

Vorbemerkung: Der Kurstext wurde im WS 06/07 neu geschrieben. Die vorher verfassten Prüfungsprotokolle und Klausuren taugen daher nur sehr eingeschränkt zur Prüfungsvorbereitung. Außerdem wurden verschiedene Passagen auch seit dieser Neufassung umgeschrieben, weshalb man sich auch auf die nachfolgenden Protokolle nicht zu 100% verlassen kann; man sollte sich zur Vorbereitung daher im Zweifel auf die neueren Fassungen verlassen. Die zwei Aufsätze, die in früheren Versionen zum Skript zusätzlich Pflichtlektüre waren, sind nun auch in den Kurstext eingearbeitet. Die Themen der Aufsätze (verhaltensbasiertes Subtyping, Alias Protection) sind aber nach wie vor zentrale Diskussionspunkte in vielen Prüfungen.

Für die mündliche Prüfung sollte man sich auf vergleichsweise sehr offene Fragen gefasst machen. Das hat mich während der Prüfung ein wenig nervös gemacht, da ich den Blicken nicht entnehmen konnte, ob ich denn auf das Richtige hinaus gehe. In meiner Note hat sich diese vereinzelt Unsicherheit aber nicht widerspiegelt. Es ist aber in jedem Fall angebracht, zu jedem der zentralen Themen frei erzählen zu können.

Die Prüfung beginnt dazu wohl immer mit der Frage: 'Was ist objektorientierte Programmierung?'. Dabei ist einem Detailgrad und Richtung der Antwort sehr offen gelassen. Wer hier ausschweifend antwortet, hat mit Glück einen Großteil der Prüfung bereits hinter sich; eine entsprechend gute Einübung eines Exposés zu dieser Antwort lohnt sich also. Die Antwort sollte sich in jedem Fall vordergründig auf das unterliegende Weltbild beziehen, so wie es in KE0 – dem Vorwort – geschildert wird. (Was mich ein wenig wundert, ist dass überspezifische Antworten anderer Gedächtnisprotokolle, die sich z.B. auf Klassen, Subtypenbeziehung und das dynamisches Binden konzentrieren, auch als richtig gewertet werden. Meines Erachtens sind dies zwar typische, aber keinesfalls notwendige Eigenschaften objektorientierter Programmierung. C++ kann bspw. mit komplett statischem Binden genutzt werden, Smalltalk hat überhaupt kein Typsystem. Bei der ergänzenden Darstellung solcher Aspekte wäre ich in meiner genauen Wortwahl deshalb vorsichtig.)

Die Fragen meiner Prüfung:

- **Was ist objektorientierte Programmierung?**
Hier habe ich betont, dass objektorientierte Programmierung eine Art zu Programmieren ist, die sich auf das Weltbild der Objektorientierung bezieht (also ein Paradigma). Das Weltbild habe ich wie im Vorkurs beschrieben erklärt, als bestehend aus Entitäten, die wir Objekte nennen mit einer individuellen Identität und einem Zustand, die zueinander in Verbindung stehen können und die sich aneinander Nachrichten schicken, wodurch sie (alleinig) ihren Zustand ändern können. Dadurch entsteht ein Objektgeflecht, das in der objektorientierten Programmierung die Bedeutung des Programmes charakterisiert. Als alternative Anwendung des objektorientierten Weltbildes habe ich die Softwarekonzeption (mittels bspw. UML) und die mathematische Spieltheorie genannt.
- **Was verstehen Sie unter Vererbung?**
Einen Mechanismus zur Mehrfachnutzung von Implementierungen zwischen Klassen. (Klassenbasierte Objektorientierung kurz erläutert.)
- **Warum war man „damals“ so begeistert von dieser Möglichkeit?**
Weil es die Wiederverwendung von Software stark fördert.
- **Was ist aber der Nachteil?**
Die starke Koppelung zwischen Basisklasse und erbender Klasse, die den Entwicklern oft nicht mal bewusst ist, insbesondere falls die Implementierung der Basisklasse dem Weiterentwickler unbekannt ist: Entwickler stellt oft implizite Anforderungen an das Verhalten der Basisklasse, die der Entwickler dieser Basisklasse in späteren Versionen nicht mehr einhält; insbesondere gefährlich im Zusammenhang mit dem dynamischen Binden. Als

