

Entwurf einer Unterrichtseinheit
„Objektorientiertes Programmieren“

vorgelegt von Daniel Jonietz
Universität Kaiserslautern
Fachbereich Informatik

Mai 1999

Inhaltsverzeichnis

1	Einleitung	2
2	Fachliche Klärung des Unterrichtsgegenstandes	3
2.1	Die Idee des Objekt-Paradigmas	3
2.2	Einordnung des Objekt-Paradigmas	4
2.3	Terminologie	5
2.4	Begrifflichkeiten	9
2.5	Programmiersprachen	9
3	Didaktische Analyse	10
3.1	Begründung des Unterrichtsvorhabens	10
3.1.1	Bedeutung des Themas im Informatikunterricht	10
3.1.2	Begründung des Themas aus dem Lehrplan	11
3.1.3	Fachspezifisch-allgemeine Lernziele	12
3.2	Auswahl und Strukturierung der Inhalte und Ziele mit Begründung	14
3.2.0	Intention der Unterrichtsreihe	14
3.2.1	Hierarchie der Objekt-Konzepte	15
3.2.2	Auswahl der Lernziele und didaktische Reduktion	15
3.2.3	Ordnung der Lernziele nach Kriterien grob / fein	17
3.3	Voraussetzungen und Vorkenntnisse	19
3.4	Zur Frage der Programmiersprache	19
3.5	Lernkontexte	21
3.6	Lehrbuch	23
3.7	Besondere Schwierigkeiten	24
4	Methodische Analyse	25
4.1	Lernmethodische Aspekte	25
4.2	Unterrichtsmethodische Aspekte	26
5	Unterrichtsplanung (Skizze)	27
5.1	Unterrichtsplanung	27
5.2	Unterrichtsplanung detailliert: Schritte und Phasen	32
A	Objekthierarchie	38
B	Objekthierarchie: Aufruf von move	39
C	Quelltexte	40
D	Arbeitsblätter	42

1 Einleitung

In dieser Ausarbeitung soll eine Unterrichtseinheit zur Objektorientierten Programmierung vorgestellt werden. Dabei gehe ich von einigen Bedingungen aus, die in einem konkreten Kurs u. U. so nicht vorliegen; Gegebenenfalls müssen Anpassungen hinsichtlich des Vorwissens der Schüler gemacht werden, indem entsprechende Änderungen vorgenommen werden oder in einem Vor-Kurs nötige Kenntnisse vermittelt werden. Wo auch immer Änderungen oder Anpassungen ohne große Umstellung der Unterrichtsreihe möglich sind, werde ich kurz die Alternativen nennen und somit zugänglich machen.

Ich möchte im Folgenden von einem fiktiven gymnasialen Grundkurs der 12. Jahrgangsstufe ausgehen, der seinen Unterricht gemäß dem rheinland-pfälzischen Lehrplan [Lehrplan 93] absolviert hat und dementsprechend bereits über Vorwissen und einige Erfahrung bezüglich der generellen Vorgehensweise bei zu lösenden Problemen verfügt. Weiter einschränkend habe dieser Kurs speziell in Turbo-Pascal einer hinreichend hohen Versionsnummer gearbeitet und sei in der Lage, abstrakte Datentypen zu entwerfen und mit ihnen umzugehen. Darüberhinaus haben die Schülerinnen und Schüler bereits elementare Grafik-Befehle von Pascal (wie z. B. das Zeichnen von Punkten, Linien und Kreisen) kennengelernt.

Ferner gehe ich von wöchentlich einer Doppelstunde und einer Einzelstunde ohne Unterrichtsausfall aus, so daß spezifische zeitliche Gegebenheiten nicht berücksichtigt werden müssen. Die räumliche und technische Ausstattung sei Ideal.

2 Fachliche Klärung des Unterrichtsgegenstandes

2.1 Die Idee des Objekt-Paradigmas

Bei der konventionellen ablauforientierten Programmierung erfolgt die Definition der Datenstrukturen und der darauf ausgeführten Operationen voneinander weitgehend unabhängig. Das Wissen um die Bedeutung und Zusammenhänge der Daten ist dezentral im ganzen Programm verteilt, indem die Datenzugriffe mit dem Wissen um die Datenstruktur programmiert werden. Dagegen erfolgt bei der Objektorientierten Programmierung die Beschreibung von Daten zusammen mit darauf auszuführenden Aktionen in einer Einheit, später sind Kenntnisse um die Datenrepräsentation nicht mehr erforderlich.

Um einen Ausschnitt der realen Welt zu modellieren, läßt sich eine solche Miniwelt (auch) als Gesamtheit von „Objekten“ (Gegenständen, Ereignissen) auffassen, die

- untereinander in Beziehungen stehen,
- sich im Verlauf der Zeit ändern und gegenseitig beeinflussen,
- beobachtet werden können,
- auf Einwirkungen von außen in spezifischer Weise reagieren.

Das Objekt-Paradigma versucht die auftretenden Objekte zu klassifizieren, ihre Beziehungen untereinander zu charakterisieren und ihre Eigenschaften und Verhaltensweisen zu beschreiben. Darüberhinaus wird den Komponenten Selbständigkeit zuerkannt, so daß die bei ablauforientierten Programmen notwendige Ablaufsteuerung sich quasi „von selbst“ aus der Komponentenstruktur ergibt. Programmierung kann sich auf die Beschreibung einzelner Komponenten sowie ihrer Kommunikationsmöglichkeiten beschränken. Diesen Sachverhalt beschreiben CRUTZEN & HEIN wie folgt:

„Jedes Objekt der Realität, das Informationen aufnimmt, speichert, verarbeitet und abgibt, kann man als Informationsobjekt beschreiben. Objekte der Realität können untereinander agieren, diese Interaktionen kann man als Kommunikation zwischen Informationsobjekten beschreiben. Ein Informationssystem ist die Realisierung einer Anzahl von Informationsobjekten — und damit ein Objekt der Realität, das man ... wiederum als Informationsobjekt beschreiben kann.“¹

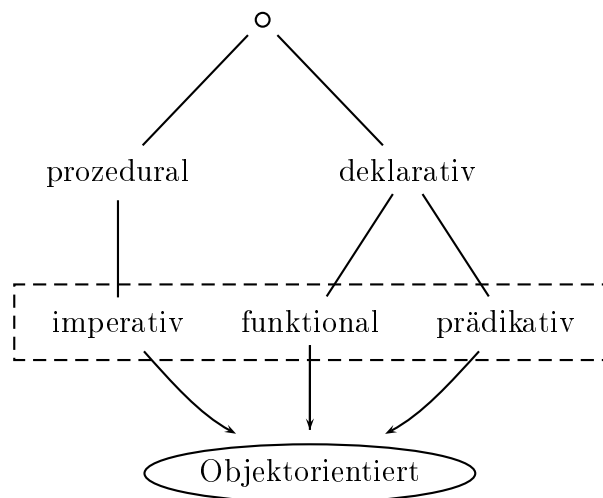
¹[Crutzen & Hein 95], S. 150

Durch die Aufspaltung komplexer Systeme in solche kleinen, in sich geschlossenen Komponenten (Objekte) erschließen sich mehrere Vorteile:

- Die Schnittstellen zwischen den Komponenten lassen sich einfach und klar verständlich angeben,
- der Programmieraufwand läßt sich durch Wiederverwendung der Gemeinsamkeiten ähnlicher Komponenten senken
- und schließlich werden viele Programmierfehler durch die Abgeschlossenheit einzelner Komponenten und die daraus resultierende eingeschränkte Sicht des Programmierers vermieden.

2.2 Einordnung des Objekt-Paradigmas

Programmiersprachen lassen sich durch charakteristische Merkmale nach drei Programmierstilen klassifizieren, diese ordnet man zwei Kategorien zu:



Als vierter Stil wird oft Objektorientiert genannt, obwohl Objektorientierte Konzepte mit allen anderen Stilen kombiniert werden können. Objektorientierte Programmierung kann eher als Erweiterung der strukturierten Programmierung denn als eigener Programmierstil aufgefaßt werden.²

²[Schwill 93], S. 10f

2.3 Terminologie³

- **Objekte**

Wie oben beschrieben läßt sich eine Miniwelt als Menge unabhängiger, kooperierender Komponenten beschreiben. Jede Komponente des Systems repräsentiert einen beliebigen Gegenstand oder Sachverhalt der Realität und besitzt erfaßbare Eigenschaften, die **Attribute** genannt werden. Alle Attribute zusammen spiegeln den Zustand der Komponente wieder. Elementare Komponenten heißen **Objekte**.

Aus Programmierer-Sicht ist ein Objekt die Zusammenfassung von Daten und Operationen zu einer Einheit.

- **Nachrichten und Dienste**

Die Dynamik des Systems wird durch den Austausch von **Nachrichten** (Botschaften) zwischen den einzelnen Komponenten gewährleistet. Eine Nachricht kann dazu führen, daß die benachrichtigte Komponente ihren Zustand ändert oder einen **Dienst** ausführt. Die Handlungsvorschrift zur Durchführung des Dienstes heißt **Methode** und entspricht einer Prozedur der ablauforientierten Programmierung.

- **Klassen**

Merkmale und Verhalten von Komponenten werden durch die Attribute und Dienste entsprechender Objekte dargestellt. Gibt es mehrere Objekte mit gleichen Merkmalen und gleichem Verhalten, so gehören diese Objekte zur gleichen **Klasse** von Objekten. Dabei gibt die Klasse die zugrundeliegende Idee, die Konzeption einer Art von Objekten an und kann als Vorlage (Template) betrachtet werden, nach der ihre Objekte aufgebaut werden. Jedes einzelne Objekt einer Klasse ist ein **Exemplar** (eine Instanz) der Klasse. Die Bildung eines Exemplars einer Klasse wird mit **Instantiierung** bezeichnet und kennzeichnet eine konkrete Ausprägung, entstanden gemäß einer Vorlage.

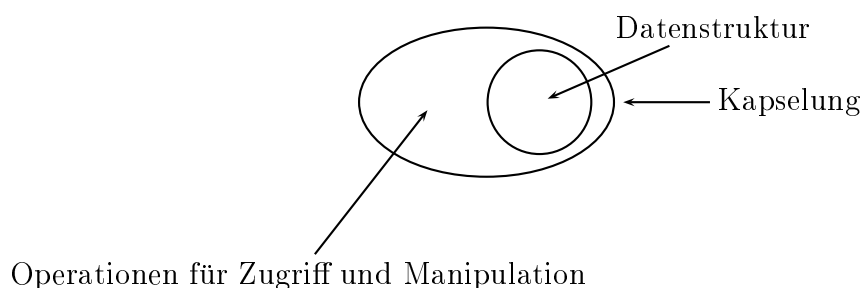
- **Kapselung**

Herkömmliche Prozeduren und Funktionen sind in der Lage Hilfsgrößen zu verstecken, so daß sie nur während der Funktionsausführung lokal zur Verfügung stehen, ihre Existenz ist auf die Ausführungszeit der Funktion und den Ort (d. h. innerhalb der Funktion) beschränkt. Die Schnittstelle der Funktion weist den Namen und evtl. benötigte Parameter aus, die Realisierung bleibt dem Benutzer ebenso wie die Hilfsgrößen verborgen. Dieses Verstecken wird mit **Kapselung** bezeichnet. Das Konzept der abstrakten Datentypen (ADT) erweitert die

³angelehnt an [Goos 96]

Möglichkeiten der Kapselung dahingehend, daß es nun auch möglich ist, lokal sichtbare Daten zu verwenden, deren Lebenszeit nicht auf die Ausführungszeit einer Funktion beschränkt ist. Dadurch wird ein hoher Grad an Abstraktion in der Beschreibung und im Umgang mit benutzerdefinierten Datentypen erreicht⁴, außerdem haben Eingriffe in die Software nur lokal beschränkte Auswirkungen: Änderungen der Implementierung haben bei stabiler Schnittstelle keinen weiteren Änderungsaufwand zur Folge.

Die Objektorientierung leistet dies mit dem Klassenkonzept: Klassen sind als implementierte abstrakte Datentypen zu verstehen. Die Klassenschnittstelle, aufzufassen als Gesamtheit aller zulässigen Zugriffs- und Manipulationsoperationen, beschränkt den Zugriff auf die Datenstruktur. Jedwede konkrete Realisierung bleibt dem Anwender verborgen. Außerhalb der Klassenhierarchie sind keine gemeinsamen Daten, keine globalen Variablen vorhanden, so daß Nebenwirkungen ausgeschlossen sind. Die Entscheidung einer Implementierung verteilt dabei ihre Konsequenzen nicht im ganzen Programm, sondern wird an einem Punkt konzentriert. GOOS nennt als Metapher für die Kapselung ein „Ei“.⁵



- **abstrakte Klassen und Vererbung**

Es sind Klassen vorstellbar, die einander ähnlich sind und sich nur in spezifischen Details unterscheiden. Wünschenswert ist es, die Gemeinsamkeiten einmal festzulegen und mehrfach zu verwenden, Unterschiede jedoch ggf. konkret anzugeben. Dazu können die entsprechenden Größen (Attribute, Methoden) lokal geändert werden, müssen jedoch nur bei tatsächlichem Bedarf angepaßt werden.

