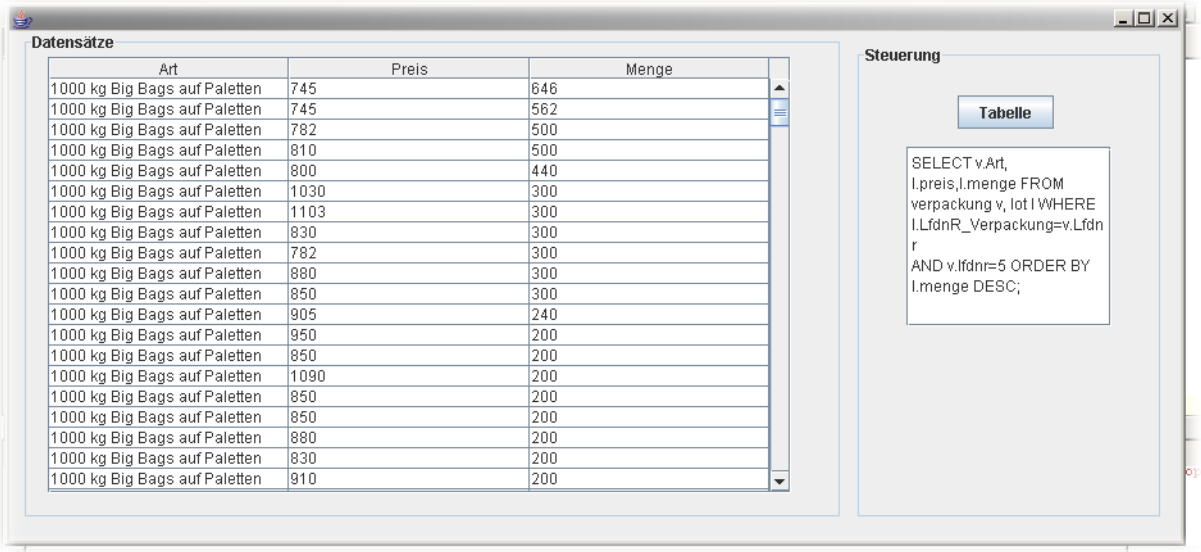
Die Klasse Datensatzzeiger

```
import javax.swing.*;
import javax.swing.table.*;
import java.awt.*;
import java.util.*;
import java.util.Vector.*;
import java.sql.*;
```

```
/**
 *
 * @author Stuhlpfarrer
 */
public class Datensatzzeiger extends javax.swing.JFrame {

    /** Creates new form Datensatzzeiger */
    public Datensatzzeiger() {
        initComponents();
        setVisible(true);
    }


    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        String query = jTextArea1.getText();
        query.replaceAll(„\n", „ „);
        query.replaceAll(„\r", „ „);
        //Abfragen(query);
    }

    public void Abfragen(String was) {
        Connection conn = Verbindung.driver();
        Statement stm = Kommunikation.stm(conn);
        ResultSet rs = Kommunikation.rs(was, stm);
        tabellen(rs);
    }

    private void tabellen(ResultSet rs) {
        DefaultTableModel dftm = new DefaultTableModel();
        String a, b, c, d;
        a = „Beitragsgruppe";
        b = „Art";
        c = „Dienstnehmeranteil";
        d = „Dienstgeberanteil";

        dftm.addColumn(a);
        dftm.addColumn(b);
        dftm.addColumn(c);
        dftm.addColumn(d);

        Vector v = new Vector();
        v.add(a);
        v.add(b);
        v.add(c);
        v.add(d);
        dftm.setColumnIdentifiers(v);

        try {
            while (rs.next()) {
                Vector vec = new Vector();
```

```
                vec.add(rs.getString(1));
                vec.add(rs.getString(2));
                vec.add(rs.getString(3));
                vec.add(rs.getString(4));
                dftm.addRow(vec);
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        JTable tab = new JTable(dftm);
        jScrollPane1.setViewportView(tab);

    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextArea jTextArea1;
    // End of variables declaration
}
```

Das Ergebnis



Reflection

Mit Reflection ist es möglich, während der Laufzeit Informationen über die Objekte zu erhalten. Die Analogie in C++ dazu ist die Runtime - Typeinformation, die im Prinzip dazu dient zur Laufzeit den Typ eines Objektes zu bestimmen. Das Java Reflection API geht darüber erheblich hinaus. Was kann man nun mit dem Reflection-API erreichen? Man kann auf Datenfelder und Methoden zugreifen, deren Name erst zur Laufzeit bekannt ist, ein Array erzeugen, das erst zur Laufzeit einen bestimmten Typ hat. Die Generics gehen genau den umgekehrten Weg, da wird genau festgelegt, welcher Typ zulässig ist und damit natürlich auch die Sicherheit erhöht. Mit dem Reflection API kann der Grad der Abstraktion erhöht werden und es können Informationen über das Verhalten zur Laufzeit abgefragt werden. Es ist möglich, Informationen über Gültigkeitsbereiche, Felder, Methoden, Konstruktoren, Basisklassen und implementierte Interfaces einer Klasse zu erhalten, damit kann auch die Instanz einer Klasse erzeugt werden, deren Name erst zur Laufzeit bekannt ist. All diese Möglichkeiten machen das Reflection API sehr mächtig, man erkauft sich damit jedoch ein Zeitproblem, das API ist langsamer, dafür hat man die Möglichkeit viel abstrakter zu programmieren, was aber durch die Gewinnung der vielen Informationen, meiner Meinung nach, mehr als aufgehoben wird.

Es kann vorkommen, dass ein Objekt unbestimmt ist und man aus irgendeinem Grund an den Klassennamen dieses Objektes kommen will. Die Auswahl könnte zum Beispiel in einem Fenster erfolgen und man möchte dies dem Benutzer zur Verfügung stellen. Es ist nun so, dass zum Zeitpunkt der Deklaration jede Klasse einen Namen besitzt. Der Name für die folgende Klasse wäre Boersenkurse.

```
public class Boersenkurse {
	String aktie;
	...
}
```

Wenn man nun zur Laufzeit den Namen in Erfahrung bringen will, gibt es die Methode getName() in der Klasse java.lang.Class. Die Nutzung zeigt das folgende Beispiel:

```
import java.lang.reflect.*;
import javax.swing.*;
public class SampleName {
	...
	static void printName(Object o) {
		Class theClass = o.getClass();
		String str = theClass.getName();
		System.out.println(str);
	}
}
```

Serialisierung - Persistenz - XML

Hat man einmal Objekte, die man auch behalten möchte, so kann man diese persistent machen, dies bedeutet dauerhaft abspeichern. Der Vorgang dafür nennt sich Serialisierung. Wird ein Objekt erst einmal serialisiert, so nennt man es danach ein „persistentes Objekt“. Es gibt viele Möglichkeiten der Serialisierung, ein Standard hat sich jedoch herauskristallisiert, nämlich XML.

Die meisten Daten werden zurzeit in relationalen Datenbanken verwaltet. Einige dieser Datenbanken unterstützen XML in unterschiedlicher Form: Zum einen bei der tatsächlichen Speicherung, zum anderen nur in Form von Abfrageschnittstellen. In den meisten Fällen muss man sich diese Abfrageschnittstellen jedoch selber programmieren. Die Herausforderung besteht darin, dass Daten in einer oder mehreren relationalen Datenbanken vorhanden sind und man diese in XML zur Verfügung stellen will.

Das folgende Beispiel stellt eine einheitliche und generische http-Schnittstelle unabhängig von Ort und Struktur der relationalen Datenbank zur Verfügung. Die Schnittstelle kann auf beliebige Datenbanken zugreifen, solange diese JDBC unterstützen und ein entsprechender Treiber im Klassenpfad zu finden ist. Damit man nicht für unterschiedliche Sichten auf die Daten im Programmcode etwas ändern muss, verfolge einen generischen Ansatz, der auf den ersten Blick nicht ganz trivial, aber im Endeffekt äußerst effizient ist. XML das empfangen wird, kann man in einem zusätzlichen Verarbeitungsschritt dann noch immer von der generischen in dieeventuell gewünschte Form transferieren. Diese generische XML Lösung ist möglich, da bei noch so komplizierten SQL - Abfragen das Ergebnis immer zweidimensional ist. Dies bedeutet nichts anderes, als dass es einfach Zeilen und Spalten gibt.

Im folgenden Beispiel zeige ich, wie man dieses zweidimensionale Ergebnis über generische Regeln in einen XML Datenstrom verwandelt.
Folgende Tabelle sei in unserer Datenbank:

LieferantID	*Firma*	*Telefon*
1	Speedy Express	+43 3423 876
2	United Package	+32 3498 234
3	Federal Shipping	+43 9852 888

Generieren eines XML Dokuments aus einer Datenbank und über http zur Verfügung stellen

Mein Ziel ist es, mit dieser Lektion eine der größten Stärken und Einsatzbereiche von Java hervorzuheben, nämlich der serverseitigen Einsatz als Middleware. Ich musste einmal etwas Ähnliches für eine Firma implementieren und es genügt vollkommen, das Beispiel nachzuvollziehen und einen Überblick zu gewinnen.

