Einstieg in die Objektorientierte Programmierung mit BlueJ und JGameGrid

DUE im Fach Informatik

von

Philipp Kupferschmied

am

Staatlichen Seminar für Didaktik und Lehrerbildung (Gymnasien) Heidelberg

Fachleiter: Theo Heußer

12. Januar 2012

Ich versichere, dass ich die Dokumentation selbstständig und nur mit den angegebenen Hilfsmitteln angefertigt habe. Zu den Stellen und Materialien, die dem Wortlaut oder dem Sinn nach anderen Werken, auch elektronischen Medien, entnommen wurden, habe ich die Quellen angegeben. Materialien aus dem Internet sind durch Ausdruck belegt.

(Philipp Kupferschmied)

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Ziel der Unterrichtseinheit 1
1.2 Objektorientierung im Informatikunterricht 2
1.3 Hinweise zum Kurs 3
1.4 Zeitliche Einordnung 4
1.5 Die verwendeten Werkzeuge 4
1.6 Struktur der Unterrichtseinheit 6
1.7 Überlegungen zum durchgeführten Projekt 7
1.8 Struktur des Projekts 8

2 Durchführung **10**
2.1 Erste Doppelstunde 10
2.2 Zweite Doppelstunde 14
2.3 Dritte Doppelstunde 18
2.4 Vierte Doppelstunde 21

3 Auswertung **24**
3.1 Analyse 24
3.2 Fazit und Ausblick 27

Literaturverzeichnis **29**

4 Anhang **30**
4.1 Foliensätze 30
4.2 Material aus der Gruppenarbeit "Klassenentwurf" 37
4.3 Codepuzzle zur Kollisionsabfrage 46
4.4 Klausur 48
4.5 Evaluationsbogen 50
4.6 Screenshots 52

Kapitel 1

Einleitung

Ziel der in dieser Arbeit dokumentierten und analysierten Unterrichtseinheit war die Vermittlung von Grundlagen der Objektorientierung bzw. objektorientierten Programmierung in Java in einem Informatikkurs in der Jahrgangsstufe 1 am Beispiel eines einfachen Computerspiels. Computerspiele oder spielähnliche Konzepte sind zum Einstieg in die objektorientierte Programmierung gut geeignet: Zum Einen sind Spiele nahe an der Lebenswelt der Schüler und werden somit wohl von den meisten als ansprechender und motivierender empfunden als beispielsweise textbasierte Konsolenanwendungen. Zum Anderen kommen selbst in einfachen Spielen in der Regel mehrere verschiedene Spielfiguren und andere Spielobjekte vor, die es erlauben, den Objektbegriff (im Sinne der objektorientierten Programmierung) zu motivieren und zu definieren.

Zur Umsetzung des Spiels wurde eine relativ junge Klassenbibliothek zur Spieleentwicklung mit Java, JGameGrid [6], verwendet. JGameGrid wurde speziell für den edukativen Einsatz konzipiert und stellt grundlegende Funktionalität bereit, um mit vergleichsweise geringem Programmieraufwand Spiele in Java entwickeln zu können. Im Gegensatz zu zahlreichen anderen primär für edukative Zwecke konzipierten Werkzeugen bringt JGameGrid keine eigene Entwicklungsumgebung mit. Ich entschied mich zum Einsatz von BlueJ [4], ein ebenfalls primär auf Ausbildungszwecke zugeschnittenes Werkzeug. Eine genauere Beschreibung sowohl von JGameGrid als als auch von BlueJ findet sich im Abschnitt 1.5.

1.1 Ziel der Unterrichtseinheit

Selbstverständlich kann eine auf 8 Stunden ausgelegte Unterrichtseinheit nicht dazu dienen, den Schülern einen tiefgehenden Einblick in alle Konzepte der objektorientierten Softwareentwicklung zu geben und gleichzeitig diese Konzepte ausreichend zu üben.

Die hier dokumentierte Unterrichtseinheit ist daher als Einstieg in die objektorientierte Programmierung konzipiert worden und hat das Ziel, wesentliche Grundkonzepte zu vermitteln und den Schülern ausreichend Zeit zum Üben bzw. Anwenden dieser Konzepte zu gewähren. Als wesentliche Konzepte sind hierbei zunächst natürlich die Begriffe "Objekt" und "Klasse" zu nennen. Den Schülern soll im Laufe der Unterrichtseinheit klar werden, was ein Objekt ist und in welchem Bezug Objekt und Klasse zueinander stehen. Die

Einführung des Objektbegriffs führt zwangsläufig zu den Eigenschaften und Fähigkeiten eines Objekts und damit zu den Begriffen "Attribut" und "Methode". Ein ebenfalls zentraler Begriff der Objektorientierung ist der Begriff "Kapselung", auch dieser soll den Schülern möglichst früh nahe gebracht werden. In den letzten Stunden der Unterrichtseinheit soll schließlich die Vererbung thematisiert werden, die jedoch aus Zeitgründen nicht weiter vertieft werden kann.

Die Umsetzung der theoretischen Konzepte in ein Programm erfolgt in der Programmiersprache Java. Neben der Syntax zur Implementierung von Klassen und zum Erzeugen von Objekten spielt hier zusätzlich der Begriff des Konstruktors eine Rolle. Außerdem sollen die Schüler in der Lage sein, Methoden eines erzeugten Objekts aufzurufen.

Nicht Teil dieser Unterrichtseinheit sind somit also insbesondere weiterführende Begriffe wie Polymorphie und dynamische Bindung, auch Details zu Konstruktoraufrufen in Vererbungsbeziehungen werden nicht genauer thematisiert.

Neben den genannten inhaltlichen Zielen soll diese Arbeit außerdem eine Einschätzung zur Tauglichkeit von JGameGrid im Unterricht liefern: Inwieweit ist es zum Einstieg in das Thema "Objektorientierung" geeignet, wie wurde die Arbeit mit dieser Bibliothek von den Schülern empfunden, welcher Vor- oder Nachteile gegenüber ähnlichen Werkzeugen ließen sich möglicherweise anhand dieser Unterrichtseinheit erkennen?

1.2 Objektorientierung im Informatikunterricht

Praktische Relevanz

Objektorientierung ist aktuell das wohl am weitesten verbreitete Entwurfs- und Programmierparadigma in der Softwareentwicklung. Von den zehn Programmiersprachen, die den TIOBE-Index im Dezember 2011 anführen, bieten beispielsweise neun Unterstützung für die objektorientierte Programmierung [7]. Obwohl die Informatikausbildung an Schulen sicher nicht in erster Linie als Berufsvorbereitung verstanden werden darf, erscheint es unter diesen Gesichtspunkten sinnvoll und wichtig, das Konzept der objektorientierten Softwareentwicklung im Informatik-Kurs in der Oberstufe zu thematisieren.

Bezug zum Bildungsplan

Der Bildungsplan für das Fach Informatik in der gymnasialen Oberstufen nennt fünf Leitideen, anhand derer sich die einzelnen Inhalte des Informatikunterrichts gliedern lassen. Die objektorientierte Programmierung taucht explizit bei der Beschreibung der Leitidee "Problemlösen und Modellieren" auf. Als zu erwerbende Kompetenzen werden hier unter anderem die Fähigkeiten, reale Probleme auf Objekte und Klassen abzubilden, die Beziehungen zwischen Klassen/Objekten zu analysieren und ein Modell in einer Programmiersprache zu realisieren genannt. Der Objektorientierung zugehörige Begriffe wie "Objekt", "Klasse", "Kapselung" und "Vererbung" tauchen ebenfalls im Rahmen dieser Leitidee auf. Als wesentliche überfachliche (und nicht speziell einer Leitidee zuzuordnende) Kompetenz nennt der Bildungsplan die Schulung des Abstraktionsvermögens durch die "Abbildung von Aufgaben der Umwelt in eine vom Rechner bearbeitbare Form" [5]. Die Behandlung

des Themas Objektorientierung bzw. der objektorientierten Programmierung ist also nicht nur durch ihre Bedeutung in Forschung und Wirtschaft begründet, sondern fußt auch auf den Vorgaben des Bildungsplans.

Auch wenn die Inhalte dieser Unterrichtseinheit somit im Wesentlichen der Leitidee "Problemlösen und Modellieren" zuzuordnen sind, werden auch Inhalte der Leitidee "Algorithmen und Daten" berührt: Kein komplexes Programm, auch nicht das in dieser Unterrichtseinheit entwickelte Spielprojekt, kommt ohne Variablen und Kontrollstrukturen wie Schleifen oder Bedingungen aus.

1.3 Hinweise zum Kurs

Die hier dokumentierte Unterrichtseinheit wurde in einem Informatikkurs in der Jahrgangsstufe 1 unterrichtet. Der Kurs hatte insgesamt 21 Teilnehmer, davon 20 männlich (zur Verbesserung der Lesbarkeit verwende ich in diesem Text das generische Maskulinum, das Wort "Schüler" steht somit stellvertretend für "Schülerinnen und Schülern"). Eine kurze Umfrage unter den Schülern zu Beginn des Schuljahres ergab, dass die meisten bisher nur sehr wenige oder keine Programmierkenntnisse hatten. Insbesondere war kein Teilnehmer bisher mit der Sprache Java vertraut.

Im Computerraum, in dem der Kurs stattfand, befanden sich insgesamt 16 Computer. Dadurch war es notwendig, dass Schüler teilweise zu zweit an einem Computer arbeiteten, was jedoch kein größeres Problem darstellte. Problematischer waren hin und wieder auftretende technische Schwierigkeiten, die manchmal dazu führten, dass einzelne Schüler nur mit zeitlicher Verzögerung die Programmieraufgaben bearbeiten konnten.

Ein weiteres Problem, das bei der Planung der Stunden, insbesondere bei der Bereitstellung von Unterrichtsmaterial, zu berücksichtigen war, war das Fehlen eines Schulbuchs. Dadurch wurde das Vermitteln neuer Inhalte und die Ergebnissicherung erschwert: Die Schüler haben kein Nachschlagewerk zum Erarbeiten oder zum Wiederholen und Vertiefen von Unterrichtsinhalten. Ich konzipierte daher für jede Doppelstunde einen Satz von Folien, die über den Beamer präsentiert wurden und damit zum Einen als eine Art "Tafel-Ersatz" dienten, auf dem Arbeitsaufträge, Beispielcode und Definitionen Platz fanden. Zum Anderen habe ich mich bemüht, die Inhalte auf den Folien ausführlich genug zu formulieren, damit sie von den Schülern als Nachschlagewerk benutzt werden können. Im Laufe jeder Doppelstunde stellte ich daher den Schülern den aktuellen Foliensatz zur Verfügung.

Darüber hinaus erstellte ich im Vorfeld der Unterrichtseinheit ein kurzes Skript, das noch einmal die wesentlichen Grundlagen und Konzepte der objektorientierten Programmierung zusammenfasst. Dieses Skript ist als Zusatzangebot für die Schüler zu verstehen und wurde ihnen erst nach Abschluss der Unterrichtseinheit zur Vorbereitung auf die Klausur zur Verfügung gestellt. Die Folien für jede Doppelstunde finden sich im Anhang und auf der beiliegenden CD, das Skript ist ebenfalls auf der CD enthalten.

1.4 Zeitliche Einordnung

Die Unterrichtseinheit wurde im Zeitraum vom 24.10.11 bis zum 21.11.11, unterbrochen durch die einwöchigen Herbstferien, unterrichtet.

Da die meisten Schüler im Kurs bisher keine oder nur sehr wenig Erfahrungen mit der Programmierung hatten, entschied ich mich, vor dem Einstieg in die objektorientierte Programmierung zunächst einige elementare Grundlagen der imperativen Programmierung zu unterrichten, nämlich:

1. Das Erstellen einfacher, sequentieller Programme
2. Variablen und die Datentypen `int` und `double` (ohne auf die interne Repräsentation dieser Datentypen einzugehen)
3. Schleifen (`for` und `while`)
4. Verzweigungen (`if-else`)
5. Methoden (nebst Parametern und Rückgabewerten)

Zur Einführung in die imperative Programmierung wurde nicht JGameGrid, sondern der Hamstersimulator [1] verwendet. Ein großer Vorteil des Hamstersimulators liegt darin, dass er sämtliche objektorientierten Aspekte vor dem Benutzer versteckt, es damit also möglich ist, rein imperative Programme zu schreiben.

1.5 Die verwendeten Werkzeuge

In diesem Abschnitt werde ich kurz die verwendeten Werkzeuge, also BlueJ und JGameGrid, vorstellen. Selbstverständlich kann und soll diese Ausarbeitung nicht als technische Dokumentation dienen. Neben einem allgemeinen Überblick wird nur insoweit auf technische Details eingegangen, als diese für die Durchführung der Unterrichtseinheit von Relevanz sind.

BlueJ

BlueJ ist eine speziell für Ausbildungszwecke konzipierte Entwicklungsumgebung für Java. BlueJ legt besonderen Wert auf die Visualisierung objektorientierter Konzepte: Das Hauptfenster zeigt ein vereinfachtes Klassendiagramm aller Klassen des aktuellen Projekts, erst ein Doppelklick auf eine Klasse öffnet einen Editor mit dem Quellcode. Darüber hinaus lassen sich mittels weniger Mausklicks Instanzen der vorhandenen Klassen, also Objekte, erzeugen. Auch die so erzeugten Objekte werden von BlueJ in Form von abgerundeten Rechtecken (angelehnt an ein UML-Objektdiagramm) visualisiert. Ein Rechtsklick auf ein solches Rechteck erlaubt es, die Attribute des jeweiligen Objekts zu inspizieren und Methoden des Objekts aufzurufen, um seinen Zustand zu ändern. Auf diese Weise ist es möglich, mit vergleichsweise wenig Programmieraufwand Grundbegriffe der objektorientierten Programmierung zu veranschaulichen. Insbesondere entfällt die Notwendigkeit,

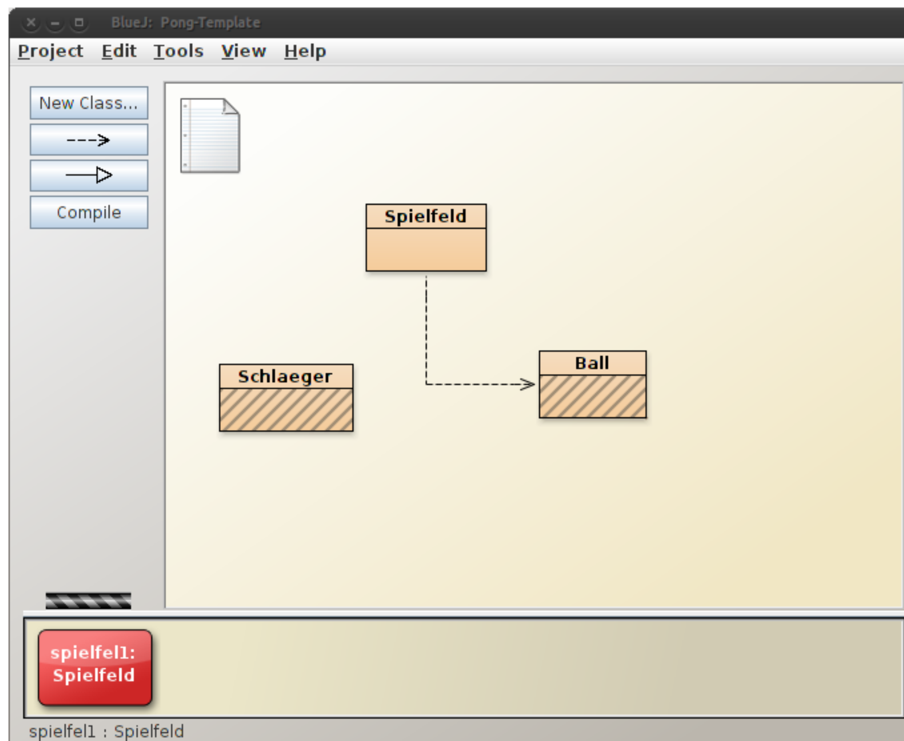


Abbildung 1.1: Die Entwicklungsumgebung BlueJ. Das Hauptfenster zeigt drei Klassen; das rote Rechteck unten stellt ein Objekt der Klasse Spielfeld dar.

ein "Rahmenprogramm" zu schreiben, das Objekte erzeugt und sich um die Ein- und Ausgabe sowie den Aufruf von Methoden kümmert. Gegenüber anderen Editoren bzw. Entwicklungsumgebungen wie beispielsweise dem im schulischen Umfeld ebenfalls beliebten Java-Editor weist BlueJ außerdem eine sehr schlanke und übersichtliche Benutzeroberfläche auf, die Anfänger nicht mit einer immensen Funktionsvielfalt verwirrt. Positiv erwähnt sei außerdem die farbige Hervorhebung von Codeblöcken im Code-Editor, die dabei hilft, die Struktur eines Programms schneller zu durchschauen.

