HS Fachdidaktik Informatik

Objektorientierte Programmierung im Anfangsunterricht

Siegfried Spolwig

Daniel Reinhold, Matrikelnummer 172491

Jörg Strehmann, Matrikelnummer 184273

Hausarbeit

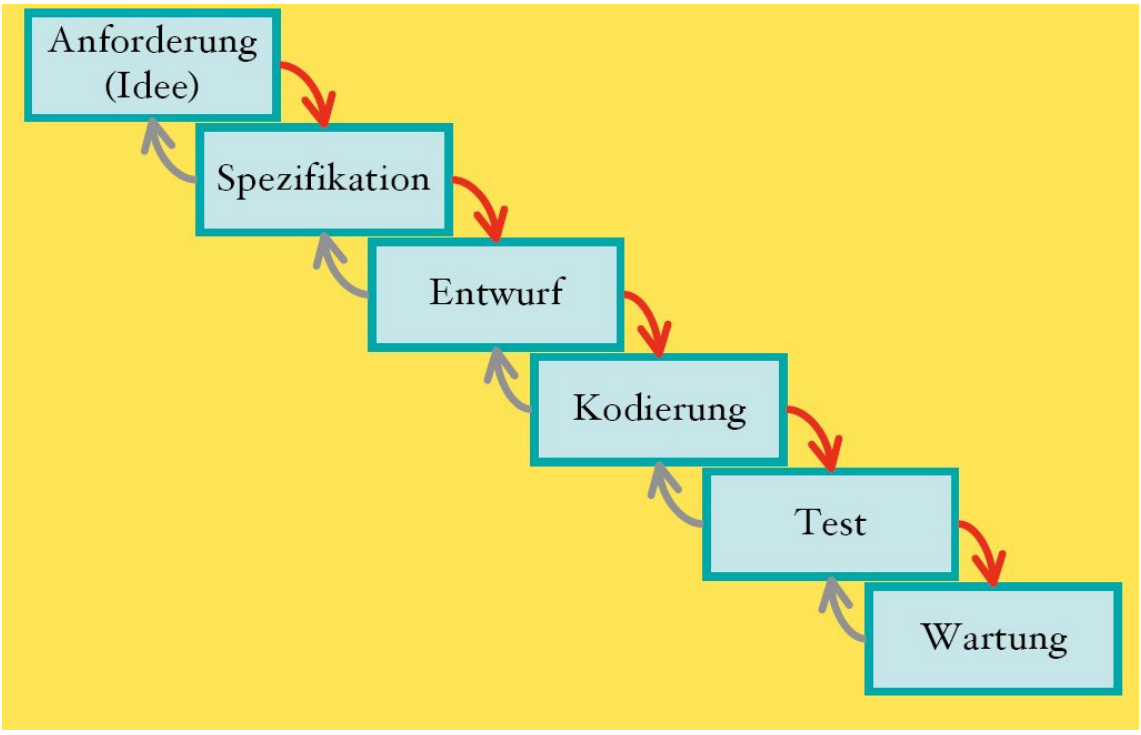
„Entwicklung und Programmierung eines Bildschirmspiels in FUJABA und Java“

Inhalt

1. Einleitung

Zunächst soll hier der Begriff des Programmierens präzisiert werden. Orientiert man sich am Wasserfallmodell der Software-Entwicklung (nachfolgende Grafik aus der VL Praktische Informatik 1, HU WS 02/03, Prof. Coy) und setzt Software-Entwicklung mit Programmierung gleich, dann wird deutlich, dass der Prozess der Codierung erst recht spät im ersten Zyklus des Wasserfalls auftritt. Aufgrund der begrenzten Zeit wird es im Unterricht vermutlich ohnehin nur eine Iteration des Wasserfalls geben, die Testphase wird verkürzt sein, da die Aufgabenstellung nicht sehr komplex ist, und Wartung wird im Laufe dieses Kurses nicht mehr notwendig sein. Bei konsequenter Anwendung des Wasserfallmodells - die sich schon deshalb anbietet, um den Schülerinnen und Schüler die realen Prozesse der Softwareentwicklung nahe zu bringen - kommt die Codierung ganz zum Schluss und erschöpft sich im durch die Syntax der Programmiersprache vorgegebenen mechanischen Abarbeiten des zuvor entwickelten Modells. Daher wollen wir der Codierung nur wenig Zeit zum Ende des Kurses einräumen und nehmen auch in Kauf, dass das lauffähige Programm in diesem Halbjahr nicht fertig wird, was auch einen zusätzlichen Anreiz darstellen dürfte, den Anschlusskurs zu besuchen.

Wasserfallmodell



Der Kurs setzt sich zu 50 Prozent aus Mädchen zusammen. Es ist festzustellen, dass die Lust, sich mit Problemen der Informatik auseinander zu setzen, bei Mädchen weniger ausgeprägt ist, als bei den Jungen. Da wir beobachtet haben, dass dies jedoch nicht mit Technikfeindlichkeit gleichzusetzen ist (Handynutzung, insbes. auch für Bildschirmspiele etc.), wollen wir mit der Auswahl des Themas eine Identifikationsmöglichkeit auch für die Mädchen schaffen. Wir schlagen vor, einen einfachen Ego-Shooter zu programmieren, dessen Ego Molly Starracer, eine Manga-Heldin ist. Dieses Vorhaben bietet auch die Möglichkeit, mit relativ wenigen Klassen zu operieren, so dass die grundsätzliche Umsetzbarkeit gegeben ist.

Als Werkzeug zur Modellierung wollen wir FUJABA verwenden, als Programmiersprache Java. Für FUJABA entscheiden wir uns trotz der harschen Kritik aus folgenden Gründen: Zwar löst FUJABA bisher nicht im Ansatz ein, was der Name suggeriert, aber die Kernkomponente, das System zur Modellierung arbeitet korrekt. Hier ist der für Schülerinnen und Schüler ideale Kompromiss zwischen begrenzter Komplexität und universeller Einsetzbarkeit gefunden worden. UML ist viel zu komplex; FUJABA kennt nur Klassendiagramme und Storydiagramme. Letztere stellen eine praxisgerechte Fusion verschiedener laufzeitbezogener Diagrammklassen in UML dar. Dass man in FUJABA aber überhaupt Prozesse darstellen kann, setzt es gegenüber anderen unterrichtsbezogenen Werkzeugen, wie z.B. BlueJ ab. Wir begrüßen auch, dass man hier Java-inhärente Begriffe wie z.B. Typdeklarationen vorwegnehmen kann (und auch muss), so dass die Schülerinnen und Schüler in der Codierungsphase, wenn sie Java lernen müssen, nicht von der Menge des Stoffes überwältigt werden. Die Sprache Java erscheint eine sinnvolle Wahl, da im Einführungskurs internetbezogen gearbeitet wurde und Applets die am besten beherrschbare Form der dynamischen Steuerung von Webinhalten darstellen. Des weiteren bietet sich über die bereits vorhandenen Klassenbibliotheken die Möglichkeit, schon nach kurzer Zeit professionell wirkende Programme zu erhalten. Da FUJABA die Übertragung von Modellen in Code nur ungenügend beherrscht, plädieren wir dafür, dass Code im Texteditor geschrieben und mit JAVAC kompiliert wird. Dies schärft auch das Bewusstsein für Erfordernisse der Java-Programmierung (Anlegen einer main-Methode in Applikationen etc.)

2. Warum Objektorientierte Programmierung?

Genau genommen geht es bei der Planung des Halbjahresverlaufs überwiegend um objektorientierte Analyse (OOA), worunter die drei Kaskaden des oben gezeigten Wasserfalls, die mit Anforderung, Spezifikation und Entwurf bezeichnet werden, zu verstehen sind. Objektorientierung sowohl im Entwurf als auch in der Programmierung lehnt sich an die menschliche Erfahrungswelt an. Wenn man davon ausgeht, dass alle Programmiersprachen in bezug auf Berechenbarkeit gemäß der Turingschen Postulate das gleiche leisten, dann besteht ihr wesentlicher Unterschied in der angebotenen Mensch-Maschine-Schnittstelle, in der verwendeten Metaphorik der Problem- und Lösungsbeschreibungen. Diese Metaphorik muss mit der Erfahrungswelt des Programmierers verknüpft sein. Für prozedurale, funktionale und logische Sprachen gilt, dass deren Konzepte nur dem mathematisch Vorgebildeten wirklich eingängig sind. Andere Nutzer können zwar streng regelgeleitet ebenfalls Grundkenntnisse der Programmierung in PASCAL, PROLOG oder CLEAN erwerben, jedoch sind diese Kenntnisse ohne Verständnis der zugrundeliegenden Philosophie kaum anwend- oder erweiterbar. Entsprechend schlechte Erfahrungen liegen für den jahrelangen Einsatz von PASCAL oder BASIC an der Schule vor.

Sicherlich haben die Entwickler objektorientierter Entwurfswerkzeuge und Programmiersprachen nicht zuerst an den Einsatz in der Schule gedacht, dennoch ist die starke Verankerung der Objektorientierung in der Erfahrungswelt mathematisch kaum vorgebildeter Menschen ein großes Plus.

Die Grundideen der Objektorientierung sind: Die Welt besteht aus voneinander verschiedenen Objekten. Viele Objekte sind einander jedoch ähnlich, verfügen über gemeinsame Eigenschaften und lassen sich deshalb zu Klassen zusammenfassen. Objekte haben Eigenschaften (=Variablen), und sie können Aufgaben verrichten oder etwas mit sich geschehen lassen (=Methoden). Klassen stehen miteinander in Beziehung. Allgemeinere Klassen können Oberklassen von spezielleren Klassen sein, Klassen können andere Klassen haben (aggregieren), Klassen können bestimmte Eigenschaften mit anderen Klassen teilen. Klassen und Objekte behalten meist für sich, wie sie eine bestimmte Aufgabe lösen, interessant ist meist nur, dass man das Gewünschte bekommt (Übergabewert) oder die Klasse tut, was sie soll (= korrekte Ausführung der Methoden). Diese Vorgehensweise nennt man Datenkappselung. Deren Vorteil liegt in der Möglichkeit der Modularisierung und der

Austauschbarkeit einzelner Module, ohne den Kontext verändern zu müssen. (Wenn alle Autos von Benzin- auf Elektroantrieb umgestellt würden, müsste man auch nicht neu Auto fahren lernen, wichtig ist nur, dass die Klasse Antrieb mit der Methode Antreiben den Rückgabewert Antriebsenergie in der gewünschten Größenordnung bereitstellt.)

Welche Vorteile diese Sichtweise hat, zeigt die Betrachtung des oben genannten Beispiels in der prozeduralen Programmierung. Hier wäre nur der Datenfluss von Bedeutung, Die Prozedur Antreiben stünde im Vordergrund und manipulierte eine Variable des Typs Antriebsenergie mit dem Index Antrieb. Zum einen „sieht“ man hier deutlich schlechter, was passiert, zum anderen wäre eine notwendige Änderung von Benzin- zu Elektroantrieb nicht ohne großen Aufwand zu bewerkstelligen, da die Prozedur nicht hinreichend vom Rest des Programms gekapselt ist.