Nun interessiert uns das folgende SQL Statement:

```
select * from Lieferanten
```

Diese Abfrage würde folgenden XML - Datenstrom ergeben. Die Kommentare wären darin natürlich nicht enthalten und sind nur als Erklärung der Transformationsregeln gedacht.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--
als Root-Element wird ein Element namens „ResultSet" instanziert.
Es hat das Attribut „onSql", welches das SQL enthält, das angefragt wurde.
-->

<ResultSet onSql="select * from Lieferanten">

<!--
für jede Zeile im ResultSet wird ein Element namens „RowSet" an das Root-
Element gehängt.
-->

<RowSet>

<!--
Innerhalb des RowSet Elements wird für jede Spalte der Ergebnismenge ein
Element instanziert, das den Namen der Spalte bekommt. Als Attribute wer-
den diverse Metainformationen wie Java-Typ, SQL Typ, Länge und Tabellenna-
men vergeben. Innerhalb des Elements taucht dann der Wert aus der Daten-
bank auf.
-->

<LieferantID java-type="java.lang.Integer" length="40"
                table="Lieferanten" type="Counter">
1
</LieferatID>
```

```
<Firma java-type="java.lang.String" length="40"
                table="Lieferanten" type="VARCHAR">
Speedy Express
</Firma>
</RowSet>
<RowSet>
<LieferantID java-type="java.lang.Integer" length="40"
                table="Lieferanten" type="Counter">
2
</LieferatID>
<Firma java-type="java.lang.String" length="40"
                table="Lieferanten" type="VARCHAR">
United Package
</Firma>
</RowSet>
<RowSet>
<LieferantID java-type="java.lang.Integer" length="40"
                table="Lieferanten" type="Counter">
3
</LieferatID>
<Firma java-type="java.lang.String" length="40"
                table="Lieferanten" type="VARCHAR">
Federal Shipping
</Firma>
</RowSet>
</ResultSet>
```

Zur Erstellung dieses XML-Datenstroms benutze ich hauptsächlich das W3C Document Object Model (DOM), welches genügend Interfaces zur Erzeugung von Elementen und Attributen und auch Möglichkeiten zum Zusammenfügen erstellter Knoten zur Verfügung stellt.
Beim Programmcode handelt es sich um ein Servlet, in dem die doGet Methode implementiert ist. Servlets sind serverseitige Java Programme, eine der größten Stärken der Programmiersprache. Der Code kann auf generische Art und Weise SQL Datenbankabfragen in XML Datenströme verwandeln. Dabei werden Metainformationen wie Feld und Tabellennamen und Typen mitgeliefert. Es wird ein http Parameter namens sql, mit der gewünschten SQL Anfrage benötigt. Die Parameter driver, url, user und pwd sind optional und können die Datenbankverbindung beschreiben, auf die das SQL abgesetzt werden soll. Zu beachten ist, dass dabei die entsprechende Treiberklasse im Klassenpfad der Servlet-Engine zu finden ist.

Das Servlet funktioniert nur, solange Spaltennamen nach well-formed XML benannt sind. Dies kann bei Bedarf sehr leicht geändert werden, indem der Spaltenname nicht über den Elementnamen, sondern über ein zusätzliches Attribut modelliert wird. Der Elementname könnte dann generisch zum Beispiel „column“ genannt werden. Für XML gibt es zwei wesentliche Qualitäts- und Funktionalitätskriterien, nämlich „valid“ und „well-formed“. Um „valid“ zu sein muss jedes Anfangs-Tag auch ein End-Tag besitzen und um „well-formed“ zu sein, muss die Datenkonsistenz stimmen.

Um die benötigten Bibliotheken verwenden zu können, habe ich die Xerces API-Library von der Website http://www.apache.org/dist/xerces/j/ heruntergeladen. In diesem jar File befinden sich auch die zwei Apache XML Bibliotheken, nämlich org.apache.xml.serialize.Outputformat und org.apache.xml.serialize.XMLSerializer. Mit einem rechten Mausklick auf die Libraries kann mit addJarFolder die Xerces Bibliothek eingebunden werden. Diese Vorgangsweise gilt auch für eigene Entwicklungen, die man weiterverwenden möchte. Diese Bibliotheksverwendungen zählen zu den größten Vorzügen der objektorientierten Programmierung und richtig eingesetzt kann damit die Entwicklungszeit drastisch verkürzt werden.

Nun legen wir eine Klasse mit dem Namen RDB2XMLConverter an. Als package habe ich stue.xml gewählt.

```
package stue.xml;

/**
 *
 * @author Stuhlpfarrer
 */
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.w3c.dom.*;
// das sind Xerces - spezifische Klassen, die benötigt werden, da
// deren Funktionsumfang vom W3C noch nicht spezifiziert ist.
import org.apache.xerces.dom.DocumentImpl;
import org.apache.xml.serialize.XMLSerializer;
import org.apache.xml.serialize.OutputFormat;
```

```
public class RDB2XMLConverter extends HttpServlet {

    private static final String CONTENT_TYPE = „text/xml“;
    private Connection connection = null;
    private String defaultDriver;
    private String defaultUrl;
    private String defaultUser;
    private String defaultPwd;

    public void init(ServletConfig config) {
        defaultDriver = config.getInitParameter(„defaultDriver“);
        defaultUrl = config.getInitParameter(„defaultUrl“);
        defaultUser = config.getInitParameter(„defaultUser“);
        defaultPwd = config.getInitParameter(„defaultPwd“);
    }

    // die doGet() Methode
    public void doGet(HttpServletRequest request, HttpServletResponse res-
ponse) throws
            ServletException, IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        String sql = request.getParameter(„sql“);
        // falls kein http-Parameter namens ‚sql‘ übergeben
        if (sql == null || sql.length() == 0) {
            out.println(„<message code=\“-1\“>please provide „
                    + „http-get-parameter named \‘sql\‘</message>“);
            return;
        }

        try {
            // eine Datenbankverbindung wird aufgebaut
            Connection con = getConnection(request);
            Document doc = null;

            // da kein Connectionpool verwendet wird, sollte
            // sichergestellt sein, das nicht mehr als ein Thread die
            // Connection verwendet
            synchronized (con) {
                doc = createDocument(con, sql);
            }
            // Auf Basis des Document-Objects wird ein OutputFormat
            // Object erzeugt. Hierbei muss das richtige Encoding
            // gewählt werden. Der letzte boolsche Wert im Constructor
            // gibt an, ob das XML eingerückt ausgegeben werden soll
            // oder nicht
            OutputFormat format = new OutputFormat(doc, „ISO-8859-1“,
true);
```

```
            // Es wird ein XMLSerializer auf Basis des Ausgabestroms
            // zum Client und des OutputFormat Objects instanziert
            XMLSerializer serial = new XMLSerializer(out, format);

            // Nun kann das Dokument zum Client geschrieben werden
            serial.serialize(doc);
            out.flush();
            out.close();
        } catch (Exception e) {
            // Im Ausnahmefall wird eine Fehlermeldung zurückgegeben
            out.println(„<message code=\"-1\">" + e + „</message>");
        }

    }
    // Diese Methode liefert auf Basis der Parameter, die in dem
    // HttpServletRequest-Object gekapselt sind, eine neue oder
    // die schon bestehende Datenbankverbindung

    private Connection getConnection(HttpServletRequest request)
            throws Exception {
        String driver = null, url = null, user = null, pwd = null;
        Connection con = null;

        // wenn der Parameter ‚url' übergeben wurde, soll das SQL
        // auf einer anderen als der Default-Datenbank ausgeführt
        // werden. Also werden die anderen Parameter auch noch ausgelesen
        if ((url = request.getParameter(„url")) != null && url.length() !=
0) {
            driver = request.getParameter(„driver");
            user = request.getParameter(„user");
            pwd = request.getParameter(„pwd");

            // Eine neue Verbindung wird mit den entsprechenden
            // Parameteren geholt
            return getConnection(driver, url, user, pwd);
        } else {
            // Ansonsten wird eine Datenbankverbindung mit den
            // Default - Werten erzeugt, oder die schon vorhandene
            // Verbindung zurückgegeben
            if (connection == null) {
                connection = getConnection(defaultDriver, defaultUrl, de-
faultUser, defaultPwd);
            }
            return connection;
        }

    }
```

```
    // Erzeugung einer neuen Datenbankverbindung
    private Connection getConnection(String driver, String url, String
user,
            String pwd) throws Exception {
        Class.forName(driver);
        Connection con = DriverManager.getConnection(url, user, pwd);
        return con;
    }

    // Ausführen der Datenbankabfrage und Erstellen des XML
    // Dies geschieht immer nach dem gleichen Schema
    private Document createDocument(Connection con, String sql)
            throws Exception {

        // Statement Objekt zum Absetzen der Anfrage
        Statement stmt = con.createStatement();

        // Ausführen der Anfrage
        ResultSet rs = stmt.executeQuery(sql);

        // Erzeugen eines ResultSetMetaData Objects
        ResultSetMetaData rsmd = rs.getMetaData();

        // Als Rückgabewert wird ein neues DocumentImp Object benötigt
        // welches das W3C Document Interface implementiert. Der W3C
        // Standard definiert nicht, wie man Objekte erzeugen soll, die das
        // Document Interface implementieren. Deswegen benutze ich hier die
        // Xerces spezifischen Objekte. Die folgende Zeile müsste also
        // ersetzt werden, sollte man sich in Zukunft für einen anderen
        // Parser entscheiden.
        Document doc = new DocumentImpl();