JGameGrid

Bei JGameGrid handelt es sich um eine von Prof. Aegidius Plüss entwickelte Java-Klassenbibliothek, die die Entwicklung einfacher 2D-Spiele erlaubt. Laut Prof. Plüss eignen sich Spiele zum Lernen besonders gut, weil sie einerseits für die Schüler motivierend sind, ihre Entwicklung andererseits aber die Anwendung informatischen Wissens erfordert [6]. Insbesondere verdeutlichen Spiele, bei denen eine Vielzahl unterschiedlicher Spielobjekte auftreten und miteinander (oder dem Spieler interagieren), die Konzepte und Vorteile der objektorientierten Programmierung.

JGameGrid stellt eine Vielzahl von Klassen bereit, um die Entwicklung eigener Spiele zu vereinfachen. Von zentraler Bedeutung ist die Klasse `GameGrid`, die für das Erzeugen und Verwalten des Spielfelds zuständig ist, sowie die Klasse `Actor`, die Funktionen zum Verwalten der Spielfiguren (Setzen der Sprite-Grafik, Verwalten der Position und Orien-

tierung) bereitstellt.

JGameGrid im Vergleich zu anderen Werkzeugen

Neben JGameGrid existieren eine ganze Reihe weiterer Werkzeuge, die speziell auf Ausbildungszwecke zugeschnitten sind und ebenfalls versuchen, Grundlagen der objektorientierten Programmierung auf spielerische Art bzw. anhand von spielähnlichen Situationen zu vermitteln.

So bietet der bereits erwähnte Hamstersimulator auch die Möglichkeit, objektorientierte Programme zu schreiben: Jeder Hamster muss als Objekt der Klasse "`Hamster`" erzeugt werden, die Vererbung erlaubt es, spezielle Hamster mit angepassten oder zusätzlichen Eigenschaften und Fähigkeiten zu implementieren. Durch die relativ enge Fixierung auf Objekte vom Typ "`Hamster`" wird allerdings nicht unmittelbar deutlich, dass die objektorientierte Programmierung ein vielseitig einsetzbares Konzept und der Objektbegriff sehr weit und vage gefasst ist. Hinzu kommt, dass der Hamstersimulator seine eigene Entwicklungs- und Simulationsumgebung mitbringt, wobei der mitgelieferte Code-Editor weit weniger komfortabel ist als beispielsweise BlueJ.

Eine weiteres mit JGameGrid vergleichbares Werkzeug ist Greenfoot [3]. Der Autor von JGameGrid, Prof. Aegidius Plüss, gibt an, dass Greenfoot die Entwicklung von JGameGrid stark beeinflusst hat. Ähnlich wie der Hamstersimulator bietet auch Greenfoot eine integrierte Entwicklungs- und Simulationsumgebung, die sich stark an BlueJ orientiert (dies ist nicht verwunderlich, da BlueJ in der gleichen Forschungsgruppe entwickelt wurde).

Gegenüber anderen Werkzeugen sehe ich die Vorteile von JGameGrid hauptsächlich in der hohen Flexibilität. Diese beginnt bei der freien Wahl der Entwicklungsumgebung, schließt aber auch eine relativ hohe Freiheit bei der Implementierung mit ein - mehr dazu in Abschnitt 1.8. Außerdem macht JGameGrid wenig Vorgaben zur Beschaffenheit der Spiele, die sich damit umsetzen lassen, das Spektrum reicht von Spielen mit gitterbasiertem Spielfeld über Spiele mit pixelbasiertem Spielfeld bis hin zur Unterstützung für Kartenspiele. Ein zusätzlicher Bonus für die zukünftige Verwendung ist die von Prof. Plüss angestrebte Portierung auf das Smartphone-Betriebssystem Android: Wenn sich damit im Unterricht erstellte Spiele mit vergleichsweise wenig Aufwand auch auf dem eigenen Smartphone spielen lassen, so stellt das sicher für viele Schüler einen zusätzlichen Motivationsfaktor dar.

1.6 Struktur der Unterrichtseinheit

Ich entschied mich dazu, den Schülern das Konzept der Objektorientierung mittels einer objektorientierten Analyse zu motivieren und zu erläutern. Diese Analyse soll an einem von mir vorbereiteten Spiel durchgeführt werden, bei dem die Schüler zunächst nur durch die Beobachtung beim Spielen die vorhandenen Spielobjekte sowie deren Eigenschaften und Fähigkeiten analysieren. Erst im Anschluss daran werden die Grundbegriffe der objektorientierten Programmierung genauer definiert und die Umsetzung in Java angesprochen. Im Laufe der Unterrichtseinheit wird dann das Spiel, das die Schüler zum Einstieg zum Analysieren erhielten, von den Schülern nachimplementiert.

Der Einstieg über eine objektorientierte Analyse erscheint mir insbesondere deswegen geeignet, weil die Schüler den Objektbegriff dadurch an einem konkreten Beispiel kennen lernen und sehen, dass das Zusammenfassen bestimmter Eigenschaften und Fähigkeiten zu einem Objekt kein willkürliches theoretisches Konzept ist, sondern sich aus der Beschaffenheit des vorgegebenen Spiels nahezu "von selbst" ergibt. So lassen sich die wesentlichen theoretischen Grundlagen der Objektorientierung anschaulich vermitteln, bevor die Schüler mit der praktischen Umsetzung in Java konfrontiert werden. Die von Hartmann et al. in [2] angesprochene Gefahr, dass ein theoretischer Zugang für die Schüler wenig motivierend und oft nicht einsichtig ist, wird durch das gewählte Vorgehen meiner Meinung nach abgeschwächt. Darüber hinaus dient das Spielprojekt als "roter Faden" durch die Unterrichtseinheit, die Schüler haben ein klar definiertes Ziel vor Augen und kennen die zu erreichenden Funktionalität bereits.

1.7 Überlegungen zum durchgeführten Projekt

JGameGrid macht nur sehr wenige Vorgaben, wie ein damit entwickeltes Spiel auszusehen hat, so reichen die auf der Homepage vorgestellten Projekte von Puzzlespielen über bekannte Klassiker wie Pacman und Breakout bis hin zu einem einfachen Jump&Run. Ein Projekt, das sich zum Einstieg in die objektorientierte Programmierung im Rahmen dieser Unterrichtseinheit eignen soll, muss jedoch gewisse Bedingungen erfüllen:

- Im Spiel sollten möglichst viele unterschiedliche und voneinander unterscheidbare Objekte vorkommen, da nur so eine sinnvolle objektorientierte Analyse möglich ist und der Sinn der objektorientierten Programmierung motiviert werden kann.
- Der programmiertechnische Aufwand muss vertretbar sein. Das bedeutet insbesondere, dass entweder keine komplexen Algorithmen und Datenstrukturen im Programm vorkommen, oder dass diese schon vorgegeben sind und von den Schülern zunächst weder genauer betrachtet noch im Detail verstanden werden müssen.

Zwei klassische Spiele schienen mir besonders geeignet zu sein:

Pac Man: Bei diesem Spiel steuert der Spieler eine Figur durch ein Labyrinth und muss dabei alle vorhandenen "Extras" einsammeln, ohne mit den Gegnern in Berührung zu kommen, die ebenfalls im Labyrinth unterwegs sind. Mit der Spielfigur selbst, den Gegnern und den "Extras" steht eine ausreichend hohe Zahl unterschiedlicher Spielobjekte zur Verfügung, deren Attribute und Methoden jedoch allesamt recht ähnlich sind. Außerdem lässt sich das Spielfeld nur schwer ohne Kenntnisse im Umgang mit zweidimensionalen Arrays umsetzen, der programmiertechnische Anspruch ist also vergleichsweise hoch.

Pong: Eine Art Tennis mit schlichter Grafik: Zwei Schläger (dargestellt durch Balken) und ein Ball. Ziel ist es, den Ball so zu spielen, dass ihn der Gegner nicht mehr erreicht. Die Zahl der Spielobjekte ist hier sehr klein, im Gegenzug ist der Implementierungsaufwand auch deutlich geringer als bei Pac Man, komplexe Datenstrukturen werden nicht benötigt.

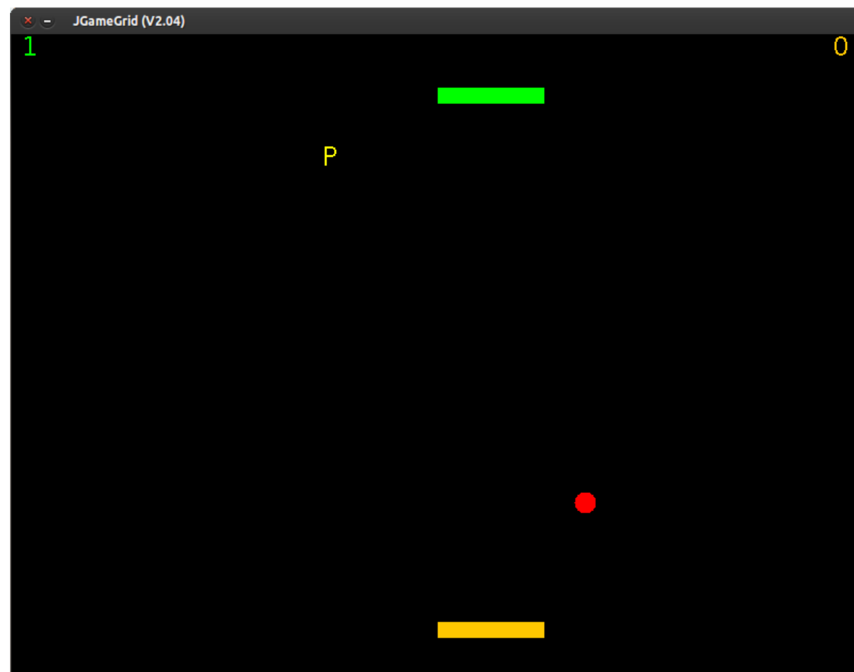


Abbildung 1.2: Screenshot des vorgegebenen Pong-Spiels

Nach reiflicher Überlegung schien mir Pong das geeignetere Projekt für die geplante Unterrichtseinheit zu sein. Um die Zahl der unterscheidbaren Arten von Objekten zu erhöhen, erweiterte ich das ursprüngliche Spielkonzept um ein "Powerup", das von jedem Spielerschläger eingesammelt werden kann und das zufällig die Breite oder die Bewegungsgeschwindigkeit des jeweiligen Schlägers verändert. Ein weiterer Vorteil dieser Modifikation ist, dass gewisse Eigenschaften eines Schläger-Objekts (seine Größe und Geschwindigkeit) dadurch, dass sie sich ändern können, vom Spieler bewusster wahrgenommen werden, was für die zunächst geplante objektorientierte Analyse hilfreich ist.

1.8 Struktur des Projekts

Wie eingangs erwähnt stellt JGameGrid die Klasse `Actor` bereit, die als Basisklasse für eigene Spielfiguren dient. `Actor` bietet dabei bereits die Funktionalität zum Setzen der Sprite-Grafik(en), ebenso Methoden zur Verwaltung der Position usw. Dieser Ansatz erlaubt das Erzeugen von Spielfiguren mit sehr geringem Implementierungsaufwand, ist jedoch im Bezug auf die geplante Durchführung der Unterrichtseinheit in zweifacher Hinsicht problematisch: So werden die Schüler zumindest auf syntaktischer Ebene sofort mit der Vererbung konfrontiert, muss doch in jeder Klassendefinition ein "`extends Actor`" auftauchen. Schwerer wiegt, dass die Klasse `Actor` sehr viel Funktionalität und Zustand "wegnimmt", der in davon abgeleiteten Unterklassen nun gar nicht mehr implementiert werden muss. Die Unterrichtseinheit soll jedoch so gestaltet sein, dass sich die Schüler zunächst Gedanken darüber machen müssen, welche Attribute und Methoden die Spielobjekte im fertigen Spiel haben könnten oder müssten, bevor sie selbst die entspre-

chenden Klassen implementieren. Würde nun von Anfang an `Actor` als Basisklasse verwendet, so würden die zuvor erarbeiteten Attribute in den selbst implementieren Klassen gar nicht auftauchen müssen - ein Sachverhalt, der in dieser frühen Phase nur schwer zu vermitteln ist. Aus diesem Grund entschied ich mich dafür, auf die Verwendung der Klasse `Actor` zu verzichten. Die "actor-lose" Umsetzung des Projekts erschien zunächst nicht ganz einfach, da die `Actor`-Klasse in JGameGrid von zentraler Bedeutung ist. Glücklicherweise ist es jedoch möglich, Grafikprimitive wie Kreise oder Rechtecke direkt in das von JGameGrid erzeugte Spielfenster zeichnen zu lassen, ohne die Klasse `Actor` verwenden zu müssen. Die Spielerschläger wurden daher als einfache Rechtecke, der Ball als ausgefüllter Kreis umgesetzt. Das Powerup wird durch den Buchstaben "P" dargestellt. Details zur Implementierung können dem Quellcode der beiliegenden CD entnommen werden, Abbildung 1.2 zeigt einen Screenshot des fertigen Pong-Spiels.

Das Pong-Projekt, das von den Schülern im Laufe der Unterrichtseinheit vervollständigt werden soll, besteht aus folgenden Klassen (siehe auch Abbildung 1.1):

`Spielfeld`: Diese Klasse dient der Verwaltung aller im Spiel vorkommenden Spielfiguren und implementiert einen Großteil der Spiellogik. Insbesondere werden hier die weiteren Spielobjekte, also Schläger und Ball, erzeugt. Auch Tastatur- und Kollisionsabfrage werden später in dieser Klasse implementiert.
Der Code zum Erzeugen des Spielfelds sowie Funktionalität, die zum Zeichnen der Grafiken notwendig ist, waren bereits von mir vorgegeben. Auch eine Instanz der Ball-Klasse wurde bereits erzeugt. Im Laufe des Projekts müssen die Schüler hier die Erzeugung weiterer Objekte, die Tastaturabfrage bzw. -steuerung und eine einfache Kollisionsabfrage zwischen Ball und Schlägern ergänzen.

`Ball`: Diese Klasse repräsentiert den Ball, mit dem gespielt wird. Dies ist die erste Klasse innerhalb des Spielprojekts, mit der die Schüler in Kontakt kommen. Zur leichteren Orientierung habe ich daher bereits relativ viel Code vorgegeben und die vorhandenen Codeteile ausführlich kommentiert. Neben den naheliegenden Attributen (Position des Mittelpunkts und Radius) verfügt die Ball-Klasse über eine Geschwindigkeit in x- und y-Richtung und eine Methode "bewege()", die den Ball über das Spielfeld bewegt. Methoden, um den Ball an den Rändern abprallen zu lassen, müssen von den Schülern vervollständigt werden.

`Schlaeger`: Diese Klasse repräsentiert einen Schläger, der von einem Spieler gesteuert wird (von der Klasse müssen also später zwei Objekte erzeugt werden). Ein Schläger ist ein einfaches Rechteck, das über die Tastatur nach rechts und links bewegt werden kann. In der Klasse `Schlaeger` war außer einer Methode zum Zeichnen eines Rechtecks kein Code von mir vorgegeben, die Klasse musste also im Laufe der Unterrichtseinheit nahezu vollständig von den Schülern implementiert werden.

Im Gegensatz zum vorgegebenen fertigen Spiel, das von den Schülern gespielt und analysiert wird, fehlt die Klasse `Powerup`. Zu Beginn der Unterrichtseinheit war nicht klar, ob es zeitlich möglich sein würde, diese Klasse im Laufe des Projekts noch zu ergänzen. Da das Spiel auch ohne diese Klasse spielbar ist, stellte diese Unsicherheit jedoch kein Problem dar.

