Hausarbeit

zum Hauptseminar „Fachdidaktik der Informatik“

Maja Blechschmidt Matrikelnr. 137629

Eileen Hilges Matrikelnr. 157779

1 Einleitung

Ziel dieser Arbeit ist die Erstellung einer Halbjahresplanung zur Einführung in die objektorientierte Programmierung, sowie zwei darin integrierte Unterrichtsentwürfe zu verschiedenen Themen des Halbjahres.

Von folgenden Voraussetzungen wird ausgegangen:
Der Informatikunterricht beginnt in der Sekundarstufe II und ist auf einen dreijährigen Kurs ausgerichtet. Der zeitliche Rahmen umfasst drei Stunden die Woche, wobei 16 Wochen für die Einführung in die objektorientierte Programmierung zur Verfügung stehen. Diese erfolgen mit Beginn des zweiten Halbjahres. Im ersten Halbjahr sind folgende Themen behandelt worden bzw. sollten behandelt worden sein:

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

Es sei angenommen, dass die zu unterrichtende Klasse zu 50% aus Mädchen besteht, die wie auch die Jungen über keine bzw. nur wenige Kenntnisse in der Programmierung verfügen. Weiterhin sei es gegeben, dass fast alle Schüler zu Hause einen PC und gleichfalls einen Internetzugang besitzen. Es besteht jedoch auch für jeden Schüler die Möglichkeit, die Computer in der Schule zur Lösung der Aufgaben zu nutzen.

Die Seminararbeit gliedert sich in zwei Hauptteile. Der erste Teil beschreibt die Halbjahresplanung in Tabellenform, sowie einer etwas ausführlicheren Textform, welche eine Begründung für diesen Einstieg in die objektorientierte Programmierung gibt. Gleichzeitig werden die Lernvoraussetzungen der Schüler geklärt und die Themenauswahl begründet.
Der zweite Teil enthält die zwei Unterrichtsentwürfe, wobei der eine dem Teilgebiet der objektorientierten Analyse entnommen wurde und der andere in das Gebiet der Kontrollstrukturen zu zählen ist.
Im abschließenden 4. Kapitel sind im Anhang der Quellcode für das Projekt und die verwendete Literatur zu finden.

2 Halbjahresplanung

2.1 Objektorientierte Modellierung als Einstieg in die objektorientierte Programmierung

Mit Beginn des zweiten Schulhalbjahres der Einstiegsphase (11. Klasse) sollen die Konzepte der objektorientierten Modellierung (OOM) erklärt werden und anhand dieser in die objektorientierte Programmierung eingeführt werden. In der Kursphase werden diese Konzepte dann später vertieft und ausgebaut.

Der Einstieg in das Thema wird mit der objektorientierten Analyse erfolgen. Dabei werden am Anfang die wichtigsten Begriffe, wie Objekte, Instanzen und Klassen unter Zuhilfenahme des life^3-Phasenmodells erläutert. Die Schüler sind bis zu diesem Zeitpunkt noch nicht mit der objektorientierten Modellierung in Berührung gekommen.
Das Grundkonzept der OOM besteht aus drei Bereichen: der Analyse, dem Entwurf und der Implementierung. Diese werden im Verlauf des Schuljahres in vorgegebener Reihenfolge bearbeitet. Vor Beginn des Projekts, welches das Abschlussziel dieses Halbjahres darstellt, wird die Einführung der Kontrollstrukturen eine wichtige Rolle spielen.

Die drei Teilbereiche der OOM sind von entscheidender Bedeutung für den Informatikunterricht, da diese in der Wirtschaft und im täglichen Leben angewendet werden. Wird zum Beispiel ein Softwareunternehmen mit der Entwicklung einer Software beauftragt, gehen der Programmierung der Software mehrere Entwicklungsphasen voraus. Diese unterteilen sich in Analyse, Design und Entwurf mit folgender Implementierung und Testphase. Mit Bestehen der Software sind die Aspekte des Einsatzes und der Wartung nicht zu vernachlässigen. Somit wird deutlich, dass die objektorientierte Analyse und der objektorientierte Entwurf von entscheidender Bedeutung für die objektorientierte Programmierung sind, da sie die Basis für eine gute objektorientierte Problemlösung darstellen.
Aus diesem Grund sollen die Aspekte in einem verkleinerten Maßstab im Informatikunterricht in der Anfangsphase aufgegriffen werden, da hier die reale Welt als Vorbild dienen soll.

Zu Beginn beschäftigt sich die Analyse mit der Identifizierung von Objekten und Klassen. Anschließend sollen die entsprechenden Attribute und Methoden mit Hilfe von Objektkarten festgehalten werden und durch ein Objektspiel, bei dem die Schüler selbst als Objekte auftreten sollen, gefestigt werden. Eine Umwandlung der Karten zu UML-Diagrammen ermöglicht es den Schülern, die Beziehungen zwischen

einzelnen Objekten sowie die Bedeutung der Schaffung von klaren Strukturen und den groben Ablauf der Algorithmen bereits vor Beginn der Implementierung zu erfassen.

Das Anlegen von Klassen sowie das Erzeugen von Objekten und die Änderung ihrer Attributwerte werden mit Hilfe von BlueJ erfolgen.
Die Einführung der Kontrollstrukturen soll mit Hilfe von JavaKara geschehen oder ersatzweise auch einer angepassten Oberfläche in BlueJ, damit die Schüler nicht ihre gewohnte Programmierumgebung ändern müssen.

2.2 Begründete Stoffauswahl

Die verwendeten Inhalte können ebenso unter 2.4 nachgelesen werden.

Das zu planende Halbjahr soll ausschließlich als Einführung in die objektorientierte Programmierung dienen, daher wird auf einige Gebiete der Objektorientierung nicht näher eingegangen, so zum Beispiel die Datenkapselung und der Polymorphismus.

Die objektorientierte *Analyse* soll lediglich zum Verständnis von Objekten und Klassen mit ihren zugehörigen Eigenschaften und Verhalten beitragen. Es wird lediglich auf einige Beziehungen zwischen den Klassen eingegangen, wie zum Beispiel der Vererbung.
Im objektorientierten *Entwurf* sollen Klassendiagramme modelliert werden, wobei auf Sequenzdiagramme und Zustandsdiagramme aus zeitlichen Gründen vorerst verzichtet wird, aber eine Ergänzung zum späteren Zeitpunkt des Kursverlaufs möglich ist.
Die objektorientierte *Implementierung* stützt sich hier größtenteils auf die einfachen Grundstrukturen einer jeden Programmiersprache sowie den Kontrollstrukturen und den spezifischen Eigenschaften der objektorientierten Umsetzung der gewählten Programmiersprache, in diesem Fall Java.

Um dem zeitlichen Rahmen des Halbjahres Rechnung zu tragen, wird auf die Verwendung von Arrays in dem hier geplanten Halbjahr verzichtet. Obgleich das abschließende Projekt „Vier gewinnt“ Arrays verwendet, soll aufgrund der Komplexität eine vertiefende Einführung erst im späteren Kursverlauf folgen. Die Verwendung von Sortier- und Suchalgorithmen spielt für das Projekt keine Rolle, daher werden auch sie erst zu einem späteren Zeitpunkt behandelt. Der Gebrauch von grafischen Oberflächen stellt gerade bei Java im Vergleich zu anderen Programmiersprachen eine größere Herausforderung dar, weil eine Umsetzung erst mit Hilfe des Verständnisses von Klassen und unter Verwendung von Bibliotheken

möglich ist. Aber als thematische Weiterführung und Ergänzung des Projekts wäre die grafische Oberfläche ein guter Anschluss im darauf folgenden Schuljahr, um die Objektorientierung fortzusetzen und zu vertiefen.