⁴[Schwill 95], S. 180

⁵[Quibeldey-Cirke 94], S. 64

Objektorientierte Beschreibung gestattet es, von einem Basistyp einer Klasse weitere Typen **abzuleiten**. Der abgeleitete Typ erbt Schnittstelle und Funktionalität, kann diese jedoch auch erweitern oder modifizieren. Die abgeleitete Klasse heißt **Subklasse** (Unterklasse), die Oberklasse **Superklasse**. Die Vererbungsrichtung Superklasse \rightarrow Subklasse beschreibt eine Spezialisierung, die Richtung Subklasse \rightarrow Superklasse Generalisierung.

Neben solchen konkreten Oberklassen sind auch sog. **abstrakte** Oberklassen erlaubt: Beschreibt eine Klasse nur Außenverhalten, so heißt sie abstrakt. Sie repräsentiert eine Menge konkreter Klassen mit gleichem Außenverhalten. Unterklassen erben Merkmale (Attribute, Methoden) von Oberklassen, dürfen aber auch eigene Merkmale in Form von Attributen oder Methoden einbringen. Eine abstrakte Klasse muß stets Oberklasse sein und eine (konkrete) Unterklasse besitzen.

Bei der Vererbung sind grundsätzlich zu unterscheiden:

- „**ist-ein**“-Vererbung
(Schüler ist ein Mensch),
- „**ist-Spezialisierung-von**“-Vererbung
(gleichseitiges Dreieck ist Spezialisierung von allgemeinem Dreieck),
- „**implementiert**“-Vererbung
(Liste realisiert Menge),
- „**hat-ein**“-Vererbung
(Ein Kreis hat einen Punkt als Mittelpunkt)
- „**verwendet**“-Vererbung
(wird ausschließlich technisch verwandt)

Erbt eine Klasse die Verhaltensweisen mehrerer (Super-)Klassen, so spricht man von Mehrfachvererbung. Die allen Subklassen einer gemeinsamen Superklasse gemeinsamen Merkmale der Subklassen heißen **abgeleitet**, die spezifischen Merkmale heißen **Kernmerkmale**. Dabei können abgeleitete Methoden überschrieben werden, indem für die Unterklasse eine neue Kernmethode angegeben wird, die die Spezifikation realisiert und die Aufgaben der abgeleiteten Methode übernimmt. Vererbung dient der Modularität und Übersichtlichkeit und fördert die Wiederverwendbarkeit von Bausteinen, da gleiche Konzepte nur einfach programmiert werden müssen und auf Grundbegriffe zurückgegriffen werden kann.

- **Polymorphie**

In monomorphen Programmiersprachen besitzen Prozeduren, Funktionen, Parameter, Operatoren und Operanden einen eindeutigen Typ, polymorphe Sprachen erlauben mehr als einen Typ.⁶ Dadurch wird die Signatur zu einer mehrelementigen Menge von Signaturen und ist nicht eindeutig. Polymorphie steht in direktem Zusammenhang zum Konzept der Vererbung. Wird von einem Objekt durch Versenden einer Nachricht ein Dienst angefordert, so entscheidet es selbständig, wie es den spezifischen Dienst erbringt. Es gilt zwischen statischer und dynamischer Bindung zu unterscheiden:

- **statische Bindung:**

die konkrete Klasse ist starr an eine abstrakte Schnittstellenklasse gebunden. So kann der Gebrauch einer Klasse entworfen werden, ohne sie im Detail zu realisieren, die Realisation wird in Form einer konkreten Unterklasse nachgereicht. Dabei werden die Methoden überladen.

- **dynamische Bindung:**

die auszuführende Methode wird erst zur Ausführungszeit bestimmt. Dadurch können Objekte mit gleichem Außenverhalten einheitlich verwendet werden: Verschiedengestaltige Objekte reagieren individuell auf gleichlautende Nachrichten. Steht zur Übersetzungszeit noch nicht fest, zu welcher Klasse ein Objekt zur Ausführungszeit gehören wird, so wird der Name der Klasse erst dann assoziiert, wenn das durch den Namen bezeichnete Objekt erzeugt wird bzw. die Methode verwendet wird. Die Bindung wird erst zur Laufzeit hergestellt, daher auch *late binding* genannt.

GALL zeigt in [Gall 95], S. 196 die Polymorphie-Typen bei der Objektorientierung auf:

- **ad hoc-Polymorphie:**

Eine Funktion kann bei gleicher Handhabung auf einer endlichen Menge von Typen in einer vom Typ abhängigen Weise operieren

- * **overloading:** Ein Name steht für unterschiedliche semantische Objekte (Funktionen, Operatoren). Die Entscheidung, welche Ausprägung verwendet wird, fällt aufgrund des Kontextes.

- * **coercion:** Jedes Funktionsargument wird in einen vorgegebenen Typ umgewandelt und erst dann übergeben.

⁶[Gall 95], S. 196

– **universal-Polymorphie:**

Funktionen auf potentiell unbeschränkter Zahl von Typen:

- * **inclusion:** Vererbung / Klassenhierarchie: Ein Objekt kann als zu unterschiedlichen Klassen gehörig betrachtet werden. Ein Objekt aus einer Subklasse kann in einem Superklassenkontext verwendet werden.
- * **parametric:** Generizität: Generische Funktion kann ihre Aufgabe mit Argumenten eines noch nicht konkretisierten Typs durchführen.

Polymorphie fördert Übersichtlichkeit und Verständnis, da für verschiedene Objekte gleiche Namen entsprechend der erwarteten Funktionalität vergeben werden können.

2.4 Begrifflichkeiten

Es sind nicht alle oben vorgestellten Konzepte erforderlich. Die wesentlichen Konzepte werden auch als **Grundsäulen** der objektorientierten Programmierung bezeichnet, dies sind Kapselung, Vererbung und Polymorphismus, als **Basiskonzepte** werden Objekte, Klassen und Vererbung genannt.⁷ Man unterscheidet nach WEGENER schlagwortartig:

- Objektbasiert = Objekte
- Klassenbasiert = Objekte + Klassen
- Objektorientiert = Objekte + Klassen + Vererbung

2.5 Programmiersprachen

Nicht alle Programmiersprachen unterstützen Objektbasiertes / Objektorientiertes Programmieren:⁸

Nicht-Objektbasiert (*non*):

LISP, Algol, C, Pascal, Modula, Ada

Nicht rein-Objektorientiert (*hybride*):

Flavors, LOOPS, CLOS, C++, Objective-C, O-O Pascal, Delphi

rein-Objektorientiert (*pure*):

Smalltalk-80, Eiffel, Actor, Oberon, Java

⁷nach [Gall 95], S. 195 und [Quibeldey-Cirkel 94], S. 118

⁸aus: [Quibeldey-Cirkel 94], S. 121

3 Didaktische Analyse

„Wer sich einmal darauf eingelassen hat, wird sehr bald sehen, daß das objektorientierte Denken zu völlig anderen Programmarchitekturen führt, die die realen Dinge unserer Umwelt viel natürlicher abbilden als ablauforientierte Programme.“⁹

3.1 Begründung des Unterrichtsvorhabens

3.1.1 Bedeutung des Themas im Informatikunterricht

Objektorientierung ist eine Entwicklung (und ein Schlagwort, wie WIRTH zu bedenken gibt), die seit Beginn der 90er Jahren verstärkt Einzug in viele Bereiche der Datenverarbeitung gehalten hat und heute kaum noch wegzudenken ist.

Jeder Benutzer eines modernen Betriebssystems arbeitet (oft unbewußt) mit Objekten: Der Bildschirm ist gespickt mit symbolischen Metaphern (Piktogrammen), die für mitunter komplexe Operationen stehen. Jeder Anwender kann sein Allgemeinwissen intuitiv einsetzen:

Auf dem „Desktop“ (= Schreibtisch) gibt es Büroutensilien wie „Papierkorb“, „Ordner“, „Dokument“, „Ablage“ und „Aktenkoffer“. Ein Dokument wird in einem Ordner abgelegt, indem es mit der Hand (mittels Maus) aufgegriffen und im Ordner fallengelassen wird.¹⁰ Der „Papierkorb“ auf dem „Desktop“ ist Sinnbild und Schnittstelle für die abstrakte Handlung „verwerfe“ oder „lösche“. Ein solcher „Papierkorb“ kann mit geringem Vorwissen und für alle Arten von Daten verwendet werden, er schluckt Kraft seiner Polymorphie sowohl Dateien als auch Textausschnitte, Klänge, Piktogramme etc., ohne daß der Benutzer irgendeine Unterscheidung treffen muß: Mit einem Mauszug verschwindet alles widerruflich in ihm, bis er „geleert“ wird. Bei der Arbeit mit anweisungsgesteuerten Betriebssystemen ist syntaktisches und begriffliches Vorwissen nötig, um solche Löschoperationen ausführen zu können.¹¹

Jeder der auch nur ein Makro zur Abkürzung oft gebrauchter Anweisungsfolgen in einer Anwendung aufzeichnen will, kommt um Objektorientierung kaum umhin. Sämtliche verbreiteten modernen Makrosprachen (wie Visual Basic u. a.) bedienen sich am Konzeptvorrat der Objektorientierung.

Jedes Wort in einer Textverarbeitung ist ein Objekt mit Attributen wie Schriftart, Stil oder Farbe und Methoden wie Kopiere, Modifiziere, Schneide aus, Lösche. Eine Tabellenzelle kann Objekte vom Typus Text, Formel,

⁹Spolwig, [Spolwig 95], S. 43

¹⁰[Quibeldey-Cirkel 94], S. 150

¹¹[Quibeldey-Cirkel 94], S. 211

Grafik, Klang oder gar eine weitere Tabelle enthalten.

Objektorientierung begegnet dem Anwender von Dateninformationssystemen auf Schritt und Tritt, wenn auch oft unbewußt und versteckt. Doch wer lernt wie Objektorientierung funktioniert, kann die Funktionalität erst in vollem Umfang erkennen und sich zu Eigen machen.

Weiterhin ist die „... Zerlegung des Systems in seine Objekte und deren Beschreibung“¹² ein starkes Mittel zur Reduzierung von Komplexität. WIRTH schildert die Vorteile:

„... und heute ist die objektorientierte Programmierung zu einem wichtigen Begriff und zu einer potenten Technik geworden. ... Es steckt mehr dahinter als das vielzitierte in den Vordergrund rücken der Daten als Objekte anstelle der Abläufe (Algorithmen), denen die Daten unterworfen werden. Es handelt sich um mehr als eine rein alternative Betrachtungsweise von programmierten Systemen.“¹³

Das Konzept der Datenkapselung ermöglicht es dem Entwickler, Daten geschützt vor indiskreten Zugriffen aufzubewahren. Daten werden nur autorisierten (*eigenen*) Methoden ausgegeben. Folglich lassen sich Aspekte des Datenschutzes vereinfacht realisieren und können bereits im Systementwurf ohne zusätzlichen Aufwand berücksichtigt werden.

3.1.2 Begründung des Themas aus dem Lehrplan

QUIBELDEY-CIRCEL zeigt in [Quibeldey-Cirkel 94], S. 12 auf, daß die Trendwende hin zur Objektorientierung um den Jahrzehntwechsel 80/90 erfolgte, der starke Aufschwung begann in den 90er Jahren. Der Lehrplanentwurf für Rheinland-Pfalz¹⁴ stammt weitgehend aus den 80er Jahren und spricht daher die zur Entstehungszeit noch geringe Richtung der Objektorientierung nicht explizit sondern nur in Ansätzen an. Dennoch ist das Objekt-Paradigma zu den aktuellen Trends in der Informatik zu zählen, und somit erlaubt und verpflichtet der Lehrplan die Einbettung der Thematik innerhalb des Komplexes „Aktuelle Entwicklungen“ der Klassenstufen 12 (zwischen dynamischen Datenstrukturen und Datenschutz anzusiedeln) oder 13.

Speziell mit den Konzepten der Objektorientierung wird den Schülern ein starkes Hilfsmittel an die Hand gegeben, so daß die Auswahl der Werkzeuge für die anstehende Projektphase fachgerecht ausfallen kann.

¹²[Spolwig 95], S. 44

¹³Wirth, Niklaus: Im Vorwort zu „Objektorientierte Programmierung in Oberon-2: Das Versprechen der Objektorientierung.“

¹⁴[Lehrplan 93]

Insgesamt wird so den folgenden Ansprüchen des Lehrplans Rechnung getragen:

- Datenschutz durch Konzept der Kapselung.
- Ergänzende Beschreibung der Darstellung von Daten.
- Kommunikation Mensch-Maschine durch Diskussion und Erfahrung der Möglichkeiten Objektorientierter Benutzer-Dialoge / Oberflächen.