Kapitel 2

Durchführung

Dieses Kapitel beschreibt für jede der vier Doppelstunden zunächst den geplanten Ablauf sowie wesentliche inhaltliche und methodische Überlegungen. Im Anschluss daran wird der tatsächliche Ablauf mit der Planung verglichen, Planabweichungen werden analysiert und mögliche Verbesserungen für die jeweilige Stunde diskutiert. Eine abschließende Analyse und Beurteilung der gesamten Unterrichtseinheit findet dann in Kapitel 3 statt.

Die inhaltlichen Schwerpunkte bezüglich der objektorientierten Programmierung sind in Tabelle 2.1 für jede der vier Doppelstunden dargestellt. Die Aufteilung entspricht dabei der tatsächlich unterrichteten, in der ursprünglichen Planung war das Thema "Konstruktoren" zunächst als weiterer Aspekt in der zweiten Doppelstunde vorgesehen.

Doppelstunde	Inhalt
1	Grundbegriffe der objektorientierten Programmierung, Grundlagen der UML
2	Verwendung von BlueJ, Klassen und Objekte in Java
3	Konstruktoren
4	Vererbung

Tabelle 2.1: Überblick über die inhaltlichen Schwerpunkte der vier Doppelstunden der Unterrichtseinheit

2.1 Erste Doppelstunde

Stundenziele

In dieser Doppelstunde sollen die Schüler mit der Grundidee der objektorientierten Programmierung vertraut gemacht werden. Insbesondere sollen nach dieser Stunde der Begriff des Objekts als Entität mit gewissen Eigenschaften (Attributen) und Fähigkeiten (Methoden) und der Begriff der Klasse als einer Art "Bauplan" für Objekte bekannt sein. Die Bedeutung des Begriffs "Kapselung" soll von den Schülern selbstständig erarbeitet werden. Ferner werden Grundlagen von UML-Diagrammen (Klassen- und Objektdiagramme) vermittelt.

Geplante Durchführung

Die Schüler erhalten eine von mir vorbereitete fertige Version des oben erwähnten Spiels "Pong". Sie bekommen Zeit, das Spiel zu spielen und erhalten dazu den Arbeitsauftrag, die im Spiel vorkommenden Spielobjekte zu bestimmen und sich zu überlegen, welche Eigenschaften und Fähigkeiten diese einzelnen Objekte haben könnten. Der Begriff "Spielobjekt" wird hierbei noch in der umgangssprachlichen Bedeutung verwendet.

Im Anschluss an diese Phase werden an der Tafel die Spielobjekte und ihre Eigenschaften gesammelt. Dabei entstehen bereits Klassendiagramme in UML-Notation. Ausgehend von dieser Analyse wird der Begriff des Objekts im Sinne der objektorientierten Programmierung eingeführt und erläutert, ebenso die Begriffe Attribut (als eine Eigenschaft eines Objekts) und Methode (als eine Fähigkeit des Objekts).

Die zwei Schläger, die im Spiel vorkommen, sind Objekte mit gleichen Eigenschaften und Fähigkeiten, deren Attribute jedoch unterschiedliche Werte haben können (und werden, schließlich befinden sich die Schläger ja an unterschiedlichen Bildschirmpositionen). Im gelenkten Unterrichtsgespräch soll dieser Sachverhalt genutzt werden, um den Begriff der Klasse einzuführen und den Unterschied zwischen einer Klasse und einem Objekt zu verdeutlichen. Die Klasse wird dabei als eine Art "Bauplan" für Objekte eines bestimmten Typs definiert.

In einer anschließenden Partnerarbeit sollen die Schüler durch eine Internetrecherche selbständig herausfinden, was unter dem Begriff der Kapselung zu verstehen ist. Die Ergebnisse werden im anschließenden Gespräch zusammengetragen, bei dieser Gelegenheit werden auch die Zugriffsrechte `public` und `private` erläutert.

Die zweite Hälfte der Doppelstunde beginnt mit einer kurzen Einführung in die UML, hier werden im Wesentlichen Klassen- und Objektdiagramme vorgestellt (Beispiele für Klassendiagramme befinden sich zu diesem Zeitpunkt ja bereits an der Tafel, der Name "Klassendiagramm" fiel bis dahin aber noch nicht). Die beiden unterschiedlichen Diagrammarten verdeutlichen nochmals die Unterscheidung zwischen der Klasse als "Bauplan" und dem Objekt als Instanz der Klasse.

Um die gerade vermittelten Grundbegriffe zu verdeutlichen, sollen die Schüler in insgesamt 7 Gruppen zu je 3 Personen für ein pro Gruppe vorgegebenes Anwendungsbeispiel/-szenario geeignete Klassen (nebst Methoden und Attributen) suchen und in einem UML-Diagramm darstellen. Die Beispielszenarien werden von mir vorgegeben und umfassen ein breites Feld an Anwendungen, beispielsweise eine Software zum Verwalten von Schülern und Schulklassen oder eine Geometriesoftware, die das Arbeiten mit einfachen geometrischen Objekten erlaubt (eine vollständige Liste aller sieben Szenarien befindet sich im Anhang). Ich wählte hier bewusst ein breites Feld an unterschiedlichen Szenarien, die auch allesamt nicht im Umfeld "Computerspiele" angesiedelt sind, um den Schülern klar zu machen, wie weit der Objektbegriff gefasst ist und wie vielfältig die objektorientierte Analyse und Modellierung eingesetzt werden können. Abhängig vom zeitlichen Verlauf sollen dann alle oder einige der gefundenen Klassen von den Schülern am Overheadprojektor präsentiert werden. Die letzte Aufgabe der Stunde ist die gemeinsame Umsetzung einer der gefunden Klassen in Java-Code. Hier kommen die Schüler erstmals mit BlueJ in Kontakt.

Der geplante Ablauf der Doppelstunde ist in Tabelle 2.2 dargestellt. Die für jede Phase

Inhalt	Dauer	Medien, Sozialform
Objektorientierte Analyse am vorgegebenen Spiel	10 min	Computer, PA
Sammeln der gefundenen Objekte (mit Eigenschaften und Methoden)	5 min	Tafel, LSG
Definition der Grundbegriffe	5 min	Beamer, Lehrervortrag
Erarbeitung des Begriffs "Kapselung"	5 min	Internet, EA/PA
Ergebnisvergleich, Begriffsdefinition "Kapselung", Zugriffsrechte	10 - 15 min	Beamer, LSG
Grundlagen der UML (Klassen- und Objektdiagramme)	5 min	Beamer, Lehrervortrag
Klassenentwurf für verschiedene Beispielszenarien	10 - 15 min	Arbeitsblatt, GA
Ergebnisvergleich	10 min	Overhead, Schülervortrag
Umsetzung einer entworfenen Klasse in Java-Code	15 min	BlueJ, LSG

Tabelle 2.2: Geplanter Ablauf der ersten Doppelstunde

angegebene geplante Dauer ist als Richtwert zu verstehen, bei den meisten Doppelstunden sind daher nicht die vollen 90 Minuten verplant.

Tatsächliche Durchführung und Analyse

Das selbstständige Spielen von Pong zu Beginn der Stunde wurde von den Schülern augenscheinlich als motivierend empfunden, es brauchte einige Aufforderungen, um sie vom Spiel loszureißen und mit der objektorientierten Analyse zu beginnen. Das Identifizieren der Spielobjekte Schläger, Ball und Powerup fiel den Schülern leicht, auch etliche Attribute wurden gefunden. Mit den Fähigkeiten bzw. Methoden taten sich die Schüler etwas schwerer, doch auch hier konnten im Gespräch etliche Methoden (z.B. die Methoden `links()` und `rechts()` der Schläger) erarbeitet werden.

Die selbstständige Recherche zum Thema "Kapselung" brachte nicht die gewünschten Ergebnisse. Zwar schienen alle Schüler bemüht, geeignete Informationen zu finden, jedoch gelang es ihnen nur sehr bedingt, die gefundenen Erklärungen mit den gerade kennengelernten Grundbegriffen Objekt, Klasse, Attribut und Methode in Einklang zu bringen. Auch weitere Erklärungen von meiner Seite brachten zunächst nicht die erhoffte Klarheit. Ein von mir nicht ausreichend vorausgesehenes Verständnisproblem war die Unterscheidung zwischen Zugriffen auf Attribute oder Methoden "von innen" (also innerhalb der Klasse) oder "von außen" (also aus anderen Programmteilen). Es erscheint mir jedoch fraglich, ob sich dieser Aspekt an dieser Stelle besser erklären lässt, ohne auf Beispielcode zurückzugreifen. Andererseits halte ich es nach wie vor für vernünftig, die Grundbegriffe wie Objekt, Klasse und Kapselung zunächst möglichst losgelöst von einer konkreten Programmiersprache zu erklären. Zum Einen würden die Schüler ansonsten nicht nur mit neuen Begriffen, sondern auch mit neuen syntaktischen Elementen konfrontiert, zum

Anderen könnte der falsche Eindruck entstehen, Objektorientierung sei ein sprachspezifisches Konzept. Für zukünftige Unterrichtsgänge anzudenken wäre eventuell, den Begriff der Kapselung auf einen Zeitpunkt zurückzustellen, an dem die Schüler bereits einfache Klassen implementiert haben. In diesem Fall besteht allerdings die Gefahr, dass sich die Schüler angewöhnen, Attribute direkt "von außen" zu ändern, ohne den vermeintlichen Umweg über die dafür vorgesehenen Methoden zu gehen.

Zufrieden war ich mit dem Verlauf der Gruppenarbeit, obwohl diese insgesamt deutlich länger dauerte als zunächst veranschlagt. Zwar erwies sich die Beschreibung mancher der sieben Szenarien als sehr offen, sodass einzelne Gruppen in die richtige Richtung gelenkt werden mussten (was sicher maßgeblich zu einer Ausdehnung dieser Phase beitrug), doch gelang es allen Schülern, für ihr Szenario eine oder mehrere Klassen zu identifizieren und diese wie gefordert in einem UML-Diagramm darzustellen. Exemplarisch sei hier auf die Ergebnisse der "Geometriesoftware-Gruppe" in Abbildung 2.2 verwiesen, Scans aller von den Schülern erstellen Folien finden sich im Anhang. Aufgrund der zeitlichen Ausdehnung der Entwurfsphase erschien es mir nicht mehr sinnvoll, die beispielhafte Umsetzung einer Klasse mit BlueJ zu besprechen, sodass der komplette Rest der zweiten Stunde für die Vorstellung und Diskussion unterschiedlicher Gruppenergebnisse verwendet wurde. Obwohl die in der Gruppenarbeit entworfenen Klassen keinen direkten Bezug zum durchzuführenden Spielprojekt hatten, halte ich diese Aufgabe auch im Rückblick für sinnvoll und auch den erhöhten Zeitaufwand noch für vertretbar. Neben einer Festigung der Begriffe Objekt und Klasse schulte diese Aufgabe überfachliche Kompetenzen wie Analysefähigkeit und Abstraktionsvermögen, aber auch die Fähigkeit zum kooperativen Arbeiten. Für die Zukunft anzudenken ist eine genauere Formulierung einzelner Aufgaben bzw. Szenarien, wobei durch die freie Formulierung auch interessante Lösungen entwickelt wurden, an die ich im Vorfeld selbst nicht dachte.

Abbildung 2.1: Die Schüler spielen und analysieren Pong.

2.2 Zweite Doppelstunde

Stundenziele

In dieser Doppelstunden lernen die Schüler, eigene Klassen in Java zu schreiben, Objekte davon zu erzeugen und Methoden dieser Objekte aufzurufen. Außerdem wird die Funktion und Implementierung eines Konstruktors thematisiert.

Geplante Durchführung

Zu Beginn der Stunde sollen in einem kurzen Gespräch die wesentlichen Aspekte der letzten Doppelstunde wiederholt werden, insbesondere natürlich die Begriffe Objekt, Klasse und Kapselung.

Da die Gruppenarbeit zur objektorientierten Analyse in der letzten Stunde länger dauerte als zunächst angenommen, kamen die Schüler bisher nicht mit der Entwicklungsumgebung BlueJ in Kontakt und wissen auch noch nicht, wie Klassen in Java implementiert werden. Losgelöst vom angestrebten Pong-Projekt soll daher zu Beginn der Doppelstunde

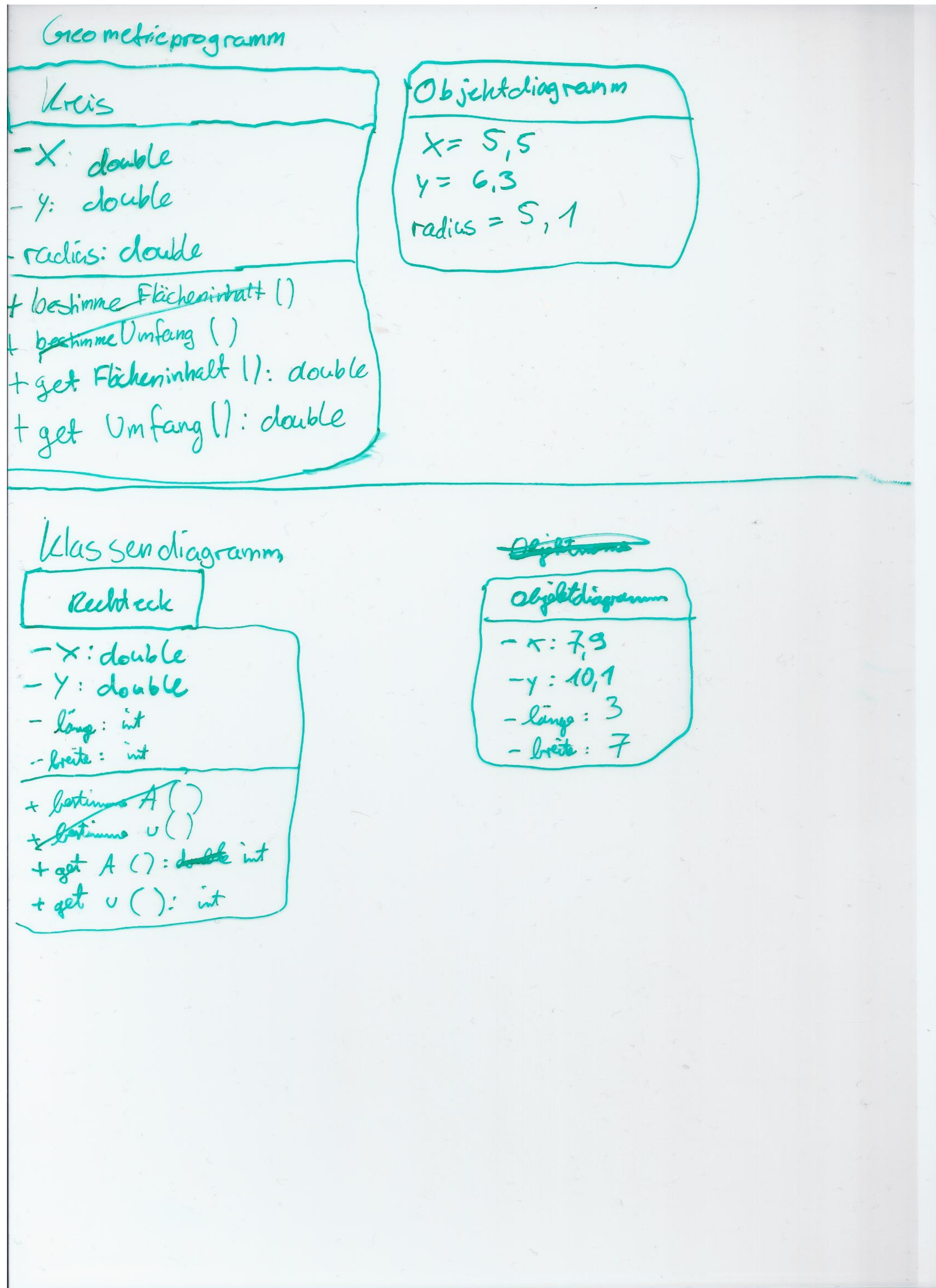


Abbildung 2.2: Von Schülern entworfene UML-Diagramme zum Szenario "Geometriesoftware"

zunächst ein Beispiel aus der Gruppenarbeit aus der letzten Doppelstunde aufgegriffen werden, nämlich die Geometriesoftware. Die Schüler in dieser Gruppe entwarfen unter anderem eine Klasse, die einen Kreis repräsentiert (festgelegt durch die Position seines Mittelpunkts und den Radius). Diese Klasse soll nun mittels BlueJ in Java umgesetzt werden. Sie erschien mir aus zwei Gründen als erstes Beispiel sinnvoll: Zum Einen ist sie einfach aufgebaut und verfügt nur über sehr wenige und nachvollziehbare Attribute. Zum Anderen ist es sehr leicht möglich, Methoden zu finden, die über reine setter/getter-Methoden hinausgehen, angedacht sind hier Methoden zur Bestimmung des Flächeninhalts und Umfangs.