        // Als Root-Element wird eine Element namens ResultSet erzeugt.
        Element root = doc.createElement(„ResultSet“);

        // Ein Kommentar soll eingefügt werden
        Comment comment = doc.createComment(„Das ResultSet Element“
                + „ kapselt das Resultat der SQL-Abfrage.“);
        // Der Kommentar und das Root-Element müssen an das Document
        // Object angehängt werden
        doc.appendChild(comment);
        doc.appendChild(root);

        // Es wird ein Attr-Object erzeugt, welches ein Attribut
        // namens ‚onSql‘ repräsentiert
        Attr sqlAttr = doc.createAttribute(„onSql“);

        // Der Wert dieses Atteibutes wird mit dem SQL belegt, das durch
```

```
        // das XML Dokument beantwortet werden soll.
        sqlAttr.setNodeValue(sql);

        // Nun muss das Attr-Object noch an das Root-Element angehängt wer-
den.
        root.setAttributeNode(sqlAttr);

        // In den folgenden Schleifen wird die Anzahl der Spalten
        // benötigt, die das Ergebnis der Anfrage hat.
        int columnCount = rsmd.getColumnCount();

        while (rs.next()) {

            // Für jeden Datensatz des Ergebnisses wird ein Element
            // namens ‚RowSet' erzeugt.
            Element row = doc.createElement(„RowSet");

            for (int i = 1; i < columnCount; i++) {
                // Für jeden Wert in jedem der Datensätze der Ergebnismeng
                // soll ein Element mit dem Namen der betreffenden Spalte
                // erzeugt werden
                Element column = doc.createElement(rsmd.getColumnName(i));

                Object object = rs.getObject(i);
                Text textNode = null;

                // Falls der Wert null war, wird an das Element der Text
                // ‚null' angehängt
                if (object == null) {
                    textNode = doc.createTextNode(„null");
                } // Ansonsten werden verschiedene Metainformationen als
                // Attribut gesetzt. Die hier angewandte Methode zum
                // Setzen von Attributen ist wesentlich komfortabler
                // als die oben angewandte.
                else {

                    column.setAttribute(„type", rsmd.getColumnTypeName(i));
                    column.setAttribute(„length", new Integer(rsmd.
getPrecision(i)).toString());
                    column.setAttribute(„java-type", object.getClass().get-
Name());

                    // Für den eigentlichen Wert muss nun ein Text - Knoten
                    // erstellt werden
                    textNode = doc.createTextNode(object.toString());
                }
                // Der Text-Knoten muss als Unterknoten an das Column
                // Element angehängt werden
```

```
                column.appendChild(textNode);

                // Das Column-Element muss wiederum an das RowSet-Element
                // angehängt werden.
                row.appendChild(column);
            }
            // Jedes entstandene RowSet-Element muss an das Root-Element
            // angehängt werden.
            root.appendChild(row);
        }

        // Das zusammengesetzte Dokument wird zurückgegeben.
        return doc;
    }
}
```

Damit wir das Beispiel testen können, brauchen wir eine Datenbank. Für dieses Beispiel verwende ich die zu Beginn der Lektion angelegte MySql Datenbank „Lieferanten“.

Eine Tomcat Installation ist Vorraussetzung für das Funktionieren von Servlets.
Im Verzeichnis WEB-INF muss eine XML - Datei namens web.xml liegen, welche die JDBC Treiber beinhaltet und den Zugang für die Datenbank.

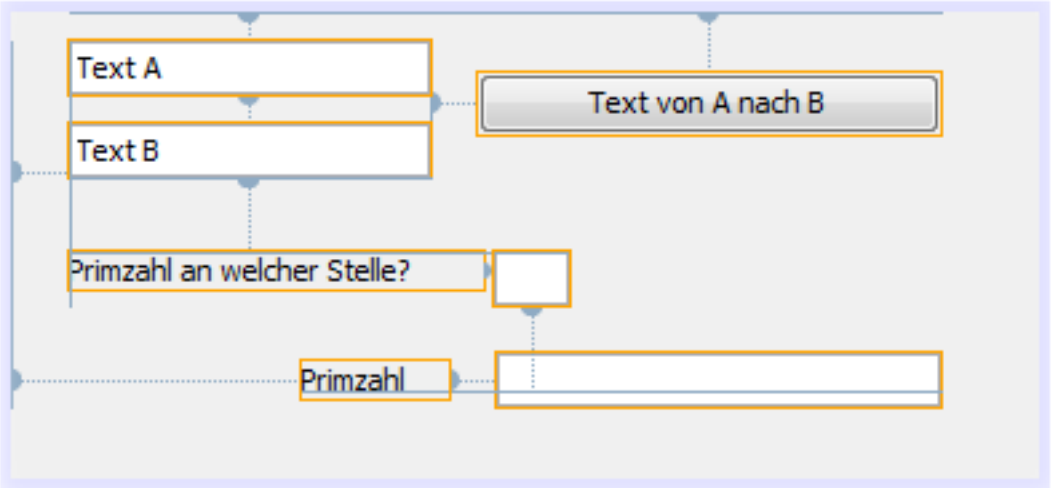
Das Ergebnis ist am Anfang der Lektion beschrieben. Man beachte, dass der XML Parser für jegliches „Select“ Statement gilt, ohne dass die Feldnamen der Datenbank bekannt sein müssen. Man nennt dies einen „generischen Ansatz“.

3.4.16 Lektion 15 - GUI mit Primzahlensuche in eigenem Thread

Die Programmierung von Threads erfordert viel Interaktion zwischen unterschiedlichen Objekten. Ein Praxisbeispiel soll das Ganze klarer werden lassen. Es geht in dem folgenden Beispiel um eine Klasse, die eine spezielle Primzahl in einer Sequenz findet. Es wird der Einsatz eines Threads demonstriert und es könnte auch jegliche andere Lösung bei der Threads erforderlich sind, damit implementiert werden. In einem einfachen GUI soll die gewünschte Primzahl eingegeben werden, zum Beispiel die tausendste und der Thread soll mit einem „Berechne“ Button gestartet werden und nach einiger Zeit, je nach dem, wie lange die Berechnung dauert, soll das Ergebnis in einem Textfeld angezeigt werden. Um zu zeigen, dass es sich wirklich um einen Thread handelt sollen nebenbei zwei Textfelder ihren Inhalt auf Knopfdruck austauschen und es sollte dabei zu keiner Blockade kommen.

Entwurf der Benutzeroberfläche

GUI Vorschlag



Code für actionPerformed „Text von A nach B“

```
// Button zum Kopieren von A nach B
    private void jButton1ActionPerformed(java.awt.event.ActionE-
vent evt) {
        this.jTextField2.setText(this.jTextField1.getText());
    }
```

Nun wird die Klasse „PrimzahlenFinder“ implementiert. Diese soll eine spezielle Primzahl in einer Sequenz finden. Dies kann ein weilchen dauern, besonders bei Primzahlen über 100.000, sodass die Primzahlensuche in einem eigenen Thread durchgeführt wird.

```
package stue.threads;

/**
 *
 * @author Stuhlpfarrer
 */
import java.lang.*;

public class PrimzahlenFinder implements Runnable {
```

```
    public long ziel;
    public long primzahl;
    public boolean beendet = false;
    private Thread runner;

    PrimzahlenFinder(long inZiel) {
        ziel = inZiel;
        if (runner == null) {
            runner = new Thread(this);
            runner.start();
        }
    }

    public void run() {
        long numPrimes = 0;
        long candidate = 2;
        while (numPrimes < ziel) {
            if (isPrime(candidate)) {
                numPrimes++;
                primzahl = candidate;
            }
            candidate++;
        }
        beendet = true;
    }

    boolean isPrime(long checkNumber) {
        double root = Math.sqrt(checkNumber);
        for (int i = 2; i <= root; i++) {
            if (checkNumber % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

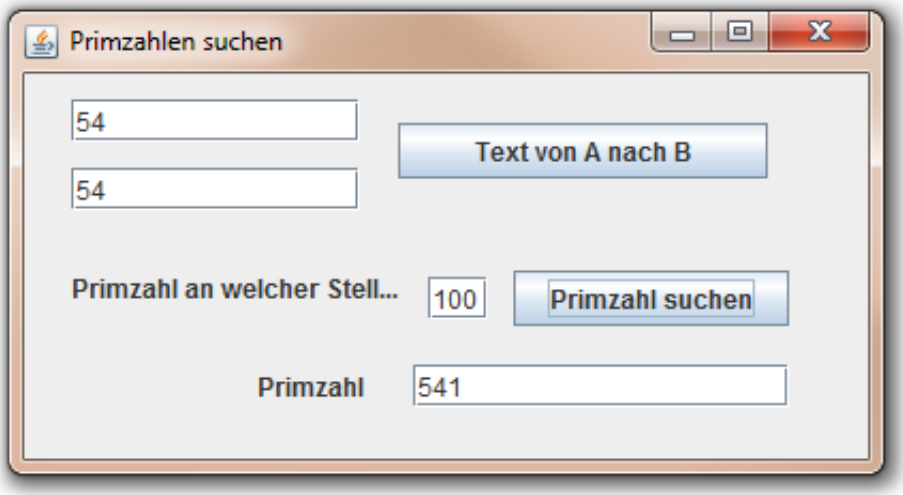
Die Klasse hat keine main-Methode und kann daher nicht als Applikation ausgeführt werden. Als nächstes folgt eine Klasse, welche den Thread verwendet. Die Klasse Primzahlenfinder implementiert die Schnittstelle Runnable und kann also als Thread ausgeführt werden.
Es gibt drei Instanzvariablen:
ziel - long gibt den Wert an, wann die spezifische Primzahl in der Sequenz gefunden wurde. Wenn man zum Beispiel die 5000ste Primzahl sucht, ist „ziel“ gleich 5000
„primzahl - long“ ist der Wert für die letzte Primzahl, den die Klasse gefunden hat.