Objektorientierte Programmierung wird durchgängig mittels höherer Programmiersprachen eingeführt, es werden höhere Sprachelemente verwendet. Generell bietet es sich an Objektorientierung in der Projektphase einzusetzen. Aus den Anforderungen des rheinland-pfälzischen Lehrplanentwurfs werden die zentralen Prinzipien Modellbildung (Suche der essentiellen Gegenstände und objektorientierte Modellierung), Modularisierung (durch systematische Zerlegung) und Strukturierung (Vererbungshierarchien) thematisiert. Das Problemlösen mit Werkzeugen wird eingeübt, das Bewußtsein, daß eine Problemlösung in verschiedenen Sprachebenen und Beschreibungsformen (Klassendiagramm, Quellprogramm) dargestellt werden kann, wird geweckt. Da das theoretische Modell der Objektorientierung ein endlicher Automat ist, lassen sich u. U. Einblicke in das Wesen von Computern gewinnen.

3.1.3 Fachspezifisch-allgemeine Lernziele

QUIBELDEY-CIRKEL nennt¹⁵ drei Tendenzen, die das Objekt-Paradigma ver-eine und die in der Informatik von zentraler Bedeutung seien:

- Entwicklung abstrakter Sprachmittel, die dem Problem näher sind als der Maschine.
- Entlastung des Entwicklers von Entscheidungen auf unteren Ebenen.
- Suche nach interdisziplinären Prinzipien des Entwerfens.

BAUMANN nennt¹⁶ Algorithmische- und Datenabstraktion als grundlegende Konzepte der Informatik, Objektorientierung hängt mit diesen eng zusammen. Außerdem sei Objektorientierung ein weiterer Schritt in Richtung auf übersichtlich strukturierte, wartungsfreundliche und leicht wiederverwendbare Software. Ideen wie strukturiertes Programmieren und Modularisierung würden zuende gedacht.

¹⁵[Quibeldey-Cirkel 94], S. 13

¹⁶[Baumann 90], S. 284ff

SCHWILL schreibt¹⁷ dem Objekt-Paradigma die Fähigkeit zur Vermittlung grundlegender Konzepte wie Abstraktion, Klassifikation und Systematisieren zu:

„Wir favorisieren ... den objektorientierten Ansatz vor allem aus drei Gründen:

Erstens erfüllt dieses Paradigma unbestrittenermaßen informatisch orientierte Forderungen nach einem zeitgemäßen Unterricht mit mächtigen Konzepten, wie Erweiterbarkeit, Anpaßbarkeit, Rekonfiguration, Vererbbarkeit, Kapselung, evolutionäre Softwareentwicklung sowie Wünsche nach einer stärkeren Anwendungsorientierung durch Betonung der Nutzung des Computers anstelle von einem vertieften Verständnis seiner Funktionsweise.

Zweitens ist dieser Ansatz ein zentrales Merkmal eines nach dem Spiralprinzip organisierten Curriculums, auf höherem Niveau beliebig ausbaufähig. ...

Drittens — und dies erscheint aus pädagogischer Sicht der wichtigste Pluspunkt — ordnet sich der objektorientierte Stil in besonderer Weise harmonisch den elementaren kognitiven Prozessen unter, die beim Denken, Erkennen und Problemlösen im menschlichen Gehirn ablaufen.“¹⁸

Daß das Objekt-Paradigma ein Mittel zur Reduktion von Komplexität ist, nennt außer SPOLWIG auch EIRUND: „... sondern kann auch als eine Menge von Konzepten zur Reduzierung der Komplexität von Systemen verstanden werden.“¹⁹

Lernpsychologische Vorteile werden neben SCHWILL in „Eine Rechtfertigung aus kognitionspsychologischer Sicht“²⁰ auch von MÜLLER vorgebracht. Er stellt in [Müller 92], S. 160 dar, daß objektorientiertes Denken eine Vielzahl von Aspekten menschlicher Vorstellungs- und Denkweisen widerspiegeln, es würden Fähigkeiten wie Abstraktion, Klassifikation und Systematisieren trainiert. Auch Kooperation, eine wesentliche Komponente menschlichen Denkens und Handelns, werde vom Objekt-Paradigma unterstützt: Da Objekte autonom sind und das Verhalten des Systems durch das Zusammenwirken solcher Gebilde zustande komme.

¹⁷[Schwill 95], S. 160

¹⁸[Schwill 95], S. 183

¹⁹[Eirund 93], S. 40

²⁰[Eirund 93], S. 44f

WIRTH warnt neben lobenden Worten aber auch vor einer übertriebenen Euphorie:

„Denn es trifft nicht zu, daß alle Probleme nutzbringend in objektorientierter Weise neu programmiert werden. Im Gegenteil, die neue Methodik kommt erst recht zum Tragen, wo komplexe Datenstrukturen und Algorithmen ins Spiel kommen. Es wäre verfehlt, die konventionelle Sicht ad acta zu legen.“²¹

QUIBELDEY-CIRKEL geht im Kapitel „Zur Psychologie der Objektorientierung“²² darauf ein, daß das Objekt-Paradigma schöpferisches Problemlösen fördere, im Gegensatz zum rein algorithmischen Lösungsweg, der sich auf „unschöpferisches Aufgabenlösen“ durch „reproduzierendes Konstruieren“ beschränke.

3.2 Auswahl und Strukturierung der Inhalte und Ziele mit Begründung

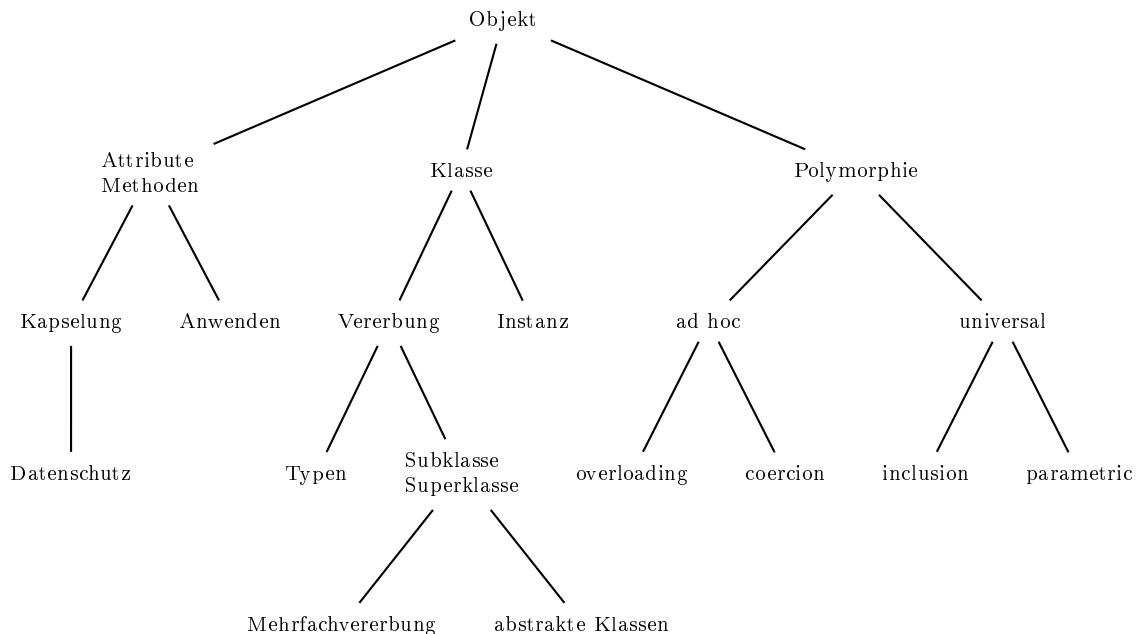
3.2.0 Intention der Unterrichtsreihe

Es soll eine erste Schüler-Begegnung mit Konzepten der Objektorientierung erreicht werden. Ein Kurs in Objektorientierter Modellierung wird erst bei großen, komplexen Problemen interessant, die sich aber leicht aus dem den Schülern bekannten oder intuitiven Kontext entfernen. Interesse wird leichter geweckt, wenn die neu erlernten Möglichkeiten nicht nur theoretisch geübt, sondern gleich praktisch angewendet werden können. Daher soll ein Kurs in Objektorientierter Programmierung gehalten werden, bei dem die Modellierungsphase zwar nicht „unter den Tisch gekehrt“ wird, sondern nur im Rahmen der Beispiele thematisiert wird. Der Schwerpunkt soll auch nicht auf der Umsetzung der Konzepte in einer Programmiersprache liegen, da die konkrete Umsetzung in jeder Sprache anders realisiert ist. Es geht um ein reines erstes Kennenlernen der Objektorientierten Mittel und Konzepte, später kann eine Erweiterung in verschiedene Richtungen erfolgen. Schüler sollen mit Objektorientierter Software in Berührung kommen und ihre Vorteile kennenlernen.

²¹Wirth, Niklaus: Im Vorwort zu „Objektorientierte Programmierung in Oberon-2: Das Versprechen der Objektorientierung.“

²²[Quibeldey-Cirkel 94], S. 124ff

3.2.1 Hierarchie der Objekt-Konzepte



Obiger Hierarchie-Baum der Objekt-Konzepte zeigt auf, daß die verschiedenen Konzepte voneinander abhängig sind. Das Betreten einer höheren Stufe ist erst nach Erreichen der Tieferen möglich.

3.2.2 Auswahl der Lernziele und didaktische Reduktion

Es ist nicht möglich Programmierung (oder Modellierung) im Sinne des Objekt-Paradigmas zu betreiben, ohne die Grundstruktur „Objekt“ zu verwenden. Schüler sollen daher lernen, mit Objekten umzugehen, Attribute und Methoden zu erkennen und Objekte sowohl anschaulich als auch in einer Programmiersprache richtig handzuhaben. Die Verwendung einer korrekten Fachsprache und der Fachtermini soll eingeübt werden.

Modellierung ist zentrales Thema der Informatik, daher soll bei dieser Gelegenheit der Schüler erneut in die Lage versetzt werden, zu einem realen Kontext Objektklassen zu modellieren, die den gestellten Anforderungen genügen. Allerdings darf die Intention dieser Unterrichtsreihe der Objektorientierten Programmierung nicht durch Modellierungsverfahren oder Notationen wie ER-Modell u. a. verdeckt werden.

Klassifikation der aufgestellten Objektklassen und geeignete Anordnung in einer Hierarchie führen ohne weiteres zu dem Mittel der Vererbung, die Termini Sub- und Superklasse oder analog Unter- und Oberklasse sollen Schülern geläufig sein, da sonst eines der stärksten Mittel der Objektorientie-

rung nicht gebraucht werden kann und die Einblicke, die diese Unterrichtsreihe bieten soll, zu stark beschränkt würden. Dabei kann zur Motivierung der Vererbungsmechanismen die erwünschte Wiederverwendung erstellter Programmteile herangezogen werden. Es reicht aus, an Vererbungsmechanismen „ist-ein“, „hat-ein“ und „ist-Spezialisierung-von“ kennenzulernen, da dies die in der Praxis am häufigsten auftretenden Möglichkeiten der Strukturierung sind.

Mehrfachvererbung ist für erste Begegnungen nicht nötig, sondern eher hinderlich, daher kann darauf verzichtet werden. Außerdem wird Mehrfachvererbung von vielen Systemen nicht unterstützt (z. B. Java). Da die Definition einer allen Subklassen gemeinsamen Schnittstelle nicht nur in einer abstrakten Oberklasse, sondern auch in den Subklassen erfolgen kann, ist der Begriff der abstrakten Klasse nicht zwangsläufig notwendig. Er kann zur vollständigen Strukturierung und Aufzeigen der Zusammenhänge leicht begrifflich eingeführt werden. Daß zu abstrakten Klassen stets eine Subklasse gehört, dürfte dann intuitiv klar sein. Der dadurch erfolgende Beschnitt der Mächtigkeit der Objektorientierung ist verschmerzlich, eine spätere Einführung kann erwartungsgemäß ohne Probleme erfolgen.

Polymorphie ist zentrales Mittel zur Erreichung von Übersicht und Ordnung, Methoden mit ähnlicher (gleicher) Funktion sollen den gleichen Bezeichner tragen, also einzig als ad-hoc-Polymorphie erkannt werden. Daß Methoden mit gleichem Bezeichner unterschiedliche Aufgaben erfüllen können, wird intuitiv durch den Umgang mit Standard-Programmen. (Icon „Scherer“ schneidet sowohl Dateien wie auch Texte usw. aus, „Tintenfaß“ (?) oder „Klembrett“ (?) fügen je nach Kontext das „richtige“, d.h. zum Kontext passenden Typ, Objekt aus der „Ablage“ ein). Verschiedene Typen der Polymorphie brauchen bei alleiniger Verwendung der ad-hoc-Polymorphie nicht unterschieden werden.

Die Mittel Vererbung und Polymorphie sollen so eingesetzt werden, daß das System maximale Übersichtlichkeit (nicht Kürze!) erhält, um Objektorientierung als Hilfsmittel zur Komplexitätsreduktion zu erkennen. Außerdem dient die Vererbung der Einsparung redundanten Quelltextes.