- Fragile-Base-Class-Problem benannt.
- **Was für eine andere Möglichkeit der Strukturierung von Klassen gibt es?**
Generalisierung/Spezialisierung als inhaltliche ausgerichtete Hierarchisierung in Bezug auf Intension und Extension der Klassen, wobei deren Bedeutung eine Rolle spielt und die Implementierung nur im Hintergrund steht.
 - **Was der beiden Möglichkeiten ist dann die bessere Art der Hierarchisierung?**
Kommt darauf an; wenn ich Klassen intern nutzen will, um mir Schreibarbeit zu ersparen, aber die Vererbung nach Außen verbergen will, dann ist reine Vererbung durchaus eine legitime Möglichkeit; z.B. das private Vererben in C++. Grundsätzlich ist Generalisierung zur Programmgliederung vor allem nach Außen aber die „richtigere“ Variante, insbesondere, da sich bei reiner Übertragung von Implementierungen Klassen inhaltlich schnell „auseinander entwickeln“ können.
 - **Was ist ein anderer Grund?**
Hier wusste ich nicht, worauf der Prüfer hinaus wollte und fragte ihn danach.
 - **Stichwort: Typen.**
Erläutert, dass Klassen gerne als Typen genutzt werden und diese zwei inhaltlich verschiedenen Konzepte mit einer ähnlichen äußeren Form nur konform sind, wenn die Klassenhierarchie als Generalisierungshierarchie gegliedert ist. In dem Zusammenhang auch das Subtyping erläutert.
 - **Was ist parametrischer Polymorphismus?**
Konzept mit Typvariablen erläutert.
 - **Was ist beschränkter parametrischer Polymorphismus?**
Problem der Unsicherheit innerhalb der Typendefinition bezüglich der Variablen erläutert. Obere Typschranke wird gesetzt, um minimale Invariante festzulegen.
 - **Wie funktioniert Subtyping im parametrischem Polymorphismus?**
Kommt auf die Sprache an: In Java nur mit Wildcards, in Scala zum Beispiel aber ko- oder kontravariant in den Typen der Typparameter, in Eiffel kovariant. (Scala nicht im Kurstext sondern aus Eigenerfahrung.)
 - **Was sind in Java diese Wildcards?**
Erlauben ein Subtyping bei zur Übersetzungszeit unbekanntem Typparametern. Repräsentieren dazu einen abstrakten Datentypen, der im Kontrast zu parametrischen Typen ohne Wildcard als Supertyp von parametrischen Objekten auftreten kann. Das aber mit der Einschränkung, dass nur lesend (Rückgabety) oder schreibender Zugriff (Parametertypen) gleichzeitig ermöglicht werden, da nur entweder ein „Minimaltyp“ oder ein „Maximaltyp“ festgelegt sein können.
 - **Gibt es in C++ parametrischen Polymorphismus?**
Wusste ich nicht genau, aber vermutete ja. Habe dann beigefügt, dass ich nicht in C++ entwickle, sondern in Java und Scala. Herr Steinmann erklärte mir daraufhin, dass es das in C++ tatsächlich eigentlich nicht gibt, der Compiler aber zur Übersetzungszeit die Templates jeweils mit ihrer Belegung auflöst und daraufhin bei Nutzung des Templates an der jeweiligen Programmstelle einen Fehler findet, nicht aber bei dessen eigentlicher Übersetzung.
 - **Wie funktioniert das in Java oder Scala?**
Kurz Typeerasure erläutert.

Abschließend sagte Herr Steinmann, dass er keine Notwendigkeit mehr sieht mich weiter zu fragen, da ich mich offensichtlich gut vorbereitet habe. Nach der Notenverkündung habe ich mich noch eine gute Weile mit Herrn Steinmann über verschiedene Themen der Informatik unterhalten, wobei er mir im Verlauf auch prompt ein Thema für die Masterarbeit angeboten hat. (Ich hatte bereits eines, gefreut hat es mich trotzdem.)

Ich kann den Prüfer empfehlen, möchte aber noch ein paar kritische Worte zum Kurs verlieren. Die

positiven Aspekte kommen hier natürlich zu kurz, was nicht heißen soll, dass es diese nicht gibt. Der Kurs hat mir viel gebracht, aber gelobt wurde auch in den anderen Prüfungsprotokollen viel. Hier daher ein paar Aspekte, die für die Entscheidung zur Belegung vielleicht eine Rolle spielen:

1. Der Kurstext ist sehr prosaisch formuliert und manchmal musste ich als deutscher Muttersprachler tatsächlich sehr lange Sätze umschreiben, um sie zu verstehen. Den Stil empfand ich zu Beginn zwar sehr erfrischend, aber bald darauf begann ich mir einen prägnanteren Ausdruck zu wünschen. Das fand ich im Verlauf des Semesters ein wenig frustrierend. Zentrale Begriffe und Erkenntnisse sollten daher im Idealfall künftig kenntlicher angezeigt werden, da diese wegen des prosaischen Stils schnell zwischen den wortgewaltigen Erklärungen verschwinden.
2. Die Struktur des Textes könnte teils auch geordneter sein, insbesondere in KE3 fand ich es stellenweise schwierig nachzuvollziehen, welche Begriffe nun logisch zusammenhängen, da Konzepte mitunter auch verstreut über verschiedene Unterabschnitte gemischt mit anderen Konzepten eingeführt werden. Als symptomatisches Beispiel möchte ich die Reihenfolge der Erläuterung von Typeinschränkung/-erweiterung und Zuweisungskompatibilität/-konformität erwähnen. Das liegt zwar so gesehen auf der Hand, geht aus dem Text aber nicht eindeutig und offensichtlich hervor. Ähnliches gilt für die Zusammengehörigkeit von Klassifikation/Instanzierung und Generalisierung/Spezialisierung. Auch das hätte ich eigentlich sogar bereits vor Bearbeiten des Kurses so in Zusammenhang gebracht; wurde in diesem Wissen durch den Kurstext aber eher wieder verunsichert, da dort die klare Ankündigung dieser Zusammenhänge und die wechselseitige Ankündigung nie vor Augen gehalten wurde.
3. Die „geschlechtsneutrale Sprache“, die sich grundsätzlich auf die weibliche Form bezieht („Einer Leserin fällt sicher auf, dass (...) Daher kann sie sehen, dass (...)“). Jedes Mal musste ich wieder kurz rätseln, wer denn nun mit „sie“ gemeint sein soll. Das ist sicher nett gemeint, reißt aber regelmäßig aus dem Lesefluss. Da würde ich das pädagogische Ziel gerne voranstellen.
4. Die Längen der Kurseinheiten sind leider unausgeglichen. Die Seitenzahl nimmt von Kapitel zu Kapitel ab; die Inhaltsdichte ist in KE2, KE3 und KE6 aber sicher am höchsten. Das macht auch das konstante Mitlernen über das Semester schwieriger, da man sich alle 14 Tage erneut auf den zu leistenden Aufwand einstellen muss.
5. Die Übungsaufgaben und eigentlich auch der Kurstext versteifen sich zu oft auf eigentlich irrelevante Detailspekte von Smalltalk oder Java, die mit dem eigentlichen Kursziel – dem Lehren von objektorientierten Konzepten – eigentlich nichts weiter zu tun haben. Das könnte man meines Erachtens gerne zurückfahren, um stattdessen den eigentlichen Inhalten mehr Raum zu bieten. Gerade die Einsendeaufgaben taugen auf Grund dessen leider eigentlich nicht zur Vorbereitung auf die mündliche Prüfung (mehr für die schriftliche), da deren Schwierigkeit eigentlich immer Implementierungsdetails betreffen. Die Aufarbeitung der theoretischen Konzepte kommt dort hingegen leider oft zu kurz.
6. Gleichzeitig hat mir eine Diskussion von wichtigen Punkten gefehlt, über die ich lieber gelesen hätte statt über Java und Smalltalk wie z.B.: Ein Ausblick zur Integration funktionaler Programmierung in objektorientierte Sprachen, wie viele moderne Sprachen die künftige Entwicklung bereits andeuten (in KE6 zumindest kurz angesprochen, gerade weil diese Closures ja eigentlich über ihren „home context“ Methoden abbilden, wie an Smalltalk eigentlich schön veranschaulicht; meine Empfehlung ist auch: Java zusammenkürzen, Scala ins Skript!); eine Diskussion von Vererbungsstrategien und Zwischenlösungen zu Einfach- und Mehrfachvererbung wie Traits und Mixins (das Fehler dieser Konzepte ist meiner Meinung nach wichtigster Grund zum „Missbrauch“ der Vererbung: Wieder empfehle ich Scala!); verschiedene Strategien zur Verwaltung von Objekten in der Garbage Collection, vielleicht auch im Kontrast zum expliziten Speichermanagement; genauere Diskussion von parametrischem Polymorphismus und den unterliegenden Strategien mit Vor- und Nachteilen wie Type Expansion, Type Erasure oder der Template-Ansatz von C++.