3. Voraussetzungen

3.1. Die Klasse

Es handelt sich um eine Berliner Klasse der Sek II, die in dreijähriger Kursfolge 3 X 45 min/ Woche Informatik bei 16 Wochen im Halbjahr hat. Die Einführung in die objektorientierte Programmierung beginnt mit dem zweiten Halbjahr der 11. Klasse. Im ersten Halbjahr wurden folgende Themen behandelt:

- Einführung in die Informatik über den internetorientierten Zugang
- Grundlagen der Rechnerorganisation, der Netze und Geschichte der Informatik
- Datenbanken und Datenschutz

Der Mädchenanteil beträgt ca. 50 Prozent. Vereinzelt sind Programmiererfahrungen vorhanden, diese sind überwiegend unsystematisch. Nahezu alle Schüler verfügen über einen heimischen PC mit Internetzugang. [http://golem14.informatik.hu-berlin.de/spolwig/hsem_03/spolwig/stoffverteilungsplan.htm]

Aus diesen Voraussetzungen ergeben sich unmittelbar zwei Folgen:
1. Die Grundlagen der Programmierung müssen tatsächlich von ersten und kleinsten Elementen ausgehend entwickelt werden; ein Einstieg über die Lebenswelt der Schülerinnen und Schüler bietet sich hier an. Eine objektorientierte Herangehensweise unterstützt diesen Einstieg.
2. Es ist belegt, dass Mädchen von Jungen trotz überwiegend besserer schulischer Leistungen im naturwissenschaftlich-technischen Unterricht marginalisiert werden, müssen sie besonders motiviert und im Verlauf des Halbjahres stärker gefördert werden. Dem trägt der hier vorgestellte Halbjahresplan Rechnung.

Die Klassenräume stellen jedem Schüler einen vernetzten Arbeitsplatz zur Verfügung. Neben einigen anderen sind auch die von uns favorisierten Werkzeuge FUJABA und JAVA vorhanden. [http://golem14.informatik.hu-berlin.de/spolwig/hsem_03/spolwig/stoffverteilungsplan.htm]

3.2. Der Berliner Rahmenplan Informatik

Zum Schuljahr 2004/05 wird in Berlin ein neuer Rahmenplan für das Fach Informatik verbindlich. Da die voraussichtlichen Veränderungen gegenüber dem derzeit gültigen Rahmenplan bereits gut dokumentiert sind, wird dieser wesentlich dem aktuellen Entwicklungsstand der Wissenschaft Informatik angepasste Plan der hier vorgestellten Unterrichtsplanung zugrunde gelegt.

Zukünftig werden Schüler mit und ohne Vorkenntnisse gemeinsam unterrichtet. Daraus erwächst eine stärkere Notwendigkeit der Binnendifferenzierung. Da jedoch gleichzeitig allen Schülerinnen und Schülern ermöglicht werden muss, Informatik als Prüfungsfach im Abitur zu wählen, kann dies nur bedeuten, dass zwar an unterschiedlichen Ausgangspunkten gestartet wird, jedoch am Ende der Einführung in die Programmierung alle den gleichen Stand erreicht haben. Wir gehen davon aus, dass sich die unterschiedliche Vorbildung im wesentlichen auf praktische Erfahrungen mit dem PC bezieht. Es ist anzunehmen, dass keiner der Schüler die theoretischen Konstrukte der Objektorientierung oder Algorithmik kennt. Daher ist die Notwendigkeit der Binnendifferenzierung im ersten Teil des Kurses, der diese theoretischen Grundlagen vermittelt, nicht so stark ausgeprägt und beschränkt sich im wesentlichen auf die Unterstützung beim Umgang mit dem Computer, wie z.B. Zurechtfinden in der Verzeichnisstruktur etc. In der Phase der Durchführung des Programmierprojekts sehen wir folgende Möglichkeiten der Binnendifferenzierung: Die Klassen zur Beschreibung des Bildschirmspiels sind unterschiedlich komplex, auch die Programmierung der darin enthaltenen Methoden ist unterschiedlich schwierig. Da die Größe des Projektes ein arbeitsteiliges Modellieren und Implementieren verlangt, werden die einzelnen Arbeitslose nach Vorbildung und den erreichten Leistungen im ersten Teil des Kurses verteilt. Zum anderen ist vorgesehen, dass das Projekt, das von den Schülerinnen und Schülern weitgehend selbständig durchgeführt werden soll, von Tutoren begleitet wird, die bei Bedarf Hilfestellungen geben, in Vergessenheit geratenes wiederholen und die Methoden oder Teile von Methoden implementieren, die ihrem Schwierigkeitsgrad nach den Rahmen einer Einführung in die Programmierung sprengen. In erster Linie ist der Lehrer der Tutor, aber in den Fällen, in denen einige Schüler bereits wissen, was andere erst noch lernen müssen, um die ihnen übertragene Aufgabe lösen zu können, sollen erstere die Aufgabe des Tutors übernehmen.

Der Entwurf des zukünftigen Rahmenplans schreibt für das erste Jahr Informatik in der Sekundarstufe II drei Lernabschnitte zwingend vor:
- L1: Grundlagen der Rechnerorganisation, der Netze und der Geschichte der Informatik (min. 15 h, LK min. 20 h
- L2: Grundlagen der Programmentwicklung (min. 30 h, LK min. 40 h)
- L3: Datenbanken und Datenschutz (min. 15 h, LK min. 20 h)
/ECKPUNKTE ZUM NEUEN RAHMENPLAN INFORMATIK SEKII/

L1 und L3 sind durch den vorgegebenen Verlauf des ersten Halbjahres abgedeckt, der vorliegende Entwurf deckt L2 ab. Der Lehrplanentwurf regt im Sinne einer kann-Bestimmung an, L1 und L2 integriert zu behandeln. Als Standalone-Programm hat das hier behandelte Programmierprojekt jedoch kaum mit Rechnerorganisation und nichts mit Computernetzen zu tun. Mit der Geschichte der Informatik lässt sich diese Projekt insofern in Einklang bringen, als die erste nicht-wissenschaftliche Anwendung eines Computers in der Tat ein Bildschirmspiel war, 1958 programmiert für die Besucher eines Tages der offenen Tür in einem staatlichen Computerlabor in den USA. [http://www.osti.gov/accomplishments/videogame.html] Das erste kommerzielle Spiel, das 1961 programmiert wurde, soll die Entwicklung des noch heute weit verbreiteten Betriebssystems UNIX maßgeblich beeinflusst haben, stellte ein Weltraumspiel dar, wirkte optisch stilbildend, und auch unser Beispielprogramm zeigt eine ähnliche Gestaltung. [http://www.cgl.uwaterloo.ca/~anicolao/sw/Spacewar/spacewar.html] Auch in der jüngeren Vergangenheit setzten Computersiele noch wichtige Impulse für die Entwicklung von Computern, z.B. stützen sich heute viele leistungsfähige Mathematikanwendungen auf die Rechenleistung von 3D-Grafikkarten, die für schnelle Spiele entwickelt wurden. Diese Tatsachen können in den ersten Stunden, wenn es um die Motivation für das Programmierprojekt geht, zur Sprache kommen.

Für den Einstieg in den Informatikunterricht ist einer von vier Wegen vorgeschrieben:
E1: Benutzung und Analyse eines dokumentierten Systems
E2: Der internetorientierte Zugang
E3: Der variablenfreie Zugang
E4: Der applikative (z.B. funktionale) Zugang (Leistungskurs)
Gemäß Vorgabe wurde E2 verwendet. Da im zweiten Halbjahr des ersten Jahres die Einführungsphase noch nicht als abgeschlossen betrachtet werden kann und wir die Wahl der Programmiersprache JAVA mit ihrer starken Verbreitung im Internet begründet haben, ist es

sinnvoll, auf die beiden grundsätzlichen Möglichkeiten des Einsatzes von JAVA-Programmen, nämlich als Applet und als Applikation einzugehen. Dies ist zugleich eine mögliche Perspektive für Klasse 12: In Vertiefung des in Klasse 11 Geleisteten könnte dann ein Website rund um das Bildschirmspiel kreiert werden,
- der auf die **historische** Dimension von Computerspielen eingeht
- auf dem der bereits fertiggestellte Quellcode des Spiels offen gelegt wird und Schülerinnen und Schüler anderer Kurse oder Schulen zum Weiterentwickeln des Spiels aufgefordert werden
- der ein Forum enthält, das, wie derzeit üblich, über eine SQL-**Datenbank** realisiert wird.
Dies entspräche den Gebieten E - Datenbanken, F - Computergrafik und G - Computernetze, aus denen (und 7 bzw. 5 weiteren in Grund- bzw. Leistungskurs) jeder Schüler zwei als Vertiefungsrichtungen in Klasse 12 auswählen muss.

Wichtig ist, dass der neue Rahmenplan (wohl aus der Erfahrung heraus, dass sich konkrete Werkzeuge in der außerschulischen Welt schnell als obsolet erweisen können) keine konkreten Programmiersprachen oder Entwicklungsumgebungen benennt, so dass die Wahl in dieser Hinsicht nicht von vorneherein beschränkt ist. Im Sinne einer soll-Bestimmung wird die Einführung in objektorientiertes Programmieren ausdrücklich begrüßt. Auf einer detailierteren Ebene schreibt die Lehrplanneufassung folgende Themen zwingend vor:

- Programmstrukturen (Algorithmik in Ansätzen)
- Variablenkonzept
- Datenstrukturen
- Anwenden elementarer Daten- und Steuerungsstrukturen
- Anwendung objektbasierter Strukturen

Diese werden daher sämtlich in unserem Entwurf berücksichtigt.

3.3. Inhalte, Kompetenzen und Ziele

Mit der objektorientierten Programmierung sind eine Vielzahl von Inhalten verbunden, aus denen man aus Zeitgründen und wegen des Charakters des Kurses als Einführung die grundlegendsten auswählen, zu komplexe oder nicht zwingend notwendige Inhalte jedoch

streichen muss. Im Rahmen des Hauptseminars, aus dem diese Arbeit hervorging, ergab sich im Ergebnis einer Diskussion folgende Liste essentieller Inhalte:

- Objekt, Klasse
- Methoden, Attribute
- Variablenkonzept
- Datentypen (einfache + Objekte)
- Algorithmen --> Kontrollstrukturen
- Syntax einer Einzelsprache (Fehlermeldungen!)
- Klassenbeziehungen --> Vererbung
- Operatoren
- Parameter
- Konzept: OOA
- graphische Darstellung von Klassen (UML) und
- Algorithmen (Struktogramme)

[http://golem14.informatik.hu-berlin.de/spolwig/hsem_03/protokolle/prot_040209.htm]

Gestrichen wurden folgende Inhalte:

- Kapselung
- Polymorphismus
- Konzept: OOD

Diese sind zwar zentral, aber für eine Einführung zu komplex. Die unmittelbaren Konsequenzen aus z.B. der Datenkapselung für unser Programmierprojekt müssen von den Schülern als gegeben hingenommen werden. Zu einem späteren Zeitpunkt (z.B. in Klasse 12) ist es ratsam, diese grundlegenden Konzepte ausführlicher darzustellen. In unserem Beispielprogramm sind sämtliche Methoden *public* und die Variablen ohne Modifier, was sie innerhalb des Packages *public* und sonst *private* macht.
Als bedingt notwendig wurden folgende Inhalte angesehen:

- Sichtbarkeit
- Konzept: Modell - View - Control

Hier bleibt es dem Lehrer überlassen, diese jetzt schon einzuführen. MVC wird in unserem Stoffverteilungsplan keine Rolle spielen, Sichtbarkeit nur am Rande, wenn Variablen behandelt werden.
Diese Inhalte stellen das Ziel der Unterrichtsplanung auf der rein kognitiven Ebene dar, es handelt sich dabei um das zu erwerbende **Wissen**.