Die Umsetzung der Klasse in Java-Code erfolgt dabei im Plenum, um den ersten Kontakt mit BlueJ zu vereinfachen. Nachdem die Klasse vollständig implementiert ist, sollen die Schüler selbstständig mit Hilfe von BlueJ Instanzen der Klasse erzeugen und Methoden aufrufen. Dies dient dazu, den Unterschied zwischen einer Klasse und einem Objekt nochmals zu verdeutlichen. Wie bereits erwähnt leistet hier die Visualisierung von BlueJ gute Dienste, da jedes Objekt auch in der Entwicklungsumgebung sichtbar und durch Anklicken manipulierbar ist.

Nach diesem Einstiegs-Projekt wenden wir uns wieder dem Spiel "Pong" zu, das die Schüler bisher nur als fertiges, benutzbares Spiel kennen. Sie erhalten jetzt den Quellcode einer stark "abgespeckten" Version, die bisher lediglich aus dem Spielfeld selbst und einer unvollständigen Ball- und Schläger-Klasse besteht. Die Schüler sollen zunächst einzeln oder zu zweit (entsprechend der Sitzordnung im Computerraum) diese Ball-Klasse um die nötigen setter- und getter-Methoden ergänzen. Der vorgegebene Code enthält bereits die Implementierung einer setter- und einer getter-Methode als Beispiel.

Außerdem sind die Methoden, die den Ball in x- bzw. y-Richtung[1] abprallen lassen, nicht vollständig implementiert. Auch hier sollen die Schüler selbstständig eine Lösung suchen - im Gegensatz zur Implementierung der setter- und getter-Methoden ist dieses Problem eher ein algorithmisch-mathematisches: Die Schüler müssen erkennen, dass ein Umkehren der Geschwindigkeit in x- bzw y-Richtung das gewünschte Ergebnis liefert. Da die Implementierung dieser Lösung auch den Zugriff auf bzw. die Manipulation von Attributen eines Objekts "von innen" erfordert , ist sie aber auch mit Blick auf die objektorientierte Programmierung sinnvoll.

In einer anschließenden Plenums-Phase werden zunächst die Lösungen verglichen. Im Anschluss wird gezeigt, wie sich in Java neue Objekte einer Klasse erzeugen und Methoden der erzeugten Objekts "von außen" aufrufen lassen. Entsprechender Code ist im Projekt bereits vorhanden, die Schüler sollen zunächst zu Testzwecken einen zweiten Ball erzeugen.

Zu Beginn der zweiten Hälfte der Doppelstunde wird der Konstruktor als spezielle Methode vorgestellt, die beim Erzeugen eines Objekts automatisch aufgerufen wird. Gemeinsam im Plenum wird dann ein Konstruktor für die Ball-Klasse implementiert, der als Parameter die initialen Werte für Position, Geschwindigkeit und Größe (Radius) des Balls erhält. Ich entschied mich hier gegen eine Implementierung in Einzelarbeit, da viele Schüler erfahrungsgemäß Probleme mit dem korrekten Umsetzen formaler syntaktischer

[1] Im fertigen Spiel soll der Ball nur noch am linken und rechten Bildschirmrand abprallen, für Testzwecke jedoch zunächst an allen vier Seiten.

Inhalt	Dauer	Medien, Sozialform
Kurze Wiederholung der Grundbegriffe der letzten Stunde	5 min	- , LSG
Umsetzen der Klasse "Kreis" in Java-Code	5 - 10 min	BlueJ, LSG
Ball-Klasse vervollständigen	10 min	BlueJ, EA/PA
Ergebnisvergleich	5 min	Beamer, Plenum
Objekterzeugung und Methodenaufrufe	10 min	BlueJ, EA/PA
Konstruktoren, Ball-Konstruktor implementieren und aufrufen	15 min	Beamer/BlueJ, Plenum
Umsetzen der Schläger-Klasse in Java-Code	20 min	BlueJ, EA/PA
Ergebnisvergleich	5 - 10 min	Beamer, LSG

Tabelle 2.3: Geplanter Ablauf der zweiten Doppelstunde

Vorgaben in Java-Code haben und somit ein gemeinsam erarbeitetes, konkretes Beispiel für einen Konstruktor hilfreich erschien.

Den Abschluss der Stunde bildet dann die Implementierung des Klasse Schläger, die dieses Mal ohne vorgegebenen Quellcode erfolgen soll (mit Ausnahme einer Methode zum Zeichnen des Schlägers). Zunächst werden im Plenum noch einmal die wesentlichen Attribute und Methoden gesammelt, die die Klasse haben soll. Anschließend sollen die Schüler mit BlueJ die herausgearbeiteten Methoden und Attribute implementieren. Analog zum Erzeugen des Balls sollen dann zwei Instanzen der Schläger-Klasse erzeugt werden. Das Implementieren einer einfachen Kollisionsabfrage ist in dieser Doppelstunde noch nicht vorgesehen, schnelle Schüler dürfen sich jedoch bereits Gedanken darüber machen.

Tatsächliche Durchführung und Analyse

Leider kam es in der ersten Stunde aufgrund einiger technischer Schwierigkeiten im Computerraum zu Verzögerungen: Bei einigen Schülern ließ sich zwar BlueJ starten, das Programmfenster war jedoch nicht sichtbar. Außerdem funktioniert der Beamer zunächst nicht. Daher begann die Stunde mit leichter Verspätung und auch das gemeinsame Implementieren der Klasse `Kreis` nahm deutlich mehr Zeit in Anspruch als geplant.

Beim anschließenden Vervollständigen der Ball-Klasse zeigte sich, dass viele Schüler mit dem Erweitern des Codes mehr Probleme hatten, als ich erwartete. Schon das Ergänzen von setter- und getter-Methoden für einzelne Attribute bereitete Schwierigkeiten, obwohl solche Methoden bereits für ein Attribut vorgegeben waren. Einige Schüler fanden vernünftige Ansätze für die Methoden zum Abprallen des Balles, eine korrekte Implementierung gelang jedoch wenigen. Ich beschloss daher, auch diese Methoden nach der Einzelarbeitsphase ausführlich im Plenum zu diskutieren und an der Tafel zu erläutern, wobei auch der mathematische Hintergrund mit einem Geschwindigkeitsvektor kurz thematisiert wurde. Aufgrund dieser Schwierigkeiten und des dadurch entstanden Zeitbedarfs ist für die Zukunft zu überlegen, ob das Abprallen des Balls von den Schülern weiterhin selbst implementiert werden soll. Denkbar wäre z.B., die von mir gewählte Reihen-

folge zu vertauschen: Die algorithmischen bzw. mathematischen Grundlagen könnten im Gespräch an der Tafel erarbeitet werden, die Schüler müssen dann "nur" noch die Methoden entsprechend ergänzen.

Beim Erzeugen von Objekten und dem Aufrufen von Methoden an geeigneten Stellen kam ein Nachteil von JGameGrid zum Tragen: Die Spielfeld-Klasse, in der der Code eingebaut werden musste, besitzt einen Konstruktor, in dem alle benötigten Objekte erzeugt werden müssen, sowie eine Methode `act()`, die automatisch periodisch aufgerufen wird und innerhalb derer Code zum Bewegen der Objekte implementiert werden muss. Trotz vorhandenem Beispielcode (das Erzeugen eines Ball-Objekts war bereits im Code enthalten) und erläuternder Kommentare hatten viele Schüler Probleme zu entscheiden, was sie wo implementieren bzw. ergänzen mussten. Da die Schüler in dieser Phase sehr engagiert arbeiteten, versuchte ich, aufkommende Fragen und Unklarheiten "vor Ort", also im Gespräch mit einzelnen Schülern oder Schülergruppen, zu klären, und nahm daher eine zeitliche Ausdehnung der Programmierphase in Kauf.

Wie sich an dieser Schilderung bereits erahnen lässt, wich der tatsächlich Ablauf der Stunde stark von der ursprünglichen Planung ab. Rückblickend ist deutlich erkennbar, dass ich für die Programmieraufgaben insgesamt zu wenig Zeit eingeplant hatte. Insbesondere den Zeitbedarf zum Ergänzen von setter- und getter-Methoden in der Ball-Klasse hatte ich deutlich unterschätzt, da diese Aufgabe eigentlich "nur" das Kopieren und Anpassen vorhandener Methoden bedeutete, von den Schülern jedoch dennoch nicht im vorgegebenen Zeitrahmen bearbeitet werden konnte.

Durch die technischen Schwierigkeiten zu Beginn der Stunde , die genauere Besprechung des Vorgehens zum Abprallen des Balls sowie insbesondere durch die notwendige zeitliche Ausdehnung der Programmierphasen war es nicht mehr möglich und sinnvoll, die Konstruktoren in dieser Stunde zu thematisieren, auch die Schläger-Klasse wurde auf die folgende Doppelstunde zurückgestellt. Die Entscheidung, verlorene Zeit nicht durch eine Erhöhung des Arbeitstempos oder eine Veränderung der Methodik aufholen zu wollen, sondern stattdessen die Arbeitsphasen wo nötig weiter auszudehnen und Schülern individuell Hilfe anzubieten, erscheint mir auch im Rückblick vernünftig, auch wenn dadurch etliche Inhalte auf die nächste Doppelstunde verschoben werden mussten.

2.3 Dritte Doppelstunde

Stundenziele

Die eigentlich bereits für den zweiten Teil der letzten Doppelstunde eingeplanten Konstruktoren sollen nun in dieser Doppelstunde thematisiert werden. Ansonsten dient diese Doppelstunde hauptsächlich der Wiederholung und Festigung: Sowohl das Implementieren einer Klasse als auch das Erzeugen von Objekten wird geübt, darüber hinaus erfordert der Einbau einer einfachen Kollisionsabfrage das Implementieren einer relativ komplexen `if`-Anweisung.

Geplante Durchführung

Nachdem sich in der letzten Stunde zeigte, dass sich viele Schüler auch mit vermeintlich einfachen Programmieraufgaben schwer taten und relativ viel Zeit brauchten, habe ich für die dritte Doppelstunde mehr Zeit für die einzelnen Programmieraufgaben vorgesehen.

Das einzig neue Konzept, das in dieser Doppelstunde besprochen werden soll, ist der Konstruktor. Nach einer kurzen Erklärung zum Sinn und Zweck eines Konstruktors soll die Ball-Klasse gemeinsam um einen solchen erweitert werden. Die Schüler lernen, wie ein Konstruktor aussieht und inwieweit er sich von einer gewöhnlichen Methode unterscheidet. Das Erzeugen der beiden Ball-Objekte aus der letzten Doppelstunde wird an den neu implementierten Konstruktor angepasst, die zahlreichen Aufrufe der setter-Methoden, um allen Attributen sinnvolle Startwerte zuzuweisen, können dann entfernt werden.

Im Anschluss soll das Spiel um die Schläger erweitert werden. Zunächst werden im Plenum nochmals die wesentlichen Attribute und Methoden der Schläger-Klasse gesammelt, bevor die Schüler die Klasse in Einzel- oder Partnerarbeit implementieren und die benötigten Objekte erzeugen. Ausgehend von der Erfahrungen der letzten Doppelstunde habe ich für die Implementierung und einen kurzen, abschließenden Ergebnisvergleich den gesamten Rest der ersten Unterrichtsstunde eingeplant.

Zu Beginn der zweiten Stunde soll das Spiel dann um die Möglichkeit, die Schläger mit der Tastatur zu steuern, erweitert werden. JGameGrid gibt verschiedene Methoden zur Tastaturabfrage vor. Eine der einfachsten ist die Methode `isKeyPressed()`, mit der der Zustand einer einzelnen Taste abgefragt werden kann. Die Deklaration der Methode wird den Schülern vorgegeben, der Einbau ins Programm soll möglichst alleine, ggf. mit weiteren Tipps, erfolgen.

Im letzten Teil der Doppelstunde soll dann das Spiel um eine einfache Kollisionsabfrage erweitert werden, sodass der Ball von den beiden Schlägern abprallen kann. Die Überprüfung auf Kollisionen soll dabei innerhalb der "Hauptschleife", also in der Methode `act()` der Spielfeld-Klasse, durchgeführt werden, der Code zur Kollisionserkennung ist also nicht Teil der Schläger- oder Ball-Klasse. Ich entschied mich für diesen Ansatz, weil er mir am verständlichsten erschien. Die Alternative, entweder die Ball- oder die Schläger-Klasse um eine geeignete Methode zu erweitern, erfordert die Übergabe eines Objekts (genauer: einer Referenz auf ein Objekt) als Parameter. Alternativ böte auch JGameGrid Methoden zur Kollisionserkennung an, die jedoch die Verwendung eines EventListeners erfordern und mir daher ebenfalls zu kompliziert erschienen. Die Überlegungen hinter der Kollisionsabfrage sollen dabei zunächst gemeinsam erarbeitet werden, eine Skizze an der Tafel dient dazu, den Bereich zu verdeutlichen, in dem sich der Ball im Bezug auf einen Schläger befinden muss, damit er abprallt.

Als Hilfestellung für die anschließende Implementierung der Kollisionsabfrage erhalten die Schüler ein von mir vorbereitetes "Code-Puzzle". Dabei handelt es sich um ein einfaches Textdokument, in dem bereits der komplette Code für die Kollisionsabfrage zwischen dem Ball und einem Schläger enthalten ist, jedoch zerlegt in einzelne "Bausteine", die von den Schülern in die richtige Reihenfolge gebracht werden müssen (das Dokument befindet sich im Anhang und auf der beiliegenden CD).

Schnelle Schüler dürfen sich Gedanken dazu machen, wie sie das Spiel um eine Möglichkeit, den aktuellen Punktestand zu verwalten, erweitern würden.

Inhalt	**Dauer**	**Medien, Sozialform**
Kurze Wiederholung: Klassen und Objekte in Java	5 min	-, LSG
Definition: Konstruktoren	10 min	Beamer, Lehrervortrag
Konstruktor der Ball-Klasse, Konstruktor-Aufruf	5 min	Beamer/BlueJ, Lehrervortrag
Implementieren der Schläger-Klasse, Erzeugen von Objekten	20 min	BlueJ, EA/PA
Vorstellen einer Schülerlösung	5 min	Beamer, Plenum
Einbau der Tastatursteuerung	15 min	BlueJ, EA/PA
Erarbeitung: Kollisionsabfrage	5 min	Tafel, Plenum
Codepuzzle Kollisionsabfrage	20 min	LibreOffice, EA/PA

Tabelle 2.4: Geplanter Ablauf der dritten Doppelstunde

Tatsächliche Durchführung und Analyse

Mehr Zeit für die Programmieraufgaben einzuplanen stellte sich als sinnvoll heraus, der Verlauf dieser Stunde entsprach im Wesentlichen der Planung. Einige Schüler brauchten jedoch für das Erzeugen der Schläger-Objekte und den Einbau der Tastaturabfrage mehr Zeit, sodass sie erst relativ spät mit dem Code-Puzzle anfingen und dieses nicht mehr vollständig "lösten". Für die Zukunft sollten an dieser Stelle also weitere Möglichkeiten zur Binnendifferenzierung angeboten werden. Eventuell wäre es z.B. sinnvoll, stärkere Schüler die Kollisionsabfrage zunächst ohne die Hilfestellung durch das Codepuzzle in Angriff nehmen zu lassen oder das Codepuzzle in unterschiedlichen Schwierigkeitsgraden anzubieten.