Die verschiedenen Möglichkeiten des Tools BlueJ werden innerhalb dieses Halbjahres nicht ausgeschöpft, was auch nicht Ziel des Unterrichts ist. Die vielfältigen Verwendungsmöglichkeiten von BlueJ können jedoch auch in den folgenden Schuljahren vertieft werden.

2.3 Übersicht

Objektorientierung

Woche	**Thema**	**Inhalt**	**Medien**
1	A. Objektorientierte Analyse …	1. Objekt	Objektkarten Klassenkarten
		a) Zustand (Attribute)	
2	… und Entwurf	b) Verhalten (Operationen) c) Objektidentität	Objektspiel UML-Diagramme
		2. Klasse	
3	B. Informationsspeicherung	Übungen mit den UML-Diagrammen	
		1. Variablen 2. Primitive Datentypen	
4	C. Objektorientierte Implementierung	1. Klassen und Objekte in Java a) Anlegen einer Klasse b) Deklaration eines Objektes	BlueJ
		c) Erzeugen von Objekten d) Ändern von Attributwerten	
5	D. Eigenschaften der Objektorientierung	2. Methoden in Java a) Erstellen von Methoden b) Aufruf von Methoden	
		1. Vererbung	
6		2. Mehrfachvererbung	
		Übung/Klausur	

Kontrollstrukturen

Woche	Thema	Inhalt	Medien
7	E. Kontrollstrukturen	1. Verzweigungen a) Bedingte Anweisung	JavaKara (oder BlueJ mit grafischer Oberfläche)
		b) Bewachte Anweisung	
8		2. Wiederholungsstrukturen a) vorprüfende Schleife	
		b) nachprüfende Schleife	
9		c) Zählschleife	
		Übungen mit den Schleifen und den Struktogrammen	

Projekt

Woche	Thema	Inhalt	Medien
10 bis 16	F. Projekt „Vier gewinnt"	Objektanalyse, Objektentwurf und Implementierung in Java	(Objektkarten) UML-Diagramme BlueJ
		Klausur	

2.4 Geplanter Verlauf des Schulhalbjahres

Ein Überblick über den Verlauf des Halbjahres konnte bereits der Übersicht entnommen werden. Dennoch sollen die einzelnen Phasen an dieser Stelle etwas ausführlicher erläutert werden.

Der Lehrplan soll in der ersten Hälfte des Halbjahres die Grundlagen der objektorientierten Analyse, des objektorientierten Entwurfs und der objektorientierten Implementierung schaffen, so dass während der Projektphase am Ende der zweiten Hälfte das Analysieren und Problemlösen, sowie das Implementieren eines eigenen Spieles unter Verwendung von Kontrollstrukturen möglich ist.

Die Erklärung des Begriffes „Objekt" bildet den Anfang der objektorientierten Analyse. Mit Hilfe der Objektkarten sollen die Schüler spielerisch das Wesen eines

Objekts begreifen und den Unterschied zwischen einem allgemeinen Objekt und einer speziellen Instanz erkennen. Gleichzeitig sollen sie den Zustand, das Verhalten und die Identität eines Objekts verstehen. Dazu wird das Objektspiel durchgeführt, um weitere Fragen zu klären und das Verständnis für den Arbeitsgegenstand zu unterstützen.
Das Objektspiel ist ein Rollenspiel, in welchem jeder Schüler die Identität eines Objektes annimmt und somit die objekteigenen Spezifikationen begreifen und die objektfremden Eigenschaften unterscheiden lernen soll. Ziel ist es, dass die Schüler die Begriffe Klasse, Objekt und Instanz und deren Unterschiede inhaltlich verstehen.
Das Objektspiel wird einerseits als Ergänzung zu den Objektkarten und andererseits zur Erklärung des Zusammenspiels der einzelnen Objekte und deren Eigenschaften sowie zur Einführung von Klassenkarten verwendet. Die Schüler erhalten dazu eine Klassenkarte mit der Charakteristik der Klasse des Objekts und eine Objektkarte, auf welcher der jeweilige Zustand des Objektes festgehalten wird. Im Laufe des Spiels müssen mehrere Objekte miteinander kommunizieren und verändern dabei ihren Zustand.
Dies soll eine Grundlage dafür schaffen, die Objektkarten und Klassenkarten im folgenden in UML-Diagramme umzuwandeln. Des Weiteren wiederholt sich eine ähnliche Darstellung der erzeugten Objekte bei BlueJ.
Mit diesem Schritt wird die Phase des objektorientierten Entwurfs eingeleitet. Auf die Verwendung UML-Tools zur Umsetzung von UML-Diagrammen soll verzichtet werden. Stattdessen werden diese mit Papier und Bleistift erarbeitet.
Dabei sollen die verschiedenen Beziehungstypen wie Vererbung, Aggregation und Assoziation kurz vorgestellt werden und eine intensivere Betrachtung zum geeigneten Zeitpunkt erfolgen.

Mit der Modellierung von Objekten und Klassen und deren zugehörigen Eigenschaften wird bei der Umsetzung in der Programmierumgebung das Problem des Variablenkonzeptes und der dazugehörigen Datentypen aufgeworfen. Dies und mehr wird im Abschnitt der Informationsspeicherung behandelt. Dabei sollen unter anderem die primitiven Datentypen und erste elementare Anweisungen eingeführt werden. Anschließend werden diese in das vorhandene UML-Diagramm eingefügt.

Im Teilbereich der objektorientierten Implementierung geht es um das Anlegen einer Klasse, das Erzeugen einzelner Objekte und deren Deklaration, sowie das Ändern von Attributwerten. Bevor in die Programmierung eingestiegen werden kann, werden damit zusammenhängende Begriffe wie Compiler und Interpreter erläutert. Im Zuge dieser Einführung soll das Umgebungstool BlueJ vorgestellt werden, mit welchem anschließend gearbeitet wird. Jedoch soll die Einführung des Tools auf die relevanten Funktionen beschränkt werden. Weitere Möglichkeiten von BlueJ sollten zum entsprechenden Zeitpunkt ergänzt werden.

Anschließend sollen die Operationen von Objekten (Methoden) in BlueJ erzeugt und aufgerufen werden. Bevor zu den Kontrollstrukturen übergegangen wird, erhalten die Schüler eine vertiefende Erläuterung zu den Eigenschaften der Objektorientierung, wie Vererbung und Mehrfachvererbung.

Ein wichtiger Punkt zur erfolgreichen Bewältigung des Projekts am Ende des Schuljahres besteht in der Einführung der dafür notwendigen Kontrollstrukturen. Aus diesem Grund ist es wichtig, dass in einer Unterrichtsstunde am Beginn dieser Phase mit den Schülern der Algorithmusbegriff geklärt wird.
Zur Motivation sollen die Kontrollstrukturen entweder in BlueJ mit grafischer Oberfläche oder mit JavaKara behandelt werden. Im Speziellen werden dabei die Verzweigungen und Wiederholungsstrukturen behandelt. Die Einführung von Struktogrammen und ihre Erstellung zu den verschiedenen Bedingungen und Schleifen stellt eine wichtige Voraussetzung zur eigenen Handhabung während der Projektphase dar. Bei der Bearbeitung des Projekts sollen die Schüler diese Strukturen selbständig erkennen und entsprechend anwenden können.

Der Projektphase wird ein verhältnismäßig großer zeitlicher Abschnitt zugeteilt, in dem es um die Problemanalyse, die Entwicklung von Lösungsansätzen und die Umsetzung in Programmcode des Spiels „Vier gewinnt“ geht.
Am Anfang wird das Projekt vorgestellt und eine Objektanalyse durchgeführt, wobei von den Schülern ein vollständiges UML-Diagramm erstellt wird. Dieses wird nach einer kritischen Analyse und erneuten Überarbeitung als Entwurf für einen Lösungsansatz festgehalten. Mit Hilfe von BlueJ soll dann ein Großteil des Programms umgesetzt werden. Die Schüler sollen weniger umfangreiche Probleme mit Hilfe der bisher gelernten Inhalte lösen, wodurch auch eine regelmäßige Wiederholung der erlernten Strukturen gewährleistet wird. Da der zeitliche Aufwand für die Implementierung des ganzen Projektes einen allzu großen Raum einnehmen würde, wird ein Teil des Programmpaketes vorgegeben, unter anderem auch die Umsetzung des Spielfeldes durch Arrays.