Ein weiteres Ziel der Unterrichtsplanung ist es, den Schülern Einstellungen zu vermitteln und sie zu Handlungen zu befähigen. Hier spielen kognitive, affektive und psychomotorische Elemente zusammen, die im Ergebnis zu den Kompetenzen der Schüler führen. m weitesten Sinne wird hier das am Ende des Kurses gewünschte Können des Schülers beschrieben. Folgende Kompetenzen wurden im Ergebnis der Diskussion im Hauptseminar als essentiell angesehen:

- Sachkompetenz

1.Modell aus einem Weltausschnitt entwickeln und in einem Klassendiagramm (nach UML) darstellen
2.Ein Anwendungsproblem objektorientiert analysieren, einen Lösungsansatz entwickeln und mit Hilfe von Programmiertechniken umsetzen
3.Eine Modul-/Klassenspezifikation (Schnittstellen) lesen können und benötigte Teile in ein Programm einbinden
4.Die für den Lernabschnitt erforderlichen Sprachelemente sicher beherrschen und anwenden
5.Ein Programm angemessener Komplexität selbstständig (unter Zuhilfenahme einer Vorlage) projektieren und realisieren

- Methodenkompetenz

6.Klassendiagramm zur Strukturierung von Programmstrukturen , Nassi-Shneiderman-Diagramm zur Entwicklung von Algorithmen einsetzen
7.Aus einer verbalen Aufgabenstellung einen Algorithmus entwickeln und in einem Nassi-Shneiderman-Diagramm darstellen
8.Ein Nassi-Shneiderman-Diagramm in Programmcode überführen
9.Einen Quelltext analysieren und in einem Nassi-Shneiderman-Diagramm darstellen

10.Auftretende Probleme durch kundiges Benutzen eines geeigneten (integrierten) Hilfesystems selbständig lösen

- Sozialkompetenz

11.Umfangreiche Aufgaben untergliedern und in einem Team arbeitsteilig lösen

12.Absprachen treffen und einhalten

- Selbstkompetenz

13.Eigene Lösungen der Gruppe vorstellen und verteidigen

14.Eigenes Arbeitstempo entwickeln und verfolgen anhand geeigneter Arbeitsmaterialien

[http://golem14.informatik.hu-berlin.de/spolwig/hsem_03/spolwig/kompetenzen.htm]

Diese angestrebten Kompetenzen werden im folgenden dahingehend modifiziert, dass Nassi-Shneidermann-Diagramme durch Diagrammarten, wie sie FUJABA einführt ersetzt werden. Der Wesensgehalt der Kompetenzen, die sich auf diese Diagramme beziehen, nämlich die Fähigkeit, sprachunabhängig Prozesse und Klassen-/Objektbeziehungen modellieren zu können, ist von dieser Entscheidung nicht berührt. Im übrigen scheinen an UML angelegte Diagrammarten klassische Struktogramme zunehmend zu ersetzen.

4. Die Werkzeuge

4.1. Fujaba

UML, die Unified Modeling Language, ist ein mächtiges Werkzeug, um alle Aspekte eines Softwareprojektes zu entwickeln. In UML lassen sich Benutzerschnittstellen, Laufzeitverhalten, statische Beziehungen und z.T. sogar der Implementierungsprozess selbst modellieren. UML stellt derzeit einen de-facto-Standard dar. Daher erscheint es sinnvoll, UML auch im Unterricht zur Einführung in die Objektorientierung zu benutzen. Andererseits ist UML außerordentlich umfangreich, und es kann von keinem Schüler, noch dazu in der Einführungsphase, erwartet werden, alle Konventionen zu lernen. Es liegt also nahe, für den Bildungsbereich eine leicht beherrschbare Teilmenge von UML zu schaffen. Weiterhin wäre ein Werkzeug wünschenswert, das das zeitraubende Zeichnen immer neuer Boxen auf Papier erspart. FUJABA ist ein solches Werkzeug, das allerdings höhere Ansprüche hat. FUJABA steht für „From UML To Java And Back“. Als Notationssystem stellt es eine Teilmenge von UML zur Verfügung, die nur Klassen- und Storydiagramme kennt. Storydiagramme sind eine vereinfachende Vereinigung von Interaktions-, Zustands- und Aktivitätsdiagramm. Üblicherweise verläuft die Analyse eines zu programmierenden Systems im Unterricht in folgenden vier Schritten ab:

- Erkennen der beteiligten Objekte, deren Attribute und Methoden
- Entwickeln einer hierarchischen Struktur durch Erkennen von Klassen, Unter- und Oberklassen
- Bestimmung der Beziehungen zwischen Objekten/Klassen
- Entdecken von Aktivitäten im System

Zur Darstellung dieser vier Schritte genügen oben genannte vereinfachte UML-Notationen völlig. FUJABA stellt eine graphische Oberfläche bereit, auf der Klassen per Mausklick erzeugt und zueinander in Beziehung gesetzt werden können. Die Idee von FUJABA ist jedoch eine weitergehende: Bei vollständiger Modellierung sowohl der Klassen und deren Abhängigkeiten als auch der Objekte und deren Verhalten zur Laufzeit soll FUJABA Java-Quelltext erzeugen. Gemäß dem zweiten Teil des Namens soll FUJABA auch erlauben, die Struktur vorhandenen Quelltexts analysieren zu lassen und über Mr. Dobs, einen graphischen Debugger, Fehler aufzuspüren. Beide Prozesse funktionieren in den aktuellen Releases von

FUJABA noch nicht zur vollen Zufriedenheit. Aus didaktischen Gründen legen wir darauf auch keinen Wert. Wichtig ist uns nur, dass mit FUJABA ein komfortables und leicht zu bedienendes Werkzeug zum Erzeugen vereinfachter UML-Strukturen zur Verfügung steht. Diese Eigenschaften machen es besonders für den Anfangsunterricht geeignet. Dass es JAVA-Quelltext erzeugt, wollen wir gerade nicht, um die Grenze zwischen Modellierung und Codierung nicht zu verwischen und um die Möglichkeit zu haben, die Schüler an die grundlegendste Weise der Codierung in JAVA, nämlich per Texteditor und JAVAC heranzuführen.

4.2. Java

Im Rahmen des internetbasierten Zugangs zur Informatik werden die Schülerinnen und Schüler gelegentlich auf Applets gestoßen sein. Vielleicht haben sie sich sogar bewusst gemacht, dass diese für dynamisches Verhalten der Websites gesorgt haben und in die HTML-Seiten eingebundene Programme sind, die beim Client, also auf ihrem eigenen Rechner ausgeführt werden. Damit sind ihnen implizit einige Vorteile von Java bereits bekannt: Man kann damit Anwendungen programmieren, die über das Internet übertragen und in einem Browser ausgeführt werden können. Diese Programme kann man leicht in HTML-Seiten einbinden. Java ist offensichtlich weit verbreitet, und es ist plattformunabhängig. Wenn man dann noch erwähnt, dass diese Plattformunabhängigkeit auch außerhalb des Browsers besteht und dass Java über riesige Klassenbibliotheken verfügt, die es ermöglichen, mit wenig Aufwand optisch ansprechende, nicht triviale Programme zu erzeugen, dann wird den Schülern klar werden, dass Java eine der geeignetsten Sprachen für den Einstieg in die Programmierung ist. Viele andere Sprachen haben ihren Schwerpunkt in der möglichst einfachen und genauen Abbildung mathematischer Konzepte. Dies ist für die Schule jedoch weniger interessant. Javas große Stärke kann man in der einfachen Erzeugung von Mensch-Maschine-Schnittstellen sehen. Daher können auch Anfänger Programme erstellen, „mit denen man etwas machen kann“ und „wo man etwas sieht“. Dies ist für die Motivation im Anfangsunterricht besonders wichtig, denn ohne schnelle Erfolgserlebnisse lassen sich lange Durststrecken, wie sie bei der Vermittlung komplexerer Strukturen zwangsläufig entstehen, nicht überwinden.

Zum anderen haben wir Java natürlich gewählt, weil diese Sprache bei relativ leicht zu lernender Syntax in großer Klarheit objektorientiert ist. Dadurch entstehen keine Brüche beim Übergang von der Modellierung zur Codierung.

5. Der Lehrplanentwurf

5.1. Sachanalyse

- Objektorientierte Analyse

Ziel des Analyseprozesses ist es, ein System von Objekten zu finden und zu arrangieren, die im gemeinsamen Zusammenspiel das reale System (Fachkonzept) abbilden und die gestellte Aufgabe mit verteilten Verantwortlichkeiten erledigen. Damit ist im Grunde die Aufgabe der OOA beschrieben, die bei allen Analysemethoden ähnlich abläuft: zunächst sind die Objekte und Klassen zu finden, dann ihre Attribute und Funktionen zu bestimmen und zuletzt die Beziehungen der Objekte untereinander festzulegen. [S. Spolwig: http://www.oszhdl.be.schule.de/gymnasium/faecher/informatik/ooa-ood/ooa_ziele.htm]

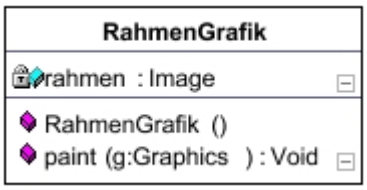
In der OOA wird beschrieben, was ein System leisten soll. Auf die konkrete Implementation, insbesondere die Verwirklichung von Konstrukten der zu verwendenden Programmiersprache, wird dabei noch nicht eingegangen

- Objektorientierte Programmierung

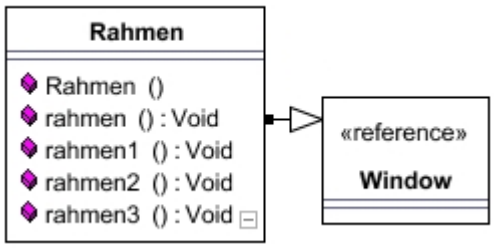
In der objektorientierten Programmierung wird ein in der OOA entwickeltes Modell in die Syntax einer Sprache übersetzt, die Objektorientierung unterstützt. In unserem Falle ist dies Java.