Das selbstständige Einbauen der Tastatursteuerung bereitete mehr Probleme als von mir erwartet. Schon im Vorfeld war abzusehen, dass die Schüler Hilfestellung bei der Frage brauchen würden, an welcher Stelle im Programm die Abfrage implementiert werden muss. Dass es hier zu Rückfragen und Problemen kam, war also nicht verwunderlich. Überrascht war ich hingegen, dass es vielen Schülern nicht gelang, die vorgegebene Methode `isKeyPressed` korrekt aufzurufen. Viele Schüler hatten die auf einer Folie vorgegebene Deklaration der Methode einfach unmodifiziert in ihr Programm übernommen, statt einen korrekten Methodenaufruf mit entsprechenden Parametern zu implementieren. Dies lag einerseits vermutlich daran, dass das Implementieren und Aufrufen von Methoden schon vor einigen Wochen (bei der imperativen Programmierung mit dem Hamstersimulator) unterrichtet worden war, andererseits war die von mir gewählte Darstellung auf der Folie wahrscheinlich für die Schüler zu ungewohnt, hier wäre ein beispielhafter Aufruf der Methode sicher hilfreicher und leichter verständlich gewesen. Der korrekte Einbau der Tastaturabfrage wurde daher nochmals im Plenum diskutiert.

Das Codepuzzle nahm wie geplant den Rest der zweiten Stunde ein, der Ergebnisvergleich fand in dieser Stunde keinen Platz mehr und wird daher Teil der kommenden Doppelstunde sein.

2.4 Vierte Doppelstunde

Stundenziele

Wesentlicher neuer Begriff dieser Doppelstunde ist die Vererbung. Dieses Konzept soll zunächst an einem einfachen Beispiel motiviert und erläutert werden.

Geplante Durchführung

Zunächst soll das Projekt von allen Schülergruppen möglichst auf den gleichen Stand gebracht werden, d.h der korrekt "sortierte" Code zur Kollisionsabfrage einer Gruppe wird vorgestellt und, falls noch nicht vorhanden, von allen implementiert. Außerdem soll die (ähnlich aufgebaute) Kollisionsabfrage für den zweiten Schläger ergänzt werden. Gruppen, die damit bereits fertig sind, sollen sich Gedanken machen, wie und wo die von jedem Spieler erzielten Punkte gespeichert werden können (die Aufgabe stand bereits in der letzten Doppelstunde zur Binnendifferenzierung zur Verfügung, wurde da jedoch von fast keiner Schülergruppe in Angriff genommen). Diese Aufgabe soll jedoch nicht im Plenum besprochen werden. Stattdessen erhalten im Anschluss alle Schüler die Aufgabe, eine neue Klasse `Spezialschlaeger` zu entwerfen. Ein Spezialschläger soll im Wesentlichen dem normalen Schläger entsprechen, aber zusätzlich nach oben und unten bewegt werden können. Eine Schülergruppe soll die entworfene Klasse danach vorstellen. Es ist zu erwarten, dass die Schüler beim Entwurf der Klasse feststellen, dass die Klasse ein sehr hohes Maß an Übereinstimmung mit der schon vorhandenen Schläger-Klasse aufweist und bei der Implementierung große Mengen an Code dupliziert werden müssten. Diese Feststellung soll in einem Lehrer-Schüler-Gespräch zunächst diskutiert werden, bevor der Begriff der Vererbung eingeführt wird.

Diese erlaubt es, Klassen zu definieren, die "automatisch" alle Fähigkeiten und Eigenschaften der Elternklasse haben, aber zusätzlich um weitere erweitert oder in den bestehenden angepasst werden können. Das Konzept der Vererbung wird den Schülern zunächst auf meinen Folien an einem einfachen Beispiel und dem zugehörigen UML-Diagramm verdeutlicht. Anschließend sollen sie die Klasse `SpezialSchlaeger` als Unterklasse der Klasse `Schlaeger` implementieren (und dabei selbst entscheiden, welche Methoden und ggf. Attribute ergänzt werden müssen). Das Java-Schlüsselwort `extends` wird den Schülern dabei selbstverständlich vorgegeben und seine Verwendung im Rahmen eines einfachen Codebeispiels demonstriert.

Beim Testen der Implementierungen für die Methoden `hoch()` und `runter()` werden die Schüler voraussichtlich über die Fehlermeldung stolpern, dass der Zugriff auf die y-Koordinate des Schlägers verboten ist. An dieser Stelle erfolgt noch einmal ein Blick auf die Möglichkeit, Zugriffsrechte zu spezifizieren, das Zugriffsrecht `protected` wird als weiteres Zugriffsrecht eingeführt. Die Schüler sollen dann die entsprechenden Änderungen an der Schläger-Klasse vornehmen und einen Spezialschläger in ihr Programm einbauen.

Beim Implementieren der Unterklasse wird außerdem das Problem auftreten, dass diese über einen eigenen Konstruktor verfügen und den Konstruktor der Oberklasse `Schlaeger` aufrufen muss, da die Oberklasse nur einen Konstruktor mit Parametern besitzt. Diese

Inhalt	**Dauer**	**Medien, Sozialform**
Kurze Wiederholung: Konstruktoren	5 min	- , LSG
Nochmaliges Anschauen des Code-Puzzles	5 min	LibreOffice, EA/PA
Vorstellen einer Schülerlösung	5 min	Beamer, LSG
Kollisionsabfrage in Programm einbauen und für zweiten Schläger ergänzen (mit kurzem Ergebnisvergleich)	15 min	BlueJ, EA/PA
Entwurf der Klasse `SpezialSchlaeger`	10 min	Papier, EA/PA
Ergebnisvorstellung und -diskussion	10 min	
Definition Vererbung	5 min	Beamer, Lehrervortrag
Implementieren der Klasse `SpezialsSchlaeger`	15 min	BlueJ, EA/PA
Zugriffsrecht `protected`	5 min	Beamer, Lehrervortrag
Klasse `Schlaeger` anpassen, vorgegebenen Konstruktor in die Klasse `SpezialSchlaeger` einbauen	10 min	BlueJ, EA/PA

Tabelle 2.5: Geplanter Ablauf der vierten Doppelstunde

Problematik soll im Rahmen der Doppelstunde jedoch nur kurz angesprochen werden, die Schüler bekommen den Code für den Konstruktor des Spezialschlägers vorgegeben.

Sollte nach dem Vergleich des Spezialschlägers noch Zeit bleiben, wird im Plenum noch das Zählen des Punktestands diskutiert und ggf. implementiert.

Tatsächliche Durchführung und Analyse

Es zeigte sich, dass weniger Schüler- bzw. Schülergruppen die Kollisionsabfrage in der letzten Doppelstunde komplett implementiert hatten als von mir gedacht. Daher nahmen der Ergebnisvergleich des Codepuzzles, die Umsetzung in das bestehende Programm sowie Einbau der Kollisionsabfrage für den zweiten Schläger (nebst kurzem Vergleich) die komplette erste Stunde in Anspruch, der Entwurf der Klasse Spezialschläger verschob sich vom Ende der ersten auf den Beginn der zweiten Stunde.

Der Entwurf der Klasse `Spezialschlaeger` funktionierte gut und zügig. Die hohe Übereinstimmung zwischen den Klassen `Schlaeger` und `SpezialSchlaeger` konnte im Gespräch herausgearbeitet und das Konzept der Vererbung somit sinnvoll motiviert werden.

Wieder war es die Umsetzung in Java, die vielen Schülern Schwierigkeiten bereitetet. Insbesondere wollten einige Schüler Attribute, die die Klasse `SpezialSchlaeger` eigentlich von der Oberklasse `Schlaeger` geerbt hat, nochmals innerhalb der Klasse `SpezialSchlaeger` definieren. Dies war insofern verwunderlich, als sich auf meinen zuvor gezeigten Folien bereits ein einfaches, konkretes Beispiel für eine Vererbungsbeziehung befand und die Schüler die Attribute und Methoden der Unterklasse korrekt genannt hatten.

Abzusehen war, dass den Schülern die Notwenigkeit, aus der Unterklasse den Konstruktor der Oberklasse aufzurufen, an dieser Stelle nicht vollständig klar werden würde.

Leider ließ es sich aus technischen Gründen nicht vermeiden, diese Problematik zumindest kurz anzusprechen: Da die in der vorherigen Doppelstunde implementierte Schläger-Klasse nicht über einen Standardkonstruktor verfügte, wäre die Klasse `SpezialSchlaeger` ohne eigenen Konstruktor nebst Aufruf von `super` nicht kompilierbar gewesen. Als Alternative wäre denkbar, die Schläger-Klasse so zu entwerfen, dass sie zusätzlich einen parameterlosen Konstruktor erhält. In diesem Fall scheint es aber schwierig, den Schülern den Sinn von zwei Konstruktoren mit unterschiedlicher Signatur zu motivieren. Ich entschied mich stattdessen dazu, die Thematik nach Abschluss der hier dokumentierten Unterrichtseinheit nochmals an einem anderen Beispiel aufzugreifen und zu vertiefen.

Kapitel 3

Auswertung

3.1 Analyse

Struktur der Unterrichtseinheit

Der von mir gewählte Unterrichtsgang motivierte die Objektorientierung anhand einer objektorientierten Analyse an einem vorgegebenen Computerspiel. Die Grundbegriffe Objekt, Klasse und Kapselung konnten so losgelöst von einer konkreten Programmiersprache erläutert werden. Für die Schüler bedeutete dies zunächst eine etwas flachere Lernkurve, weil sie sich auf die Konzepte konzentrieren konnten, ohne sich gleichzeitig mit der Java-Syntax befassen zu müssen. Bei der Erarbeitung der Kapselung führte dieser Weg jedoch zu Verständnisproblemen, die vermutlich nicht - oder zumindest weniger ausgeprägt - aufgetreten wären, wenn die Schüler zu diesem Zeitpunkt bereits Kontakt mit der Implementierung in Java gehabt hätten. Für die Zukunft ist also anzudenken, die Kapselung zunächst etwas zurückzustellen, wobei dann ein Weg gefunden werden muss, die in 2.1 erwähnten Nachteile abzuschwächen oder zu vermeiden.

Erst in der zweiten Doppelstunde kamen die Schüler mit der objektorientierten Programmierung in Java in Kontakt und lernten am Beispiel eines Computerspiels, wie sich in Java Klassen definieren lassen, wie man Objekte von diesen Klassen erzeugt und wie man mit diesen Objekten interagiert, sprich, deren Methoden aus anderen Programmteilen aufruft. Trotz der in Kapitel 2 angesprochenen zeitlichen Schwierigkeiten hauptsächlich in der zweiten Doppelstunde erscheint mir die grundlegende Struktur der Unterrichtseinheit nach wie vor sinnvoll und geeignet - in Zukunft sollte die Zeitplanung der Einzelstunden aber dem tatsächlichen Ablauf in dieser Unterrichtseinheit entsprechen. Schnellere Schüler (oder Kurse) können dabei jederzeit mit zusätzlichen Aufgaben wie beispielsweise der schon angesprochenen Punktezählung oder der Klasse `PowerUp` betraut werden.

Klar ist, dass eine Einzelstunde zum Thema Vererbung nicht mehr leisten kann, als das grundlegende Konzept vorzustellen. Nicht thematisiert wurden beispielsweise Vererbungshierarchien, das Überschreiben von geerbten Methoden oder auch die dynamische Bindung. Inwieweit sich das gewählte Spielprojekt und JGameGrid für derartige weiterführende Aufgaben eignen, wird in den nächsten Abschnitten noch kurz angesprochen. Auch fehlten weitere Anwendungsbeispiele und Übungsaufgaben zum Erkennen möglicher Vererbungsbeziehungen und zur Umsetzung in Java-Code. Wäre die letzte Doppelstunde komplett für

die Vererbung zur Verfügung gestanden, so hätte man beispielsweise eine Aufgabe ähnlich der Gruppenarbeit aus der ersten Stunde anbieten können, bei der die Schüler für verschiedene Szenarien Vererbungsbeziehungen suchen sollen. Für die Zukunft anzudenken ist also, die Unterrichtseinheit weiter zu strecken, sodass die Vererbung erst in einer fünften Doppelstunde thematisiert wird. Die vierte Doppelstunde könnte dann dazu genutzt werden, das Spiel (zunächst ohne Vererbung) zu vervollständigen, also beispielsweise die Punktezählung und ggf. die Klasse `PowerUp` einzubauen und verbleibende Bugs zu suchen und zu beseitigen. Führt man das Projekt über diese vier bzw. möglicherweise fünf Doppelstunden weiter fort, so ließen sich sukzessive weitere Möglichkeiten von JGameGrid umsetzen, insbesondere natürlich die bisher nicht verwendete Klasse `Actor`. Diese könnte dazu verwendet werden, gemeinsamen Zustand aus den Klassen `Schlaeger`, `Ball` und ggf. `PowerUp` "herauszuziehen" und so eine weitere Vererbungshierarchie zu schaffen.

Tauglichkeit von JGameGrid

JGameGrid wurde als Klassenbibliothek speziell für Ausbildungszwecke entworfen. Insbesondere aufgrund seiner Natur als reine Bibliothek ohne unterstützende Entwicklungsumgebung werden die Schüler selbst bei einfachen Projekten mit Codeteilen konfrontiert, deren Bedeutung und Zusammenspiel sich ihnen nicht unmittelbar erschließt. Es zeigte sich jedoch in der Evaluation durch die Schüler, dass den meisten der vorgegebene Coderahmen wenig Probleme bereitete (siehe 3.1). Den von mir vorgegebenen Coderahmen durch die Schüler selbst entwickeln zu lassen, erscheint mir aus zwei Gründen nicht sinnvoll: Zum Einen wäre der zeitliche Aufwand dafür immens hoch, zum Anderen würden die Schüler etliche Konzepte benutzen müssen, die sie zu diesem Zeitpunkt noch nicht verstehen können.

Der von mir gewählte Einstieg über eine objektorientierte Analyse und das anschließende Implementieren der gefundenen und entworfenen Klassen passte insofern nicht zu JGameGrid, als die Bibliothek ihre Benutzer eigentlich von Anfang an mit der Vererbung konfrontiert: Alle Klassen für Spielfiguren sind Unterklassen der Klasse `Actor`. Erfreulicherweise bietet JGameGrid jedoch die nötige Flexibilität, um auch ohne diese frühe Einführung der Vererbung auszukommen. Ein solches Vorgehen wäre bspw. mit Greenfoot nicht möglich gewesen oder zumindest weit schwieriger geworden: Die dort ebenfalls vorhandene `Actor`-Basisklasse ist noch sehr viel tiefer im Gesamtsystem verwurzelt.

In dieser Unterrichtseinheit wurde nur ein geringer Bruchteil der Möglichkeiten von JGameGrid ausgeschöpft. Verwendet man beispielsweise die Klasse `Actor` als Basisklasse aller Spielfiguren, so erhält man eine einfache Möglichkeit, Rastergrafiken für die Figuren zu verwenden. Darüber hinaus bietet JGameGrid beispielsweise Funktionen zur Kollisionserkennung, vielfältige Möglichkeiten der Tastaturabfrage und vieles mehr. Diese Möglichkeiten sind von Schülern, die erst seit wenigen Wochen programmieren, natürlich nur bedingt einsetzbar. Allerdings wird JGameGrid dadurch auch für Schüler mit mehr Programmiererfahrung interessant, beispielsweise im Rahmen eines Informatikkurses in der Jahrgangsstufe 2. Hier hat man die Möglichkeit, komplexere Spielprojekte zu schaffen, die mehr Gebrauch machen von den Möglichkeiten, die JGameGrid bietet. Ein solches

Spielprojekt ließe sich vermutlich auch sinnvoll in einzelne Teilprojekte zerlegen, die dann von einzelnen Schülergruppen bearbeitet werden.

Nicht unerwähnt bleiben soll außerdem der ausgezeichnete Support für JGameGrid: Im Rahmen der Vorbereitung dieser Unterrichtseinheit wandte ich mich mehrfach an Prof. Plüss, der meine eMails immer schnell beantwortete und auch in kürzester Zeit von mir vorgeschlagene Veränderungen an der Bibliothek vornahm.

Klausurergebnisse

Zwei Wochen nach Abschluss der hier dokumentierten Unterrichtseinheit wurde im Informatik-Kurs die erste Klausur geschrieben. Die objektorientierte Programmierung nahm dabei den kleineren Teil der Klausuraufgaben ein, da die zuvor vermittelten Grundlagen der imperativen Programmierung ebenfalls abgeprüft wurden.