Der Begriff der Kapselung soll so eingesehen werden, daß die Daten nur über Objekt-eigene Methoden erreicht werden können und nach außen unsichtbar und unerreichbar sind. Dadurch können wichtige Aspekte der Datenschutzproblematik angesprochen werden, außerdem sollen Schüler einsehen, daß dies die Wartungsfreundlichkeit von Softwaresystemen erhöht, da unerwünschte Nebenwirkungen mit Sicherheit ausgeschlossen sind.

Schließlich können Objekte in fertigen Systemen (Betriebssystem, Office-Pakete o.ä.) erkannt und richtig genutzt werden.

3.2.3 Ordnung der Lernziele nach Kriterien grob / fein

- **Groblernziele**

Die Schülerinnen und Schüler sollen

- den Begriff des Objektes kennenlernen und ihn richtig anwenden können,
- das Konzept der Datenkapselung verstehen,
- die Idee der Objektorientierten Modellierung unter besonderer Berücksichtigung der Vererbung anwenden können,
- das Konzept der Polymorphie kennen,
- Vorteile Objektorientierter Programmierung erfahren.

- **Feinlernziele**

Die Schülerinnen und Schüler sollen

- wissen, daß ein Objekt aus Attributen und Methoden besteht und für einen realen Gegenstand oder Sachverhalt steht,
- ein einzelnes Objekt in Turbo-Pascal deklarieren können,
- das Aufrufen der Methoden eines Objektes beherrschen,
- eine Klasse von Objekten in Turbo-Pascal definieren können,

- erfahren, was Kapselung ist und wie sie funktioniert,
- erkennen, warum Daten gekapselt werden und
- wissen, wie diese Daten trotzdem geändert und ausgegeben werden können,
- Vorteile der Kapselung kritisch aufzeigen,

- einfache Miniwelten Objektorientiert modellieren können und
- das Konzept der Vererbung dabei sinnvoll anwenden,
- erkennen, daß das Konzept der Vererbung die Wiederverwendung von Softwarebauteilen erleichtert und damit fördert,
- anhand einer Objekthierarchie das Zusammenspiel der Objekte erkennen,
- Programmieren als Modellbildung und deren Realisation auf dem Rechner verstehen,

- die Strukturierung von Wissen als Mittel der Entflechtung von Komplexität erkennen,

- wissen, daß es mehrere Methoden und Attribute mit gleichem Bezeichner geben kann, die aber ein unterschiedliches Verhalten aufweisen können,
- erkennen, daß Polymorphie die Verständlichkeit von komplexen Systemen erhöht,
- Beispiele für sinnvoll angewandte Polymorphie nennen und erläutern können,

- die Konzepte der Objektorientierung anwenden und
- Zusammenhänge aufzeigen können,
- ihre Programmierfertigkeit verfeinern,

- lernen, in Teams (Gruppen) zu arbeiten und sich gegenseitig auszutauschen,
- in ihrer Kooperationsbereitschaft gefördert werden,
- trainieren, Ergebnisse einer Gruppe zu präsentieren.

3.3 Voraussetzungen und Vorkenntnisse

Da der Unterrichtsblock in der Klassenstufe 12 angesiedelt wird, verfügen die Schüler (gemäß Lehrplan) über Vorwissen aus den Bereichen der strukturierten Programmierung und können mit Prozeduren und Funktionen umgehen. Einfache und zusammengesetzte Datentypen sind bekannt, die Schüler besitzen Fertigkeiten in Auswahl, Definition und Umgang mit diesen Datentypen. Außerdem haben sie bereits Erfahrungen mit Modellierungsvorgängen. SPOLWIG berichtet, daß es „Von unschätzbarem Nutzen ist ..., wenn vorher eine Textverarbeitung in der Weise eingeführt wurde, daß es Textobjekte gibt (Zeichen, Wort, Zeile usw.), auf denen Operationen (Löschen, Einfügen, Kopieren) möglich sind.“²³

3.4 Zur Frage der Programmiersprache

Oben ist ausgeführt, daß nicht nur über Objekte gesprochen, sondern Objekte auch programmiert werden sollen. Es bleibt die Frage der Programmiersprache, zur Auswahl steht prinzipiell eine der pure- oder hybride- Programmiersprachen. Von Vorteil ist es, wenn keine neue Programmiersprache eingeführt werden muß, sondern die den Schülern aus dem Anfangsunterricht bekannte Sprache bereits das notwendige Rüstzeug mitbringt, um damit objektorientiert programmieren zu können. Dies gewährleisten z. B. die recht weit verbreitete Sprache Turbo-Pascal ab Version 5.5. Der Umstieg von zusammengesetzten Datentypen (`record` in Pascal) auf Objekte erweist sich als besonders einfach, da zu den einzelnen Datenfeldern nur noch die Operationen hinzugefügt werden müssen. Die Syntax bleibt weitgehend erhalten, Objekt-Methoden werden analog zu Datenbereichen mit dem `.` (Punkt-) Operator erreicht.

Eine Alternative ist es, von Turbo-Pascal auf Delphi umzusteigen. Dies hat den Vorteil, daß Delphi weitreichende Möglichkeiten zur Manipulation von bestehenden Objekten bietet; aus umfangreichen Klassenbibliotheken können nützliche Module ausgewählt werden, die sofort zur Verfügung stehen. Dafür besteht die Gefahr, daß das sehr umfangreiche Delphi den Einstieg durch die Komplexität der Programmierumgebung erschwert.

Von WIRTH u. a. wird Oberon empfohlen, auch der verfügbare Kurs²⁴ führt in Oberon ein. Da diese Unterrichtseinheit nur in die Konzepte einführen will, ist wohl das Erlernen einer kompletten Sprache mit der Mächtigkeit von Oberon unangemessen; soll jedoch zukünftig mehr mit Objekten gearbeitet werden, kann zweifelsohne Oberon gewählt werden.

²³[Spolwig 95], S. 48

²⁴[Hermes & Stein 96]

Aus aktuellem Anlaß ist Java als eine der modernsten Programmiersprachen zu nennen, deren Unterrichtstauglichkeit von BAUMANN angezeigt wird²⁵. Allerdings geht er von einer Einführung im Anfangsunterricht aus und hat damit andere Voraussetzungen als diese Reihe.

Sollte die den Schülern bekannte Sprache C sein, kann auf Java wie auch auf C++ umgestiegen werden, syntaktische Differenzen sind klein.

Da die fiktive Klasse Turbo-Pascal beherrscht, wird im folgenden Turbo-Pascal als Sprache verwendet, zum einen, weil sie die notwendige Struktur innehat, zum anderen, weil sich der Umstieg auf Objekte besonders einfach gestaltet. Die anfänglichen Umstellungsschwierigkeiten von ablauforientierten Programmen werden klein gehalten, da immer noch ein Hauptprogramm vorhanden ist und notfalls eine ähnliche Funktion übernehmen kann.

Ein rein objektorientiertes Programm besitzt kein Hauptprogramm mehr, die Objekte reagieren autonom. Da Turbo-Pascal hybride ist, gibt es ein Hauptprogramm, das aber lediglich den Charakter eines Steuermoduls besitzt und nicht zwangsläufig den Datenfluß regelt.

Eine Beschreibung der nötigen Objekt-Syntax findet sich im Anhang, ausführlich in [Tischer].

Doch auch die Nachteile von Turbo-Pascal sollen nicht verschwiegen werden:

- Mit dem Schlüsselwort `OBJECT` wird eine Klassendefinition gekennzeichnet.
- Ein Objekt als Exemplar einer Klasse wird als Variable im `VAR`-Block deklariert.
- Kapselung wird im strengen Sinne nur durch das zusätzliche Schlüsselwort `PRIVATE` erreicht.
- Um dynamisches Binden (late binding) zu erreichen, müssen die Methoden explizit als `VIRTUAL` deklariert werden, `procedure` muß für Initialisierungsmethoden durch `constructor` ersetzt werden.
- Polymorphie kann nicht voll ausgenutzt werden. Jedes Objekt darf Methodennamen in Turbo-Pascal nur einmal enthalten, unabhängig von deren Signatur.
- Es gibt kein `instanceof`, mit dem rasch die Klassenzugehörigkeit eines Objektes festgestellt werden kann. Dies ist zwar im strengen Sinn nicht nötig, da jedes Objekt seine Klassenzugehörigkeit kennt und selbst die nötigen Entscheidungen trifft, kann aber in manchen Situationen zu einer Erhöhung der Übersichtlichkeit dienen, wenn konkrete Fallunterscheidungen durchgeführt werden.

²⁵cf. [Baumann 97]

Trotz der hier geschilderten Nachteile soll Turbo-Pascal verwendet werden. Dies mag zwar in Anbetracht der „modernen“ Objektorientierten Sprachen Delphi, Java und Oberon wie ein Kompromiß erscheinen, bietet aber unter diesen Umständen eine ideale Ausgangsbasis für jedwede spätere Orientierung. Sind die Konzepte erst einmal bekannt, kann ebenso leicht auf die syntaktisch C-ähnliche Sprache Java wie auch auf die syntaktisch Pascal-ähnlichen Sprachen Delphi und Oberon umgestiegen werden. Es mag die Frage aufkommen, warum dann nicht gleich eine dieser Sprachen gewählt wird, sondern erst im altbekannten Turbo-Pascal begonnen wird. Dies kann dahingehend beantwortet werden, daß

1. diese Reihe wirklich nur *einführen* will und die Konzepte dazu in ausreichendem Maß von Turbo-Pascal geboten werden,
2. der Umstieg auf eine andere Sprache nicht zu weiterem Erkenntnisgewinn führt,²⁶
3. dem Schüler die Neuen Konzepte deutlicher werden, wenn er nicht noch eine gesamte, mächtigen Programmiersprache kennenlernen und sich nicht in eine neue, unbekannte Entwicklungsumgebung einarbeiten muß, sondern die ihm bekannte Syntax, Befehlsvielfalt und die erprobte Entwicklungsumgebung weiterverwenden kann,
4. in Zukunft (z. B. in der Projektphase) gerade durch die Beschränkung wirkliche Auswahl zwischen allen Sprachen besteht, da alle Grundmittel und -konzepte bekannt sind und eingeübt wurden.

Außerdem soll nocheinmal darauf hingewiesen werden, daß, sollte Turbo-Pascal den Schülern *nicht* bekannt und eine Neuorientierung notwendig sein, gleichwohl auf Java, Delphi oder Oberon umgestiegen werden kann. Jede dieser Sprachen²⁷ hat Vor- und Nachteile, die Entscheidung kann nicht generell gefällt, sondern muß im Kontext der bisherigen und der angestrebten Ausbildung getroffen werden.

3.5 Lernkontexte

An den Lernkontext werden verschiedene Anforderungen gestellt. Die Komplexität soll dem Leistungsvermögen des Kurses angemessen sein, zugleich

²⁶allenfalls die Aufteilung der Klassen auf jeweils eigene Dateien läßt bei Java erkennen, wie autonom die Objekte wirklich sind, und daß die Dynamik ganz aus den Objekten selbst entsteht; dies kann aber – wenn gewünscht – in Turbo-Pascal mittels `{ $I datei }` simuliert werden, falls alle Klassen mit ihren Methoden in eigenen Dateien abgelegt werden.

²⁷und gegebenenfalls auch weitere, diese Aufzählung soll keinen ausschließenden Charakter haben

aber auch beim Einstieg nicht zu hoch, damit der Schwerpunkt den neuen Stoff erfaßt. Um den Zugang zum Systemverständnis leicht zu halten, sollte der Lernkontext keine langwierigen Systemanalysen nach sich ziehen und aus der Erfahrungswelt der Schüler stammen. Außerdem sollte das für das spezifische Problem notwendige Vorwissen möglichst gering sein, um die Problematik nicht zu verdecken. Müller meint dazu: „So wird im Unterricht mehr nach einem geeigneten Beispiel gesucht, das eine algorithmische Grundstruktur bzw. einen ausgewählten Datentyp günstig einführt. Da zusätzlich die praktische Arbeit nicht zu kurz kommen soll, darf die Problemanalyse und -Strukturierung nicht zu viel Zeit in Anspruch nehmen.“²⁸ Um möglichst viele Konzepte einüben und die Vorteile der neuen Techniken einsehen zu können, sollte die Objektstruktur möglichst plastisch sein, dabei aber nicht künstlich konstruiert wirken. Es bieten sich verschiedene Gebiete an:

- Simulation eines (Brett-)Spiels.
- System zur Bruchrechnung.
- Grafiksystem.