3 Stundenentwürfe

3.1 Unterrichtsentwurf: Objekte, Attribute und Instanzen

3.1.1 Thema der Unterrichtseinheit

Zum Beginn der objektorientierten Programmierung sind keine besonderen Begriffe oder Fähigkeiten vorauszusetzen. Lediglich der Umgang mit alltäglichen Gegenständen und Kenntnisse über deren Aussehen und Verhalten sind notwendig, was jedoch keiner weiteren Überprüfung bedarf.
Datenbanken arbeiten ebenfalls mit Objekten und ihren Attributen, so dass hier die Möglichkeit zur Verknüpfung von inhaltlichen Ideen besteht.
Die vorliegende Unterrichtsstunde ist die erste Stunde der objektorientierten Programmierung. Diese stellt eine von vier möglichen Ansätzen zur Einführung der Programmierung dar. Diese Unterrichtstunde findet im zweiten Halbjahr des ersten Informatikkurses statt, wobei im ersten Halbjahr einführende Bereiche der Informatik behandelt werden, so zum Beispiel Datenstrukturen, Datenschutz, Datenbanksysteme, Geschichte der Informatik etc.
Im Anschluss an diese Stunde werden an verschiedenen Objekten die neuen Begriffe im Objektspiel angewendet und gefestigt. Da Objekte die grundlegende Idee dieser Art von Programmierung darstellen, bilden sie die Basis für den weiteren Unterrichtsablauf.

3.1.2 Sachanalyse

<u>Themenüberblick der Informatik</u>

- theoretische (Komplexität, Sortieralgorithmen, Suchalgorithmen, Graphentheorie, Logik, etc.)
- praktische (Software-Entwicklung, Software-Engineering, Datenbanksysteme, etc.)
- technische (Zahlensysteme, Computertechnik, Schaltungslogik, etc.)
- angewandte (Geschichte, Datenschutz, etc.)

<u>Einordnung der Programmiersprachen</u>

- imperative
- funktionale
- logische (wissensbasiert)
- objektorientierte

Begriffe

Ein *Objekt* ist allgemein ein Gegenstand des Interesses, insbesondere einer Beobachtung, Untersuchung oder Messung. In der objektorientierten Software-Entwicklung ist ein Objekt ein individuelles Exemplar von Dingen, Personen oder Begriffen der realen Welt oder der Vorstellungswelt.
Ein Objekt besitzt *festgelegte Eigenschaften* und reagiert mit einem *definierten Verhalten* auf seine Umgebung. Außerdem besitzt jedes Objekt eine Identität, die es von allen anderen Objekten unterscheidet.

Die *Objektidentität* ist die Eigenschaft, die ein Objekt von allen anderen Objekten unterscheidet. Aufgrund ihrer Existenz sind alle Objekte unterscheidbar, auch wenn sie zufällig identische Attributwerte besitzen. Die Identität eines Objektes kann sich ändern.
Ein konkretes Objekt wird auch als *Instanz* bezeichnet.

Die Eigenschaften eines Objektes werden durch seine *Attributwerte* bestimmt, sein Verhalten durch eine Menge von *Operationen*. Die *Attribute* beschreiben die Daten bzw. die Eigenschaften einer Klasse. Alle Objekte einer Klasse besitzen dieselben Attribute, jedoch unterschiedliche Attributwerte.

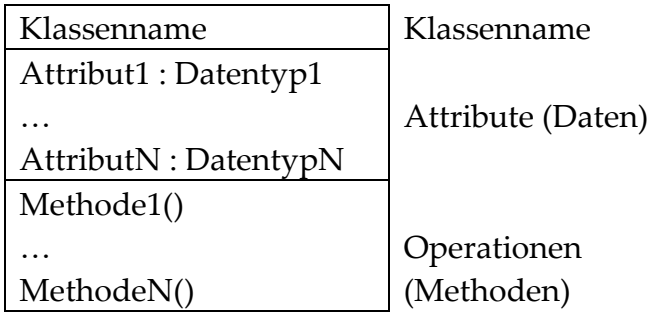
Objektoperationen, kurz *Operationen* genannt, beschreiben die Dienstleistung, die ein Objekt seiner Umgebung zur Verfügung stellt. Sie legen die Funktionalität bzw. das Verhalten des Objekts fest. Eine Operation ist eine ausführbare Tätigkeit im Sinne eines Algorithmus. Sie kommuniziert mit dem aufrufenden Objekt über Ein- und Ausgabeparameter.

Eine *Klasse* ist allgemein eine Gruppe von Dingen, Lebewesen oder Begriffe mit gemeinsamen Merkmalen. In der objektorientierten Welt spezifiziert eine Klasse die Gemeinsamkeit einer Menge von Objekten mit denselben Eigenschaften und demselben Verhalten. Eine Klasse besitzt einen Mechanismus, um Objekte zu erzeugen, der sogenannte Konstruktor[1].

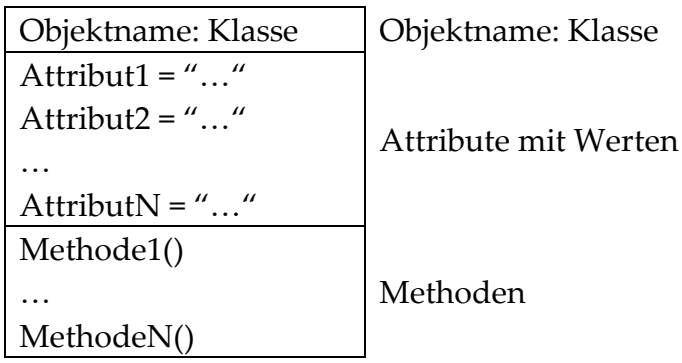
[1] Balzert, H.: *Lehrbuch Grundlagen der Informatik*. Konzepte, Notation in UML, Java, C++, Algorithmik und Software-Technik, Heidelberg – Berlin – Oxford, 1999, S. 106f.

UML-Diagramme

UML (Unified Modeling Language)[2] – Notation zur grafischen Darstellung objektorientierter Konzepte. Zur grafischen Darstellung gehören Klassendiagramme[3], Objektdiagramme[4], Kollaborationsdiagramme[5] und Sequenzdiagramme[6].



Instanz



[2] vgl. Balzert, H., S. 123.
[3] stellt die objektorientierten Konzepte Klasse, Attribute, Operationen und Beziehungen zwischen Klassen in grafischer Form dar
[4] grafische Darstellung von Objekten und ihren Verbindungen; erlaubt eine Momentaufnahme eines laufenden Programms
[5] Erweiterung des Objektdiagramms um Botschaften; durch eine hierarchische Nummerierung, die Angabe des Operationsnamens und der Botschaftsrichtung durch einen Pfeil sind Ablaufsequenzen darstellbar
[6] grafisch, zeitbasierte Darstellung mit vertikaler Zeitachse von Botschaften zwischen Objekten und Klassen; Botschaften werden durch horizontale Linien, Objekte und Klassen durch gestrichelte Linien präsentiert

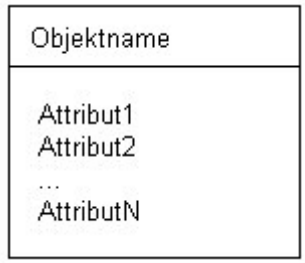
Umsetzung in Java

```
class Klassenname
{
//Attribute
   Datentyp Attributname = Anfangswert;

//Konstruktor
   Klassenname(Parameterliste)
  {
   …
   }

//Methoden
    Rückgabetyp Methodenname(Parameterliste)
   {
    …
    }
}
```

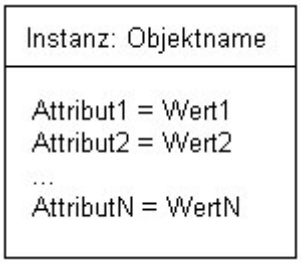
Karten

Die folgenden Karten sollen zur Grundlage der UML-Klassendiagramme dienen, welche im Anschluss daran folgen. Diese Karten enthalten jedoch noch keine Übersicht über die verwendeten Datentypen. Die erweiterten Karten werden durch die Operationen ergänzt.