- UML

Die Unified Modeling Language ist eine Sprache zur Beschreibung von Softwaresystemen. Sie hat zum einen vorher bestehende Notationssysteme, zum anderen die Darstellung für verschiedene Einsatzgebiete vereinheitlicht. Sie enthält verschiedene Diagrammarten, von denen die wichtigste das Klassendiagramm ist. Ein Klassendiagramm enthält Boxen für jede Klasse, in der Klassenname, Klassenvariablen, deren Sichtbarkeit sowie Methoden und deren Sichtbarkeit aufgeführt werden. Eine solche Box sieht z.B. so aus:



Darüber hinaus werden im Klassendiagramm auch die Beziehungen zwischen Klassen dargestellt. Dies erfolgt über Linien und Zeichen an den Enden der Linien, die die Art der Beziehung beschreiben. Die wichtigste Beziehung ist die Vererbungsbeziehung und wird wie folgt dargestellt:

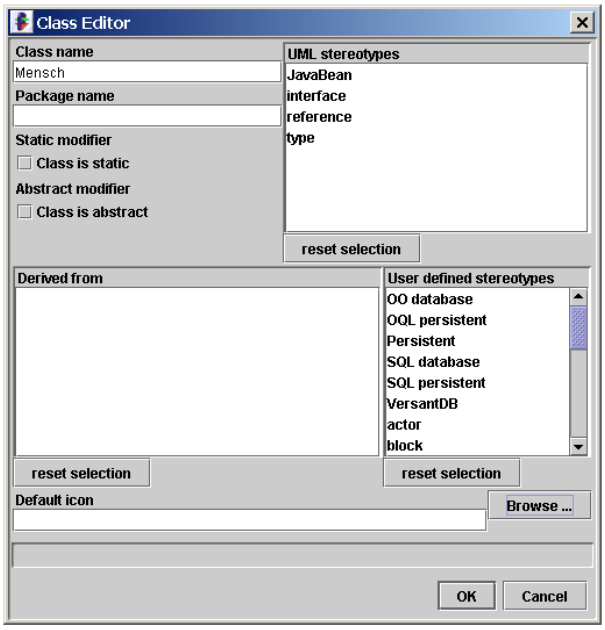


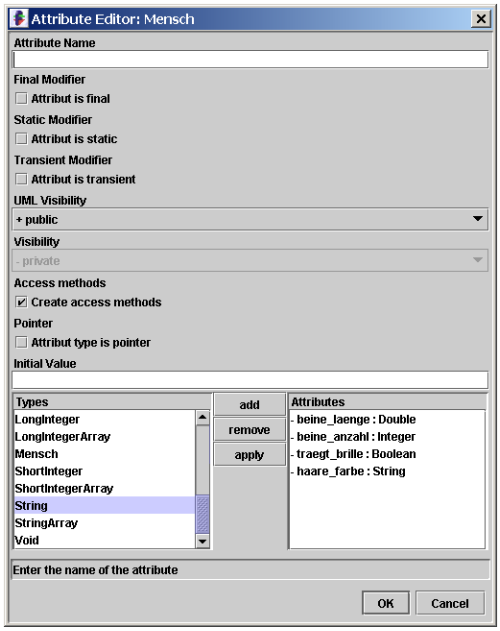
Diese Darstellung bedeutet, dass die Klasse Rahmen alle Variablen und Methoden der Klasse Window erbt, welche hier nicht dargestellt sind, da Window aus einer Klassenbibliothek von Java stammt.

Weitere Diagrammarten sind Use-Case-Diagramme, Interaktionsdiagramme, Package-Diagramme, Zustandsdiagramme, Aktivitätsdiagramme und Implementierungsdiagramme. Eine gute Einführung in UML findet man hier:
[http://www.fokus.gmd.de/research/cc/ecco/projects/OKS/WS9899/umlkompt.htm]

- FUJABA

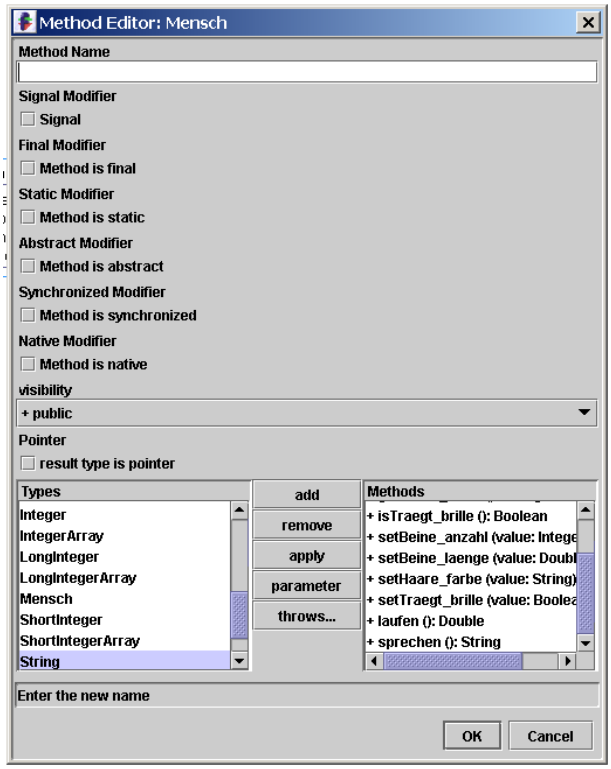
Was FUJABA leistet, wurde bereits in 4.1. besprochen. Die Bedienung erfolgt über Menüs, die entweder über die Kopfleiste oder über die rechte Maustaste aufgerufen werden. Im folgenden werden die gebräuchlichsten Menüs dargestellt, um eine Klasse zu erzeugen. In ähnlicher Weise können Beziehungen zwischen Klassen erstellt oder Aktivitätendiagramme erzeugt werden.

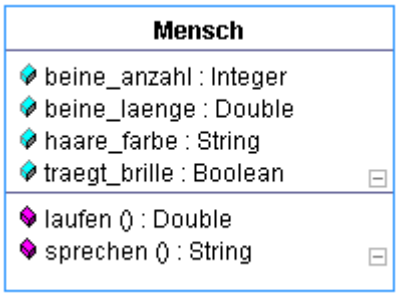




erzeugt eine Klasse

erzeugt Attribute einer Klasse





erzeugt Methoden einer Klasse

Darstellung der Klasse in FUJABA

- Variablen und Datentypen

Variablen beschreiben Eigenschaften von Objekten. Sie können als Behälter für Werte eines bestimmten Datentyps aufgefasst werden. Der Datentyp, auch Datenstruktur genannt, bezeichnet die Menge aller gleichartigen Daten, auf denen eine Sammlung von Operationen definiert ist. Eine Operation ist dabei eine Verknüpfung, die einer festen Anzahl von

Eingabedaten ein Ergebnis zuordnet. Die Verwendung eines bestimmten Datentyps in der Programmierung hat die Reservierung eines wohldefinierten Speicherbereichs zur Folge. Erzeugt eine Operation einen Wert der außerhalb des Wertebereichs des Datentyps liegt, entstehen Fehler. In Java gibt es einfache und strukturierte Datentypen. Die einfachen sind: boolean, char, byte, short, int, long, float, double. Alle array-, class- und interface-Datentypen sind strukturierte Datentypen.

- Methoden

Methoden beschreiben Verhalten von Objekten. Sie stellen Algorithmen zur Manipulation des Zustands von Objekten dar. Sie manipulieren Daten eines vorher festgelegten Typs und geben Daten eines festgelegten Typs zurück.

- Java-Klassenbibliotheken

Eine besonders gewünschte Folge des Konzeptes der Objektorientierung ist die einfache Modularisierung umfangreicher Probleme. Es wäre unökonomisch, oft auftretende Teilprobleme immer wieder selbst zu lösen. Für viele in einem Projekt auftretende Probleme gibt es bereits Module, Klassen, die Java in Klassenbibliotheken zusammengefasst hat. Diese Bibliotheken, auch Packages genannt, müssen vor Anwendung importiert werden. So wird für Anwendungen mit graphischer Oberfläche die Bilbliothek java.awt und für Ein-/Ausgaben java.io benötigt.

- Java-Syntax

Die Beachtung der Syntax ist in Java wie in jeder Programmiersprache von besonderer Bedeutung. Der Compiler JAVAC kann Syntaxfehler nicht automatisch beheben (anders als z.B. viele Webbrowser, die Konventionen zur Deutung fehlerhaften HTMLs implementiert haben) JAVAC gibt jedoch Fehlermeldungen aus, die auf Syntaxfehler und die Zeile des Codes, in der diese auftreten, hinweisen. Einige wichtige Konventionen sind: Die Bezeichnung der Klasse muss mit dem Namen der Datei, in der sie gespeichert ist, übereinstimmen. Anweisungen werden mit Semikolon beendet. Anweisungen werden mit geschweiften Klammern verschachtelt. Es gibt Schlüsselwörter, die nicht als Bezeichnung für Variablen und Methoden verwendet werden dürfen, da diese für bestimmte Anwendungen,

z.B. zur Typdeklaration von Variablen, reserviert sind. Der Typ einer Variablen muss vor, spätestens zu einer Zuweisung deklariert werden.

- Kontrollstrukturen

Als Kontrollstruktur kann die bedingte oder wiederholte Ausführung von Code bezeichnet werden. Im engeren Sinne werden als Kontrollstrukturen Schleifen und deren Abbruchsbedingungen verstanden. In Java gibt es Verwirklichungen der vorprüfenden und der nachprüfenden Schleife. Einige Informatiker vertreten jedoch die Auffassung, dass weitere Strukturen zu den Kontrollstrukturen zu rechnen sind.

- Vorprüfende Schleife: Diese Struktur dient zur Wiederholung einer Anweisung bis eine Abbruchbedingung eintritt. Ob diese Abbruchbedingung vorliegt, wird geprüft, bevor die Anweisung ausgeführt wird. In Java hat die vorprüfende Schleife folgende Syntax: „while(Bedingung){Methode};“ bzw. „for(startbelegung laufvariable, abbruchbelegung laufvariable, transformation laufvariable){Anweisung};“

- Nachprüfende Schleife: Diese Struktur dient zur Wiederholung einer Anweisung bis eine Abbruchbedingung eintritt. Ob diese Abbruchbedingung vorliegt, wird geprüft, nachdem die Anweisung einmal ausgeführt wurde. In Javas Syntax wird die nachprüfende Schleife wie folgt geschrieben: „do {Anweisung} while (Bedingung);“ .

- Weitere Kontrollstrukturen: H.P. Gumm und M. Sommer sehen auch folgende Strukturen als Kontrollstrukturen an:

  - Hintereinanderausführung, in Javas Syntax “**;**”Der Code zeigt hier an, dass eine Anweisung beendet ist.
  - Klammern des Programms oder der Prozedur, in Javas Syntax „{Anweisung}“ Der Code zeigt hier Anfang und Ende eines Programms oder einer Prozedur an.
  - Bedingte Anweisung, in Javas Syntax „if(Bedingung){Anweisung}else {Alternative};“ Der Code führt in Abhängigkeit des Wahrheitswertes für eine Bedingung eine Entscheidung zwischen zwei Prozeduren herbei.