Die einzelnen Vorschläge sollen nun kurz diskutiert werden:

Die Simulation eines Spiels bietet eine ausgesprochen plastische und detaillierte Vererbungsstruktur der einzelnen Figuren, die in [Glaeser] kurz vorgestellt wird. Insbesondere Schach ist sehr reizvoll, aber auch recht komplex und wird nicht von allen Schülern gespielt. Zugoperationen und Regelwerk sind schwierig, hinzu kommt eine notwendige, umfangreiche Realisierung der Spielumgebung, die Züge gestattet und erfolgte Züge nachvollziehen läßt. Polymorphie kann besonders eindrucksvoll an einer Methode zug gezeigt werden, die je nach Figur ein anderes Verhalten zeigt. Andere Spiele bieten oft keine ausreichende Klassenhierarchie oder sind ähnlich Schach zu komplex. Es ist aber möglich, ein eigenes Kunst-Spiel – z. B. durch Weglassen oder Hinzunehmen von Figuren oder Regeln – zu kreieren, das den Anforderungen gerecht wird.

Ein System zur Bruchrechnung, angegeben von BAUMANN in [Baumann] ist auch nicht sehr plastisch, dafür aber von sehr einfacher Struktur. Polymorphie wird erst bei komplexeren Strukturen nötig, es sind Hilfsfunktionen wie ggT und kgV nötig. Die notwendige Theorie ist jedem Gymnasiasten bekannt. Ausgesprochen motivierend wirkt ein solches Problem aber nicht, das System Bruchrechnung eignet sich daher eher als vertiefendes Beispiel zur Übung und Kontrolle denn als einführendes Beispiel.

²⁸[Müller 92], S. 158

Grafische Objekte sind jedem Schüler aus der Mathematik geläufig, sowohl zweidimensionale als auch dreidimensionale. Auch notwendige Berechnungen sind einfach genug, um nicht vom thematischen Schwerpunkt abzulenken. Strukturen können individuell aufgebaut und nahezu unbegrenzt erweitert werden, um unterschiedliche Schwierigkeitsgrade zu realisieren, Beispiele sind aus dem Bereich der Mathematik genügend vorhanden und anschaulich. Zwei Grundideen sollen angegeben werden:

1. Von verschiedenen dreidimensionalen Objekten wie Pyramiden auf quadratischer Grundfläche, Kegeln und Tetraeder gekennzeichnet durch ein Grundmaß und die Höhe soll der Oberflächen- und Volumeninhalt berechnet werden. Die Anschauung ist begrenzt durch die räumliche Vorstellung, Erweiterungsmöglichkeiten sind gegeben, werden jedoch rasch komplex.
2. Zweidimensionale Objekte wie gleichseitige Dreiecke, Quadrate oder Kreise gekennzeichnet durch ihren Mittelpunkt und einen Radius sollen auf dem Bildschirm gezeichnet werden. Methoden zum Erzeugen (Zeichnen), Verschieben und Löschen sollen angegeben werden. Erweiterungen sind in beinahe unbegrenztem Maße möglich: beliebige regelmäßige Polygone, Flächenberechnungen, Einfärbungen, Schraffuren usw. sind mögliche Vorschläge. Die Operationen zum Zeichnen der Objekte sind nicht schwierig, wenn entsprechende Vorgaben gemacht werden und führen schnell zu Erfolgsergebnissen, die zu weiteren Experimenten ermutigen. Notwendige Kenntnisse der Geometrie und Flächenberechnung sind aus der Sekundarstufe I in ausreichendem Maße vorhanden und gehen nicht über den Satz des Pythagoras und die Kreisformel hinaus.

Dieses letzte Beispiel soll zum motivierenden Einstieg verwendet werden.

Formeln

Den Flächeninhalt eines Quadrates $F_Q = g^2$ sollte jeder Gymnasiast der 12. Klasse berechnen können, ebenso die Fläche eines Kreises $F_K = \pi \cdot r^2$. Dagegen dürfte der Ellipseninhalte $F_E = \pi \cdot r_1 \cdot r_2$ den meisten unbekannt sein und muß daher vorgegeben werden.

3.6 Lehrbuch

Verschiedene Schulbücher gehen auf Objektorientierung ein, hier seien exemplarisch genannt:

- [Harbeck & Thode] nennt Objektklasse und führt Vererbung als Fortführung abstrakter Datentypen ein.
- [Baumann] geht auf Grundbegriffe der objektorientierten Programmierung ein, zur Einführung werden dreidimensionale geometrische Gebilde verwendet.
- VON HERMES & STEIN liegt ein Kurs „Objektorientierte Programmierung“ in Form eines Heftes [Hermes & Stein 96] vor, der insbesondere auf Grafik und Oberon eingeht.

Von diesen Bücher wird hier keines explizit verwendet, statt dessen werden eigene, speziell auf das zu erstellende Graphiksystem abgestimmte Materialien an die Schüler ausgegeben.

3.7 Besondere Schwierigkeiten

Schwierigkeiten sind beim Begriff der Polymorphie zu erwarten, da Schülern u. U. gar nicht auffällt, daß mehrere Methoden den gleichen Bezeichner tragen und daß dies nicht selbstverständlich ist. Hier kann ergänzend darauf hingewiesen werden, daß bereits früher Polymorphie verwendet wurde, z. B. ist das Additionssymbol $+$ überladen, außer ganzen Zahlen können auch Dezimalzahlen und Zeichenketten „addiert“ werden.

Problematisch ist auch die Verwendung von Subklassenmethoden im Superklassenkontext mittels des Schlüsselwortes `self`, was Verwirrung (Verwendung von *late binding* und den sog. *virtuellen* Methoden) stiften dürfte. Umgehen lässt sich dies durch ein sauber strukturiertes Klassendiagramm, in dem die Hierarchie deutlich hervorgehoben ist. Durch Pfeile kann markiert werden, wer wann auf welche Methode zugreifen möchte und welches Objekt welcher Hierarchie dazu angesprochen werden muß.

Auch Kapselung wird u. U. als überflüssig empfunden, weshalb und vor wem sollen Datenzugriffe geschützt sein? Hier kann motivieren, daß es sich nicht nur um einen Schutz vor Lese-, sondern auch vor Schreibzugriffen handelt, so daß das Programm vor Fehleingaben geschützt wird.

4 Methodische Analyse

4.1 Lernmethodische Aspekte

„Man kann objektorientiertes Denken besonders gut lernen, wenn man entsprechend den vorgestellten didaktischen Linien die Gelegenheit hat, ein selbst oder von anderen implementiertes informationelles Modell interaktiv zu ändern, zu verfeinern, zu erweitern und mit anderen implementierten Modellen zu kombinieren. Auch wenn man damit ganz klein und ohne viele Voraussetzungen beginnen kann, ist objektorientiertes Programmieren ein weittragendes Denkinstrument.“²⁹

Der Lernprozeß kann auf zwei Methoden angelegt werden:

Beim synthetischen Zugang wird ein zu erstellendes System von Grund auf neu konstruiert, die kennenzulernenden Konzepte werden bei der Konstruktion eingesetzt und ihre Notwendigkeit direkt eingesehen. Diese Vorgehensweise ist bei der Einführung der Objektorientierung schwierig, da kein unmittelbarer Bedarf vorliegt, die Vorteile der Objektorientierung können ja nicht im Vorfeld erkannt werden. Außerdem würde eine unmotivierte abstrakte Einführung von Konzepten notwendig.

Beim analytischen Zugang wird ein bestehendes System analysiert, erweitert und verbessert. Zuerst kann mit den Objekten eines fertigen Systems experimentiert werden. Dabei werden die Möglichkeiten der Objektorientierung direkt angewendet und so durch praktisches Anwenden erlernt. Dieser Zugang wird von SPOLWIG in [Spolwig 95] an einem praktischen Beispiel (Fahrplanautomat) beschrieben. Ein Vorteil liegt darin, daß erste Ergebnisse sehr schnell vorliegen, da die Vorlage ja „nur“ angepaßt zu werden braucht. Durch die Möglichkeit, direkt das bestehende System zu erweitern, wird individuell kreatives Arbeiten erlaubt.

Werden beide Zugänge gemischt, können die Vorteile beider Vorgehensweisen kombiniert werden: Analytisch beginnend soll ein vorgegebenes Minimalsystem untersucht und an ersten minimalen Erweiterungen erprobt werden, eine anschließende Besprechung soll die Konzepte klären und benennen. Anschliessend kann synthetisch, d. h. die richtigen Mittel bewußt anwendend, fortgeführt werden. Später werden weitere Konzepte bei Bedarf eingeführt. Dadurch, daß zu Beginn ein Grundsystem gegeben ist, sind Fortschritte und Ergebnisse schnell sichtbar und wirken motivierend auf die Schüler. In der zweiten Phase ist das Wissen um die Anwendung der Konzepte vorhanden

²⁹[Crutzen & Hein 95], S. 158

und führt mit Hilfe den in der ersten Phase gemachten Erfahrungen schnell zu korrekten Ergebnissen.

4.2 Unterrichtsmethodische Aspekte

Die Unterrichtseinheit ist als Dialog zwischen wissendem Lehrer und erprobenden Schülern angelegt. Die zu Beginn gestellte Aufgabe stellt sich schnell als schwierig und umfangreich heraus, der Lehrer liefert einen „Anfang“, der – als Anreiz und Quelle dienend – ausgebaut werden soll. Anhand dieses Impulses werden Schüler am Rechner auf neue, unbekannte Konzepte und Techniken gestoßen, die im weiteren Verlauf immer weiter ausgebaut werden. Durch direktes Arbeiten mit den Mitteln am Rechner werden die ungewohnten Konzepte in ihrer Anwendung erlernt, anschließend im Gespräch fachlich korrekt abgeklärt und fixiert. Jedes Ziel der Unterrichtsreihe wird so Stück für Stück kennengelernt, eingeübt, fixiert und vertieft sowohl durch weitere praktische Tätigkeit als auch eher theoretische Überlegungen.

Dabei überwiegen die Sozialformen der Partnerarbeit am Rechner und Gruppenarbeit zu gemeinsamen theoretischen Überlegungen; gegenseitig kann Neues leichter verstanden und erklärt werden. Im Unterrichtsgespräch werden die kennengelernten Konzepte gemeinsam fachlich korrekt erörtert und vertieft, Beispiele aufgezeigt und Grenzen bestimmt. Die Einzelarbeit in der Hausaufgabe schließlich dient der selbständigen individuellen Beschäftigung mit den Konzepten sowie einer Wiederholung und Vorbereitung.

Für die praktische Arbeit werden Rechner mit der Sprache Turbo-Pascal zur Programmierung eingesetzt, Tafel und Hefte dienen der Fixierung von Definitionen, Merksätzen, Anwendungsbeispielen und allgemeinen Überlegungen. Tageslichtprojektor und Folien werden für mehrfach verwendete Grafiken wie z. B. die Klassenhierarchie verwendet. Auch zur Kontrolle von Hausaufgaben und der gegenseitigen Vorstellung entwickelter Lösungen können Folien dienen.

Der Lernerfolg wird regelmäßig kontrolliert, für diesen Zweck werden Übungsaufgaben vergeben und überprüft. Als Übungsaufgaben werden dazu Teilprobleme ausgewählt, die die entsprechenden Aspekte in ausreichendem Umfang berücksichtigen. Darüberhinaus dient die Hausaufgabe der Vorbereitung des nächsten Themenschwerpunktes, sie soll die Problemlösefähigkeit fördern.

5 Unterrichtsplanung (Skizze)

5.1 Unterrichtsplanung

Einführung

Zur Einführung wird das Problem gestellt, ein Computergeometriesystem zu entwickeln, das es erlauben soll:

- Punkte, Kreise, Quadrate u. ä. geometrische Objekte zu zeichnen,
- diese zu verschieben und zu löschen,
- ihren Flächeninhalt rechnerisch zu bestimmen und auszugeben.

Die Aufgabenstellung wird als Arbeitsblatt ausgeteilt.

Nachdem gemeinsam rasch festgestellt wird, daß das gegebene Problem recht komplex ist, wird als erster Schritt der Entzerrung an die Grafik-Befehle von Turbo-Pascal erinnert und eine entsprechende Zusammenfassung ausgeteilt. Eventuell kann ein Schüler kurz referieren und eine entsprechende Zusammenfassung erstellen, falls der Zeitrahmen und der Ablauf der vorherigen Stunden dies erlauben.

Im Anschluß wird ein knapp gehaltenes, minimales Programm ausgeteilt, das bereits mit Punkten als geometrischen Objekten umgehen kann. An Methoden stehen nur Initialisieren und Zeichnen zur Verfügung. Zu diesem Programm gibt es eine konkrete Aufgabenstellung zur Herausarbeitung der neuen Begriffe. Anhand dieses Programmes erfolgt die erste Begegnung der Schüler mit Objekten; Es können neue Punkte erstellt und gezeichnet werden, dabei wird intuitiv mit den Begriffen und der Syntax umgegangen. Der den Schülern bereits bekannte Punkt-Operator führt vom Objekt zu seinen Methoden. Schließlich werden die neuen Schlüsselworte an der Tafel gesammelt und ggf. geklärt.