Gedächtnisprotokoll: Kurs 1814 Objektorientierte Programmierung WS11/12

Datum: 09.03.2012

Note: 1.3

Prüfer: Prof. Dr. Steimann; Beisitzerin Fr. Dr. Keller

Vorbereitung: Skript gelesen, alle Übungsaufgaben bearbeitet und eingeschickt. Zu jedem Kapitel die Wichtigsten Definitionen und Sachverhalte stichpunktartig herausgearbeitet und diese mit den bereits existierenden Prüfungsprotokolle abgeglichen.

Prüfung: freundliche Atmosphäre; Prüfung hatte normalen Gesprächscharakter.

Was ist objektorientierte Programmierung?

Was unterscheidet SMALLTALK von Java?

Sind Klassen auch Objekte?

Welche Beziehungen gibt es zwischen Objekten?

Wie werden Beziehungen ausgedrückt?

Welche Arten von Beziehungen gibt es?

Wie wird die Komposition in den Programmiersprachen C++, C# und Eiffel umgesetzt?

Was ist ein Typ?

Was heißt typisiert?

Welche Gründe gibt es für Typisierung?

Was sind statische und dynamische Typprüfungen?

Was ist Zuweisungskompatibilität?

Wo können Laufzeitfehler in Java auftreten?

Was sind und wozu dienen der einfache und beschränkte parametrische

Polymorphismus?

Was sind Wildcards und wozu werden diese benötigt?

Was ist Substituierbarkeit und welche Bedingungen werden daran geknüpft?

Fazit: Uneingeschränkte Empfehlung für den Prüfer. Man sollte die Stoffmenge nicht unterschätzen und sich genügend Freiraum schaffen (mind. 1 Woche Vorbereitungszeit einplanen)

Prüfungsprotokoll zur mündlichen Fachprüfung

Kurs: 01814 - Objektorientierte Programmierung

Prüfer: Prof Dr. F. Steimann

Beisitzer: Dr. Keller

Datum: 09.03.2012

Note: 1,0

Herr Prof. Steimann fragte: Was ist Objektorientierte Programmierung?

Ich bot an mit einer Abgrenzung zur prozeduralen Programmierung zu beginnen.

Folgende Themen wurden dann von mir in einer Art Vortrag erläutert:

- Objekte
- Identität vs. Gleichheit
- Aliasing
- Wertsemantik vs. Verweis- oder Referenzsemantik
- Call by value vs. Call by reference
- Klassen
- Vererbung
- Abstrakte Klassen
- Typen
- Typen vs. Klassen
- Subtyping
- Statische vs. Dynamische Bindung

Hiernach fragte Herr Prof. Steimann nach Polymorphie, und ich erläuterte folgende Punkte

- Polymorphie
- parametrischer Polymorphismus
- beschränkter parametrischer Polymorphismus

Hierauf wurde nach WildCardTypen gefragt, und ich erläuterte Wildcards mit oberer und unterer Schranke, sowie mit beiden.

Hiernach sprach ich noch einige Probleme der OOP an

1. Das Problem der Substituierbarkeit (Liskov-Substitutions-Problem)
Hier wurde konkreter nach Kontra- und Kovarianz gefragt
2. Das Fragile-base-class-Problem
3. Das Problem der schlechten Tracebarkeit
4. Das Problem der eindimensionalen Strukturierung
5. Das Problem der mangelnden Kapselung

Die Prüfung verlief in einer ruhigen, angenehmen Atmosphäre. Herr Prof. Steimann lies mich erzählen und fragte nur an 2 Stellen nach, wahrscheinlich um festzustellen, ob das Problem nur gelernt oder verstanden wurde.

Herr Prof. Steimann ist als Prüfer sehr zu empfehlen (01853 hatte ich auch bei ihm die Fachprüfung).

Als Prüfungsvorbereitung diente der sehr gute Studientag durch Herrn Paap und die von mir erstellte, nachfolgend beigefügte Lernunterlage.

01814 Objektorientierte Programmierung

prozedurale Programmiersprachen

klare Trennung zwischen

- Modellierung von Informationen
- Modellierung der Verarbeitung

OOP

- Analogie zu Gegenständen der materiellen Welt
- Programmausführung als System kooperierender Objekte
- Objekte verfügen über Fähigkeit, auf Nachrichten zu reagieren, durch Zustandsänderung und/oder Auskunft über diesen geben
- *Objekte*
 - haben Zustand (Attribut), Identität und Lebensdauer
 - können Nachrichten verarbeiten, durch Methodenausführung
 - sind selbständige aktive Einheiten, können unabhängig und parallel agieren (Inhärente Parallelität des OO-Paradigmas)
 - bilden Einheit aus Daten und darauf definierten Operationen
 - Schnittstelle beschreibt Interaktionsmöglichkeiten, Katalog der Methoden = Protokoll
 - über Referenzen ansprechbar (mehrere Referenzen auf dasselbe Objekt = Aliase)

Identität vs. Gleichheit

Zwei Objekte können zwar gleich, aber nie dasselbe sein, oder es sind nicht zwei Objekte, sondern eins!