[Gumm/Sommer, S. 87]

- Spezifikation, Entwurf, Codierung

Eine Spezifikation ist eine vollständige, detailierte und unzweideutige Problembeschreibung. Dabei heißt eine Spezifikation vollständig, wenn alle Anforderungen und relevanten Rahmenbedingungen angegeben worden sind, detailiert, wenn klar ist, welche Hilfsmittel, insbesondere welche Basisoperationen zur Lösung zugelassen sind und unzweideutig, wenn klare Kriterien angegeben werden, wann eine vorgeschlagene Lösung akzeptabel ist. [Gumm/Sommer, pp. 82-83]
Entwurf ist der Prozess, in dem alle Komponenten des Softwaresystems, das das in der Spezifikation beschriebene Problem lösen soll, festgelegt sowie deren Beziehungen und Schnittstellen untereinander und zur Außenwelt definiert werden. Die in der objektorientierten Programmierung angewandte Entwurfsmethode ist die Objektorientierte Analyse, OOA.
Codierung ist der Prozess des Übersetzens des Entwurfs in die Syntax der Zielsprache. Spezifikation, Entwurf, Codierung, Test und Wartung sind Zyklen, die miteinander verzahnt sind und je nach Notwendigkeit mehrfach durchlaufen werden können.

-Test

Neben den beiden typischen Fehlerarten Syntaxfehler und Wertebereichsüberschreitung, die vom Compiler gefunden werden, können Programme auch logische Fehler aufweisen. Diese können vom Compiler nicht erkannt werden. Es ist für umfangreichere Programme auch nicht möglich, die Richtigkeit der Ausgabe für jede Eingabe mathematisch zu beweisen. Daher muss ein fertiges Programm getestet werden. Damit ein Test unvoreingenommen und zweckdienlich ist, soll er vor der Implementierung des Programms bereits geplant werden. Er soll zeigen
- ob das Programm die gewünschten Ausgaben für jede mögliche Eingabe, insbesondere auch Grenzfälle liefert
- ob es terminiert, falls der Algorithmus dies vorsieht
- ob Speichernutzung und Rechenzeit als akzeptabel geltende Werte nicht überschreiten

Üblicherweise wird die Testphase, je nach dem zur Anwendung kommenden Entwicklungsmodell, mehrfach durchlaufen.

5.2. Didaktische Analyse

In Abschnitt 3 ist ein Teil der didaktischen Analyse vorweggenommen worden, indem gezeigt wurde, dass die vorliegende Planung Zusammensetzung und Vorbildung der Klasse berücksichtigt, sich auf den kommenden Rahmenplan Informatik abstützt und sich Inhalte und Ziele zu eigen macht, die zwischen den zukünftigen Informatiklehrerinnen und -lehrern des Hauptseminars als konsensfähig erwiesen haben. Darüber hinaus erbringt die didaktische Analyse folgende Ergebnisse:

Es wird von vorneherein eine Verschränkung der Sozialformen angestrebt. Wenn sich bei oberflächlicher Betrachtung für den ersten Teil des Kurses bis zum Beginn des Programmierprojekts der Lehrervortrag als hauptsächliche Sozialform anbietet, da zunächst massiv Wissen zu vermitteln sei, so ist dies doch abzulehnen, da hierbei ausschließlich kognitive Fertigkeiten angesprochen werden und bei der Menge neuer Information auch diese schnell überschritten werden dürften. Es ist unbedingt nötig, auch affektive und psychomotorische Fähigkeiten anzusprechen, um die Kompetenzziele zu erreichen. Die erste Woche, in der grundsätzliche Eigenschaften von Objekten erarbeitet werden, wird überwiegend im Plenum bzw. in Gruppenarbeit, jeweils mit dem Lehrer als Moderator absolviert. In der zweiten Woche kommt dann Computerarbeit hinzu, denn das Werkzeug FUJABA soll durch „Learning by Doing“ nutzbar gemacht werden. Der Lehrer ist hier nur mittelbar durch Arbeitsblätter präsent, gegenseitige Hilfestellungen und Gruppenarbeit sind ausdrücklich erwünscht. In den nächsten Wochen wird es kurze einführende Lehrervorträge zu den theoretischen Konzepten „Variablen und Datentypen“, „Methoden“ und „Java-Klassenbibliotheken verwenden“ geben, diese umfassen jedoch höchstens ein Viertel jeder Unterrichtsstunde, darüber hinaus werden diese Konzepte gemeinsam und am Computer durch graphische Umsetzung in FUJABA geübt. Das Projekt wird in Gruppenarbeit durchgeführt, auf Wunsch können einzelne Methoden auch von einzelnen Schülern erarbeitet werden. Begleitet wird das Projekt durch Tutorien. Diese müssen von Schülern bei Schwierigkeiten beantragt werden und werden dann gewährt, wenn die Schüler das Problem, das sie haben, klar umreißen können. Schüler, die ihre Bewertung verbessern wollen, können auch selbst Tutorien geben. Dies schließt leistungsschwächere Schüler nicht von vorneherein aus, da diese auch nach entsprechender Vorbereitung um ein bis zwei Unterrichtsstunden zeitversetzt zur Anfrage gegeben werden können. Der Lehrer moderiert diesen Prozess. Generell ist es schwierig, Lehr- und Leistungsraum klar zu trennen. Wähnen sich Schüler generell in einer Bewertungssituation, werden sie weniger risikofreudig beim Ausprobieren

von Lösungsstrategien und weniger kooperativ im Team sein. Daher denken wir, dass Sozialformen, bei denen der Lehrer sich im Hintergrund hält, auch für bessere Ergebnisse sowie deren bessere Bewertbarkeit sorgen.

Die vorliegende Halbjahresplanung folgt einem Top-Down-Schema, das heißt, ausgehend von allgemeinen Konzepten der Objektorientierung wird eine Problemlösung und schließlich eine Implementierung der Problemlösung durch Schreiben von Quelltext einer konkreten Programmiersprache. Dieser Ansatz stellt den von der Schule geforderten Zukunftsbezug des Unterrichteten her. Bei entsprechend umgekehrter Vorgehensweise bestünde das Risiko, dass die eingesetzte Programmiersprache nach einiger Zeit in der außerschulischen Welt nicht mehr eingesetzt würde, so dass die konkreten Kenntnisse der Syntax nichts mehr wert wären und dass die Konzepte der Objektorientierung als eine Folge einer konkreten Syntax wahrgenommen würden und somit nicht verallgemeinerungsfähig wären. In Abschnitt 8 wird dargestellt, dass der Halbjahresplan auch an den weiterführenden Unterricht in Klasse 12 angeschlossen ist. Darüber hinaus bietet er einigen fächerübergreifenden Nutzen, auch wenn fächerübergreifender Unterricht bei dieser Planung keine hohe Priorität hatte. Zuallererst werden natürlich mathematische Konzepte vermittelt. Um sich den Variablenbegriff der Informatik zu erschließen, ist es hilfreich, eine Vorstellung vom Variablenbegriff in der Mathematik zu haben. In der Graphikprogrammierung sind darüber hinaus Kenntnisse über Funktionen, im weiterführenden Teil auch über Vektorräume von Nutzen oder können an dieser Stelle, falls in Mathematik noch nicht behandelt, vermittelt werden. Wenn es um die Gestaltung graphischer Objekte geht, kann man Fähigkeiten aus der Bildenden Kunst mit einbeziehen. Bei einem Projekt dieser Größenordnung ist als Minimum an Dokumentation eine ausreichende Kommentierung des Quelltextes nötig, was die in Deutsch gefragten Fähigkeiten der Textproduktion fördert. Und die passive Beherrschung der englischen Sprache wird sich durch das Lesen von Dokumentationen auch verbessern. Hier sei insbesondere auf die Java API verwiesen, die von SUN selbst nicht ins Deutsche übertragen wurde.

Folgende Medien werden eingesetzt: Der Kurzfilm „Molly Starracer“ zur Motivation der Mädchen und zur Identifikation von Objekten in einem virtuellen (weil fiktionalen) Kontext, Lego Technics zum „Erfassen“ des Objektbegriffs, der Computer für die Arbeit mit FUJABA, Texteditor, Grafikprogramm und JAVAC sowie zum Nutzen von WWW-Ressourcen, außerdem PDF-Arbeitsblätter, die über das LAN bereitgestellt werden, Tafel.

5.3. Der Stoffverteilungsplan

Die folgende Tabelle zeigt die Wochenplanung des Kurses. Im Anschluss daran wird stundenweise erläutert, welche Methoden und Medien eingesetzt werden, um die Inhalte der jeweiligen Woche vermitteln zu können und die Schüler zu befähigen, die oben genannten Kompetenzen zu erwerben. Die Tabelle zeigt auch, dass eine wirklichkeitsnahe Planung nicht davon ausgehen darf, dass die nominelle Anzahl von Stunden auch tatsächlich zur Verfügung steht. Hier wird davon ausgegangen, dass durch eine Klassenfahrt eine Woche nicht zur Verfügung steht. Um auf nicht vorausplanbare Verzögerungen reagieren zu können, ist eine Woche als Puffer vorgesehen. Leistungsbewertung ist ebenfalls notwendig, daher wird in der vorletzten Woche eine Klausur eingeplant.

☐ Grobplanung:

Wo.	**LE**	**Thema**		**Bemerkungen / zu lesen**
1	1.0	Objekte der realen Welt mit Lego Technics und anhand des Motivationskurz-films, Klassen Vorstellung des Kursziels	Einführung, Analyse	=> Kurzfilm Molly Starracer
2	2	Klassen, Objekte und deren Eigenschaften	FUJABA Einführung	=> LAN: PDF zu Fujaba; vom Lehrer zu erstellen
3	.	Klassen, Objekte und deren Eigenschaften	Variablen und Datentypen	=> LAN: PDF zu Fujaba; vom Lehrer zu erstellen
4		Klassen, Objekte und deren Eigenschaften	Methoden	=> LAN: PDF zu Fujaba; vom Lehrer zu erstellen

5	3	Modularität	Java-Klassenbibliotheken verwenden	JAVA API
6	4	Projektvorschau Übung	Spezifikation / Entwurf	
7	5	Programmierpraxis	Java-Syntax	
8		Programmierpraxis	Kontrollstrukturen	
9	6	Projekt	Spezifikation, Entwurf, Codierung	
10		Projekt	grundsätzlich freie Zeiteinteilung,	permanente Begleitung durch individuelle Tutorien
11		Projekt	aber Codierung erst ca. im letzten Drittel nach Abnahme des Entwurfs	
12	-	-	-	--- Klassenfahrt ------
13		Projekt		
14				
15		Projekt	Testphase	
16		Projekt	Präsentation	
17		**KLAUSUR**		
18		Puffer		

Es wird davon ausgegangen, dass von den drei Stunden pro Woche wenigstens zwei zusammenhängen.

1. Woche:

Zur Motivation wird „Molly Starracer“ gezeigt, ein Kurzfilm im Stile japanischer Manga-Cartoons. Molly Starracer ist Raumschifftechnikerin und wird zur Pilotin in einem intergalaktischen Rennen. Anschließend provoziert der Lehrer mit der Frage: „Können Frauen das? Ist das realistisch?“ Intendiert ist eine kurze Debatte, zu erwarten sind provokante Äußerungen der Kursteilnehmer, nach 10 Minuten soll feststehen, dass Mädchen das können, und folglich können sie auch Programmieren lernen, und zwar mindestens genauso gut wie Jungen.