Ergänzung zur Einführung

Die obiger Vorgehensweise leicht abwandelnd ist es auch möglich, daß der Lehrer seinem Kurs ein fertiges Objekt übergibt, daß die Aufgabe des Hauptprogramms übernimmt und es gestattet, mit Hilfe der Tastatur (Cursor-Tasten) oder der Maus interaktiv geometrische Objekte auf dem Monitor zu erstellen. Dadurch müßte der Schüler keine eigenen Objekte instantiieren und initialisieren und kann sich mehr auf den Modellierungsprozeß und die Programmierung der Klassen konzentrieren. Sinnvoll ist eine Mischung beider Vorgehensweisen, so daß in einer ersten Phase alle Objekte selbst instantiiert, initialisiert und gezeichnet werden (die Kontrolle liegt voll beim

Schüler), und anschließend ein Kontroll-Objekt vorgegeben wird. Ein solches Kontroll-Objekt wäre vom Lehrer zu realisieren und als Black-Box in seiner Funktionalität vorzustellen, es müßte neu entstehende und bereits entstandene Objekte dynamisch verwalten.

Alternative Einführung

Besonders deutlich wird die Notwendigkeit neuer Konzepte, wenn die Schüler sie selbst erfahren. Dazu wird analog obiger Einführung das Problem gestellt, das Computergeometriesystem zu entwickeln. Statt im Vorfeld den Aufwand zu klären, sollen die Schüler einfach „drauflos programmieren“, ohne das Problem genau zu analysieren. Dabei können sie selbständig durch direkte Anwendung die notwendigen Grafikbefehle wiederholen. Wird dann problematisiert, daß, sobald zwei Kreise nacheinander gezeichnet werden, die Koordinaten des ersten unerreichbar sind, um ihn beispielsweise zu verschieben, wird die Komplexität der neu einzuführenden Datenstruktur, die dieses Problem beheben soll, sehr schnell deutlich.

Dieser Weg, bei dem die Schüler selbst wiederholen und dabei die Notwendigkeit neuer Verfahren entdecken, ist recht zeitaufwendig und wird daher „nur“ als Alternative behandelt. Die Vorteile liegen auf der Hand: Selbständig wird erinnert, selbständig wird die Notwendigkeit aufgrund zu hoher Komplexität erkannt. Anschließend können die Grundstrukturen anhand des Programmfragmentes erarbeitet werden, die weitere Vorgehensweise kann genau wie beschrieben erfolgen.

Einarbeitung

Nach einer kurzen Besprechung der weiteren Vorgehensweise und der zu Hause entwickelten Ideen werden die Methoden `hide` und `move` für Punkte fertiggestellt. Das Verstecken eines Punktes wird z. B. dadurch gelöst, daß er einfach schwarz übermalt wird. Das hierzu nötige Wissen ist klein, es wird außer der Idee nur der Befehl zum Zeichnen eines Punktes gebraucht, die Methode kann analog der zum Zeichnen entstehen, ggf. sogar abkopiert und nur umbenannt werden. Dabei wird auf die Objekt-Attribute zugegriffen, und es ist durchaus denkbar, daß Schüler dies mit nicht-Objekt-eigenen Methoden versuchen. Da die Attribute durch das Schlüsselwort `private` explizit gekapselt sind und von aussen unzugänglich verborgen liegen, werden die Schüler im ersten Ansatz vermutlich scheitern. Dieses „Scheitern“ erfolgt bewußt, denn nur wenn Schüler selbst erfahren, was es heißt das Daten gekapselt sind, erkennen sie die Vorteile und problematisieren dies.

Dies läßt sich nutzen, gleich auf die Kapselung als wesentliches Konzept der Objektorientierung zu sprechen zu kommen, es bietet sich an auch gleich auf Fragen des Datenschutzes und den Vorteil für die Anwendung nicht-selbst-erstellter Klassen einzugehen. Da nur autorisierte Objekt-eigene Methoden Daten ausliefern, lassen sich Daten wesentlich einfacher vor unerlaubter Manipulation schützen. Ebenso wird verhindert, daß Attribute Werte zugewiesen werden, für die sie nicht ausgelegt wurden, es kann z. B. leicht verhindert werden, daß Wertebereiche überschritten werden. Die Kontrolle für einzulesende Daten liegt nicht mehr beim Eingebenden, sondern beim einlesenden Objekt.

Alternativ ist es möglich `hide` so zu realisieren, daß man Punkte mit einem weiteren Attribut versieht, das entweder die Farbe oder aber einen Zustand (sichtbar/unsichtbar) beinhaltet. Dann muß `show` gerufen werden und den Punkt mit der geänderten Farbe oder aber mit der Hintergrundfarbe übermalen. Dazu muß die Hintergrundfarbe einheitlich festgelegt werden, konventionell auf schwarz. Es ist auch möglich, `show` als Parameter die zu verwendende Farbe zu übergeben, so daß die Methode `hide` auf eine Zeile gekürzt werden kann, ohne daß `show` wesentlich komplexer würde.

Modellierung

Nachdem Punkte als geometrische Objekte jetzt weitgehend fertiggestellt sind, reicht die Erfahrung aus, um die Miniwelt zu modellieren. Dazu wird festgestellt, daß Kreise und Quadrate beide einen Mittelpunkt und ein Grundmaß besitzen, das beim Kreis *Radius*, beim Quadrat *halbe Grundseite* heißt. Dabei wird arbeitsteilig vorgegangen, eine Gruppe sucht die für Kreise, die andere die für Quadrate notwendigen Daten und stellt die Methoden auf. Dabei hilft eine entsprechend formulierte Aufgabenstellung, daß auch „Quadrat“ mit einem Punkt als Aufhänger modelliert wird und nicht z. B. alle Eckpunkte notwendig sind. Anschließend stellen die Gruppen sich gegenseitig die Ergebnisse vor, ein günstig vorformatiertes Tafelbild oder eine Folie helfen, die Gemeinsamkeiten zu erkennen. Folglich lassen sich sowohl Kreis als auch Quadrat mit den gleichen Attributen beschreiben, werden aber völlig unterschiedlich gezeichnet. Zusätzlich ist auch der Flächeninhalt unterschiedlich, wie aus Klassenstufe 10 bekannt ist. Motiviert durch die Einsparung von weiteren Prozeduren wird eine neue Klasse `flaeche` geschaffen, die die Attribute Mittelpunkt und Grundmass sowie die Methoden `move` und `hide`, hat. Von `flaeche` werden dann `kreis` und `quadrat` mit jeweils eigenen Methoden `show` und `inhalt` abgeleitet.

Diese Klassenhierarchie wird dann implementiert, wobei mit der Oberklasse `flaeche` begonnen werden sollte. Sinnvoll ist es, `hide` als letztes zu

implementieren, da dort zusätzliches Wissen investiert werden muß: Das zu `flaeche` gehörende `hide` muß wissen, ob es ein gegebenes Objekt mit `show` der Klasse `kreis` oder mit `show` der Klasse `quadrat` zeichnen soll. Da es kein `instanceof` gibt, mit dem die Klassenzugehörigkeit festgestellt werden könnte, bleiben zwei Möglichkeiten:

- Jede der Klassen `kreis` und `quadrat` erhält ein eigenes `hide`, damit zwangsläufig aber auch ein eigenes `move`. Damit sind einige der Vorteile Objektorientierter Programmierung hinfällig, dieser Weg wird wohl von Schülern eingeschlagen werden (da er offensichtlich nicht abwegig ist), es sollte aber zumindest auf die bestehende Alternative hingewiesen werden.
- Erlaubt man Turbo-Pascal Methoden dynamisch zu binden, ist das Problem gelöst. Dazu müssen die entsprechenden Methoden als `virtual` deklariert werden, auf die jeweils zur Klasse des aktuellen Objektes zugehörigen Methoden kann dann mit `self` zugegriffen werden. (Dazu dürfen die `init`-Methoden nicht als `procedur`, sondern müssen als `constructor` deklariert sein.)

Dieses durchaus etwas undurchsichtige Aufrufverfahren sollte mittels geeigneter Folien — in denen die Aufruf-Pfade innerhalb einer Objekthierarchie markiert werden — erläutert werden.³⁰

Erweiterungsmöglichkeiten

Kleine Ergänzungen sind rasch einzufügen, z. B. kann ein zusätzliches Attribut eingefügt werden, das die Farbe festlegt. Durch diese kleinschrittigen Einführungen neuer Bestandteile lernt der Schüler mit der wachsenden Komplexität seines Systems den Umgang kennen und kann die Zusammenhänge zwischen einzelnen Bestandteilen verstehen und erfassen. Wird ein weiteres geometrisches Objekt eingefügt, z. B. eine Ellipse (bei der zur Beschränkung auf ein Grundmaß Halbmesser doppelt so lang ist wie der andere), so kann dies sehr rasch geschehen: Es müssen nur Methoden zum Zeichnen und Berechnen des Flächeninhalts programmiert werden, dennoch können Ellipsen auch versteckt oder verschoben werden! Hier lassen sich besonders eindrucksvoll die Vorteile (Wiederverwendbarkeit, Übersichtlichkeit) der Objektorientierung erfahren.

Andererseits ist es aber auch möglich, für die Ellipse einen zweiten Halbmesser als neues, Ellipsen-eigenes Attribut einzuführen, oder gleich eine ganze Oberklasse von Flächen, die zwei Grundmaße brauchen. Damit wären dann

³⁰cf. Anhang B

neben Ellipsen und Rechtecken z. B. auch gleichschenklige Dreiecke abgedeckt.

Auch die Linie (eigentlich: die Strecke), ein geometrisches Grundelement, kann leicht eingeführt werden, sie besteht nur aus zwei Endpunkten. Daher kann `move` von Punkten weiterverwendet werden, neu zu erstellen ist – wie bei Kreisen, Quadraten und Ellipsen auch schon – die `show`-Methode.

Sinnvoll erweitern läßt sich ein solches System in vielen Richtungen, es können z. B. Drehwinkel gegen die Horizontale eingeführt werden. Dabei wird aber – neben dem notwendigen mathematischen Wissen – stets im Wesentlichen die Methode „Zeichne“ komplizierter, der Modellierungsvorgang wird nur ein geringes Stück vorangetrieben.

Von Nachteil ist es in diesem System noch, daß beim Löschen eines geometrischen Objektes die verdeckten – räumlich scheinbar dahinter liegenden – geometrischen Objekte auf dem Bildschirm teilweise zerstört aussehen, außerdem muß noch für jedes einzelne geometrische Objekt eine eigene Variable eingeführt werden. Werden alle Objekte dynamisch in einer Liste (oder einer anderen sinnvollen Datenstruktur) gespeichert, können nach einem `hide`-Vorgang die anderen überlappten geometrischen Objekte neu gezeichnet werden, indem für jedes der Reihe nach `self.show` aufgerufen wird. So kann leicht auch ein geometrisches Objekt „in den Vordergrund gerückt werden“, indem es in der Liste auf einen letzten Platz verschoben wird und somit erst nach – also über – den anderen Objekten gezeichnet wird.

Abschluß

Zum Schluß soll eine kurze Diskussion geführt werden, in der Vor- und Nachteile der Objektorientierten Programmierung zur Sprache kommen sollen. Die Schüler sollen selbst die gemachten Erfahrungen kritisch reflektieren und zur Sprache bringen.

5.2 Unterrichtsplanung detailliert: Schritte und Phasen

1. Schritt³¹

1. Phase: Problemstellung

- Inhalt/Ziel:
 - Aufgabenstellung: Geometrieprogramm
 - Was ist für ein solches Programm notwendig ?
 - Diskussion, Lösungsansätze
 - Gliederung: Grafikprogrammierung, Geometrie, „eigentliches Problem“.
- Medien: Arbeitsblatt AB1, ggf. Tafel
- Sozialform: Unterrichtsgespräch.

2. Phase: Erinnerung an Grafik

- Inhalt/Ziel:
 - Bereitstellung der notwendigen Grafikbefehle
- Medien: Arbeitsblatt „Grafik mit Turbo-Pascal“, ggf. Folie
- Sozialform: Unterrichtsgespräch
- Alternativ: Schülervortrag, Kurz-Referat.

3. Phase: Erkunden und Ausprobieren des vorgegebenen Programms

- Inhalt/Ziel:
 - Bekanntwerden mit den neuen Schlüsselworten
 - Einarbeiten in die Struktur Objektorientierter Programme
 - Begegnung mit dem Objektbegriff
- Medien: Arbeitsblatt AB2, Rechner, Pascal, Programm, Papier
- Sozialform: Gruppenarbeit, Partnerarbeit

4. Phase: Ergebnisse sammeln

- Inhalt/Ziel:
 - Zusammentragen der neuen Schlüsselworte

³¹cf. die vorgestellten Alternativen zu Phase 1 und 2.

- wie werden Objekte in TP definiert ?
- wie wird mit Objekten gearbeitet ?
- Sicherung dieser Ergebnisse
- Medien: Tafel1, Heft.
- Sozialform: Unterrichtsgespräch, Schülervortrag, ggf. Lehrvortrag zu einzelnen Stichworten.