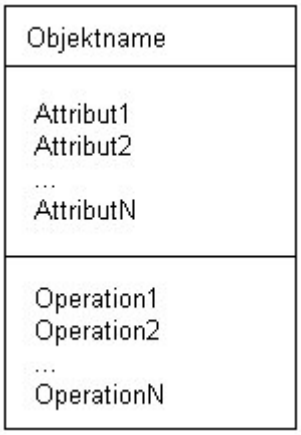
Klassenkarten



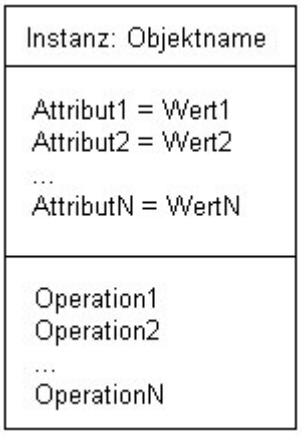
Objektkarten



erweiterte Klassenkarte



erweiterte Objektkarte



3.1.3 Unterrichtsziele

Grobziele

Die Schüler sollen

- Objekte, ihre Attribute und die zugehörigen Klassen an gegebenen Beispielen bestimmen.

Feinziele

Die Schüler sollen

- die zugehörigen Attribute eines Objektes bestimmen können.
- anhand von Instanzen die zugehörigen Klassen bestimmen und die Objekte beschreiben können
- den Unterschied von Objekt und Instanz darlegen können.

3.1.4 Didaktisch-methodische Überlegungen

Falls beim Thema Datenbanken die Begriffe Objekt, Instanz und Attribute schon geklärt wurden, kann hier eine Verknüpfung mit dem neuen Themenabschnitt in Betracht gezogen werden.
Dennoch ist es sinnvoll, die neuen Begriffe durch Erklärungen mit Inhalt zu füllen und an verschiedenen Beispielen zu erläutern.

Umsetzung

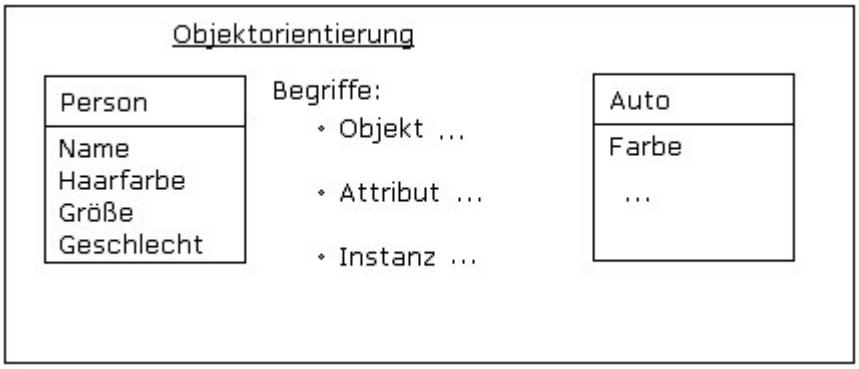
Phase I: (Einführungsphase) In dieser Phase soll ein Überblick über die verschiedenen Themen der Informatik gegeben werden und die Einordnung der folgenden Stunden in dieses Schema.
Phase II: (Erarbeitungsphase) Zusammen mit den Schülern sollen gemeinsam die Attribute eines bestimmten Objektes (hier: Person) bestimmt werden, und eine Besprechung der Unterschiede von Objekten im Allgemeinen und Instanzen im Konkreten stattfinden.
Phase III: (Wiederholungsphase) Da im vorangegangenem Halbjahr bereits Datenbanken behandelt wurden, soll hier eine Verknüpfung mit den bereits bekannten Begriffen der Entität und Attribut stattfinden.
Phase IV: (Übungsphase) Die Schüler sollen nun selbständig die Eigenschaften eines weiteren Objektes (hier: Auto) in Gruppenarbeit ermitteln.
Phase V: (Festigungsphase) Am Ende der Stunde sollen noch einmal die neuen Begriffe und der Unterschied von Objekt und Instanz wiederholt werden, wobei noch bestehende Fragen geklärt werden sollen.

3.1.5 Unterrichtsverlauf

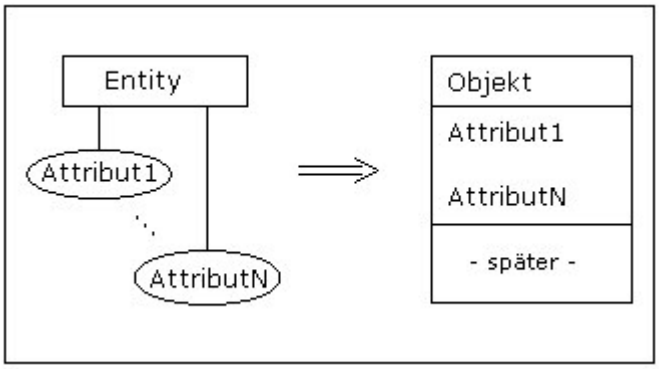
Zeit	**Phase**	**Inhalt**	**Unterrichtsform/ Medien**
2 Min		- Begrüßung, Organisation	
3 Min	Phase I	- Themenüberblick der Informatik - Einordnung der Programmiersprachen	Lehrervortrag
10 Min	Phase II	- Klärung der Begriffe Objekt und Attribut - Bestimmung der Attribute einer Person	Unterrichtsgespräch
5 Min		- Bestimmung der Attributwerte einer bestimmten Person (Instanz) - Klärung des Begriffs Instanz	Unterrichtsgespräch
5 Min	Phase III	- Wiederholung von Entitäten und Attributen - Entitätsidentität	Folie 1
5 Min		- Einführung der Klassen- und Objektkarten	Lehrervortrag Klassenkarten Objektkarten
5 Min		- Verteilung von Spielzeugautos - Wiederholung der Begriffe Objekt und Instanz sowie den Unterschied durch die Schüler	Unterrichtsgespräch Spielzeugautos
5 Min	Phase IV	- Ermittlung der Attribute des Objektes Auto - Notieren dieser Attribute auf Klassenkarten - Ermittlung der Attributwerte der eigenen Instanz - Notieren der Attribute auf Objektkarten	Partnerarbeit Spielzeugautos Klassenkarten Objektkarten
2 Min		- Zusammentragen der ermittelten Attribute	Klassengespräch Tafelanschrift
3 Min	Phase V	- Wiederholung der neuen Begriffe - Klärung von Fragen	

3.1.6 Unterrichtsmaterialien

Tafelanschrift



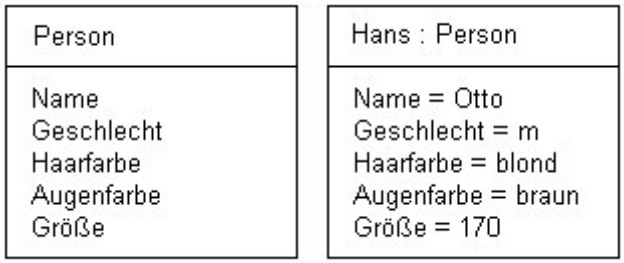
Folie 1 – Beziehung von Entitäten und Objekten



Karten

Auto
Modell Farbe Tank fahrtüchtig Kilometerstand

Ferrari : Auto
Modell = Ferrari Farbe = rot Tank = Diesel fahrtüchtig = ja Kilometerstand = 1000



Spielzeugautos

Die Spielzeugautos haben die Eigenschaft, dass sie verschiedene Automarken und unterschiedliches Aussehen aufweisen, so dass möglichst zahlreiche Attribute durch die Schüler ermittelt werden können.