Der Lehrer befragt die Schüler, welche Gegenstände und Figuren sie in dem Film gesehen hätten. Dabei kann Film nochmals gezeigt werden. Lehrer schreibt als Protokollant die genannten Objekte in einigem Abstand zueinander an die Tafel. Um jedes Objekt kommt eine Box mit Platz nach unten. Dann erfragt der Lehrer Eigenschaften dieser Objekte. Mögliche Antwort: Molly hat schwarze Haare. Lehrer schreibt unter Molly: Haare: schwarz. Wiederholung für weitere Eigenschaften. Lehrer: Was können die einzelnenen Objekte tun? Raumschiff kann schießen, Molly kann lenken. Lehrer steuert mit Fragen, so dass für einige Objekte gleiche Methoden zustande kommen: Kann der Pirat lenken? Feststellung der Schüler: Einige der Objekte haben gleiche Kategorien von Eigenschaften und gleiche Handlungsmöglichkeiten. Lehrervorschlag: Man könnte zur Beschreibung aller Menschen in dem Film einen allgemeinen Menschen namens Mensch erfinden, der auch lenken kann, Haare hat etc. Um z.B. Molly zu beschreiben, bräuchte man nur noch ihre besonderen Eigenschaften aufzuzählen, ihre allgemeinen (hat wie jeder Mensch zwei Beine) und die Dinge, die sie tun kann, nicht. Weitere Betrachtung: Die Klassen stehen wie die zu ihnen gehörigen Objekte zueinander in Beziehung. Ein Raumschiff hat einen Menschen als Besatzung, Mollys Raumschiff hat Molly als Besatzung. Zum Ende der ersten Doppelstunde wird das Ziel des Kurses vorgestellt: Die Programmierung eines Bildschirmspiels, in dem man selbst ein Raumschiff steuert, Kometen und Asteroiden ausweichen oder sie mit der Bordkanone abschießen muss und die anderen Raumschiffe bekämpft, die ihrerseits auf einen schießen. Auf das mögliche Argument, ein solches Vorhaben fördere Gewalttätigkeit wird hier nur mit einem Satz eingegangen: Gewalt findet in diesem Spiel nur auf einer fiktionalen, sehr abstrakten Metaebene statt, Gewaltdarstellungen gibt es völlig unumstritten im Deutsch- und Fremdsprachenunterricht in den behandelten Texten, und falls Schüler selbst an der

Gewaltdarstellung in dem zu programmierenden Spiel Anstoß nehmen, räumen wir Zeit zur Diskussion ein. Im Lehrervortrag wird dargestellt, dass die Programmierung sorgfältige Planung voraussetze und in Form der Modellierung vorweggenommen werden müsse, und dass es dazu nötig sei, alle Objekte zu identifizieren, die in einem solchen Spiel vorkommen. Die dritte Stunde wird mit einer Wiederholung des Objektbegriffs verbracht. Dazu werden Tüten mit Logo Technics ausgegeben. (Es gibt in dieser Serie kleine Tüten, mit deren Inhalt man genau ein Fahrzeug, einen Roboter etc. bauen kann) In diesen Tüten wird jeweils genau ein Teil fehlen. Die Schülerinnen und Schüler sollen nun analysieren, welches Teil fehlt, und zwar nicht über die Inhaltsliste, sondern über die Schnittstellen zu den anderen Teilen. Dazu ist es nötig, das fehlende Teil als Objekt zu beschreiben („müsste rot sein“, „hat 6 Noppen“ etc.) Bei korrekter Beschreibung wird das fehlende Teil ausgehändigt und so eine Belohnung gegeben. Die fertiggestellten Fahrzeuge werden wieder als Objekt betrachtet. Ein Objekt kann also aus Objekten bestehen. Dann werden die Fahrzeuge zerlegt und Klassen von Bauteilen gebildet. Dabei entsteht eine Sortierung wie z.B. 4 Motoren, 16 Räder, 16 Reifen, 28 Steine mit 6 Noppen, 24 Steine mit 8 Noppen usw. Einige der Tüten wird es mehrfach geben. Es gibt offenbar eine Klasse, die z.B. einen Dünenbuggy erzeugt. Indem die Steine nun wieder in Bezug auf ihre Zugehörigkeit zu einem bestimmten Bausatz sortiert werden, erhält man die Elemente, die die jeweilige Klasse benötigt, um ein Objekt zu erzeugen.

2. Woche

Die Eigenschaften von Klassen werden formalisiert. An dieser Stelle wird der Begriff des Konstruktors eingeführt und als Analogie die Bauanleitung für die Legosets benutzt. Aus der recht umfangreichen Bedienungsanleitung von FUJABA destilliert der Lehrer mit Screenshots unterstützte Bedienungsanleitungen für die Erstellung von Diagrammen in FUJABA. Über Beamer und Tafel zeigt der Lehrer die in 5.1. dargestellten Menüs. Als klassische Aufgabe bietet es sich hier an, eine Klasse Auto anlegen zu lassen und darum herum Unterklassen (z.B. Cabrio) oder assoziierte Klassen (z.B. Räder).

3. Woche

Im Klassengespräch wird zurückgegriffen auf das Beispiel des Films aus der ersten Woche, und es wird nochmals über die Eigenschaften gesprochen, die den Objekten zugeordnet werden konnten. Damals wurde ein abstrakter Mensch definiert, von dem alle realen Menschen abgeleitet werden konnten. Diese Eigenschaften können je nach konkretem Menschen verschieden belegt sein, sie sind also variabel. Eine Variable ist also ein Behältnis

für verschiedene in einem bestimmten Kontext in Frage kommende Werte. Nun wird darüber gesprochen, welches Format diese Eigenschaften haben. Wäre es zum Beispiel sinnvoll, für die Anzahl der Beine alle rationalen Zahlen zuzulassen? Sicher gibt es kein Wesen, das 2.5 Beine hat. Daher macht die Beschränkung auf natürliche Zahlen Sinn. Diese Eigenschaft der Variable ist ihr Wertebereich oder Typ. An dieser Stelle erklärt der Lehrer, welcher Wertebereich gemeint ist, wenn eine Eigenschaft als Integer bezeichnet wird. Der Lehrer trägt vor, dass in der Programmierung die Typen der Variable vor deren Benutzung erklärt werden müssen, damit der Computer weiß, wie viel Speicher er reservieren muss. Allen Schülern wird einleuchten, dass ein Behältnis, das Kugeln, die in ganzzahligen Schritten von 1 bis 100 nummeriert sind, aufnehmen soll, kleiner sein kann, als eines, das Kugeln, die von 0,001 bis 100,000 in tausendstel Schritten nummeriert sind, fassen muss. In der folgenden Stunde erläutert der Lehrer alle einfachen Datentypen und deren Wertebereich. In der Arbeit mit FUJABA bekommen nun die Klassen im Automodell Eigenschaften zugewiesen, und die Variablen werden gemäß der Natur der Eigenschaften initialisiert. Um zu testen, ob die Schüler die Wertebereichsgrenzen verinnerlicht haben, bekommen sie folgende Aufgabe:

Gegeben sei folgendes Java-Codefragment:

```
.
int i = 50000;
i *= 50000;
.
```

Welchen Wert hat `i` *nach Ausführung dieser Anweisungen? Erklären Sie dies*
(Dass es sich hier um Java-Code handelt, hat noch keine Bedeutung, lediglich „*=“ muss erklärt werden.)

4. Woche

Hier wird analog der Einführung in der dritten Woche verfahren: Rückgriff auf das Einführungsbeispiel der ersten Woche, dann Formalisierung. Erarbeitung in der Gruppe. Wichtig ist, dass die Schüler erkennen, dass Methoden Datenstrukturen modifizieren und dass ihnen mitgeteilt werden muss, welchen Typs die Ein- und Ausgabewerte sind, selbst wenn nichts ausgegeben wird (*void*).

5. Woche

Diese Woche steht im Zeichen des interdisziplinären Unterrichts. Zunächst wird erläutert, dass man längst nicht alles selbst programmieren muss, da viele Klassen, die man oft braucht, bereits von SUN programmiert wurden und in den Klassenbibliotheken zur Verfügung gestellt wird. Eine vollständige Dokumentation findet man hier. Und auch, wenn man noch keine Zeile Java programmiert hat und die Syntax noch nicht kennt, macht die Auseinandersetzung mit der Dokumentation Sinn. Diese ist hier zu finden:

http://java.sun.com/j2se/1.4.2/docs/api/index.html

Offiziell liegt sie nur auf Englisch vor, also wird dies gleichzeitig eine Übung zum Leseverständnis in Englisch. In Absprache mit dem Englischlehrer kann diese Woche interdisziplinär unterrichtet werden (dadurch gewänne man sogar etwas zusätzliche Zeit).

Anhand folgender Fragen könnten sich die Schüler durch die Java API bewegen:

1. In welchen Packages befinden sich die Klassen StringBuffer und Scrollbar ?
2. Welche Klassen befinden sich im Package 'java.util.prefs'?
3. Welche Interfaces werden im Package 'java.applet' definiert?
4. Welche Klassen erweitert die Klasse Frame (a.k.a. von welchen erbt sie)? Geben Sie die komplette Vererbungshierarchie an.
5. Welche Interfaces implementiert die Klasse MouseMotionAdapter ?
6. Wieviele Konstruktoren hat die Klasse Hashtable ?
7. Wieviele Methoden mit Namen join hat die Klasse Thread ?
8. Von welchem Typ ist der einzige Parameter g der Methode paint der Klasse Canvas ?
9. Was bewirkt der Aufruf der Methode removeShutdownHook ? Zu welcher Klasse gehört sie?
10. Mit welchen Werten werden capacity und load factor vorbelegt, wenn der parameterlose Konstruktor von Hashtable aufgerufen wird?

Zusatzfrage für ganz schnelle:

• Welche Ausnahmen (Exceptions) können die parametrisierten wait-Methoden der Klasse 'java.lang.Object' auslösen (man spricht in Anlehnung an das englische "throw" auch von "auswerfen")?

(Originalaufgaben aus der Übung Praktische Informatik 1, WS 02/03, Prof. Coy; zu einem Zeitpunkt gestellt, als auch die Studenten noch keine Java-Syntax kannten. Dies ist eine Aufgabe zum Bekanntmachen mit der API, zum Aufnehmen der Tatsache, dass die meisten Dokumentationen in der Informatik in Englisch verfasst sind und zum Üben der Recherche in geordneten Datensätzen)

6. Woche

Der Lehrer stellt die einzelnen Stufen eines Entwicklungsmodells, z.B. Wasserfallmodell, vor und erklärt die Abgrenzung. Wichtig: Spezifikation ist noch nicht Modellierung. Anhand eines einfachen, in zwei Stunden zu bewältigenden Problems wird das Erstellen einer Spezifikation und der Entwurf geübt. Eine möglich Problemstellung wäre, aus zwei der Lego-Technics-Bausätze, die eigentlich Autos werden sollten, ein Flugzeug zu bauen. Die Spezifikation umfasst eine Beschreibung der Eigenschaften eines Flugzeugs. Die vorhandenen Steine geben den Rahmen für den Entwurf vor. Lego-Syntax ist im wesentlichen, dass Noppen in Löcher gesteckt werden müssen. Darüber hinaus gibt es in Lego-Technics sowohl Einschränkungen als auch Erweiterungen. Die Modellierung des Entwurfs soll in FUJABA erfolgen.