HA:

- Wie kann man einen Punkt wieder verstecken oder verschieben?
(Idee)

2. Schritt

1. Phase: Besprechung

- Inhalt/Ziel:
 - Besprechung der HA: Wie kann ein Punkt „versteckt“ werden ?
 - Idee: Ein Punkt kann durch Übermalen versteckt werden.
 - Idee: Verschieben durch Verstecken, Ändern der Daten und anschließendes Neumalen.
 - Gewöhnung an den Umgang mit der (neuen) Fachsprache
- Medien: ggf. Tafel
- Sozialform: Unterrichtsgespräch mit Diskussion

2. Phase: Implementierung

- Inhalt/Ziel:
 - Einüben des Umgangs mit Objektorientierten Hilfsmitteln
 - Erzeugen der Methoden `hide` und `move` für Punkte.
 - Erstellen eigener Methoden, Erprobung.
- Medien: Rechner, Programm
- Sozialform: Partnerarbeit

3. Phase: Kapselung

- Inhalt/Ziel:
 - Daten sind durch `private` nicht nach aussen zugänglich.
- Medien: Rechner, Programm, Tafel2, evtl. Folie.
- Sozialform: Partnerarbeit, Unterrichtsgespräch

4. Phase: Sicherung u. Reflektion

- Inhalt/Ziel:
 - Wozu ist Kapselung gut ?
 - Definition der neuen Begriffe
- Medien: Tafel2, Heft
- Sozialform: Unterrichtsgespräch, ggf. Lehrvortrag

3. Schritt

1. Phase: Modellierung

- Inhalt/Ziel:
 - Wodurch unterscheiden sich Quadrate und Kreise ? (→ Methoden für Flächeninhalt und Zeichnen)
 - Zusammentragen der nötigen Methoden und Daten für Kreise und Quadrate
 - Zusammenfassen ähnlicher Eigenschaften ist sinnvoll
 - Vererben der Eigenschaften an alle Flächen, d. h. insbesondere an Kreis(e) und Quadrat(e).
 - Aufstellen der Objekthierarchie.
- Medien: Arbeitsblatt AB3, Tafel3, Tafel4, Folie „Objekthierarchie“
- Sozialform: Unterrichtsgespräch, Schülervortrag

2. Phase: Implementierung

- Inhalt/Ziel
 - Erste eigene Implementierung einer Klasse
 - Einüben der Definitionen
 - Methoden `init` und `show` schreiben, anschliessend `hide`
 - ggf. : Kennenlernen von „hat ein“-Vererbung, mittels Fläche hat ein(en) Mittelpunkt.
 - Problem: `hide` muß wissen, mit *wessen* `show` es übermalen soll
- Medien: Rechner, Programm
- Sozialform: Partnerarbeit

4. Schritt

1. Phase: Polymorphie und virtuelle Methoden

- Inhalt/Ziel:
 - Woher kennt eine Superklasse die Zugehörigkeit eines Objektes zu einer der Subklassen?
 - Input: `virtual` mit `self`.
- Medien: Tafel5, Tafel6, Folie „Objekthierarchie: Aufruf von `move`“
- Sozialform: Schülervortrag, Lehrvortrag

2. Phase: Implementierung

- Inhalt/Ziele
 - Umsetzung der neuen Konzepte: Oberklasse kann Unterklasse eindeutig aufrufen.
 - Realisierung der restlichen noch offenen Methoden
 - Sicherung
- Medien: Rechner, Programm
- Sozialform: Partnerarbeit

HA: Was ist notwendig für Erweiterung zu neuer Fläche Ellipse?

5. Schritt

1. Phase: Besprechung

- Inhalt/Ziel:
 - Vorstellung der Ellipsen-Formeln
- Medien: Tafel7 oder Folie mit Tafel7
- Sozialform: Schülervortrag, Lehrvortrag, Unterrichtsgespräch

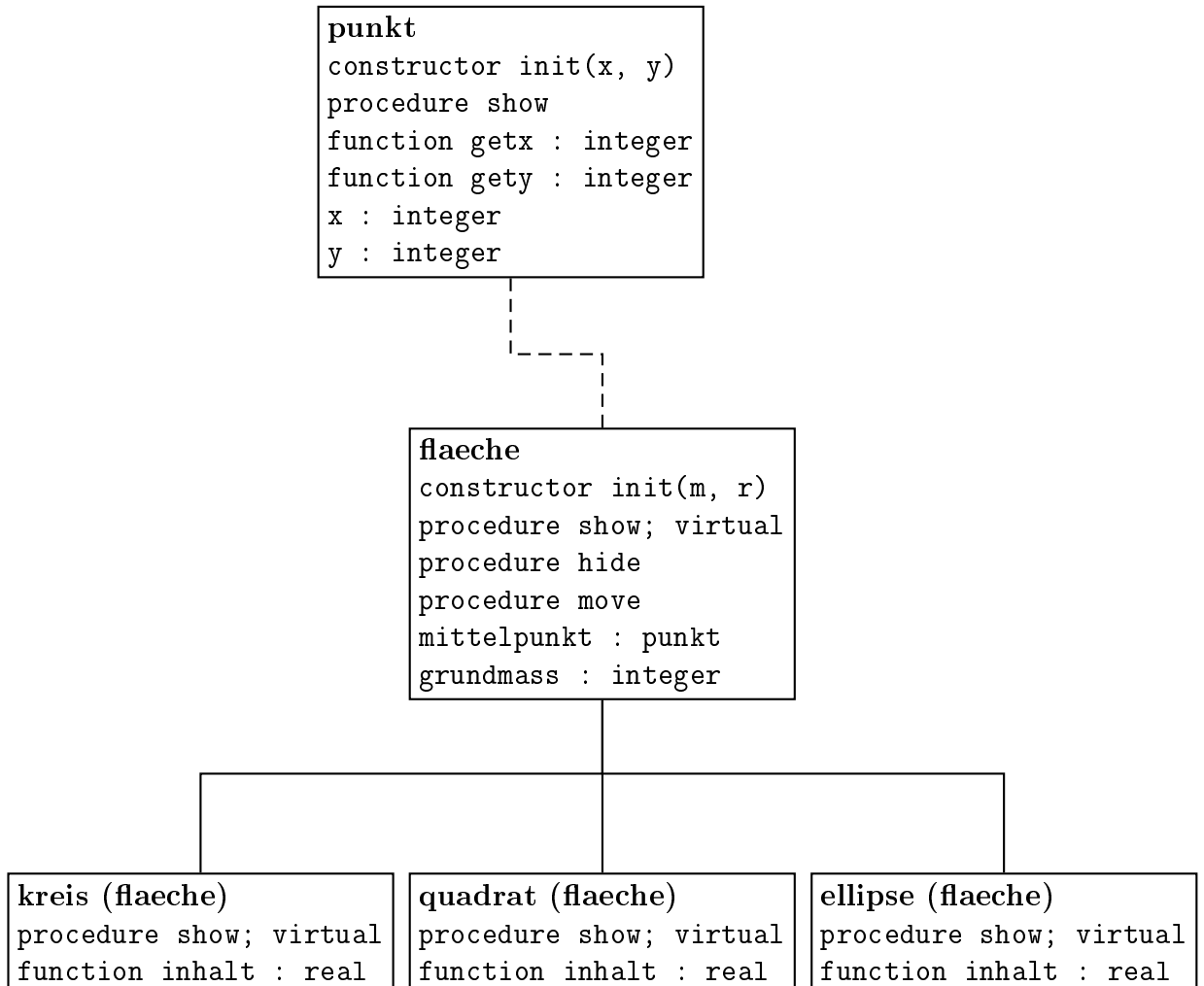
2. Phase: Realisierung

- Ziel/Inhalt
 - Nur noch `inhalt` und `show` müssen geschrieben werden!
 - Besprechung der Hausaufgabe
 - Vorteile der Vererbung erfahren
- Medien: Rechner, Programm, Tafel8
- Sozialform: Partnerarbeit

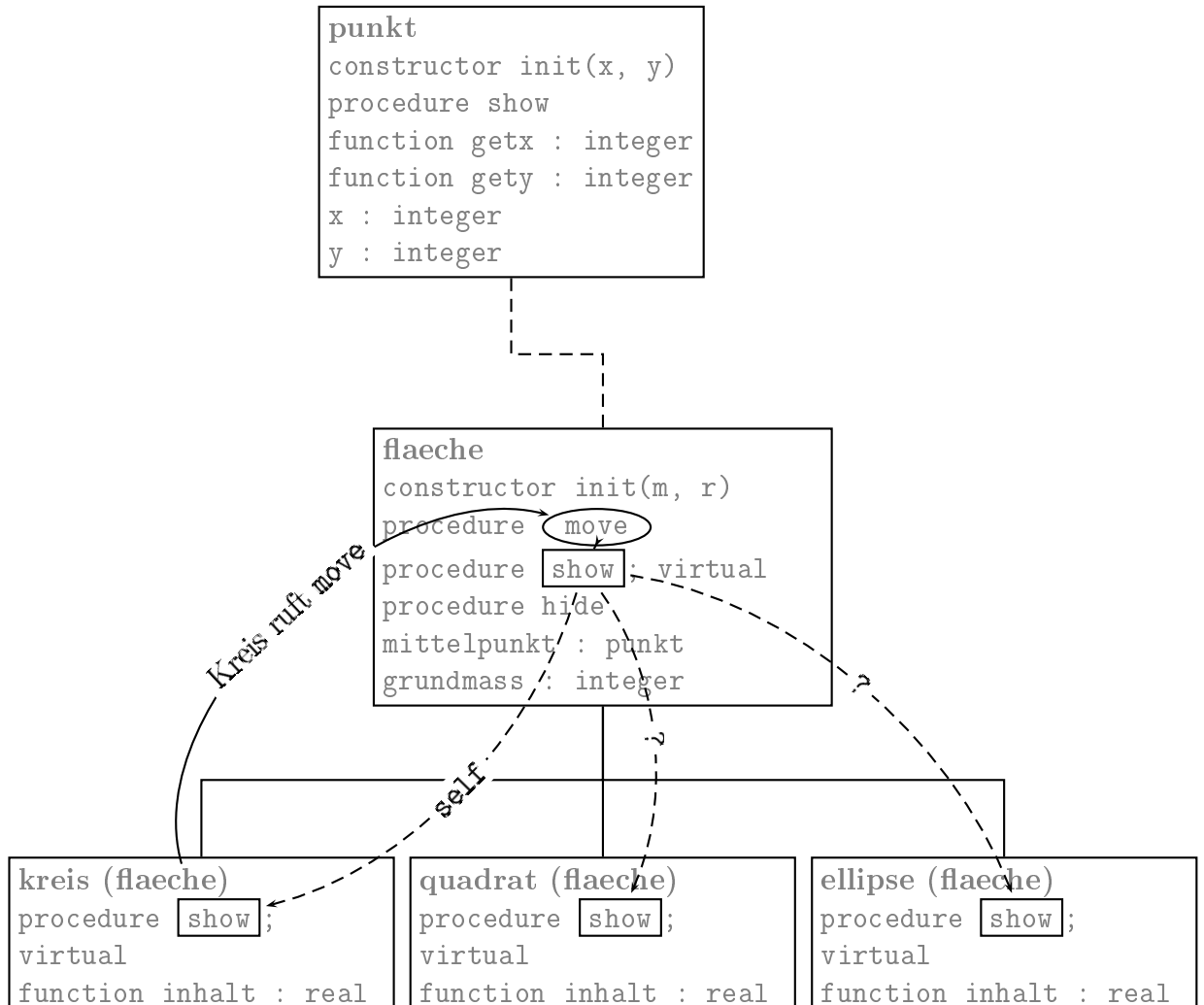
3. Phase: Abschluß

- Ziel/Inhalt:
 - Zusammenfassung der wesentlichen Konzepte sowie ihrer Vor- und Nachteile
 - Abschluß des Themas
- Medien: Tafel9
- Sozialform: Unterrichtsgespräch, Diskussion

A Objekthierarchie



B Objekthierarchie: Aufruf von move



C Quelltexte

```
01: PROGRAM
02:   geometrie;
03:
04: (* Stellt eine Klasse klasse_punkt zur Verfuegung *)
05: (* Punkte koennen angelegt und gezeigt werden   *)
06: (* x-, und y-Koordinaten von Punkten koennen ab- *)
07: (* gefragt werden                                 *)
08:
09:
10: USES
11:   crt, graph;
12:
13: TYPE
14:   klasse_punkt = OBJECT
15:     constructor init ( x_, y_ : integer );
16:     procedure show;
17:     function getx : integer;
18:     function gety : integer;
19:   PRIVATE
20:     x, y : integer;
21:   end;
22:
23: CONSTRUCTOR klasse_punkt.init ( x_, y_ : integer );
24: (* Initialisiert einen Punkt *)
25: begin
26:   x := x_;
27:   y := y_;
28: end;
29:
30: PROCEDURE klasse_punkt.show;
31: (* Zeigt den Punkt an *)
32: begin
33:   putpixel( x, y, white );
34: end;
35:
36: FUNCTION klasse_punkt.getx : integer;
37: (* liefert die x-Koordinate des Punktes zur"uck *)
38: begin
39:   getx := x;
40: end;
41:
```

```
42: FUNCTION klasse_punkt.gety : integer;
43: (* liefert die y-Koordinate des Punktes zur"uck *)
44: begin
45:   gety := y;
46: end;
47:
48: VAR
49:   einpunkt : klasse_punkt;
50:   andererpunkt : klasse_punkt;
51:   grDriver : Integer;
52:   grMode   : Integer;
53:
54: begin
55:   grDriver := Detect;
56:   InitGraph( grDriver, grMode, 'c:\tp\tp6\bgi' );
57:   (* Pfad ggf. anpassen *)
58:
59:   einpunkt.init( 100, 100 );
60:   (* einpunkt wird mit Werten versorgt *)
61:   andererpunkt.init( 173, 276 );
62:   (* andererpunkt wird mit Werten versorgt *)
63:   einpunkt.show;
64:   (* und schliesslich angezeigt *)
65:   andererpunkt.show;
66:
67:   repeat until keypressed;
68:   closegraph;
69: end.
70:
```

D Arbeitsblätter

AB1: Aufgabenstellung

Aufgabe: Es soll ein Grafiksystem entworfen werden:

Auf dem Monitor sollen Punkte, Kreise, Quadrate und Ellipsen gezeichnet, verschoben und wieder gelöscht werden können. Außerdem soll der Flächeninhalt solcher Figuren ausgegeben werden.