3.2 Unterrichtsentwurf: Einführung von bedingter und bewachter Anweisung

3.2.1 Thema der Unterrichtseinheit

Nachdem die Einführung in die Objektorientierung abgeschlossen ist, werden die Schüler jetzt mit dem Gebiet der Kontrollstrukturen vertraut gemacht. Am Anfang soll die Erklärung von Bedingungen stehen, die erst allgemein an Struktogrammen veranschaulicht und anschließend konkret in einem kleinen Java-Programm umgesetzt werden sollen. Die Programmierumgebung wird JavaKara sein, die schon eine fertige grafische Oberfläche mitbringt.
Begriffe wie Algorithmus, Compiler und Interpreter, sind bereits in den vorangegangenen Stunden geklärt worden. Dennoch sollte die Zeit genutzt werden, um den Terminus des Algorithmus zu wiederholen, da Bedingungen und Schleifen als Möglichkeit zur Lenkung des Ablaufes zu seinen Hauptbestandteilen zählen.
Die Theorie soll an der Umsetzung einer kleinen Formel geübt werden, wobei die Verwendung der bedingten Anweisung (*if*-Anweisung) im Vordergrund stehen soll. Als weiterführende Variante kann die Umsetzung des gleichen Programms mit einer bewachten Anweisung (*switch*-Anweisung) erfolgen[7].

3.2.2 Sachanalyse

Algorithmus

Ein Algorithmus ist ein Plan zur Lösung eines Problems mit Hilfe einer Folge von eindeutigen, endlichen, ausführbaren Schritten. Diese werden sequentiell abgearbeitet und ausgeführt.
Für einen Algorithmus sind zwei mächtige Werkzeuge charakteristisch: Schleifen, mit deren Hilfe der Computer Wiederholungen von bestimmten Schritten effektiv und übersichtlich ausführen kann, und Bedingungen, mit deren Hilfe in mehrere Fälle unterschieden werden kann.

Struktogramm

In einem Struktogramm werden die nötigen Sequenzen mit Hilfe von Strukturblöcken dargestellt, die von oben nach unten abgearbeitet werden. Dieses Verfahren bringt den Vorteil mit sich, dass Algorithmen unabhängig von

[7] Die Umsetzung einer bewachten Anweisung wird in der vorbereiten Stunde nicht erfolgen, dennoch soll in der Sachanalyse bereits daraufvorbereitet werden.

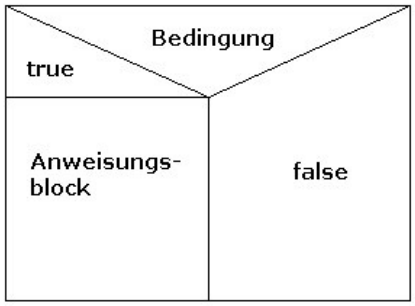
spezifischen Programmiersprachen abgebildet werden können. Auf diese Weise bilden Struktogramme eine wichtige Grundlage für die Umsetzung von Algorithmen von der natürlichen Sprache zum fertigen Programm.

Bedingte Anweisung

Die bedingte Anweisung, die in Java mit dem Schlüsselwort *if* eingeleitet wird, gliedert sich in zwei Teile: erstens die zu überprüfende Bedingung und zweitens eine Anweisung bzw. ein Anweisungsblock, die ausgeführt werden, wenn die Bedingung erfüllt ist. Wenn nötig können andere Anweisungen ausgeführt werden, falls sich die Bedingung als falsch herausstellt. Diese werden durch das Schlüsselwort *else* initiiert. Bei der Überprüfung der Bedingung muss daraufgeachtet werden, dass die Vergleichsoperatoren sowie logischen Operatoren und deren Anwendung den Schülern bekannt sind.

a) Auswahl ohne Alternative

- Struktogramm



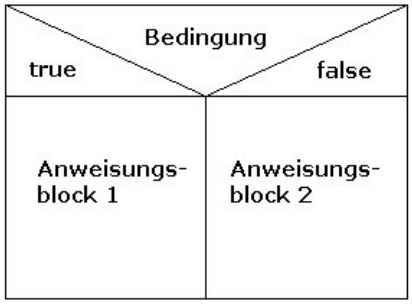
- wird die **Bedingung** mit *true* ausgewertet, wird der **Anweisungsblock** ausgeführt, sonst wird die bedingte Anweisung verlassen
- die **Bedingung** muss bei der Auswertung einen Wert des Typs *boolean* liefern
- der **Anweisungsblock** umfaßt eine Folge von Anweisungen und Ausdrücken

- Umsetzung in Java

```
if (Bedingung)
{
    Anweisungsblock
}
```

b) Auswahl mit Alternative

- Struktogramm



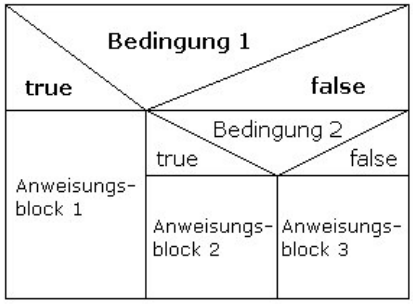
- wird die **Bedingung** mit *true* ausgewertet, wird der **Anweisungsblock 1** ausgeführt, falls die **Bedingung** *false* liefert, wird der **Anweisungsblock 1** übersprungen und der **Anweisungsblock 2** bearbeitet
- die bedingte Anweisung wird anschließend verlassen

- Umsetzung in Java

```
if (Bedingung)
{
    Anweisungsblock 1
}
else
{
    Anweisungsblock 2
}
```

c) Mehrfachalternative

- Struktogramm



- der Anweisungsblock in einer bedingten Anweisung kann selber wieder eine bedingte Anweisung enthalten

- Umsetzung in Java

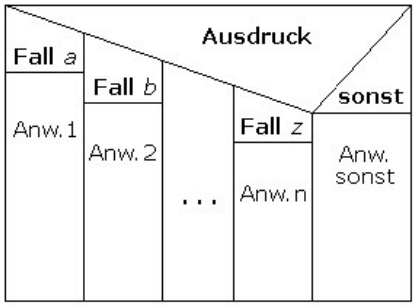
```
if (Bedingung 1)
{
    Anweisungsblock 1
}
else
{
    if (Bedingung 2)
    {
        Anweisungsblock 2
    }
    else
    {
        Anweisungsblock 3
    }
}
```

Bewachte Anweisung

Bei der bewachten Anweisung handelt es sich um eine Mehrfachverzweigung, die in Java durch das Schlüsselwort *switch* eingeleitet wird. Diese kann bei mehreren

Wahlmöglichkeiten für eine Variable mehrere *if*-Anweisungen ersetzen. Der Programmcode kann sich dadurch übersichtlicher gestalten.