Die von den Schüler zu bearbeitenden Aufgaben aus diesem Themengebiet lassen den Schluss zu, dass die theoretischen Grundlagen von einem Großteil der Kursteilnehmer verstanden und gelernt wurden: Eine Aufgabe, bei der die Schüler passende Attribute für eine bestimmte Klasse identifizieren und ein Klassen- und Objektdiagramm zeichnen sollten, wurde von den meisten Schülern gut gelöst. Auch das Konzept der Kapselung zu erkennen und in eigenen Worten zu erklären gelang vielen. Interessanterweise bereitete auch in der Klausur die korrekte Implementierung in Java-Code die meisten Schwierigkeiten. Sowohl beim Implementieren eines einfachen Konstruktors als auch beim Implementieren einer einfachen, zu einer Klasse gehörenden Methode machte die Mehrheit der Schüler syntaktische Fehler. Auch der geforderte Java-Code zum Erzeugen eines Objekts wurde nur von sehr wenigen Schülern vollständig richtig angegeben. Dies hat mich insbesondere insofern verwundert, als die Implementierung nicht nur im Rahmen der hier dokumentierten Unterrichtseinheit mehrfach thematisiert und geübt, sondern auch in der vor der Klausur angesetzten Übungsstunde nochmals explizit angesprochen wurde. Zur Relativierung sei gesagt, dass die Schüler oft an Kleinigkeiten scheiterten: So wurde beim Implementieren einer Methode die Angabe des Rückgabetyps vergessen oder der falsche Typ angegeben, in der Parameterliste fehlte der Datentyp der Parameter, usw. Offensichtlich tun sich also viele Schüler mit den strengen syntaktischen Vorgaben einer Programmiersprache schwer, während theoretische Konzepte weniger Probleme bereiten.

Evaluation durch die Schüler

Nach der Rückgabe der Klausur erhielten die Schüler noch einen von mir erstellten Evaluationsbogen, mit dem ich ermitteln wollte, wie die Schüler selbst die Unterrichtseinheit empfanden. Sie sollten dazu verschiedene Aussagen auf einer Skala von -2 (trifft nicht zu) bis +2 (trifft voll zu) bewerten. Der Evaluationsbogen wurde von 20 der 21 Kursteilnehmer ausgefüllt (ein Schüler war an diesem Tag krank). Exemplarisch möchte in in diesem Abschnitt einige Ergebnisse der Evaluation vorstellen.

19 von 20 Schülern beantworteten die Frage, ob sie die Entwicklung eines Spiels als motivierend empfanden, mit +1 oder +2, stimmten der Aussage also zu. Lediglich ein Schüler kreuzte hier den Wert 0 an. Alle Schüler waren der Meinung, dass die einführende

Analyse des Spiels geeignet war, das Grundkonzept der Objektorientierung zu verdeutlichen. 3 Schüler bewerteten die Aussage, dass sie den vorgegebenen Coderahmen verwirrend fanden, mit +1, weitere 4 Schüler mit 0. Ein Schüler enthielt sich bei dieser Frage aus nicht weiter spezifizierten Gründen komplett. 5 Schüler gaben an, dass sie für die Programmieraufgaben mehr Zeit gebraucht hätten (Zustimmung +1 oder +2), 3 Schüler hätten sich mehr oder genauere Erklärungen gewünscht (Zustimmung ebenfalls +1 oder +2). In der Selbsteinschätzung kamen die meisten Schüler zu dem Schluss, die besprochenen Konzepte und deren Implementierung verstanden zu haben (Zustimmung +1 oder +2). Dies deckt sich teilweise nicht mit den Ergebnissen der Klausur, in der sich insbesondere bei der Umsetzung in Java mehr oder weniger große Probleme zeigten.

Insgesamt beurteilten die Schüler die Unterrichtseinheit also positiv.

3.2 Fazit und Ausblick

Das Ziel, den Schülern die Grundlagen der objektorientierten Programmierung am Beispiel eines einfachen Computerspiels zu vermitteln, wurde im Wesentlichen erreicht. Der Zeitrahmen von insgesamt vier Doppelstunden erwies sich jedoch für die geplanten Themen als etwas zu knapp, die Grundlagen der Vererbung sollten besser in eine fünfte Doppelstunde ausgelagert werden, um auch hier mehr Zeit für zusätzliche Übungsaufgaben zu haben. Der enge zeitliche Rahmen sowie das ohnehin anspruchsvolle Thema führten an mancher Stelle zu einer steilen Lernkurve und zu einem von manchen Schülern als zu hoch empfundenen Tempo. Für die Zukunft sind hier weitere Möglichkeiten zur Binnendifferenzierung zu suchen.

Das Konzept der Vererbung, insbesondere die Notwendigkeit von Konstruktor-Aufrufen der Basisklasse, wurde in zwei Einzelstunden nach der Klausur nochmals aufgegriffen und vertieft. Die Vermittlung weiterführender Konzepte wie Objektreferenzen, Polymorphie und dynamische Bindung war nicht Teil dieser Unterrichtseinheit, hierfür müssen weitere Stunden eingeplant werden.

Die Kombination aus der Entwicklungsumgebung BlueJ und der Klassenbibliothek JGameGrid erwies sich als sinnvoll. Im Gegensatz zum ähnlich gearteten Greenfoot punktet JGameGrid vor allem durch die höhere Flexibilität, durch die es möglich war, die Vererbung zunächst komplett auszuklammern und den Fokus auf die Implementierung einfacher, eigenständiger Klassen zu richten. Im Gegensatz zum "trockenen" Implementieren und Testen von Klassen mit BlueJ allein "belohnte" JGameGrid die Arbeit der Schüler mit direkt sichtbaren, grafischen Resultaten und am Ende der Unterrichtseinheit sogar mit einem (hoffentlich) funktionierenden, spielbaren Spiel.

Die Auswahl eines geeigneten Spiels zur Analyse und Nachimplementierung war nicht trivial, sollte das Spiel doch auf der einen Seite möglichst mehrere unterschiedliche Arten von Spielobjekten bieten, auf der anderen jedoch für die Schüler mit entsprechender Hilfestellung implementierbar sein. Die gewählte Variante des Spiels Pong stellt meiner Meinung nach einen sinnvollen Kompromiss zwischen diesen Anforderungen dar - lediglich der Spezialschläger zur Motivation der Vererbung mag etwas "konstruiert" wirken. Erweitert man die Unterrichtseinheit auf einen größeren Zeitrahmen, so bietet das Pong-Projekt durchaus Potential für weiterführende Themen und Techniken. So könnte man

nach Einführung der Vererbung die vorhandenen Klassen `Ball` und `Schlaeger` dahingehend modifizieren, dass die Klasse `Actor` als Basisklasse verwendet wird. Diese Modifikation ließe sich zunächst sogar motivieren, ohne die Klasse `Actor` vorzustellen, indem man die Schüler gemeinsamen Zustand der bisher implementierten Klassen identifizieren lässt. Weiterhin macht die in der hier dokumentierten Unterrichtseinheit implementierte Lösung einige Kompromisse, in denen einfach zu verstehender Code einer aus Sicht der objektorientierten Programmierung "schöneren" Lösung vorgezogen wurde. Zu nennen ist hier vor allem die Kollisionsabfrage, die momentan in der "Hauptschleife" des Spiels stattfindet. Bei einer Weiterentwicklung des Projekts könnte man stattdessen die Kollisionsabfrage in eine der vorhandenen Klasse, entweder `Ball` oder `Schlaeger`, "hineinziehen". Dies würde insbesondere die Verwendung von Objektreferenzen erfordern, die an dieser Stelle thematisiert werden könnten. Alternativ dazu könnte das Projekt auf die von JGameGrid bereitgestellten Methoden zur Kollisionserkennung über EventListener umgestellt werden.

Weniger gut geeignet erscheint das Pong-Projekt zur Vermitteln weiterführender Konzepte der objektorientierten Programmierung. Schon der Spezialschläger war eine etwas erzwungen anmutende Erweiterung des Spielprinzips, um eine sinnvolle Klassenhierarchie zu schaffen. Das Finden von geeigneten Unterklassen, die beispielsweise vorhandene Methoden überschreiben, gestaltet sich meiner Meinung nach schwierig. Daher wird man vermutlich Mühe haben, Polymorphie bzw. dynamische Bindung in diesem Szenario zu thematisieren.

Mit den in dieser Ausarbeitung herausgearbeiteten Anpassungen und Änderungen gegenüber der ursprünglichen Planung erscheint mir die vorgestellte Unterrichtseinheit auch für zukünftige Jahre und Kurse geeignet, um damit die Grundlagen der objektorientierten Programmierung zu unterrichten.

Literaturverzeichnis

[1] BOLES, D. Hamstersimulator. http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/index2.html, 2008.

[2] HARTMANN, W., NÄF, M., AND REICHERT, R. *Informatikunterricht planen und durchführen*. Springer, 2007.

[3] KÖLLING, M. GreenFoot. http://www.greenfoot.org, 2006.

[4] KÖLLING, M., AND ROSENBERG, J. BlueJ. http://www.bluej.org, 1999.

[5] MINISTERIUM FÜR KULTUS J. U. S. Bildungsplan Informatik (Gymnasium).

[6] PLÜSS, A. JGameGrid. http://www.aplu.ch/home/apluhomex.jsp?site=45, 2010.

[7] TIOBE. TIOBE Programming Community Index. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html, Dezember 2011.

Kapitel 4

Anhang

Im Anhang finden sich

1. Die in den vier Doppelstunden verwendeten Folien
2. Material aus der Gruppenarbeit "Klassenentwurf"
   (a) Das Arbeitsblatt mit den Arbeitsaufträgen
   (b) Scans der von den Schülern entworfenen UML-Diagramme
3. Die Vorlage für das Codepuzzle zur Kollisionsabfrage
4. Die nach der Unterrichtseinheit geschriebene Klausur
5. Der Evaluationsbogen
6. Screenshots der im Literaturverzeichnis genannten Webseiten

Aus Platzgründen ist das von mir geschriebene Skript zur objektorientierten Programmierung nicht im Anhang enthalten. Es befindet sich jedoch auf der beiliegenden CD.

4.1 Foliensätze

Informatik K1 2011/12

Philipp Kupferschmied

HGS

24.10.2011

Aufgabe

- Partnerarbeit: Startet die Datei Pong.jar durch Doppelklick und spielt ein wenig gegeneinander
  - Spieler oben: Pfeil links/rechts
  - Spieler unten: A und D
- Beantwortete folgende Fragen:
  - Welche Spielobjekte gibt es?
  - Welche Eigenschaften und Fähigkeiten haben diese?

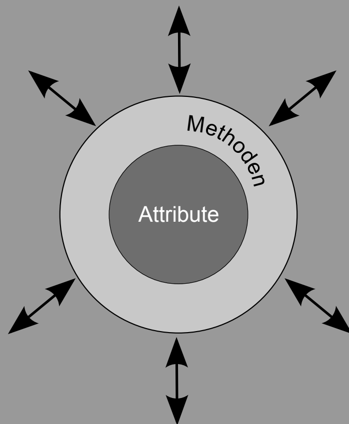
Objektorientierte Programmierung

- Grundidee:
  - Zusammenfassen von Daten und den darauf anwendbaren Methoden zu einem Datentyp, dem Objekt
- Objekt:
  - "Ding" mit gewissen Eigenschaften (Attributen) und Fähigkeiten (Methoden)
  - Häufig eine Nachbildung/Modellierung eines realen Objekts
- Klasse:
  - "Bauplan" eines Objekts
  - legt Attribute und Methoden fest, aber noch nicht deren konkrete Ausprägung
- Von einer Klasse lassen sich beliebig viele Objekte erzeugen!
  - Objekte bezeichnet man auch als Instanz der zugehörigen Klasse

Aufgabe

- Partnerarbeit: Findet heraus, was unter dem Begriff Kapselung (im Bezug auf objektorientierte Programmierung) zu verstehen ist
- Welche Konsequenzen ergeben sich daraus für die gefundenen Klassen?

Kapselung (1)



Kapselung (2)

- Verändern/Auslesen von Attributen über spezielle Zugriffsmethoden, oft setter/getter genannt
  - Beispiel:
    - `void setBreite( int neueBreite )`
    - `int getBreite()`
- Zugriffsrechte spezifizieren, wer worauf zugreifen darf

  public Jeder darf auf die Methode oder das Attribut zugreifen

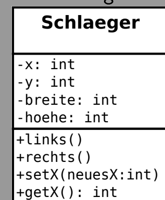
  private Zugriff ist nur innerhalb der Klasse möglich

UML-Diagramme (1)

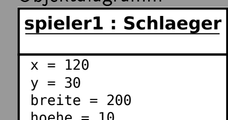
- UML (Unified Modeling Language)
  - graphische Modellierungssprache
- Klassendiagramm
  - Stellt eine Klasse dar, beinhaltet Klassenname, Attribute (ggf. mit Datentyp), Methoden (ggf. mit Parametern und Rückgabetyp) und Zugriffsrechte (+ für public, - für private)
- Objektdiagramm
  - Stellt ein konkretes Objekt dar, kann Werte aller oder bestimmter Attribute zeigen

UML-Diagramme (2)

Klassendiagramm



Objektdiagramm



Aufgabe

- Sieben 3er-Gruppen
- Für das jeweilige Beispielszenario geeignete Klasse(n) identifizieren und deren Eigenschaften (Attribute) und Methoden (Fähigkeiten) ermitteln.
- Klasse(n) in einem Klassendiagramm darstellen (ohne setter/getter)
- EIN beispielhaftes Objektdiagramm zeichnen
- Ergebnisse auf Folie darstellen

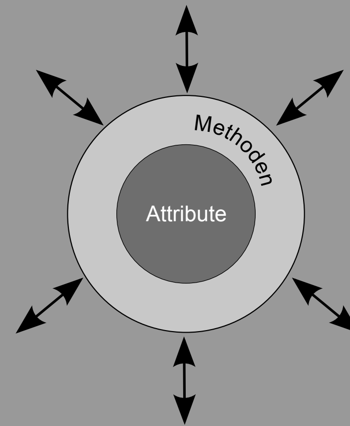
Informatik K1 2011/12

Philipp Kupferschmied

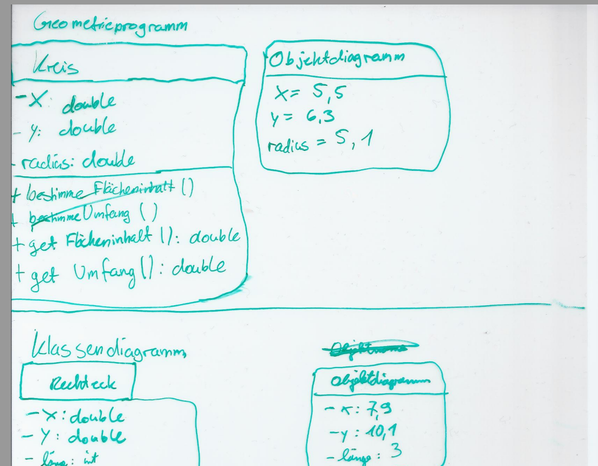
HGS

7.11.2011

Was war das doch gleich?



Geometrie-Software



Aufgabe

Vervollständige die Ball-Klasse:

- Ergänze fehlende setter/getter-Methoden
- Analysiere die Methode `bewege()`
  - Welche Abmessungen (in Pixeln) hat das Spielfeld?
- Vervollständige die Methoden `abprallenX()` und `abprallenY()`

Aufgabe

- Weise den weiteren Attributen des erzeugten Ball-Objekts sinnvolle Wert zu
- Rufe an einer geeigneten Stelle die Methode `bewege()` des Ball-Objekts auf
- Erzeuge ein zweites Ball-Objekt und lasse dieses sich auch bewegen

Konstruktoren

- Spezielle Methode, die beim Erzeugen eines Objekts automatisch aufgerufen wird
- Kann benutzt werden, um z.B. Werte für Attribute des erzeugten Objekts festzulegen
- Definition ähnlich wie gewöhnliche Methode, aber
  - Name muss dem Namen der Klasse entsprechen
  - KEINE Angabe eines Rückgabetyps (auch nicht `void`)

Aufgabe

Vervollständige die Schlaeger-Klasse:

- Implementiere einen geeigneten Konstruktor
- Ergänze fehlende Attribute und Methoden
- Erzeuge zwei Schlaeger-Objekte an geeigneten Positionen
- Wo könnte man den Code zur Steuerung/Tastaturabfrage einbauen?