„beendet“ - ist ein boolscher Wert, der angibt, ob das Ziel erreicht wurde.

Außerdem gibt es eine Instanzvariable „runner“, die das Thread-Object beinhaltet, in dem die Klasse laufen wird. Dieses Object sollte gleich null sein, solange der Thread nicht gestartet wurde.

Der Konstruktor startet den Thread, falls dieser noch nicht läuft. Sobald die start() Methode des Threads aufgerufen wird, ruft diese wiederum die run() Methode der Klasse mit dem Thread auf.
Die run() Methode erledigt den größten Teile der Arbeit des Threads, was typisch für eine nebenläufige Klasse ist. Man legt die rechenintensivsten Aufgaben in einen Thread, damit diese den Rest des Programmes nicht ausbremsen. Die Methode verwendet zwei neue Variablen, nämlich „numPrimes“ für die gefundenen Primzahlen und „candidate“ für die Zahl, die möglicherweise eine Primzahl ist. Diese Variable fängt bei der ersten möglichen Primzahl „Zwei“ an.
Als erstes wird überprüft, ob der aktuelle Kandidat eine Primzahl ist, was durch den Aufruf der isPrime() Methode geschieht, die true zurückgibt, wenn die Zahl eine Primzahl ist und ansonsten false.
Wenn „candidate“ eine Primzahl ist, dann wird numPrimes um eins erhöht und die Instanzvariable „primzahl“ wird auf diese Zahl gesetzt. Danach wird die Variable „candidate“ um eins inkrementiert und die Schleife läuft weiter. Sobald die richtige Anzahl an Primzahlen gefunden worden ist, endet die while-Schleife und die Instanzvariable „beendet“ wird auf true gesetzt. Dies bedeutet, dass das „PrimzahlenFinder“ - Objekt die richtige Primzahl gefunden hat und damit die Suche abgeschlossen ist. Am Ende der run() Methode hat dann der Thread seine Arbeit vollbracht.
Die isPrime() Methode entscheidet, ob eine Zahl eine Primzahl ist, indem sie den Operator % (Modulo gibt den Rest zurück) verwendet. Wenn eine Zahl durch 2, oder durch eine andere höhere Zahl teilbar ist (d.h. wenn der Rest 0 ist), dann ist sie keine Primzahl.
Nun folgt eine Klasse zum Testen, die eine Konsolenausgabe generiert. Im zweiten Teil wird dann die Benutzeroberfläche berücksichtigt.

Das Ergebnis



Der Code für den Button Primzahl suchen:

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent
evt) {

        String countStr = this.jTextFieldStelle.getText();
        long countLo = Long.parseLong(countStr);
        PrimzahlenFinder finder = new PrimzahlenFinder(countLo);
        System.out.println(„Suche nach Primzahl: „ + countLo);

        try {
            Thread.sleep(1000);
        } catch (InterruptedException ie) {
            System.out.println(„Ich tue nichts!“);
        }

        long primLo = finder.primzahl;
        String primStr = String.valueOf(primLo);
        this.jTextFieldErgebnis.setText(primStr);

        System.out.println(finder.primzahl);
    }
```

3.4.17 Lektion 16 - Tabelle anlegen mit Swing (Beispieldaten: Ansprüche des Dienstnehmers im Krankheitsfall)

Ziel und Erklärung

Eine Tabelle ist eine der komplexesten Swing-Komponenten. Sie beinhaltet eine interne Aufteilung zwischen der Darstellung und dem Inhalt. Durch diese Trennung können sehr elegant dynamische Inhalte dargestellt werden. Den Umgang mit diesen beiden Aufteilungen werde ich in der nächsten Lektion behandeln. Die Aufgabe ist nicht ganz trivial. Zum Glück existiert aber auch eine intuitivere Vorgehensweise, wie eine Tabelle erstellt werden kann. Hierzu legt man die Inhalte in einem 2D-Array an und übergibt die beiden Arrays dem Konstruktor der jTable Klasse.
Legt man die Tabelle nun noch in den ViewPort einer ScrollPane, werden sowohl Überschriften als auch die Inhalte sichtbar. Würde man die Tabelle direkt auf den ContentPane legen, könnte man nur den Inhalt sehen.
Anlegen der Tabelle über die Properties:

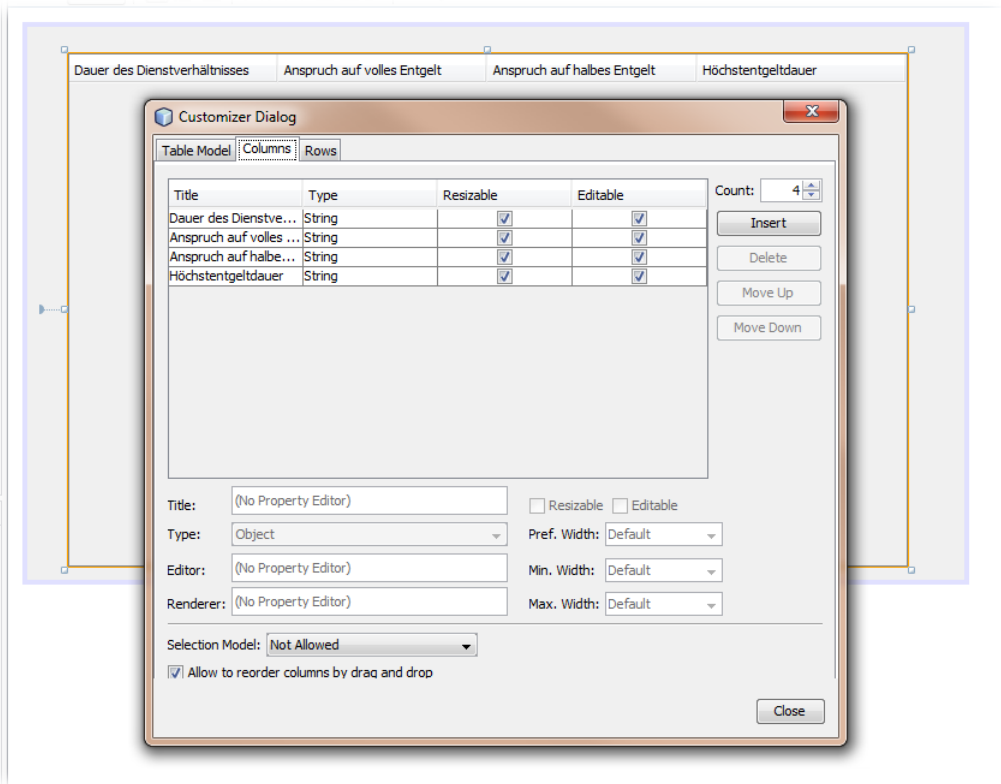
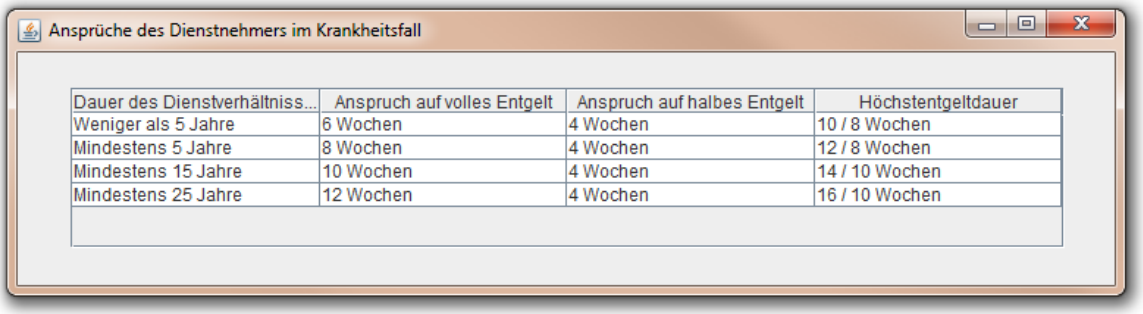


Abb.: 22 Ansprüche des Dienstnehmers im Krankheitsfall

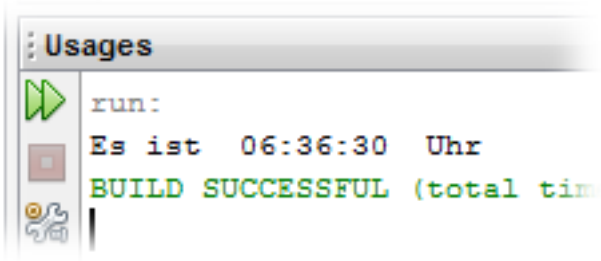


Das Tabellenmodell:

```
setTitle(„Ansprüche des Dienstnehmers im Krankheitsfall“);

        jTable1.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {
                {„Weniger als 5 Jahre“, „6 Wochen“, „4 Wochen“, „10 / 8 Wo-
chen“},
                {„Mindestens 5 Jahre“, „8 Wochen“, „4 Wochen“, „12 / 8 Wo-
chen“},
                {„Mindestens 15 Jahre“, „10 Wochen“, „4 Wochen“, „14 / 10
Wochen“},
                {„Mindestens 25 Jahre“, „12 Wochen“, „4 Wochen“, „16 / 10
Wochen“}
            },
            new String [] {
                „Dauer des Dienstverhältnisses“, „Anspruch auf volles Ent-
gelt“, „Anspruch auf halbes Entgelt“, „Höchstentgeltdauer“
            }
        ) {
            Class[] types = new Class [] {
                java.lang.String.class, java.lang.String.class, java.lang.
String.class, java.lang.String.class
            };

            public Class getColumnClass(int columnIndex) {
                return types [columnIndex];
            }
        });
        jScrollPane1.setViewportView(jTable1);
```

3.4.18 Lektion 17 - Laufende Uhr in einem Frame anzeigen

Es kommt wirklich oft vor, dass man die aktuelle Uhrzeit irgendwo im Fenster anzeigen lassen will. Im nächsten Beispiel zeige ich einen eleganten objektorientierten Weg, wie diese Uhr-Funktion in eine spezielle Klasse ausgelagert werden kann. Für die Uhranzeige wird ein neuer Thread gestartet, damit nicht die Funktionsweise der bestehenden Anwendung beeinträchtigt wird.
Da die ausgelagerte Klasse neben der Runnable-Funktionalität auch noch ein Label ist, also von der Klasse Label erbt, lässt sich diese Komponente sehr leicht über add() in eine bestehende GUI einbauen.



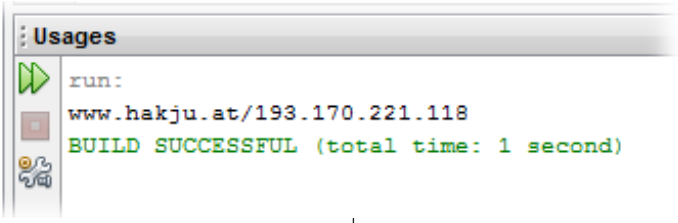
Die Klasse Zeitgeber:

```
package stue.timelabel;

/**
 * Die Klasse Zeitgeber zeigt fortlaufend
 * die Uhrzeit an
 * @author Stuhlpfarrer
 */
import java.util.Date;
import java.awt.Label;
import java.text.SimpleDateFormat;

public class Zeitgeber extends Label implements Runnable {
    // Festlegen des Datumsformates

    private SimpleDateFormat sdf = new SimpleDateFormat(
            „'Es ist , hh:mm:ss , Uhr'");
    private String dateString;
    private Date d = new Date();
```

```
    public Zeitgeber() {
        setAlignment(Label.CENTER);
        setText(getDateString());
        System.out.println(getDateString().toString());
    }

    // liefert das formatierte Datum
    public String getDateString() {
        d.setTime(System.currentTimeMillis());
        dateString = sdf.format(d);
        return dateString;
    }
    // die run() Methode wird ausgeführt, wenn
    // der Thread gestartet wurde

    public void run() {
        // Endlos - Schleife
        while (true) {
            // Dem Label wird die Beschriftung mit dem aktuellen
            // Wert zugewiesen
            this.setText(getDateString());
            // da nur die Sekunden angezeigt werden, kann sich
            // der Thread ohne Auswirkungen für eine knappe
            // Sekunde schlafen legen
            try {
                Thread.currentThread().sleep(995);
            } catch (InterruptedException ie) {
                ie.printStackTrace();
            }
        }
    }
}
```

3.4.19 Lektion 18 - Zu einer URL gehörige IP-Adresse ermitteln

Es kommt in der Praxis öfter vor als man glaubt, dass man zu einer URL die dazugehörige IP-Adresse ermitteln muss, sei es, weil ein DNS Service ausgefallen ist, oder aus performanten Gründen. Um nun die IP-Adresse einer bekannten URL herauszufinden, muss zuerst ein „InetAdress“- Objekt über die Methode getByName() generiert werden. Der Methode wird die bekannte URL als Zeichekette übergeben.



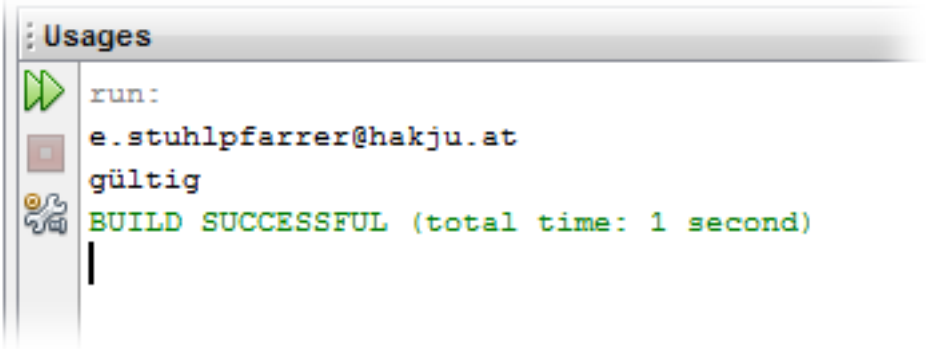
```
package stue.internet;

/**
 *
 * @author Stuhlpfarrer
 * IP Adresse zu einer URL wird auf die Konsole geschrieben
 */
import java.net.*;

public class Url2Ip {

    public static void main(String[] args) throws Exception {
        InetAddress addr1 = InetAddress.getByName(„www.hakju.
at“);
        System.out.println(addr1);
    }
}
```

3.4.20 Lektion 19 - Überprüfung einer eMail - Adresse

Bei geschäftlichen Interaktionen im Web, kann es schon vorkommen, dass eine automatische Überprüfung der Korrektheit einer eMail - Adresse erfolgen muss.
Der Aufbau einer Mail - Adresse wird durch die RFC822 definiert[1]. Demnach besteht eine Mail immer aus drei Teilen, dem Namen des Benutzers, dem @ Zeichen und einer IP - Adresse, bzw. einem Rechnernamen oder einer Domain, bei der das Postfach des Benutzers liegt. Für den Namensteil dürfen die 26 Buchstaben des Alphabetes, Zahlen, sowie Punkte (.) und Unterstriche bzw. Trennstriche (_ bzw. -) verwendet werden. Außerdem muss der Namensteil mindestens zwei Buchstaben enthalten.
Möchte man nun eine eMail Adresse auf Korrektheit überprüfen, muss entsprechend geprüft werden, ob die einzelnen Teile einer Mailadresse korrekt sind. Mit Hilfe der folgenden Klasse ist es möglich, eine eMail Adresse, die dem Programm als Parameter mitgegeben wird zu überprüfen. Um den regulären Ausdruck nicht zu komplex werden zu lassen, habe ich darauf verzichtet, auf die in der Praxis selten genutzte Möglichkeit eine IP Adresse zu verwenden, verzichtet,

1 www.ietf.org/rfc/rfc822.txt

```
package stue.internet;

/**
 *
 * @author Stuhlpfarrer
 */
import java.util.regex.*;

public class EmaiAdressePruefen {

    public static void main(String[] args) {

        String eMail = „e.stuhlpfarrer@hakju.at“;