7. Woche

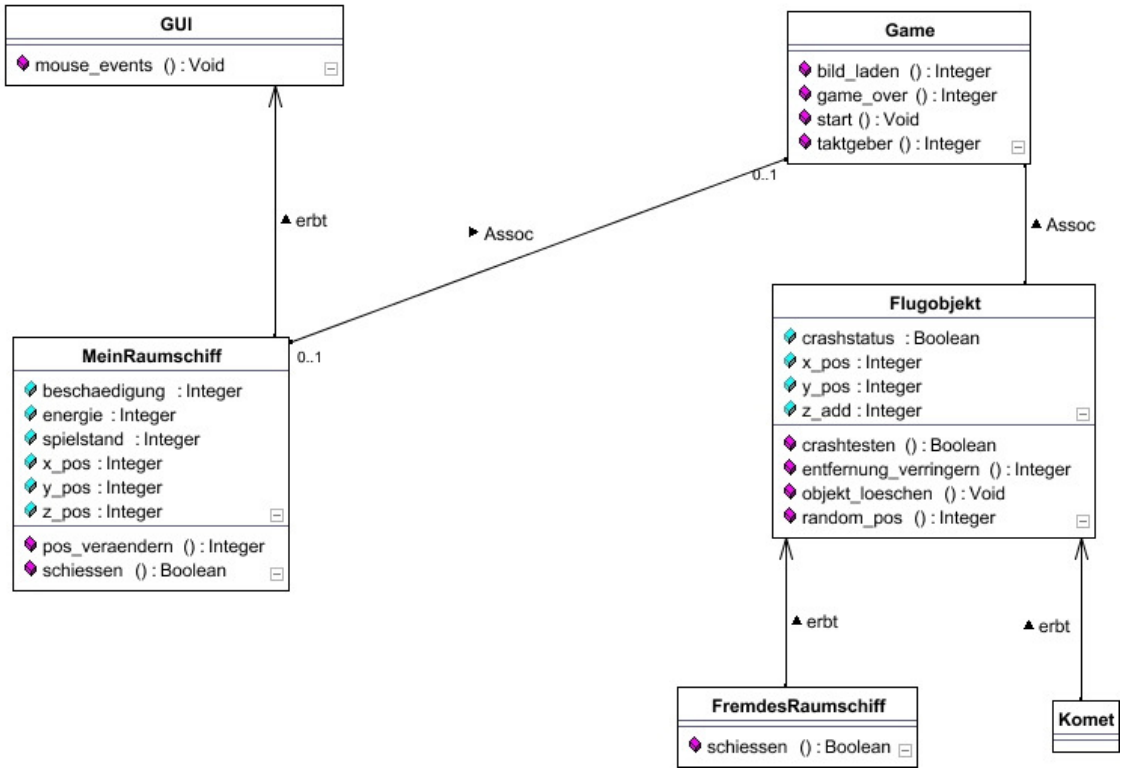
Die Einführung in die Java-Syntax muss weitgehend frontal und am Computer erfolgen. Dazu gehört eine Einweisung in die Benutzung von Texteditor, MS-DOS-Eingabeauffordrung und JAVAC.

8. Woche

Schüler bekommen die Lego-Beutel und die Aufgabe alle roten Teile auszusortieren. Dann sollen sie aufschreiben, wie sie vorgegangen sind. Dabei wird etwas wie das folgende herauskommen: „Ich nehme ein rotes Teil aus der Tüte, schaue, ob noch ein rotes Teil in der Tüte ist, wenn ja, nehme ich es heraus. Diesen Vorgang wiederhole ich, bis ich feststelle, dass kein rotes Teil mehr in der Tüte ist. Nun bin ich fertig“. Lehrer stellt dar, dass man das auch formalisieren kann. Der Schüler hat eine nachprüfende Schleife angewandt. Seine Vorgehensweise wäre gescheitert, wenn kein rotes Teil in der Tüte gewesen wäre, denn er sagte ja: „Ich nehme ein rotes Teil aus der Tüte...“ Im Klassengespräch ergibt sich die nachprüfende Schleife als Alternative. Der Lehrer gibt einen weiteren Algorithmus vor: Nimm ein Teil aus der Tüte, wenn es rot ist, lege es nach links, sonst nach rechts. Wenn kein Teil mehr in der Tüte ist, lege alles, was rechts ist, in die Tüte zurück. Hier wird (neben einer Schleife) die bedingte Alternative verwendet. Anschließend gibt der Lehrer die Syntax für die vorprüfende Schleife (sowohl *while* als auch *for*) und die bedingte Alternative (*if...else*) in Java an. Da nun Syntaxkonstrukte bekannt sind, werden einfache Aufgaben mit Kontrollstrukturen von den Schülern in Java gelöst. (z.B. alle Zahlen von 1 bis 10 ausgeben, nur die Zahlen von 1 bis 100 ausgeben, die durch 7 teilbar sind etc.)

ab 9. Woche:

Die Schüler diskutieren, welche Klassen benötigt werden, um das Spiel zu beschreiben. Wenn die Klassen gefunden sind, müssen die Beziehungen zwischen den Klassen, Methoden und Variablen benannt werden. Die Ergebnisse der Diskussion werden in FUJABA dargestellt. Die Schüler sollen Abläufe anhand der Diagramme durchspielen, um logische Brüche aufzudecken. Ein vorläufiger erster Funktionsentwurf sollte nach ca. 2 Wochen vorliegen und könnte wie folgt aussehen:



Daran muss nun weitergearbeitet werden, insbesondere müssen die verschiedenen Methoden entwickelt werden. An dieser Stelle sollen die Schüler auch Algorithmen entwickeln. z.B. kann ein Flugobjekt feststellen, ob es mit dem Objekt von MeinRaumschiff kollidiert ist, indem die Methode crashtesten den Wert von crashstatus ermittelt. Wenn dieser true ist, dann sind die beiden Objekte zusammengestoßen, und das Objekt von Flugobjekt wird über objekt_loeschen gelöscht. Ein sinnvoller Algorithmus zur Umsetzung von crashtesten wäre:

IF (x_pos[MeinRaumschiff]=x_Pos[Flugobjekt]
and y_pos[MeinRaumschiff]=y_Pos[Flugobjekt]
and z_pos[MeinRaumschiff]=z_Pos[Flugobjekt])

THEN crashstatus=true
ELSE crashstatus=false

Da an dieser Stelle sprachunabhängig modelliert werden soll, wird ein Pseudocode verwendet. Aus diesem Beispiel wird sofort deutlich, dass das Modell erweitert werden muss. Man benötigt mindestens eine Methode in MeinRaumschiff, die die Positionsdaten übergibt. Wenn das Modell fertig ist, moderiert der Lehrer zwischen den Schülern und den Notwendigkeiten der Programmierung. Im fertigen Modell wird es wahrscheinlich Fälle von Mehrfachvererbung geben, welche Java nicht unterstützt, flüssige Grafikprogrammierung erfordert die Benutzung von Threads. Der Lehrer wird die notwendigen Änderungen des

Modells erläutern und die Schüler bitten, dies zunächst hinzunehmen. Nun wird die Codierarbeit verteilt. Die Schüler sollen kleinere Klassen und einige Methoden einer sich abzeichnenden großen Klasse programmieren. Der Lehrer stellt den Rest zur Verfügung.

13. Woche

Nunmehr stehen zwei Wochen zum Codieren zur Verfügung. Die Schüler werden permanent dazu angehalten, den Quelltext auszukommentieren. Auf diese Weise lassen sich bei Problemen schnell die Zeilen finden, die aufgrund eines Denkfehlers das fehlerfreie Compilieren vereiteln. In diesem wie im vorangegangenen Abschnitt stehen der Lehrer und, wo möglich, andere Schüler als Tutoren zur Verfügung. Wenn alle Klassen fehlerfrei compilieren, werden sie in einem Verzeichnis gespeichert, und die gemeinsame Testphase beginnt.

14. Woche

Auch im Zusammenwirken fehlerfrei kompilierbarer Klassen könne Fehler auftreten. In der Regel sind diese logischer Natur. Das Verhalten der graphischen Objekte gibt Hinweise auf die Natur dieser Fehler. Sollte z.B. ein Objekt an immer der selben Stelle des Bildschirms die Crashsequenz aufrufen, aber das optisch eindeutige Ineinanderfliegen zweier Objekte keine Wirkung zeigen, stimmt etwas mit der Implementierung der Methode crashtesten nicht. Eventuell ist aber auch die Methode, die die Position eines der Objekte übergibt fehlerhaft, möglicherweise ist sie so implementiert, dass die Koordinaten nur einmal gesetzt werden und dann den Charakter einer Konstanten haben. Klar ist, dass im Ergebnis des Testens teilweise neu codiert werden muss. Dies geht jetzt jedoch schneller, da nun die ganze (Schul-)Klasse Einblick in eine fehlerhafte Methode nimmt.

15. Woche

In einer Klausur werden die Konzepte, die in der ersten Hälfte des Halbjahres theoretisch erarbeitet und in der zweiten Hälfte praktisch gefestigt wurden, abgeprüft. Anschließend erfolgt eine Evaluation des Kurses durch die Schüler und eine Diskussion, welche Inhalte die Schüler in der 12. Klasse nochmals behandelt wissen wollen.