Informatik K1 2011/12

Philipp Kupferschmied

HGS

14.11.2011

Konstruktoren

- Spezielle Methode, die beim Erzeugen eines Objekts automatisch aufgerufen wird
- Kann benutzt werden, um z.B. Werte für Attribute des erzeugten Objekts festzulegen
- Definition ähnlich wie gewöhnliche Methode, aber
  - Name muss dem Namen der Klasse entsprechen
  - KEINE Angabe eines Rückgabetyps (auch nicht `void`)

Aufgabe

- Implementiere die Schläger-Klasse
  - Konstruktor
  - Attribute/Methoden
- Erzeuge zwei Objekte der Klasse

Aufgabe

- Innerhalb der Spielfeld-Klasse steht die Methode `boolean isKeyPressed( int keyCode )` zur Verfügung, die `true` zurück liefert, wenn die Taste mit dem entsprechenden Keycode aktuell gedrückt ist.
- Keycodes sind z.B. `KeyEvent.VK_LEFT` (Pfeil links), `KeyEvent.VK_A` (Taste 'A'), ...
- Wo innerhalb der Spielfeld-Klasse sollte man die Tastaturabfrage einbauen?

Aufgabe

- Bringe den Code für die Kollisionsabfrage zwischen Ball und Schläger in die richtige Reihenfolge.

Informatik K1 2011/12

Philipp Kupferschmied

HGS

21.11.2011

Aufgabe

- Bringe den Code zur Kollisionsabfrage im "Codepuzzle" in die richtige Reihenfolge
  - 5 Minuten, dann Vergleich

Aufgabe

- Implementiere die Kollisionsabfrage für beide (!) Schläger und teste sie
  - 10 Minuten
- Falls Du früher fertig bist, mache Dir Gedanken zum Punktestand
  - Wer bekommt wann einen Punkt?
  - Wie/wo kann man speichern, wer wie viele Punkte hat?

Aufgabe

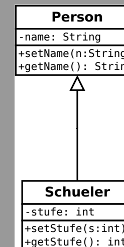
- Entwirf eine Klasse `SpezialSchlaeger`. Ein Objekt dieser Klasse soll sich nicht nur nach links und rechts, sondern auch nach oben und unten bewegen können.
  - Zeichne ein UML-Diagramm
  - Implementiere die Klasse (nicht am PC, auf Papier)
  - setter/getter können weggelassen werden

Vererbung

- Erlaubt es, von einer Klasse eine sog. Unterklasse abzuleiten
- Die Unterklasse erbt alle Eigenschaften und Methoden der Elternklasse
- Die Unterklasse kann um zusätzliche Attribute und Methoden ergänzt werden
- Die Unterklasse kann geerbte Methoden überschreiben

Zwischen Unterklasse und Oberklasse besteht in der Regel eine "Ist-Ein"-Beziehung

Vererbung in der UML



- Welche Attribute und Methoden hat ein Objekt von der Klasse `Schueler`?

Vererbung in Java

- Schlüsselwort `extends` definiert Vererbungsbeziehung
- Z.B.:

```
class Schueler extends Person
{
    private int stufe;
    public void setStufe( int neueStufe )
    {
        stufe = neueStufe;
    }
}
```


Aufgabe

- Implementiere die Klasse `SpezialSchlaeger` als Unterklasse der Klasse `Schlaeger`
- Ersetze einen der Schläger im Spiel durch einen Spezialschläger
  - Erweitere die Tastaturabfrage entsprechend

Zugriffsrechte

public: Jeder darf auf die Methode oder das Attribut zugreifen
private: Zugriff ist nur innerhalb der Klasse möglich
protected: Zugriff ist innerhalb der Klasse und aus Unterklassen erlaubt.

Konstruktoren und Vererbung (1)

- Konstruktoren werden nicht vererbt!
- Beim Erzeugen eines Objekts der Unterklasse wird automatisch der Standardkonstruktor der Oberklasse aufgerufen
- Hat die Oberklasse keinen Standardkonstruktor, so muss der vorhandene Konstruktor "von Hand" aufgerufen werden
  - Der Aufruf des Konstruktors der Oberklasse erfolgt mittels `super( <Parameterliste> );`

Konstruktoren und Vererbung (2)

```
class SpezialSchlaeger extends Schlaeger
{
    // Konstruktor des SpezialSchlaegers
    public SpezialSchlaeger( int x, int y, int b, int h )
    {
        // Aufruf des Konstruktors der Klasse Schlaeger
        super( x, y, b, h );
    }
}
```


Aufgabe

- Erweitere die Klasse `Schlaeger` um ein Attribut `punkte` und geeignete Methoden
- Wo/wie kann man die Punkte zählen?

4.2 Material aus der Gruppenarbeit "Klassenentwurf"

Szenario 1
Eine Software, die Schüler und Schulklassen verwaltet.

Szenario 2
Ein Geometrieprogramm, das das Arbeiten mit verschiedenen geometrischen Grundobjekten (Kreis, Rechteck, ...) erlaubt.

Szenario 3
Eine Software, mit der ein Gebrauchtwagenhändler seinen Fuhrpark verwalten kann.

Szenario 4
Der Hamstersimulator mit mehreren Hamstern.

Szenario 5
Eine Software zur Verwaltung des Warenbestands eines Supermarkts.

Szenario 6
Ein Mediaplayer, der Musikdateien verwaltet und abspielt.

Szenario 7
Eine „elektronische Fernsehzeitschrift“.

Schulklassen

- Lehrer : string
- Anzahl_Schüler : int
- klassenraum : int
- Stufe : int
- Name : string
+ krankheitsfälle (Anzahl : int)
+ Raumverlegung (neuer_Raum : int)
+ klassendurchschnitt (Durchschnittsnoten : int) : int
+ neues_Schuljahr ()
+ Ausflug ()

Schüler

- Geburtstag : string
- Name : string
- Noten : int
- Sitzposition : int
- Wohnort : string
- Telefonnummer : int
- Fehlstunden : int

+ Durchschnittsnote (Noten : int) : int
+ Alter (Geburtstag : ~~int~~ string) : int
+ Versetzung (Durchschnittsnote : int) : boolean
+ krankheitsfall ()

Geometrieprogramm

Kreis
- x: double - y: double - radius: double
+ ~~bestimme Flächeninhalt~~ () + ~~bestimme~~ Umfang () + get Flächeninhalt (): double + get Umfang (): double

Objektdiagramm
x = 5,5 y = 6,3 radius = 5,1

Klassendiagramm

Rechteck
- x : double - y : double - länge : int - breite : int
+ ~~bestimme~~ A () + ~~bestimme~~ U () + get A () : ~~double~~ int + get U () : int

~~Objektname~~

Objektdiagramm
- x : 7,9 - y : 10,1 - länge : 3 - breite : 7

Klassendiagramm:

Eine Software, mit der ein Gebrauchtwagenhändler seinen Fuhrpark verwalten kann.

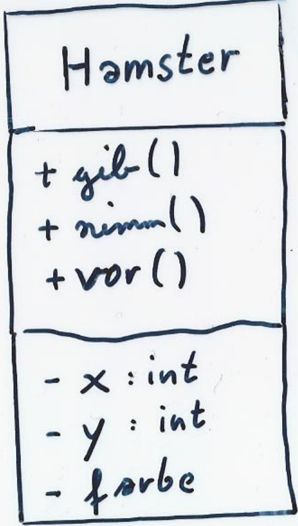
Class Auto
- Modell : int - Farbe (Bsp.: 0x6) - Stellplatz (x|y) : int - Kilometerzahl : double - Baujahr : int - Verhandlungsbasis : int
+ Auf Lager () : boolean + verkauft () : boolean + in Reparatur () : boolean

Objektdiagramm:

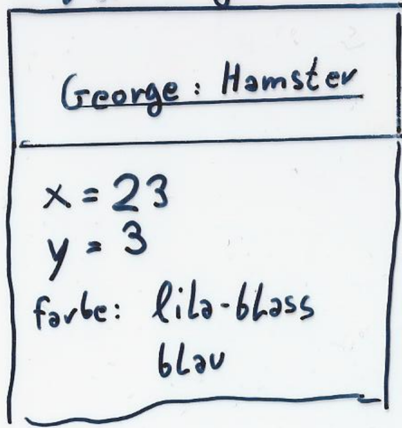
Auto: BMW M3
Modell = 1 Farbe = 0x6 Stellplatz = (5|3) Kilometerzahl = 2.0000,5 Baujahr = 2005 Verhandlungsbasis = 10000

Ich bin das lustige Äffchen

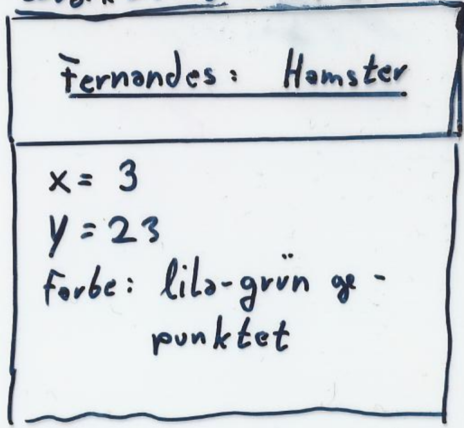
Klassendiagramm:



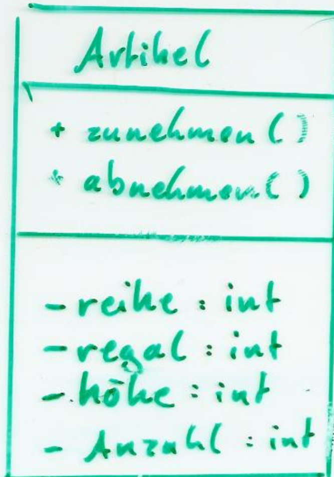
Objektdiagramm 1:



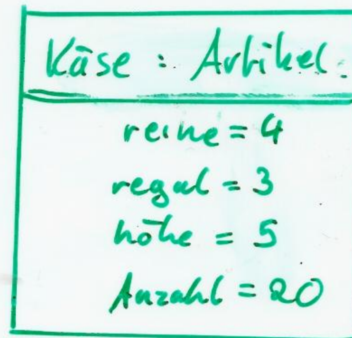
Objektdiagramm 2:



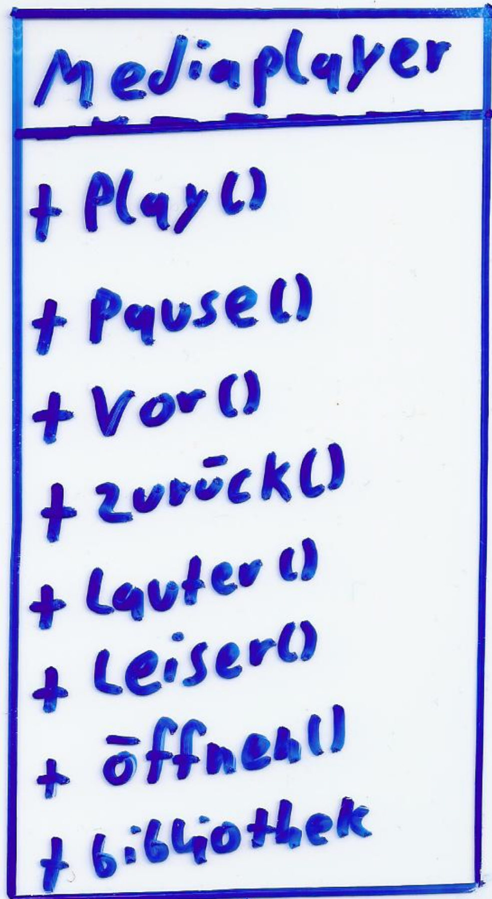
Klassendiagramm:



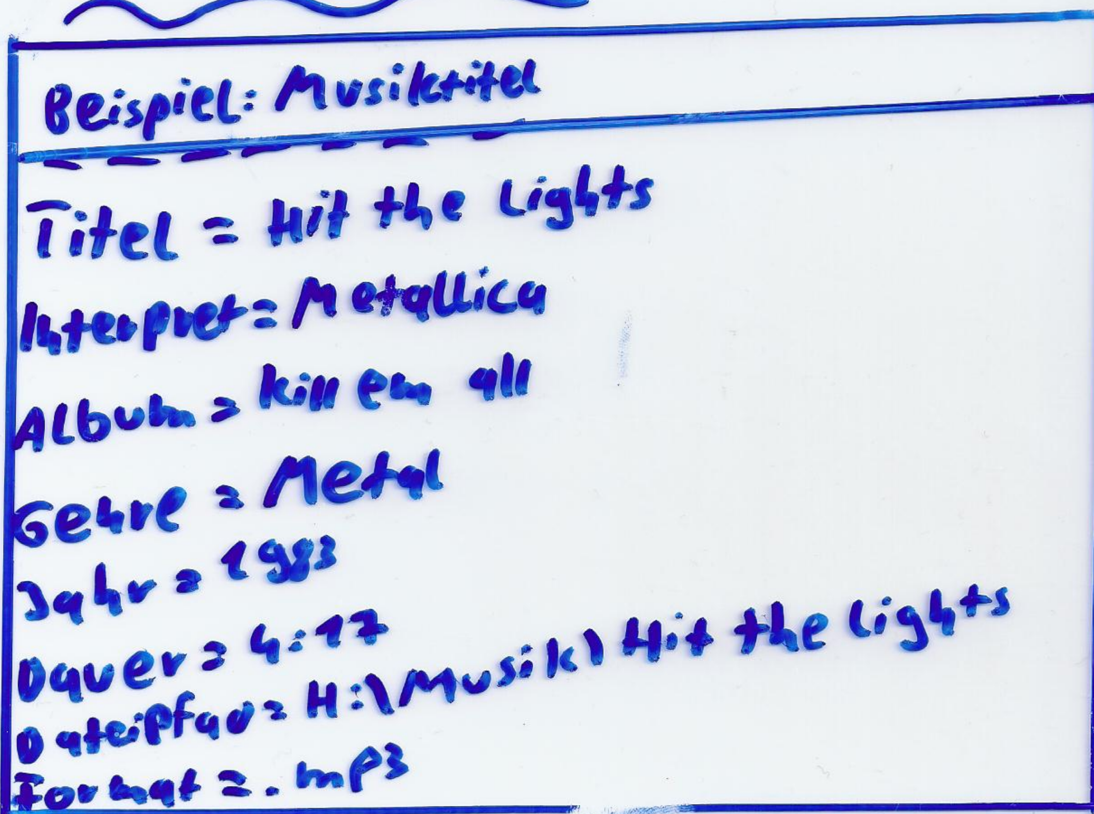
Objektdiagramm:



Klassendiagramm



Objektdiagramm



Klassendiagramm

Sendung
- Sender : string - Name : string - Uhrzeit : Uhrzeit - Beschreibung : string - Bewertung : string

Objektdiagramm

Dodo
Sender: RtL Name: Dodo und der Schoko bär Uhrzeit: 20.15 Uhr Beschreibung: Dodo hat einen Bär aus Schokolade Bewertung: 5/5 sterne

4.3 Codepuzzle zur Kollisionsabfrage

Die Schüler erhielten dieses Dokument in elektronischer Form, sodass sich die einzelnen Teile ohne "Bastelarbeit" ordnen ließen.

```
// Attribute in temporäre Variablen
// kopieren – spart Schreibarbeit...
```

```
int b1 = spieler1.getBreite();
```

```
.getX();
```

```
ball.getRadius();
```

```
.getY();
```

```
int h1 = spieler1.getHoehe();
```

```
int bal1Y =
```

```
ball.getY();
```

```
int sx1 = spieler1
```

```
ball.getX();
```

```
int ballX =
```

```
int ballR =
```

```
int sy1 = spieler1
```

```
// Die eigentliche Kollisionsprüfung
```

```
ballX > sx1 - ballR
```

```
ballY >
```

```
{
```

```
&&
```

```
sy1
```

```
)
```

```
ball.abprallenY();
```

```
&&
```

```
}
```

```
&&
```

```
if (
```

```
+ h1
```

```
ballY < sy1 + h1 + ballR
```

```
ballX <
```

```
sx1 + b1 + ballR
```

4.4 Klausur

Dies ist die Klausur, die zwei Wochen nach Abschluss der hier dokumentierten Unterrichtseinheit geschrieben wurde. Wie bereits im Textteil erwähnt, liegt der Schwerpunkt auf der imperativen Programmierung, die Aufgaben 7 bis 9 beziehen sich jedoch auf die objektorientierte Programmierung.