- Struktogramm



- der **Ausdruck** wird ausgewertet
- stimmt der Wert des **Ausdruck**s mit einem **Fall** überein, werden alle **Anweisungsblöcke** ab dem entsprechenden Fall ausgeführt
- stimmt kein Wert mit dem Ergebnis vom **Ausdruck** überein und ist ein **Fall** für **sonst** vorhanden, wird dieser **Anweisungsblock** ausgeführt
- der Wert des Ausdrucks ist vom Typen *byte*, *short*, *int* oder *char*
- **a**, **b**, … **z** sind unterschiedliche Konstanten des Typs, den der **Ausdruck** liefert
- anschließend wird die bewachte Anweisung verlassen

- Umsetzung in Java

```
switch (Ausdruck)
{
    case a: Anweisungsblock 1
    case b: Anweisungsblock 2
    …
    case z: Anweisungsblock n
    default: Anweisungsblock def
}
```

3.2.3 Unterrichtsziele

Grobziele

Die Schüler sollen

- den Algorithmusbegriff wiederholen und festigen.
- die Bedingungen in Java erläutern und anwenden können.
- ein Struktogramm erarbeiten können.

Feinziele

Die Schüler sollen

- die Merkmale des Algorithmus, Schleifen, Selektionen und Sequenzen werden erklären können.
- in *if*- und *switch*-Anweisungen unterscheiden können.
- Vergleichsoperatoren und logische Ausdrücke erkennen und differenzieren können.
- die Darstellung von Bedingungen in Strukturblöcken beherrschen können.
- ein Struktogramm zur Lösung der Aufgabe erstellen.
- das Struktogramm in ein lauffähiges Java-Programm mit JavaKara umsetzen.

3.2.4 Didaktisch-methodische Überlegungen

Der Verwendung von logischen Operatoren und Vergleichsoperatoren sollte besondere Aufmerksamkeit gewidmet werden, da die Unterscheidung dieser von entscheidender Bedeutung für Umsetzung des Programms sein wird.

Umsetzung

Phase I: Zu Stundenbeginn wird kurz der Stundenverlauf vorgestellt.
Phase II: Der „Algorithmus"-Begriff wird mit Hilfe eines kleines spontan ausgewählten Beispieles wiederholt. Dadurch wird die sofortige Verwendung des Begriffes gewährleistet. Aus dem Beispiel wird weiterhin die Notwendigkeit von Schleifen und Bedingungen erkannt. Nun werden die *if*- und *switch*-Anweisungen eingeführt.
Phase III: Im nächsten Schritt wird die zu lösende Aufgabe gestellt. Dann werden die einzelnen Sequenzen anhand von Strukturblöcken verdeutlicht und es wird im Folgenden erklärt, wie Bedingungen in einem Struktogramm dargestellt werden. Daran anschließend geht es an die Umsetzung in den Java-Quellcode.

Phase IV: In dieser Phase wird die neue Programmierumgebung JavaKara kurz vorgestellt. Anschließend erhalten die Schüler eine schriftliche Aufgabenstellung, die sie in lauffähiges Java-Programm umwandeln sollen. Die feststehenden Methoden des JavaKara werden vorgegeben[8].
Phase V: Diese Phase richtet sich allein an die Schüler, die bereits vor dem Ende der Stunde mit dem Lösen der ersten Aufgabe fertig sind. Sie erhalten eine Zusatzaufgabe, die als Hausaufgabe beendet werden soll.

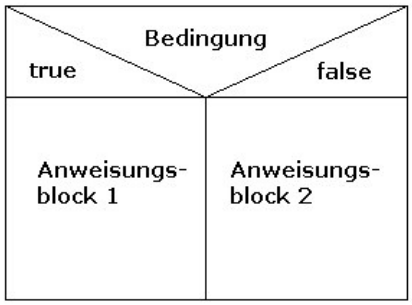
3.2.5 Unterrichtsverlauf

Dauer	**Phase**	**Inhalt**	**Unterrichtsform/ Medien**
2 min.	Phase I	- Begrüßung - Ausblick auf die Stunde wird gegeben	Lehrervortrag
10 min.	Phase II Erarbeitung I	- Wiederholung und Festigung des Algorithmus-Begriffes - Einführung von Bedingungen	Unterrichtsgespräch
10 min.	Phase III Erarbeitung II	- Erläuterung von Bedingungen in Struktogrammen - Erarbeitung des dazugehörigen Java-Codes	Lehrervortrag und Unterrichtsgespräch
18 min.	Phase IV Festigung I	- Vorstellung des neuen Programmiertools mit den nötigen Methoden usw. - Umsetzung der Aufgabe in ein Java-Programm	Lehrervortrag Computerarbeit Partnerarbeit
5 min.	Phase V Festigung II	- Umsetzung der Zusatzaufgabe in ein Java-Programm - Aufgeben der Hausaufgabe	Computerarbeit Partnerarbeit möglich

[8] Die Beispiele sind folgender Website entnommen worden:
http://www.educeth.ch/informatik/karatojava/javakara/

3.2.6 Unterrichtsmaterialien

Folie 1- Struktogramm zu Bedingungen



Folie 2 – Umsetzung in Java-Quellcode

```
if (Bedingung)
{
    Anweisungsblock 1
}
else
{
    Anweisungsblock 2
}
```

Folie 3 – Aufgabenstellung

Drücke diese Anweisungen in Java für KARA aus: „Wenn vorne kein Baum steht, aber ein Pilz steht, so gehe einen Schritt vor und drehe nach rechts, sonst gehe einen Schritt nach links, falls dort kein Baum steht."

Folie 4 – wichtige Methoden von KARA

KARA – Kommandos	**KARA – Sensoren**
kara.move()	kara.treeFront()
kara.turnLeft()	kara.treeLeft()
kara.turnRight()	kara.treeRight()
kara.putLeaf()	kara.mushroomFront()
kara.removeLeaf()	kara.onLeaf()

Folie 5 – Lösung zur Aufgabe

```
if (! kara.treeFront() && kara.mushroomFront())
 {
        kara.move();
        kara.turnRight();
 }
else    {
                if (! kara.treeFront())
                        {
                                kara.turnLeft();
                                kara.move();
                        }
        }
```

Folie 6 – Zusatzaufgabe

Kara steht vor einer Schlange von Kleeblättern, an deren Ende ein Baum steht. Kara soll bis zum Baum laufen. Immer, wenn er auf ein Kleeblatt kommt, soll er rechts daneben ein weiteres Blatt legen. Verwende zur Lösung der Aufgabe eine Methode *legeRechts()*, die Kara dazu veranlasst, rechts ein Kleeblatt hinzulegen und dann auf den Weg zurückzukehren.

Folie 7 – Lösung

```
Import JavaKaraProgram;
Public class RechtsDanebenLegen extends
JavaKaraProgram

{
        void legeRechts()
   {
        kara.turnRight();
        kara.move();
        kara.putLeaf();
        kara.turnLeft();
        kara.turnLeft();
        kara.move();
        kara.turnRight();
   }
        public void myProgram()
        {
          if (kara.onLeaf())  { legeRechts(); }
          while (!kara.treeFront())
          {
                kara.move();
                if (kara.onLeaf())  { legeRechts(); }
          }
        }
}
```

4 Anhang

4.1 Projekt „Vier gewinnt“

4.1.1 Klassendiagramme

VierGewinnt
fenster – Fenster spiel – Spiel
public main() – void

Spiel
aktuellerspieler – Spieler spieler1 – Spieler spieler2 – Spieler spielfeld – Spielfeld
public Spiel() public start() – void public spielausgabe() – void private spielerwechsel() – void private ueberpruefung() – boolean

Spieler
farbe – Color name – String
public Spieler() public chipsetzen() – int public getfarbe() – Color public getname() – String

Spielfeld
spielfeld – Chip[7][6]
public Spielfeld() public gewinner() – Spieler public gewinnerfarbe() – Color public istvoll() – boolean public gewinnzug() – boolean private gewinnzeile() – boolean private gewinnspalte() – boolean private gewinndiagonale() – boolean public chipsetzen() – boolean public spielfeldausgabe() – void public paint() – void private zeichneSpielfeld() – void private zeichneChip() – void private zeichneGesetzteChips() – void