AB2: Arbeitsauftrag

Untersuche das gegebene Programm. Berücksichtige dabei die folgenden Punkte und notiere stichwortartig die Ergebnisse!

- Ändere die Koordinaten des Punktes `einpunkt`. Beobachtung ?
- Führe einen neuen Punkt `noch_ein_punkt` mit den Koordinaten (200|320) ein und laß ihn anzeigen!
- Welche neuen Schlüsselworte enthält das Programm ?
- Welche Bedeutung haben sie ? Stelle begründete Vermutungen an!
- Nenne Gemeinsamkeiten und Unterschiede der neuen Struktur innerhalb des `TYPE`-Blockes mit der Definition eines `record`!

AB3: Gruppenarbeit

Gruppe A Modelliere einen „Kreis“. Stelle dazu die notwendigen Daten zusammen, die einen Kreis eindeutig beschreiben. Gib kurz die notwendigen Methoden (Zeichnen, Flächeninhalt, ...) an und skizziere die Funktionsweise.

Stelle anschließend die Ergebnisse vor.

Gruppe B Modelliere ein „Quadrat“. Stelle dazu die notwendigen Daten zusammen, die ein Quadrat eindeutig beschreiben. Gib kurz die notwendigen Methoden (Zeichnen, Flächeninhalt, ...) an und skizziere die Funktionsweise.

(Hinweis: Versuche mit minimalen Daten auszukommen! Es ist nicht notwendig, die vier Eckpunkte anzugeben.)

Stelle anschließend die Ergebnisse vor.

Tafel0: Objekte

Ein Objekt ist die Zusammenfassung von Daten und Operationen zu einer Einheit. Die Daten nennt man auch Attribute, die Operationen auch Methoden.

Tafel1: Definition von Klassen

Eine Klasse von Objekten wird in Turbo-Pascal so definiert:

```
TYPE
  name = OBJECT
    methoden ...
  PRIVATE
    Attribute ...
  end;
```

Dabei werden für `methoden` die Funktionen und Prozeduren mit ihrer *Signatur* angegeben, so wie sie sonst in der Kopfzeile einer Prozedur oder Funktion stehen.

Jede Klasse braucht i. A. eine Prozedur, die die Attribute erstmalig mit Werten versorgt:

```
constructor init statt procedure ...
```

Ein Objekt deklarieren: `VAR object : klasse;`

Methoden ausführen: `object.methodenname(parameter);`

Tafel2: Kapselung

Die Attribute eines Objektes sind nach außen gekapselt, d. h. nicht zugänglich, sie dürfen nur innerhalb der zu diesem Objekt gehörenden Methoden verwendet werden. Dies wird durch das Schlüsselwort `PRIVATE` erreicht.

Tafel3: Kreis — Quadrat

	Kreis	Quadrat
Gemeinsamkeiten	Radius Mittelpunkt	Grundseite Aufhängepunkt
Unterschiede	$F = \pi * r^2$ circle...	$F = g^2$ line...

Tafel4: Vererbung

Klassen können ihre Attribute und Methoden Vererben. Ein Erbe heißt dann Subklasse, die beerbt die Superklasse. Die Subklasse darf die ererbten Attribute und Methoden ändern oder überschreiben.

Vererbung wird in Turbo-Pascal so angegeben:

```
TYPE
  kreis_klasse = OBJECT (flaeche_klasse)
```

...

Jetzt hat jedes Objekt der Klasse `kreis_klasse` die Attribute und Methoden von `flaeche_klasse` übernommen. Diese Vererbungsrichtung wird mit Spezialisierung bezeichnet, denn ein Kreis ist eine spezielle Fläche.

Tafel5: Polymorphie

Verschiedene Methoden verschiedener Klassen dürfen den gleichen Bezeichner tragen. Sinnvoll ist dies, wenn sie eine ähnliche Funktionalität besitzen, also von ihnen ein ähnliches Verhalten erwartet wird.

Tafel6: Virtuelle Methoden

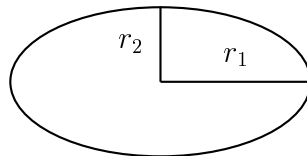
Damit die Methode `move` in der Superklasse `flaeche` weiß, von welcher der abgeleiteten Klassen sie gerufen wurde, und welches `show` sie dementsprechend verwenden muß, wird `show` als virtuelle Methode deklariert. Dies geschieht in Turbo-Pascal mit dem Schlüsselwort `virtual`; hinter dem Methodenkopf, z. B.:

```
procedure kreis.show; virtual;
```

Dann kann in den Superklassenmethoden `flaeche.move` und `flaeche.hide` mit `self.show` die jeweils richtige Subklassenmethode gerufen werden.

Tafel7: Ellipsenformeln

Ellipsen:



zwei *Halbmesser* r_1 und r_2

Flächeninhalt: $F_{\text{Ellipse}} = \pi \cdot r_1 \cdot r_2$

Tafel8: Klasse ellipse

Wie kann eine Ellipse modelliert werden?

→ sei (*der Einfachheit halber*) $r_2 := 2 \cdot r_1$

also: ellipse hat einen Mittelpunkt `mitte` und einen Halbmesser `r`.

⇒ ellipse kann von `flaeche` abgeleitet werden! Was muß ergänzt werden?

→ `show`

→ `inhalt`

→ alles andere funktioniert schon!

Tafel9: Diskussion:**Vor- und Nachteile der Objektorientierung**

<u>Vorteile</u>	<u>Nachteile</u>

Grafik mit Turbo-Pascal

- **Initialisieren des Grafik-Systems**

Dem System muß durch Angabe von `graph` im `USES`-Block die Unit `graph` bekannt gemacht werden, im Variablenblock werden Variablen deklariert:

```
grDriver : integer; und grMode : integer;
```

Vor Verwendung der ersten Grafikbefehle dann

```
grDriver := Detect;
```

```
InitGraph(grDriver, grMode, 'c:\tp\tp6\bgi');
```

wobei `'c:\tp\tp6\bgi'` durch den Pfad zu den `.bgi`-Dateien ersetzt wird.

Jetzt steht das Grafik-System zur Verfügung, bis es mit `closegraph`; wieder geschlossen wird.

- **Punkte**

Punkte werden mit

```
putpixel(x, y, color);
```

gesetzt, wobei `x` und `y` die Koordinaten angeben, und `color` die zu verwendende Farbe festlegt. Für `color` dürfen Zahlen von 0 bis 15 oder aber Farbkonstanten wie z. B. `black`, `white`, `red`, `yellow` usw. eingesetzt werden.

- **Linien**

Eine Linie wird mit

```
line(xa, ya, xe, ye);
```

gezogen. Dabei stehen `xa` und `ya` für die Start-Koordinaten, `xe` und `ye` für den Endpunkt der Linie. Die Farbe kann (muß nicht) vorher mittels `setcolor(color)`; eingestellt werden, sie bleibt dann solange aktiv, bis eine neue Farbe ausgewählt wird.

- **Kreise**

Der Befehl `circle(x, y, r)`; zieht einen Kreis um den Mittelpunkt `(x | y)` mit dem Radius `r` und der momentan aktiven Farbe.

- **Ellipsen**

Eine Ellipse hat zwei Halbmesser r_1 und r_2 .

Mit `ellipse(x, y, 0, 360, r1, r2)`; wird sie um `(x | y)` gezeichnet.

- **Text**

`OutTextXY(x, y, text)`; schreibt den Text `text` (vom Typ `string`) an die Position `(x | y)`.

Literatur

- [Baumann] Baumann, R.: Informatik für die Sekundarstufe II, Bd. 2: Höhere Datentypen, Automaten, Sprachen.
Stuttgart, Klett 1993.
- [Baumann 90] Baumann, R.: Didaktik der Informatik.
Stuttgart, Klett 1990 und Klett 1996²
- [Baumann 97] Baumann, R.: Objektorientierte Programmierung.
aus: Java - Stimulans für den Informatik-Unterricht. in: LOG IN 17 (1997), H5, S. 23f
- [Crutzen & Hein 95] Crutzen; Hein: Objektorientiertes Denken als didaktische Basis der Informatik.
in: Schubert, S.: Innovative Konzepte für die Ausbildung. Berlin, Springer 1995, S. 149-158
- [Eirund 93] Eirund, H.: Objektorientierte Programmierung.
in: LOG IN 13 (1993), H4, S. 40-48
- [Freytag & Brauer 97] Freytag, J.; Brauer, W.: Objektorientierung in der Ausbildung.
in: Informatik-Spektrum 20:326-327 (1997)
- [Gall 95] Gall, H.: Objektorientierte Konzepte in der Ausbildung.
in: Informatik-Spektrum 18:195-202 (1995)
- [Glaeser] Glaeser, G.: Von Pascal zu C/C++ — Gemeinsamkeiten und Unterschiede der beiden Sprachen.
München, Markt&Technik-Verlag 1993
- [Goos 96] Goos, G.: Vorlesungen über Informatik, Bd. 2: Objektorientiertes Programmieren und Algorithmen.
Berlin, Springer 1996
- [Harbeck & Thode] Harbeck, G.; Thode, R. u. a.: Metzler Informatik
Stuttgart, Metzlersche Verlagsbuchhandlung 1990².
- [Hermes & Stein 96] Hermes & Stein: Objektorientierte Programmierung.
Ein Zugang in Oberon mit Anwendung von Listen und Grafik.
Stuttgart, Klett 1996
- [Humbert] Humbert, L.: Objektorientierung in der Schule.
<http://schulen.hagen.de/IF/00/welcome.html>

- [Lehrplan 93] Ministerium für Bildung und Kultur (Hrsg.): Lehrplanentwurf Informatik Rheinland-Pfalz.
Mainz, Ministerium für Bildung und Kultur, 1993.
- [Müller 92] Müller, K.: Neue Programmierkonzepte für die Schule?
in: Bosler, U.: Schulcomputer-Jahrbuch 93/94. Stuttgart, Metzler & Teubler 1992, S. 157-189,220
- [Quibeldey-Cirkel] Quibeldey-Cirkel, K.: Paradigmenwechsel im Software-Engineering: Auf dem Weg zu objektorientierten Weltmodellen.
<http://www.ti.et-inf.uni-siegen.de/staff/Quibeldey/Publikationen/SW-Technik-Trends.ZIP>
- [Quibeldey-Cirkel 94] Quibeldey-Cirkel, K.: Das Objekt-Paradigma in der Informatik.
Stuttgart, Teubner 1994
- [Schwill] Schwill, A.: Objektorientierte Programmierung. Eine Rechtfertigung aus kognitivpsychologischer Sicht.
in: LOG IN 13 (1993), H4, S. 44
- [Schwill 93] Schwill, A.: Programmierstile. Ein Überblick.
in: LOG IN 13 (1993).
- [Schwill 95] Schwill, A.: Programmierstile im Anfangsunterricht.
in: Schubert, S.: Innovative Konzepte für die Ausbildung. Berlin, Springer 1995, S. 178-187
- [Spolwig 95] Spolwig, S.: Objektbasierte Programmierung im Anfangsunterricht.
in: LOG IN 15 (1995), H3, S. 43-49
- [Tischer] Tischer, M.: Turbo-Pascal 6.0
Düsseldorf, Sybex 1990.
- [Wolff von Gutenberg 93] Gutenberg, J.W. von: Objektorientiert programmieren von Anfang an.
Mannheim, BI-Wissenschaftsverlag 1993.