1. Aufgabe: 2 P

   (a) Gib Java-Code an, der eine Variable mit dem Namen 'wert' erzeugt, in der ganzzahlige Werte gespeichert werden können.
   (b) Warum würde die anschließende Zuweisung `wert = 42.5;` zu einem Compilerfehler führen?

2. Aufgabe: 5 P
   Gegeben ist folgender Ausschnitt aus einem Programm:

```
1  int x;
2  x = 3;
3  int y = x;
4  x = x + 1;
```

   (a) Erläutere die Bedeutung **jeder** Zeile des Programmausschnitts.
   (b) Gib die Werte an, die x und y haben, nachdem obiger Code ausgeführt wurde.

3. Aufgabe: 6 P
   Gegeben ist folgender Codeausschnitt, der einige Lücken (gekennzeichnet durch ???) aufweist:

```
int i = ???;
while ( i < ??? )
{
        schreib( "" + i );
        ???;
}
```

   (a) Ergänze den Code, sodass die Schleife 10 mal durchlaufen wird. Gib dazu **hinter** jeder Zeile an, durch was die ??? ersetzt werden sollen.
   (b) Gib eine zu dieser while-Schleife äquivalente for-Schleife an.

4. Aufgabe: 5 P
   Der folgende Programmcode wurde unter leichtfertiger Missachtung der Formatierungsratschläge von Herrn K. geschrieben.

```
1  void main()
2  {
3          int x = 0;
4  while ( x < 5 )
5          {
6          x++;
7                  if ( x == 3 )
8                  {
9  schreib( "Drei" );
10
11         } schreib( "" + x );
12
13 }
14
15 }
```

   (a) In welcher Zeile endet die Schleifenrumpf der while-Schleife?
   (b) Was gibt das Programm aus?

5. Aufgabe: 4 P
   Schreibe eine Methode `int fakultaet( int n )`, die die Fakultät der Zahl n berechnet und zurückgibt. Zur Erinnerung: Die Fakultätsfunktion ist definiert als $n! = 1 \cdot 2 \cdot \ldots \cdot n$. Außerdem gilt $0! = 1$.
   Der Fall, dass die Methode mit einem negativen Parameter aufgerufen wird, muss nicht berücksichtigt werden.

6. Aufgabe: 3 P
   Gegeben ist das folgende Hamster-Programm. Der Hamster befindet sich zunächst an der im Screenshot angegebenen Stelle.

```
void main()
{
        while ( vornFrei() )
        {
                while ( kornDa() )
                {
                        nimm();
                        linksUm();
                }
                vor();
        }
}
```

   (a) Zeichne in den Screenshot den Pfad ein, den der Hamster zurücklegt, wenn obiges Programm ausgeführt wird.
   (b) Kennzeichne durch einen Pfeil auf dem "Zielfeld" die Blickrichtung des Hamsters, nachdem das Programm zu Ende gelaufen ist.

7. Aufgabe: 4 P
   Ein Computerprogramm soll Bücher verwalten, jedes Buch soll dabei ein Objekt der Klasse "Buch" sein.

   (a) Gib vier (sinnvolle) Attribute (einschließlich Datentyp) an, über die die Klasse "Buch" verfügen könnte. Verwende für Zeichenketten/Text den Datentyp "String".
   (b) Zeichne ein Klassendiagramm der Klasse Buch sowie ein konkretes Objektdiagramm. Es müssen keine Methoden eingezeichnet werden!

8. Aufgabe: 4 P

   (a) Was bedeuten die Schlüsselwörter public und private?
   (b) Welches Grundkonzept der Objektorientierten Programmierung lässt sich damit umsetzen? Erläutere dieses Prinzip in wenigen Worten.

9. Aufgabe: 6 P
   Gegeben ist die Implementierung einer Klasse "Rechteck"

```
class Rechteck
{
        private double a;
        private double b;
}
```

   (a) Implementiere einen Konstruktor, der zwei Parameter erwartet und die übergebenen Werte den Attributen a und b zuweist.
   (b) Implementiere eine Methode "berechneFlaeche", die den Flächeninhalt des Rechtecks berechnet und zurückgibt.
   (c) Gib Java-Code an, der ein Rechteck-Objekt mit den Seitenlängen 2.5 und 3.14 erzeugt und die Methode "berechneFlaeche" des erzeugten Objekts aufruft. Das Ergebnis soll in einer Variable von geeignetem Typ gespeichert werden.

10. Aufgabe: 1 P
    Vergewissere Dich, dass Du auf jedes Blatt (einschließlich dem Aufgabenblatt) Deinen Namen geschrieben hast.

Ges: 40 P

4.5 Evaluationsbogen

Der folgende Bogen wurde an die Schüler des Kurses ausgeteilt, um Rückmeldungen über die unterrichtete Einheit zu erhalten.

Evaluation der Unterrichtseinheit zur OOP

Bewerte bitte die einzelnen Punkte auf einer Skala von -2 (trifft nicht zu) bis +2 (trifft voll zu).

	-2	**-1**	**0**	**+1**	**+2**
Das Ziel, ein eigens Spiel zu entwickeln, war motivierend.					
Die einführende Spielanalyse war geeignet, das Grundkonzept der Objektorientierung zu verdeutlichen.					
Ich habe verstanden, was ein Objekt ist und woraus es besteht.					
Ich habe den Unterschied zwischen einem Objekt und einer Klasse verstanden.					
Ich habe verstanden, was es mit Kapselung auf sich hat.					
Ich weiß, wie man eine Klasse in Java implementiert.					
Das Implementieren/Vervollständigen der Klassen „Ball“ und „Schlaeger“ war vom Schwierigkeitsgrad angemessen.					
Die einzelnen Teile des Spielprojekts zu überblicken fiel mir leicht.					
Ich fand den vorgegebenen Coderahmen verwirrend.					
Ich habe das Grundprinzip eines Konstruktors verstanden.					
Ich weiß, wie man einen Konstruktor implementiert.					
Ich habe das Grundkonzept der Vererbung verstanden.					
Ich weiß, wie man eine Unterklasse einer Klasse definiert.					
Ich hätte für die Programmieraufgaben mehr Zeit gebraucht.					
Ich hätte mehr/genauere Erklärungen gebraucht.					
Ich habe versucht, nicht Verstandenes zuhause nachzuvollziehen.					
Ich habe das Skript gelesen und fand es hilfreich.					

Sonstige Anmerkungen/Rückmeldungen/Verbesserungsvorschläge:

4.6 Screenshots

In diesem Abschnitt folgen Screenshots der Startseite von allen im Literaturverzeichnis genannten Webseiten. Aufnahmedatum aller Screenshots war der 2. Januar 2012.

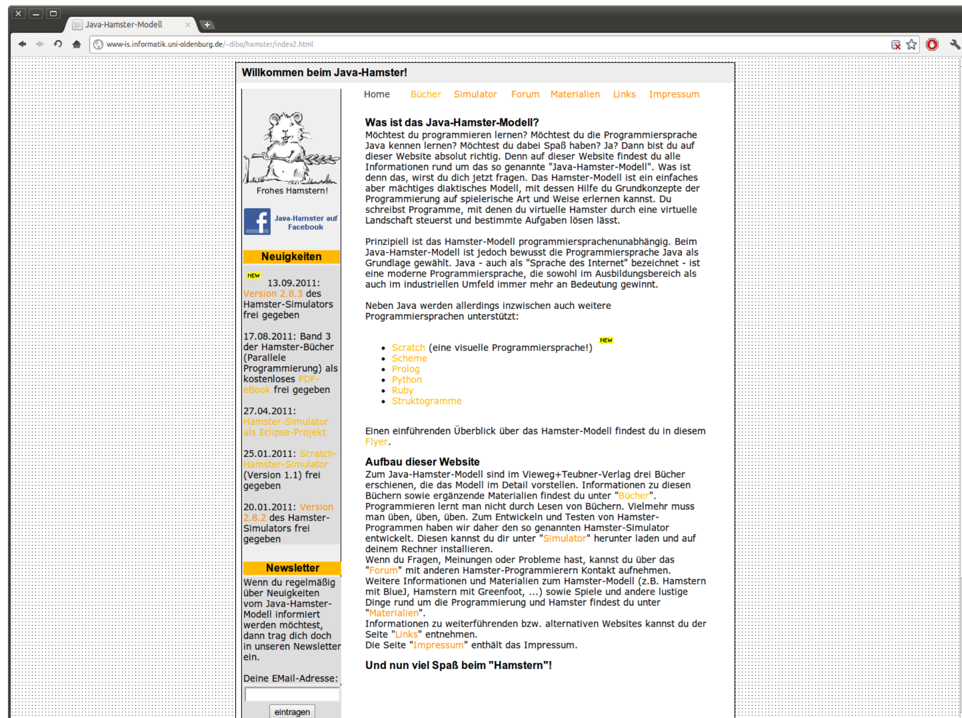


Abbildung 4.1: Hamstersimulator: `http://www-is.informatik.uni-oldenburg.de/~dibo/hamster/index2.html`

Abbildung 4.2: Greenfoot: `http://www.greenfoot.org`

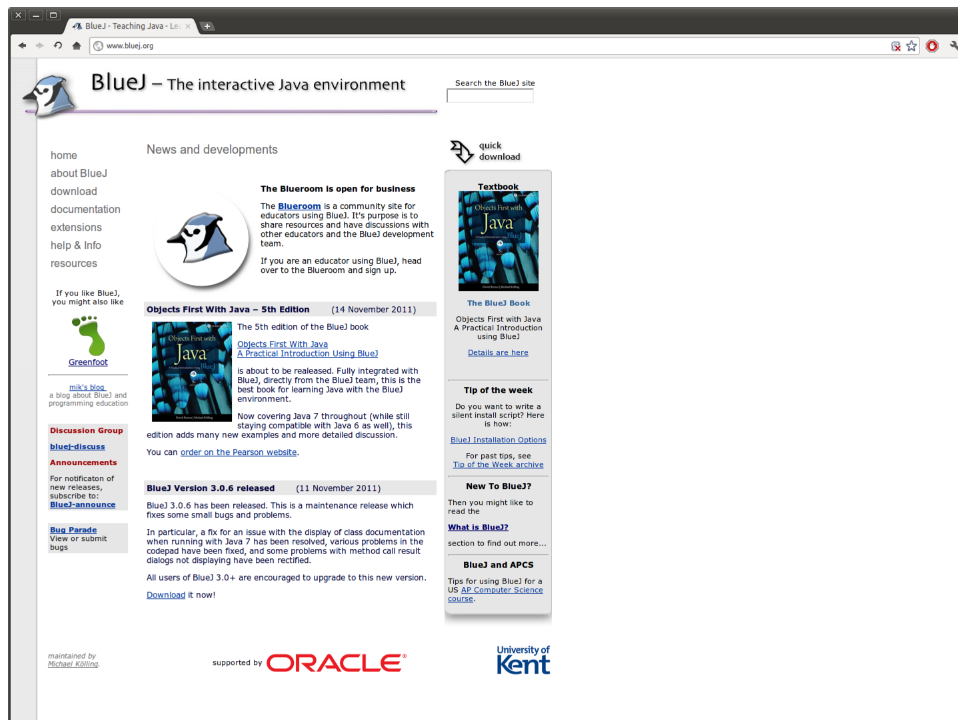


Abbildung 4.3: BlueJ: `http://www.bluej.org`

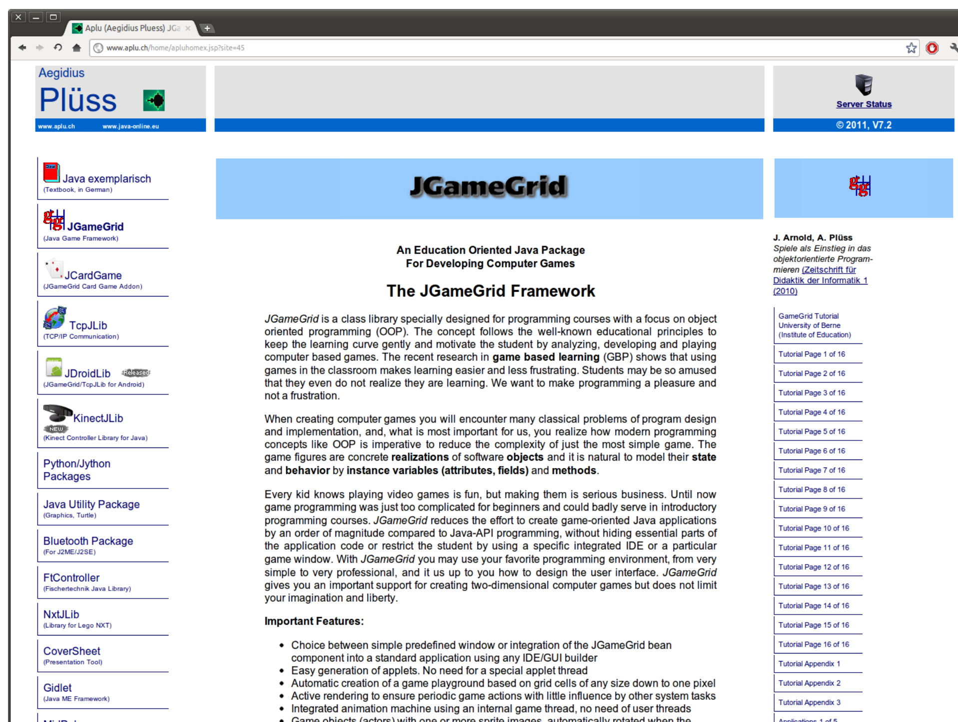


Abbildung 4.4: JGameGrid: `http://www.aplu.ch/home/apluhomex.jsp?site=45`

TIOBE Software: Tiobe Ind

www.tiobe.com/index.php/content/paperinfo/tpci/index.html

© 2012 TIOBE Software BV | Privacy Statement

TIOBE SOFTWARE THE CODING STANDARDS COMPANY

code. TIOBE code.

Home Company Products TICS Paper & Info

Paper & Info > Tiobe Index

TIOBE Programming Community Index for December 2011

December Headline: C++ about to be dethroned by C#

Since the beginning of the TIOBE index back in 2001, the programming language C++ has been number 3 of the chart in a very consistent way. Perl, Visual Basic and PHP have been number 3 too, but these languages could keep this position only for a few months. Now C# is knocking on the door. It will certainly be a tough battle again. C# is Microsoft's most active and evolving programming language, whereas Microsoft recently announced to revive C++ in favor of... C#. We will see what will happen the next few months.

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. The popular search engines Google, Bing, Yahoo!, Wikipedia, YouTube and Baidu are used to calculate the ratings. Observe that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Position Dec 2011	Position Dec 2010	Delta in Position	Programming Language	Ratings Dec 2011	Delta Dec 2010	Status
1	1	=	Java	17.561%	-0.44%	A
2	2	=	C	17.057%	+0.98%	A
3	3	=	C++	8.252%	-0.76%	A
4	5	↑	C#	8.205%	+1.52%	A
5	8	↑↑↑	Objective-C	6.805%	+3.56%	A
6	4	↓↓	PHP	6.001%	-1.51%	A
7	7	=	(Visual) Basic	4.757%	-0.36%	A
8	6	↓↓	Python	3.492%	-2.99%	A
9	9	=	Perl	2.472%	+0.14%	A
10	12	↑↑	JavaScript	2.199%	+0.69%	A
11	11	=	Ruby	1.494%	-0.29%	A
12	10	↓↓	Delphi/Object Pascal	1.245%	-0.93%	A
13	13	=	Lisp	1.175%	+0.11%	A
14	23	↑↑↑↑↑↑↑↑↑	PL/SQL	0.803%	+0.24%	A
15	14	↓	Transact-SQL	0.746%	-0.03%	A

Search

Mail a

Sitemap

Contact

Distributor

Abbildung 4.5: TIOBE: `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`