Chip
spieler – Spieler farbe – Color
public Chip() public getspieler() – Spieler public getfarbe() – Color public chipausgabe () – void

Fenster
buttonpanel – Panel endepanel – Panel ende – Button knopf1 – Button knopf2 – Button knopf3 – Button knopf4 – Button knopf5 – Button knopf6 – Button knopf7 – Button farbe – Color
public Fenster() public ausgabe() – void public actionPerformed() – void private farbewechsel() – void private gewonnen() – void private unentschieden() – void private zugausfuehren() – void

4.1.2 Implementierung

Klasse VierGewinnt

```
class VierGewinnt
{
	public static Spiel spiel = new Spiel();

	//für die grafische Ausführung
	public static Fenster fenster = new Fenster(spiel.spielfeld);

	public static void main(String [] arg)
	{
		//spiel.start();			//ohne grafische Ausführung
	}
}
```

Klasse Spiel

```
import java.awt.Color;

class Spiel
{
	//Attribute
	public Spielfeld spielfeld;
	public Spieler spieler1;
	public Spieler spieler2;
	public Spieler aktuellerspieler;

	//Konstruktor
	public Spiel()
	{
		spielfeld = new Spielfeld();
		spieler1 = new Spieler(Color.blue);
		spieler2 = new Spieler(Color.red);
		aktuellerspieler = spieler2;
	}

	//Methoden
	public void start()
	{
		spielausgabe();
		int spalte;
		do
		{
			spielerwechsel();
			spalte = aktuellerspieler.chipsetzen();
			spielfeld.chipsetzen(spalte, aktuellerspieler);
		spielausgabe();
		}while(!ueberpruefung(spalte));
	}

	private boolean ueberpruefung(int spalte)
	{
		if(spielfeld.istvoll())
```

```
                {
                        System.out.println("Das Spielfeld ist voll!");
                        return true;
                }
                if (spielfeld.gewinnzug(aktuellerspieler, spalte))
                {
                        Spieler spieler = spielfeld.gewinner(spalte);
                        System.out.println("Der Spieler "+ spieler.getname()  +" hat gewonnen");
                        return true;
                }
                return false;
        }

        private void spielerwechsel()
        {
                if (aktuellerspieler == spieler1)
                        aktuellerspieler = spieler2;
                else
                        aktuellerspieler = spieler1;
        }

        public void spielausgabe()
        {
                spielfeld.spielfeldausgabe();
        }

}
```

Klasse Spieler

```
import java.awt.Color;

public class Spieler
{

        //Attribute
        public Color farbe;
        public String name;

        //Konstruktor
        public Spieler (Color neuefarbe)
        {
                farbe = neuefarbe;
                if(neuefarbe == Color.red)
                        name = "Rot";
                else
                        name = "Blau";
        }

        //Methoden
        public Color getfarbe()
        {
                return this.farbe;
        }

        public String getname()
```

```
	{
		return this.name;
	}

	public int chipsetzen()
	{
		System.out.println("Bitte Spalte angeben, " + this.getname() + "!");
		int spalte = Keyboard.readInt();
		return spalte;
	}
}
```

Klasse Spielfeld

```
import java.awt.Color;
import java.awt.Panel;
import java.awt.Graphics;

public class Spielfeld extends Panel
{
	//Attribute
	Chip[][] spielfeld;

	//Konstruktor
	public Spielfeld ()
	{
		spielfeld = new Chip[7][6];
		for (int j=0; j<=5; j++)
		{
			for (int i=0; i<=6; i++)
			{
				spielfeld[i][j]=null;
			}
		}
	}

	//Methoden
	public boolean istvoll()
	{
		for (int i = 0; i < spielfeld.length; i++ )
		{
			if (spielfeld[i][0] == null)
			{
				return false;
			}
		}
		return true;
	}

	public void spielfeldausgabe()
	{
		System.out.println();
		Chip chip = null;
		for(int i = 0; i<6; i++)
		{
			System.out.print("| ");
```

```
                        for(int j = 0; j<7; j++)
                        {
                                chip = spielfeld[j][i];
                                if(chip == null)
                                        System.out.print(" ");
                                else
                                        chip.chipausgabe();
                                System.out.print(" | ");
                        }
                        System.out.println();
                }
                System.out.println("-----------------------------");
                System.out.println("  1   2   3   4   5   6   7");
                System.out.println();
        }

        public boolean chipsetzen(int spalte, Spieler aktuellerspieler)
        {
                for(int zeile = 5; zeile>=0; zeile--)
                {
                        if(spielfeld[spalte-1][zeile] == null)
                        {
                                Chip chip = new Chip(aktuellerspieler);
                                spielfeld[spalte-1][zeile] = chip;
                                return true;
                        }
                }
                return false;
        }

        public Spieler gewinner(int spalte)
        {
                int zeile = 0;
                while (spielfeld[spalte-1][zeile] == null)
                {
                        zeile++;
                }
                Chip chip = spielfeld [spalte-1] [zeile];
                return chip.getspieler();
        }

        public boolean gewinnzug (Spieler spieler, int spalte)
        {
                Color farbe = spieler.getfarbe();
                spalte = spalte - 1; //Feldindex

                if (gewinnspalte (farbe, spalte) || gewinnzeile (farbe, spalte) || gewinndiagonale (farbe,
spalte))
                        return true;
                return false;
        }

        private boolean gewinnspalte(Color farbe, int spaltenindex)
        {
                int j;
                for (j=0 ; j<3 ; j++)
                {
                        if (spielfeld [spaltenindex] [j] != null)
```

```
                {
                        for (int i=j+1 ; i<=j+3 && i<6; i++)
                        {
                                Chip chip = spielfeld [spaltenindex] [i];
                                if(chip.getfarbe() != farbe)
                                {
                                        return false;
                                }
                        }
                        System.out.println("Gewinn in Spalte");
                        return true;
                }
        }
        return false;
}

private boolean gewinnzeile (Color farbe, int spaltenindex)
{
        int cnt = 1;
        int zeile = 0;

        while (spielfeld[spaltenindex][zeile] == null) zeile++;

        int i = 1;

        //nach rechts prüfen
        while (i<(7-spaltenindex) && spielfeld[spaltenindex+i][zeile]!=null
                        &&  spielfeld[spaltenindex+i][zeile].getfarbe() == farbe)
        {
                cnt++;
                i++;
        }

        if (cnt >= 4) return true;

        i = 1;

        //nach links prüfen
        while (i <= spaltenindex && spielfeld[spaltenindex-i][zeile] != null
                        && spielfeld[spaltenindex-i][zeile].getfarbe() == farbe )
        {
                cnt++;
                i++;
        }

        if (cnt >= 4)
        {
                System.out.println("Gewinn in Zeile");
                return true;
        }

        return false;
}

private boolean gewinndiagonale(Color farbe, int spaltenindex)
{

        int zeile=0;
```

```
while (spielfeld[spaltenindex][zeile] == null)
{
        zeile++;
}

int h=1; //Hauptdiagonale (der gesetzte Chip zählt mit)
int n=1; //Nebendiagonale

//Überprüfung nach rechts unten
int x=spaltenindex+1;
int y=zeile+1;
while (x<7 && y<6 && spielfeld[x][y] != null
                && spielfeld[x][y].getfarbe()==farbe)
{
        y++;
        x++;
        h++;
}

//Überprüfung nach links oben
x=spaltenindex-1;
y=zeile-1;
while (x>=0 && y>=0 && spielfeld[x][y] != null
                && spielfeld[x][y].getfarbe()==farbe)
{
        y--;
        x--;
        h++;
}

//Überprüfung nach rechts oben
x=spaltenindex+1;
y=zeile-1;
while (x<7 && y>=0 && spielfeld[x][y] != null
                && spielfeld[x][y].getfarbe()==farbe)
{
        y--;
        x++;
        n++;
}

//Überprüfung nach links unten
x=spaltenindex-1;
y=zeile+1;
while (x>=0 && y<6 && spielfeld[x][y] != null
                && spielfeld[x][y].getfarbe()==farbe)
{
        y++;
        x--;
        n++;
}

if (h>=4 || n>=4)
{
        System.out.println("Gewinn in Diagonale");
        return true;
}
else
```

```
                return false;
        }


//------------------
//fuer grafische Ausfuehrung

        public boolean chipsetzen(int spalte, Color farbe)
        {
                for(int zeile = 5; zeile>=0; zeile--)
                {
                        if(spielfeld[spalte-1][zeile] == null)
                        {
                                Chip chip = new Chip(farbe);
                                spielfeld[spalte-1][zeile] = chip;
                            return true;
                        }
                }
                return false;
        }

        public Color gewinnerfarbe(int spalte)
        {
                int zeile = 0;
                while (spielfeld[spalte][zeile] == null)
                {
                        zeile++;
                }
                Chip chip = spielfeld [spalte] [zeile];
                return chip.getfarbe();
        }

        public boolean gewinnzug (Color farbe, int spalte)
        {
                if (gewinnspalte (farbe, spalte-1) || gewinnzeile (farbe, spalte-1)
                        || gewinndiagonale (farbe, spalte-1))
                        return true;
                else
                        return false;
        }

        public void paint (Graphics graphic)
        {
                zeichneSpielfeld(graphic);
                zeichneGesetzteChips(graphic);
        }

        private void zeichneSpielfeld (Graphics graphic)
        {
                for (int i = 0; i <= 7; i++)
                {
                        graphic.drawLine(56 + 330/7* i, 320- 330/7* 7, 56+ 330/7* i, 270);
                }
                graphic.drawLine(56, 270, 56+ 330, 270);
                graphic.drawString("Spieler Blau ist am Zug!", 165, 290);
        }

        private void zeichneGesetzteChips (Graphics graphic)
```

```
        {
                for (int zeile = 0; zeile < 6; zeile++)
                {
                        for (int spalte = 0; spalte < 7; spalte++)
                        {
                                if ( (spielfeld[spalte][zeile] != null))
                                {
                                        zeichneChip(graphic, spalte, zeile, spielfeld[spalte][zeile]);
                                }
                        }
                }
        }

        private void zeichneChip(Graphics graphic, int spalte, int zeile, Chip chip)
        {
                Color c = graphic.getColor();
                graphic.setColor(chip.getfarbe());
                graphic.fillOval((56 + 1) + spalte * 330/7,270 - (330/7 - 1) - (5 - zeile) * (330/7 - 2),
                        (330/7 - 2), (330/7 - 2));
                graphic.setColor(c);
        }

}
```