6. Das Programmbeispiel

Das Programm „Spaceman“ weist folgende Funktionalitäten auf: Nach Start des Spiels ist der eigene Name einzugeben. Dieser wird benötigt, um am Ende des Spiels gegebenenfalls einen neuen Highscore zu speichern. Auf dem Bildschirm befindet sich eine Ansicht des Sternenhimmels, rechten Bildschrimrand befindet sich das eigene Raumschiff. Dieses kann durch Ziehen der Maus auf der y-Achse bewegt werden. Vom linken Bildschirmrand nähern sich Flugobjekte, die zufällig erzeugt werden und entweder Raumschiffe oder Himmelskörper darstellen, von denen es jeweils zwei verschiedene Ausprägungen gibt. Die Flughöhe der Objekte wird ebenfalls zufällig erzeugt. Die Flugkörper bewegen sich nur auf der x-Achse. Nach Start des Spiels ist der eigene Name einzugeben. Am oberen Bildschirmrand befinden sich Anzeigen für die verbleibende Zeit, die verbleibenden Leben und die noch vorhandene Munition. Auf dem Spielfeld ist ein Fadenkreuz zu sehen, das vertikal gemeinsam mit dem eigenen Raumschiff bewegt werden kann und auch, im Gegensatz zum eigenen Raumschiff, horizontal bewegt werden kann. Befindet sich ein Flugobjekt im Fadenkreuz und wird die linke Maustaste betätigt, trifft man das Flugobjekt und zerstört es. Kollidiert man mit einem Flugobjekt oder wird vom Schuss eines Raumschiffes getroffen, verringert sich die Anzahl der Leben. Wenn die Zeit abgelaufen ist, wird aus der Zahl der getroffenen Flugkörper, der Zahl der eigenen noch vorhandenen Leben und der noch vorhandenen Munition der Punktestand entsprechend den einzelnen Objekten zugewiesenen Wertigkeiten errechnet. Als Erstspieler hat man nun einen Highscore erreicht, und der eigene Name wird zusammen mit dem Punktestand in einer Datei gespeichert. Bei jedem weiteren Spiel wird zunächst geprüft, ob man eine höhere als die gespeicherte Punktzahl erreicht hat, bevor gegebenenfalls diese gespeichert wird. Der untere Spielfeldrand weist zudem Buttons mit den Funktionen „Start“, „Pause“ und „Exit“ auf. Ein Hilfesystem wurde ebenfalls integriert. Zur Komplexität: Das Spiel ist am oberen Rand dessen, was im Anfangsunterricht zumutbar ist. Wir haben bereits Abstriche an der Verwirklichung des graphischen Konzepts gemacht. In den ersten Stufen der Modellierung schwebte uns noch dreidimensionale Vektorgrafik vor, dies erschien uns jedoch zu komplex und ist mit dem Wissen eines Schülers der elften Klasse nicht vereinbar. Darüber hinaus hat der Prozess von Spezifikation über Modellierung bis hin zur Codierung etwa 8 Personenarbeitswochen in Anspruch genommen. Geht man davon aus, dass im vorliegenden Konzept 50 Prozent der Codierungsarbeit und im Wege der Korrektur des Schülerentwurfs 30 Prozent der Modellierungsarbeit vom Lehrer geleistet wird, dann müsste eine Klasse von angenommen wenigstens 20 Schülern ein ähnliches Programm erstellen können.

7. Probleme bei der praktischen Umsetzung

Ein grundsätzliches Problem war die anzustrebende Komplexität des Programmierprojekts. Wir waren uns einig, dass ein zu primitives Projekt von Anfang an für mangelnde Motivation bei Schülerinnen und Schülern sorgen würde. Da die Stärke von Java mit seinen umfangreichen Klassenbibliotheken gerade darin besteht, mit relativ wenig Aufwand professionell wirkende Programme zu erstellen, sollte ein ambitioniertes Projekt durchgeführt werden. Die Idee, ein Computerspiel zu programmieren, lag nahe, da dies eine Softwareart ist, die von den meisten Schülerinnen und Schülern selbst sehr gern benutzt wird und daher die Teilnahme an der Programmierung eines solchen ein hohes Sozialprestige in der Gruppe der Gleichaltrigen mit sich bringt. Dies ist eine starke Motivation. Der dabei beschrittene Grat ist jedoch schmal, da eine zu komplexe Aufgabe in der Durchführung schnell Frustration hervorbringen kann. Was uns selbst zu Beginn des Projekts nicht klar war, war die Notwendigkeit der Thread-Programmierung. Dieses Konzept ist eindeutig zu komplex für den Anfangsunterricht, es lässt sich auch nicht aus einem allgemeinen Objektbegriff ableiten. Die Programmierung direkt aus dem ersten Modell mit klarer Klassenstruktur heraus führte zu ruckeliger und langsamer Grafik, die Jugendliche in einem Computerspiel nicht hinzunehmen bereit wären. Der angestrebte professionelle Eindruck wäre verloren gegangen. Daher muss der Thread als gegeben, als eine „Black Box“, die man zu akzeptieren habe, eingeführt werden und entsprechender Code inklusive Anweisungen zur Integration des von Schülern geschriebenen Codes vom Lehrer vorgegeben werden. Dies ist eine Schwäche des Ansatzes, über lange Zeit sprachunabhängig zu modellieren und sich die Syntax der Umsetzungssprache „en passant“ aneignen zu müssen. Der entgegengesetzte Ansatz, das ausschließliche Lernen einer Programmiersprache mit allen syntaktischen Konstrukten, jedoch ohne die zugrunde liegenden Verfahren der Softwareentwicklung, widerspräche dem Allgemeinbildungsgebot der Schule und scheidet somit aus.

Aus der Tatsache, dass in Java Mehrfachvererbung nicht gestattet ist, folgte, dass die meisten Methoden in nur einer Klasse zusammengefasst werden mussten. Der in der Java-Programmierung gängige Weg, Mehrfachvererbung über Instanzenbildung oder Interfaces zu programmieren, wäre für den Anfangsunterricht nicht zu vermitteln gewesen. Daher unterscheidet sich das tatsächliche Klassendiagramm des Beispielprogramms sehr von dem von den Schülern entwickelten Modell. Dies könnte kontraproduktiv für die angestrebte Erfahrung sein, dass sich umfangreiche Softwareprojekte leichter bewältigen lassen, wenn man sie zunächst sprachunabhängig modelliert.

8. Ausblick

Aufgrund seiner modularen Beschaffenheit bietet das vorgestellte Projekt zahlreiche Anknüpfungspunkte für den Unterricht in Klasse 12.

Im Rahmenplan für die Klasse 12 ist das Thema „Grundlagen großer Programmsysteme“ vorgeschrieben. Das Bildschirmspielprojekt steht bereits an der Grenze der Definition eines großen Programmsystems. Hier soll ein Einblick in die professionelle Softwareentwicklung gegeben werden. Da alle Verfahrensmodelle zyklisch aufgebaut sind, kann man hier durchaus auf ein begonnenes Projekt zurückgreifen. Nach den Ferien wird einiges in Vergessenheit geraten sein, und so könnte man mit der Analyse des vorliegenden Programms wesentliche Aspekte der Objektorientierung wiederholen. Unter der Rubrik „Einblicke in wesentliche Anwendungsbereiche der Informatik“ werden Computerspiele explizit genannt. Viele der Konzepte, die im Rahmenplan für die Klasse 12 vorgeschrieben sind, wurden durch unseren Entwurf bereits vorweggenommen und können nun vertieft werden: Datentypen, strukturierte Datentypen, abstrakte Datentypen/ Klassen, Prozedur- und Modulkonzept, Klassen und Pakete, Parameterübergabe, Ereignissteuerung etc.

Das in Klasse 11 erstellte Programm bietet viele Erweiterungsmöglichkeiten:

- Im Zusammenspiel mit dem Mathematikunterricht könnte aus der Steuerung in der Ebene eine 3D-Vektorsteuerung werden. Hier wird das objektorientierte Modell erst richtig überzeugend, denn obwohl sich optisch das Spiel in Gänze ändert, müssen in der Programmierung tatsächlich nur die Darstellung und eine Methode zur Berechnung von Treffern / Kollisionen geändert werden.
- Es wäre schön, wenn man sich im Anschluss an das Spiel in eine Bestenliste eintragen könnte. Diese zu programmieren erfordert die Beschäftigung mit Arrays.
- Eine Multiuser-Steuerung könnte implementiert werden, so dass man in verschiedenen Raumschiffen gegeneinander spielen kann.
- Aus der Applikation könnte ein Applet erzeugt werden.
- Wir haben eine Klasse speed implementiert, die lediglich die Methode sleep der Klasse Thread (aus java.runable) beinhaltet. Mit sleep lassen sich Prozesse für eine definierte Zeit unterbrechen. Das Problem ist, dass sleep von der Systemgeschwindigkeit abhängig ist, und so das Programm auf schnelleren Rechnern auch schneller läuft. Hier zeigt sich ein Vorteil der Objektorientierten Programmierung, denn hier bietet sich eine systemgeschwindig-

keitsunabhängige Implementierung von speed an, ohne dass sonst irgend etwas geändert werden müsste.

Aus dem letzten Punkt ergeben sich weitere Möglichkeiten, die über das Programmierprojekt hinausgehen:
- Das Spiel könnte in ein Website-Projekt eingebunden werden, das neben der Möglichkeit, hier das Spiel online zu spielen, auch eine Dokumentation über die Geschichte der Computerspiele und den Java-Quellcode des Spiels inklusive vieler Kommentare enthält. Dies gäbe wiederum anderen Schülerinnen und Schülern (auch an anderen Schulen) die Möglichkeit, den Quelltext zu analysieren (auch ein möglicher Zugang im Anfangsunterricht), ihn zu verbessern oder neue Klassen dafür zu schreiben.
- Wenn das Spiel als Applet gespielt wurde, könnte der erreichte Punktestand in einer Online-Highscore-Liste in Form einer SQL-Datenbank gespeichert werden. Diese zu erstellen wäre eine Aufgabe für Schülerinnen und Schüler der 12. Klasse mit Schwerpunkt Datenbanken.
- Schülerinnen und Schüler mit Schwerpunkt Computergrafik könnte für eine optisch ansprechende Gestaltung der graphischen Objekte im Spiel sorgen.

9. Quellenverzeichnis

9.1. Print

- Barnes, D.J./Kölling, M.: Objektorientierte Programmierung mit Java, [Pearson Studium] München, 2003
- Gumm, H.P./Sommer, M.: Einführung in die Informatik, [Oldenbourg] München, 2002
- Hubwieser, P.: Didaktik der Informatik, [Springer] Berlin, 2001
- Louis, D./ Müller, P.: Jetzt lerne ich Java, [Markt+Technik Verlag] München, 2001
- Meyer, H.: Leitfaden zur Unterrichtsvorbereitung, [Cornelsen Scriptor] Berlin, 1993

9.2. WWW-Ressourcen

- Skripte und Folien zur Vorlesung Praktische Informatik 1, WS 02/03, HU Berlin, Prof. Coy, W.: http://waste.informatik.hu-berlin.de/Lehre/vorherige/ws02-03/pi1/resources/default.html
- Skripte und Folien zur Vorlesung Praktische Informatik 1, WS 03/04, HU Berlin, Prof. Schlingloff, H.: http://informatik.hu-berlin.de/PI1/vorlesung/index.html
- Materialen zur Einführung in die Objektorientierte Programmierung und Beispiele aus der Schulpraxis, OSZ Handel II Berlin, S. Spolwig et al: http://www.oszhdl.be.schule.de/gymnasium/faecher/informatik/oop/index.htm
- Materialien über die Verbindung von Computerspiel und Fortschritte der Informatik: http://www.osti.gov/accomplishments/ videogame.html und http://www.cgl.uwaterloo.ca/~anicolao/ sw/Spacewar/spacewar.html
- Einführung in UML: http://www.fokus.gmd.de/research/cc/ecco/projects/OKS/WS9899/umlkompt.htm
- Website des Hauptseminars „Objektorientierte Programmierung im Anfangsunterricht, WS 03/04, S. Spolwig mit Grundlagen und Voraussetzungen für diese Arbeit: http://golem14.informatik.hu-berlin.de/spolwig/hsem_03/index.html