        String pattern = „([a-zA-Z0-9_\\-\\.]+)“ + // Benutzer
                „@“ + // @ Zeichen
                „([a-zA-Z0-9_\\-\\.]{2,})“ + //Domain - Subdomain
                „\\.“ + // Punkt
                „([a-zA-Z]{2,5})“;                // TLD
        System.out.println(eMail);
        if (Pattern.matches(pattern, eMail)) {
            System.out.println(„gültig“);
        } else {
            System.out.println(„ungültig“);
        }
    }
}
```



3.4.21 Lektion 20 - Überprüfung einer Kreditkartennummer

Ziel und Erklärung

In automatisierten Webapplikationen mit integriertem Zahlungsverkehr ist es ein wichtiges Kriterium, Kreditkartennummern auf die Richtigkeit zu überprüfen.
Kreditkarten bestehen zumeist aus 13 - 16 Ziffern, die zu je 4 Ziffern in einem Block zusammengefasst sind. Die ersten Ziffern dienen dazu, eine Kartennummer einem Herausgeber zuordnen zu können. Ich habe die Formate für die großen vier Hersteller zusammengefasst.

Hersteller	Anfang	Gesamtlänge
Visa	4	13
Master	51, 52, 53, 54, 55	16
Diner‘s Club	30, 36, 38	14
American Express	34, 37	15

Abb.: 23 Kreditkartennummern

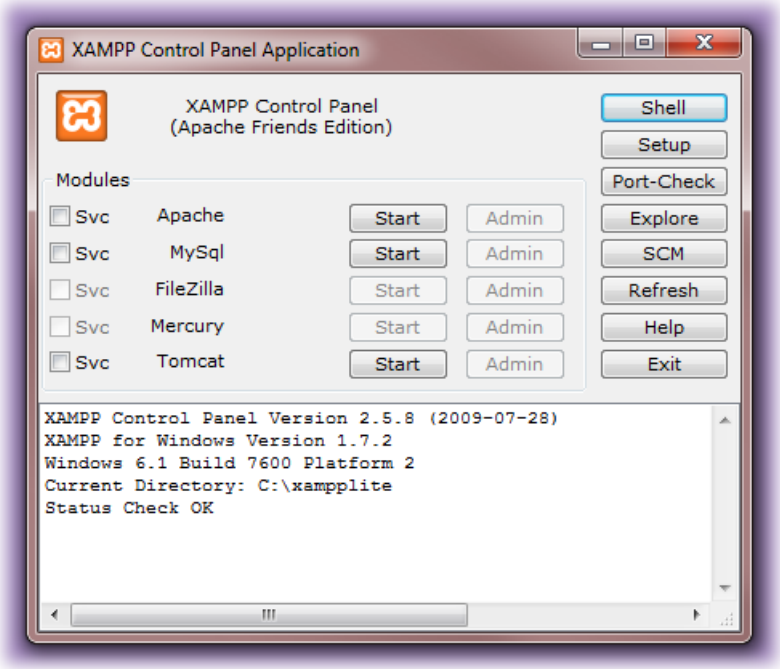
Die letzte Ziffer der Kartennummer ist oftmals eine Prüfsumme, die sich aus den anderen Ziffern nach einem Algorithmus bestimmen lässt. Das folgende Beispiel überprüft die Korrektheit einer Kreditkartennummer. Ich lasse hierbei allerdings die Überprüfung einer möglichen Prüfsumme außer Acht, es lassen sich solche Zahlen nämlich nicht innerhalb eines regulären Ausdruckes berechnen.

```
import java.util.regex.Pattern;

public class KreditKartenNummernTest {

    public static void main(String[] args){
        String KreditkartenNummer = „“;
               String pattern =
                „(4\\d{3}[- ?]\\d{4}-?\\d)“ +   // Visa
                „|“ +                           // oder
                „(5[1-5]\\d{2}[- ]\\d{4}[- ]?\\d{4})“ + // Master
                „|“ +                           // oder
                „(3[47]\\d{2}[- ]?\\d{4}[- ]?\\d{4}[- ]?\\d{2})“ + //
Diners
                „|“ +                           // oder
                „(3[47]\\d{2}[- ]?\\d{4}[- ]?\\d{4}[- ]?\\d{3})“; //
Amex

        if(Pattern.matches(pattern, KreditkartenNummer)){
            System.out.println(„gültig“);
        } else {
            System.out.println(„ungültig“);
        }
    }

}
```

3.4.22 Lektion 21 - Web Server - Servlets

Ziel und Erklärung

Als Servlets bezeichnet man Java Implementierungen, die auf einem Server laufen. Diese Kategorie von Programmen benötigt man für dynamische Websites.
Um ein Servlet zu starten, wird ein Servlet - Container benötigt. Dieser stellt eine definierte Ausführungsumgebung für Servlets und Java Server Pages zur Verfügung. Professionelle Server gibt es als Download bei Sun, als semiprofessionelle Trainingsumgebung eignet sich der Open Source Server XAMPP mit der Tomcat - Erweiterung. XAMPP bietet standardmäßig den Apache - Webserver und eine MySql - Datenbank an. Mit der Tomcat - Erweiterung, ebenfalls ein Open Source Projekt lassen sich auch dynamische Java Websites hosten.
Der derzeit aktuelle Downloadlink lautet: http://www.apachefriends.org/de/xampp.html. Auf diese Website ist auch das Tomcat - addOn zu finden. Hervorragende Dienste leistet auch der in der Netbeans IDE integrierte Glassfish - Server.



Nach der Installation stellt Tomcat ein Verzeichnis „webapps“ bereit, in dem die einzelnen Webapplikationen liegen. Eine Webapplikation besteht meist aus Servlets, html - Seiten, Grafiken und JSP‘s, sowie weiteren Dateien zum Beispiel Stylesheets. Um die problemlose Übertragbarkeit von einem Server zum anderen zu gewährleisten gibt es eine vorgegebene Verzeichnisstruktur. Das Verzeichnis WEB-INF enthält die Klassen und Konfigurationsdateien einer Web - Applikation. Die Datei web.xml enthält die Konfigurationsdaten, anhand der Server die einzelnen Servlets identifiziert. Im Verzeichnis classes werden die Klassendateien angelegt. Das Verzeichnis lib enthält apllikationsspezifische Java - Archiv - Dateien (jar).
Ein Servlet muss kompiliert werden, bevor es der Server ausführen kann. Manche Server erledigen das selbst, aber davon sollte nicht ausgegangen werden. Ein Servlet, das direkt im Verzeichnis classes angelegt wird, also ohne Package - Angabe, wird im Tomcat mit der URL http://localhost:8080/appname/servlet/Test aufgerufen. „appname“ steht für den Verzeichnisnamen der Applikation. Tomcat stellt einen vordefinierten Bereich „servlet“ zur Verfügung, über den Servlets aufgerufen werden können. Ist ein Servlet in einem Package untergebracht, so müssen bei obigem Aufruf alle Package - Angaben durch Punkte getrennt vor den Namen gestellt werden.

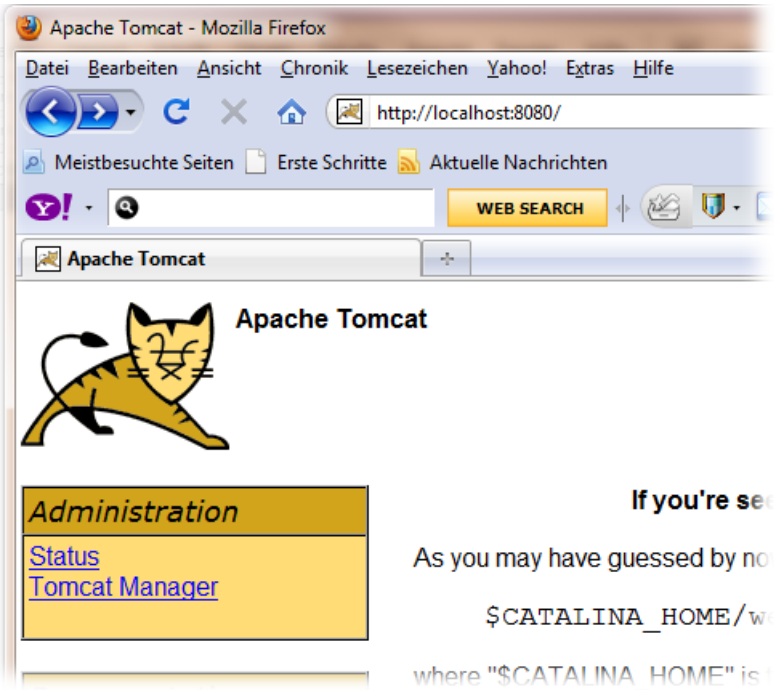
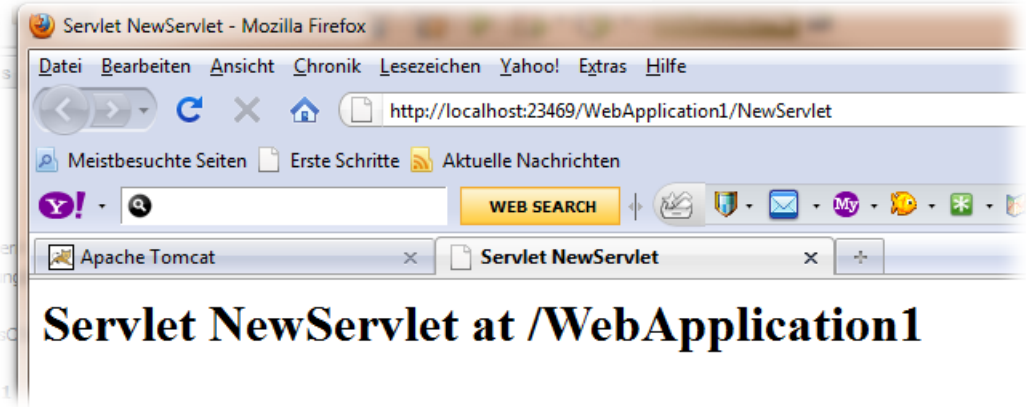


Abb.: 24 Erfolgreiche Tomcat - Installation

3.4.23 Lektion 22 - Der Einsatz von Servlets

Ziel und Erklärung

Ein Servlet ist normalerweise eine Unterklasse der abstrakten Klasse javax.servlet.HttpServlet. Sie definiert die grundlegenden Methoden, mit denen Aufrufe per HTTP Protokoll beantwortet werden. Die wichtigsten Methoden sind doGet() und doPost(), die bei den entsprechenden HTTP - Anfragetypen aufgerufen werden. Dabei ist die doGet() Methode dazu gedacht, Seiten aufzurufen, während die doPost() Methode dazu dient, Informationen an den Server zu schicken. Dies ist im Browser leicht zu erkennen, bei der get Methode sind alle Parameter nach dem Aufruf in der Adresszeile des Browsers zu sehen, bei der post Methode nicht.
Damit möchte ich gleich davor warnen, die get Methode nicht für den Versand von sensiblen Daten, wie zum Beispiel von Passwörtern, oder Benutzerdaten zu verwenden.
Beide Methoden erhalten als Parameter jeweils ein HttpServletRequest und ein HttpServletResponse - Objekt. Diese Klassen definieren die Schnittstelle zu einer HTTP Anfrage (HttpServletRequest) und der Antwort an den Browser (HttpServletResponse). Über das Request Objekt lassen sich Anfrageparameter ermitteln, der Ausgabekanal wird vom Response - Objekt bereitgestellt, über den eine Antwort an den Browser gesendet wird.
Ein Servlet, das keine Informationen auswertet, muss nur die doGet() Methode überschreiben. Dort wird eine html - Seite erzeugt und an den Browser geschickt. Die Klasse „EinfachesHtml“ erzeugt eine einfache html - Seite innerhalb der doGet() Methode.
Das Ergebnis des ersten Servlets:



Die wichtigsten Teile des Servlets:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author Stuhlpfarrer
 */
@WebServlet(name = „NewServlet“, urlPatterns = {„/NewServlet“})
public class NewServlet extends HttpServlet {

        protected void processRequest(HttpServletRequest request, HttpServ-
letResponse response)
            throws ServletException, IOException {
        response.setContentType(„text/html;charset=UTF-8“);
        PrintWriter out = response.getWriter();
        try {

            out.println(„<html>“);
            out.println(„<head>“);
            out.println(„<title>Servlet NewServlet</title>“);
            out.println(„</head>“);
            out.println(„<body>“);
            out.println(„<h1>Servlet NewServlet at „ + request.getContext-
Path() + „</h1>“);
            out.println(„</body>“);
            out.println(„</html>“);

        } finally {
            out.close();
        }
    }
```