Klasse Chip

```
import java.awt.Color;

public class Chip
{

        //Attribute
        public Color farbe;
        public Spieler spieler;

        //Konstruktor
        public Chip (Spieler neuerspieler)
        {
                farbe = neuerspieler.getfarbe();
                spieler = neuerspieler;
        }

        //Methoden
        public Color getfarbe()
        {
                return this.farbe;
        }

        public Spieler getspieler()
        {
                return this.spieler;
        }

        public void chipausgabe()
        {
                if(this.getfarbe() == Color.red)
```

```
                        System.out.print("r");
                else
                        System.out.print("b");
        }

//--------------------
//fuer grafische Ausfuehrung

        //Konstruktor
        public Chip(Color neuefarbe)
        {
                farbe = neuefarbe;
                spieler = new Spieler(neuefarbe);
        }

}
```

Klasse Fenster

```
import java.awt.Frame;
import java.awt.Panel;
import java.awt.Button;
import java.awt.Color;
import java.awt.Label;
import java.awt.Graphics; //abstrakte Klasse
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class Fenster extends Frame implements ActionListener
{
        //Attribute
        public static Button ende, knopf1, knopf2, knopf3, knopf4, knopf5, knopf6, knopf7;
        public static Panel buttonpanel, endepanel;
        public static Spielfeld feldpanel;
        public static Color farbe = Color.red;

        //Konstruktor
        public Fenster(Spielfeld spielfeld)
        {
                setTitle("VierGewinnt");
                setSize (450, 400);
                setLocation (200, 200);
                setVisible(true);

                // Fenster schliessen
                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent e)
                        {
                                System.exit(0);
                        }
                });

                // Button-Panel (oben)
```

```
		buttonpanel = new Panel();
		endepanel = new Panel();

		ende = new Button ("Spiel beenden!");
		knopf1 = new Button("1");
		knopf2 = new Button("2");
		knopf3 = new Button("3");
		knopf4 = new Button("4");
		knopf5 = new Button("5");
		knopf6 = new Button("6");
		knopf7 = new Button("7");

		ende.addActionListener(this);
		endepanel.add(ende);
		knopf1.addActionListener(this);
		buttonpanel.add(knopf1);
		knopf2.addActionListener(this);
		buttonpanel.add(knopf2);
		knopf3.addActionListener(this);
		buttonpanel.add(knopf3);
		knopf4.addActionListener(this);
		buttonpanel.add(knopf4);
		knopf5.addActionListener(this);
		buttonpanel.add(knopf5);
		knopf6.addActionListener(this);
		buttonpanel.add(knopf6);
		knopf7.addActionListener(this);
		buttonpanel.add(knopf7);

		add("North", buttonpanel);
		add("South", endepanel);

		//Spielfeld-Panel (unten)
		feldpanel = spielfeld;
		add("Center", spielfeld);

		setVisible(true);
	}

	//---------------------------
	//ActionListener
	public void actionPerformed(ActionEvent event)
	{
		Button knopf = (Button)event.getSource();
		if(knopf == ende)
		{
			System.exit(0);
		}
		int spalte = java.lang.Integer.valueOf(knopf.getLabel()).intValue();
		zugausfuehren(spalte);
	}

	private void zugausfuehren(int spalte)
	{
		if (feldpanel.chipsetzen(spalte, farbe))
		{
			if(feldpanel.gewinnzug(farbe, spalte))
			{
```

```
                                spielgewonnen(feldpanel.gewinner(spalte).getname());
                        }

                        if(feldpanel.istvoll())
                        {
                                unentschieden();
                        }
                        farbwechsel();
                }
                feldpanel.repaint();
        }

        private void farbwechsel()
        {
                if(farbe == Color.red)
                        farbe = Color.blue;
                else
                        farbe = Color.red;
        }

        private void spielgewonnen(String name)
        {
                Button gewonnen = new Button("Spieler " + name + " hat gewonnen!");
                gewonnen.addActionListener(
                        new ActionListener(){
                                public void actionPerformed(ActionEvent e){
                                        Runtime.getRuntime().exit(0);
                                }
                        });
                remove(buttonpanel);
                buttonpanel.add(gewonnen);
                add("North", gewonnen);
        }

        private void unentschieden ()
        {
                Button unentschieden = new Button(" Alle Steine gesetzt: Unentschieden!");
                unentschieden.addActionListener(
                        new ActionListener() {
                                public void actionPerformed(ActionEvent e){
                                        Runtime.getRuntime().exit(0);
                                }
                        });
                remove(buttonpanel);
                add("North", unentschieden);
        }
}
```

4.2 Literatur

Eckpunkte zum Berliner Rahmenplan unter:
http://bebis.cidsnet.de/faecher/fach/informatik/

Balzert, H.: *Lehrbuch Grundlagen der Informatik.* Konzepte, Notation in UML, Java, C++, Algorithmik und Software-Technik, Heidelberg – Berlin – Oxford, 1999.

Hubwieser, P.: *Didaktik der Informatik. Grundlagen, Konzepte, Beispiele.* Berlin, Heidelberg, New York: Springer Verlag, 2000.

http://www.educeth.ch/informatik/karatojava/javakara/