3.4.24 Lektion 23 - J2EE - Java Enterprise Edition - Überblick

Eine EE5 Applikation besteht aus mehreren Komponenten, die in einem oder mehreren Containern verwaltet werden, die Architektur ist mehrschichtig und die Module lassen sich auf mehrere Systeme verteilen, was insgesamt eine hervorragende Skalierbarkeit zur Folge hat.
Die EE5 Applikation unterscheidet sich in einigen wesentlichen Punkten zu einer herkömmlichen Applikation. In der herkömmlichen Architektur wird der Programmierer dazu verleitet, ohne passende Architektur gleichzeitig an der Programmsteuerung und an der Fachebene zu arbeiten, bei der Java Enterprise Entwicklung werden mehrere Aufgaben mit klar umrissenen Rollen definiert und die vorgegebene Architektur ermöglicht eine bessere Konzentration auf Fachspezifika, während der Container die Steuerung des Programms übernimmt. Wenn man das erste Mal an einer Enterprise Entwicklung mitarbeitet, so wird auffallen, dass es keinen zentralen Einstiegspunkt in das Programm gibt, stattessen arbeitet eine Enterprise Applikation anfragebasiert, diese können von Webbrowsern kommen, indem der Benutzer auf bestimmte Links klickt, oder von „native Clients“ die dem Benutzer eine grafische Benutzeroberfläche, wie zum Beispiel „Swing“ zur Verfügung stellen. Ist ein Java System nicht von Anfang an für den Betrieb auf mehreren Systemen ausgelegt, so ist es im Nachhinein nicht so ohne weiteres ohne Refactoring[1] verteilbar. Durch eine implizite Nutzung von Remote[2] - Schnittstellen lässt sich eine JEE5 Anwendung durch Deployment jederzeit auf mehrere Container verteilen.

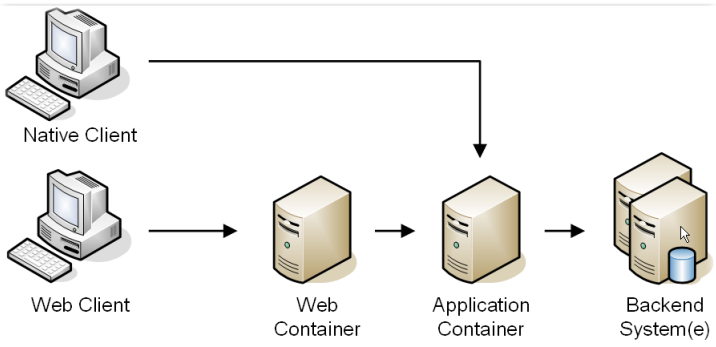


Abb.: 25Skalierbarkeit

1 Refactoring (deutsch auch Refaktorisierung, Refaktorierung, Restrukturierung oder schlicht Umgestaltung) bezeichnet in der Software-Entwicklung die manuelle oder automatisierte Strukturverbesserung von Programm-Quelltexten unter Beibehaltung des beobachtbaren Programm-Verhaltens. Dabei sollen die Lesbarkeit, Verständlichkeit, Wartbarkeit und Erweiterbarkeit verbessert werden, mit dem Ziel, den jeweiligen Aufwand für Fehleranalyse und funktionale Erweiterungen deutlich zu senken. Quelle: Wikipedia, 15.9.2009

2 Schnittstelle zur Fernwartung

Liegen die Container auf physikalisch unterschiedlichen Rechnern, dann könnte zum Beispiel ein „native Client“ (eine Java Swing Applikation) direkt auf den Application Container zugreifen und der WebClient bedient sich des Web Containers und beide Container benötigen das Datenbank Management System des Backend Systems. Beim WebClient funktioniert das so, dass der Benutzer via http eine Anfrage an den Web Container stellt und dieser ein Programm laufen hat, welches die Anfrage bearbeitet und html Code an den Client zurückschickt. Diese serverseitigen Programme nennt man in der Java - Welt „Servlets“. Man bezeichnet diese Systeme als „Mehrschichtarchitektur“. Punkto Skalierbarkeit ist auch ein Clustering möglich, das heißt, bei steigenden Lasten kann man die einzelnen Container auf verschiedene Instanzen verteilen. So kann die Rechenleistung von mehreren Maschinen für eine Anwendung genutzt werden.

J2EE ist eine Sammlung von Werkzeugen und Techniken, welche für Firmenlandschaften das EDV - technische Rüstzeug bereitstellen sollen. Die Problematik besteht jedoch in der Komplexität, je spezieller die Anforderungen werden. Damit kommen auch wieder die Kosten ins Spiel, da für diese Aufgaben natürlich Spezialisten ans Werk gehen müssen.
Am Wichtigsten ist es jedoch meiner Meinung nach, die Dinge nicht aus den Augen zu verlieren und bei Bedarf gezielt einzusetzen. Den Überblick über die „Enterprise“ - Produktpalette zu behalten erfordert ziemlichen Einsatz, der aber möglicherweise mit einem guten und skalierbaren Design belohnt wird, was im Endeffekt der Firma mehr Effizienz und Kostenersparnis bringt, obwohl es am Anfang gar nicht diesen Anschein hat.

4 Zusammenfassung

Die Aufgabe, Lehrunterlagen über objektorientierte Programmierung zu erstellen und zwar für ein bestimmtes, zu erzielendes Niveau von Handelsakademieabsolventen im IT - Zweig, hat mir einiges an Phantasie abverlangt. Zum einen ist die Materie an sich schon ziemlich komplex und zum anderen macht die Theorie ohne Praxis wenig Sinn. Ziel war es einen Mittelweg zu finden, von der objektorientierten Theorie, über die Planung und das Design von Software, bis zum Einsatz einer, den Anforderungen der objektorientierten Theorie genügenden, Programmiersprache. Ich habe Java als Programmiersprache gewählt, weil diese Sprache modern und plattformübergreifend ist und auf Grund meiner beruflichen Erfahrung in den meisten Produktionsfirmen eingesetzt wird. Die Konkurrenzsprache C# von Microsoft ist noch neuer als Java und wurde von Grund auf neu entwickelt, hat aber bei den Firmen noch nicht wirkliche Akzeptanz gefunden. Die Programmiersprache hatte ich nun, aber was ist für einen „IT - Zweig Handelsakademieabsolventen“ wichtig, um im Berufsleben Fuß fassen zu können? Viele Firmen schätzen natürlich die Vorzüge um das kommerzielle Wissen der HAK - Absolventen und setzen dies natürlich ein, aber es kommt noch ein Aspekt dazu. Es läuft nichts mehr ohne EDV. Viele Firmen planen neue Abläufe und neue Systeme einzuführen und genau da brauche ich heute jemanden, der zumindest ansatzweise eine Ahnung von dem hat, wofür er vielleicht die Kosten ausrechnen soll und sich aus bestimmten Gründen möglicherweise auch für ein System entscheiden soll. Dieses Wissen, nein fast schon ein Gefühl habe ich versucht mit meinen Lektionen zu vermitteln, wobei für mich der wichtigste Teil am Anfang „First make a plan“ fast schon zur Gebetsfloskel im Unterricht geworden ist. Es gibt arbeitstechnisch nichts Schrecklicheres, als Softwareentwickler, die sich nach oberflächlicher Beschäftigung mit einer Aufgabe, ihrer Tastatur und Maus hingeben und von einem nicht vorhandenen Konzept nicht mehr abzubringen sind. Darum habe ich die Analyse und das Design besonders hervorgehoben, wobei zu betrachten ist, dass heutzutage ohne objektorientiertes Hintergrundwissen sowieso kein schlüssiges Softwaredesign möglich ist. Als weiterer wichtiger Aspekt erschien mit der Ansatz einige Methoden aufzuzeigen, sich das Leben für diese Aufgaben leichter zu machen, denn einfach ist es sowieso nicht, vor allem, wenn man in einem Team arbeitet helfen oft die besten Techniken nicht, wenn die Mischung des Teams nicht passt. Aber dafür kann die objektorientierte Programmierung nichts, sie ist nur ein Hilfsmittel. Die Einführung der wichtigsten Begriffe musste einfach sein, um ein gemeinsames Ausgangsniveau zu erhalten.

Diese Begriffe, wie Vererbung, Polymorphie und Interfaces waren bis vor kurzem noch die absoluten Zauberworte in der Softwareentwicklung. Jeder Softwaredesigner, der etwas auf sich hielt, war bestrebt, zumindest die „Vererbung“ exzessiv zu nutzen. Wofür das alles? In einem Kapitel habe ich beschrieben, dass es viel sinnvoller ist, Software lesbar zu schreiben und damit meine ich, lesbar für denjenigen der die Sache warten und nachbearbeiten muss, als alle mystischen Tricks der objektorientierten Programmierung einzusetzen. Firmen brauchen Transparenz und kostengünstige Wartbarkeit. Die Wartung der Software darf nicht so teuer werden, dass man gleich eine Neuentwicklung in Erwägung ziehen könnte. Auf der einen Seite werden wiederverwertbare Komponenten gefordert, frei nach den „Black - Box“ denken, auf der anderen Seite will man die Möglichkeit haben, überall eingreifen zu können. Es ist immer eine Gradwanderung und wie gesagt, die objektorientierte Programmierung ist dafür ein Hilfsmittel, aber sicher nicht der heilige Gral. Es gab eine Zeit, in der Software „Design Patterns“ groß in Mode waren. Dies sind Entwurfsmuster für Softwareentwicklung, welche sich für immer wiederkehrende grobe Aufgabenstrukturen, Klassenhierachien und Strukturen der Implementierung anbieten. Das war schon gut so, aber der Einsatz hat sich trotzdem nicht bewährt. Zum einen, weil die Softwareentwickler einige Basispatters sehr gut beherrschten und nicht mehr nach links oder rechts schauten und zum anderen hätte es für viele Aufgaben effizientere Lösungen gegeben, wenn man sich besonnen hätte, dass im übertragenen Sinne, viele Wege nach Rom führen.
Ich glaube mit meiner Meinung richtig zu liegen, dass unsere Absolventen nicht die großen Softwareentwickler werden, mit möglichen Ausnahmen, aber darauf hätte ich mit meiner Arbeit sicher keinen Einfluss gehabt, aber sie müssen den Überblick haben und eine Ahnung davon was möglich ist. Aus diesem Grunde sind die Lektionen sehr exemplarisch und sollen selbst implementierbar sein und vielleicht Lust auf mehr machen. Dies meinte ich mit der anfangs erwähnten Gradwanderung. Meine Arbeit ist als softwaretechnisch gesehenes „Framework“ aufzufassen, welches einen guten Rahmen für weitere Entwicklungen bieten soll.
Wie lange im Bereich der Softwareentwicklung eine Strömung anhält weiß niemand genau zu sagen, doch ich denke, eine gute Basis hat immer Bestand.
Um nochmals Albert Einstein zu zitieren und das Werk zu beenden:
„Man soll die Dinge so einfach wie möglich machen, aber nicht einfacher!“
Nach Veröffentlichung sind die Lehrunterlagen im Login - Bereich der Website mit der URL www.hakju.at zu finden.

Ehrenfried Stuhpfarrer, Dezember 2009

Literaturverzeichnis

(1) Adobe After Effects CS3 - Philippe Fontaine - ISBN 978--38362-1165-9

(2) Das Java Codebook, Addison - Wesley, Donnermeyer, S 850, ISBN 3-8273-2059-3

(3) Flash MX und ActionScript Kompendium - ISBN 3-8272-6186-4

(4) Graphic Java, Mastering the JFC, David M. Geary, S 1600, ISBN 0-13-079667-0

(5) Implementation Patterns, Kent Beck, 191 S, ISBN 978-3-8273-2644-7

(6) JAVA 2, Schritt für Schritt zum Profi, Markt + Technik, S 881, ISBN 3-8272-6723-4

(7) Java ist auch eine Insel (8. Auflage), Ullenboom, 1.475 S, Galileo Computing, ISBN 978-3-8362-1371-4

(8) Handbuch der Java Programmierung, Guido Krüger, S 1235, ISBN 3-8273-2120-4

(9) Objektorientierte Programmierung; 656 S., Galileo Computing; ISBN 978-3-8362-1401-8

(10) http://de.wikipedia.org/wiki/Objektorientierte_Programmierung

(11) http://de.wikibooks.org/wiki/Programmieren:_Paradigmen:_Objektorientierte_Programmierung

(12) www.objektorientierte-programmierung.de

(13) Projekt - Management - STEP BY STEP - SBXNr. 130763; www.wissenistmanz.at

(14) Thinking in Java, Bruce Eckel, S 1900, Mindview Inc., http://www.mindviewinc.com

(15) uml http://www.parlezuml.com/tutorials/umlforjava/java_ocl.pdf

(16) Wirtschaftsinformatik, Baier, Bauer, Wurzer..., MANZ, S 236, SBNr. 120895

(17) PV08, Mag. Thomas Schöpf, Schulbuchnummer 116.474, www.rw-interaktiv.at

Verwendete Programme

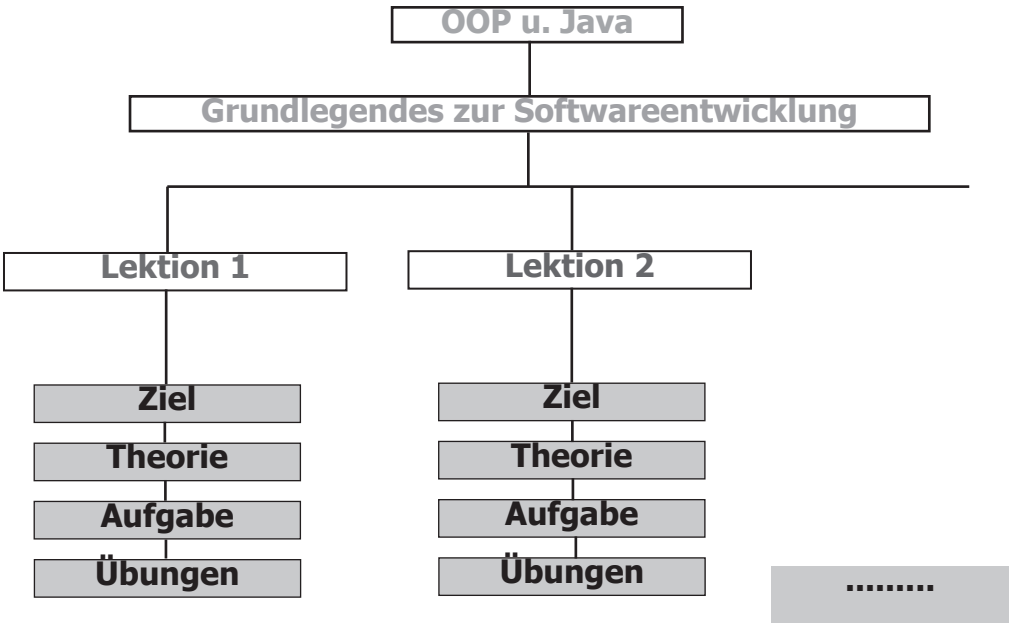
DreamweaverCs3 ® Adobe
InDesignCs3 ® Adobe
Excel ® Microsoft
Modeilio - UML Designer ® Modeliosoft
Java Netbeans IDE 6.8 ® Sun
PhotoshopCs3 ® Adobe
Together - CaseTool ® TogetherSoft
Tomcat ® Jakarta

Das Projektlogo

Das objektorientierte Konzept ist zwiebelschalenartig aufgebaut. Ein Objekt beinhalte wieder ein Objekt, dieses enthält wieder ein Objekt etc.
Auf meinem Schreibtisch steht ein Muranoglas welches mich durch den zwiebelschaligen Aufbau motivierte, dieses als Projektlogo zu fotografieren, entsprechend zu bearbeiten und in die Projektdokumentation einzubauen.
Die kleinen Bälle, mit verschiedenen Buchstaben, die zusammen das Wort „class“ ergeben, stehen als Synonym für den wichtigsten Begriff in der objektorientierten Programmierung, der Klasse.

Sitemap

Der Aufbau der Website erfolgt ebenfalls als Framework. Die Struktur ermöglicht es, relativ einfach Seiten hinzuzufügen und Änderungen vorzunehmen. Alle Lektionen und Zusatzinformationen sind auch als PDF abrufbar.



Abbildungsverzeichnis