Aliasing

- Mehrere Referenzen auf dasselbe Objekt = *Alias*
- Änderung über einen Alias ändert das Objekt
- Sichtbarkeit typischerweise auf Ebene der Variablen geregelt, nicht auf der der Objekte.
Bsp. Entry-Objekte

Wertsemantik vs. Verweis- oder Referenzsemantik

- *Wertsemantik* – Variable enthält direkt den Wert, z.B. Java Primitivtypen
- *Verweis- oder Referenzsemantik* – Variable enthält Referenz auf Objekt
- Referenzsemantik wegen wachsender Objekte + Subtyping (wie viel Platz?)

Wertsemantik

- Java: Zeichen, Zahlen, boolesche Werte
- SMALLTALK: Zeichen, kleine Ganzzahlen

Call by value vs. Call by reference

Call by value

- reine Eingabeparameter
- formalem Parameter wird der aktuelle Parameter als Wert zugewiesen

Call by reference

- Ein- und Ausgabeparameter
 - formalem Parameter wird Referenz auf aktuellem Parameter als Variable (nicht als Objekt) zugewiesen
-
- SMALLTALK, Java - call by value
 - C++ - call by value + call by reference

Klassen

- Klassendefinition – bildet Art Vorlage für Objekte mit Eigenschaften (Instanzvariablen und Methoden)
- *Intension* – Summe der Merkmale, die die Objekte charakterisieren
- *Extension* (Ausdehnung oder Erstreckung) – Menge der Objekte, die darunterfallen
- *Metaklasse* – beschreibt für jede Klasse, welche Attribute + Methoden sie hat
- SMALLTALK: Klasse selbst ist ein Objekt
- *Klassifikation* – Zuordnung von Objekten zu Klassen
- *Klassenhierarchie* – Hierarchie von Klassen, Standard-Superklasse = Object
- Superklasse ist *Generalisierung* (Abstraktion) ihrer Subklassen
- Subklasse ist *Spezialisierung* ihrer Superklasse
- *Instanziierung* – Vorgang, bei dem ein neues Objekt entsteht
- *Klonen* – Kopieren auf Basis eines bereits bestehenden Objektes

Vererbung

- Mechanismus zur Übertragung von Eigenschaften und Methoden einer Generalisierung auf ihre Spezialisierung
- Überschreiben – bereits bestehende Methode des Supertyps in Subtyp redefiniert
- Überladen – neue Methode mit vorhandenem Methodennamen aber unterschiedlicher Parameterliste

Abstrakte Klassen

- nicht instanziiierbare Klassen
- in der Regel fehlen Teile der Verhaltensspezifikation
- können spezifizieren, dass individuelle Implementierung von best. Verhalten von Subklassen erwartet wird
- *offene Rekursion* – aus implementierter Methode der abstrakten Klasse wird „fehlende“ abstrakte Methode per dynamischer Bindung auf konkretem Subtyp aufgerufen

Typen

- was - primitiver Begriff, vergleichbar Menge in Mengentheorie
- warum - Wunsch nach statischer Typsicherheit und Flexibilität
→ Lösung (mit Einschränkungen): Subtyping mit Auseinanderfallen des statischen / dynamischen Typs
→ Dynamische Methodenwahl, dynamisches Binden
- Intension: Definition des Typs
- Extension: Wertebereich

Typsysteme und Typisierung

- umfasst Typausdrücke, Wertausdrücke, Regeln
- Motivation für Typsysteme – Sicherstellung, dass einem Objekt nur Nachrichten gesendet werden, die es versteht.

Gründe für Typisierung

- Regelt Speicherlayout
- erlaubt effizientere Ausführung eines Programms
- erhöht Lesbarkeit eines Programms
- ermöglicht automatisches Finden von logischen Fehlern

Typäquivalenz und Typkonformität

- Intension – Verhältnis der Typdefinitionen
- Extension – Zusammenhang der Wertebereiche der Typen
- nominal Typäquivalenz:
sich auf den Namen beziehend (Namensäquivalenz)
- strukturelle Typäquivalenz:
erwartet dass Typen paarweise gleich definiert sind (Strukturäquivalenz)

Typprüfung

- statisch: Typkorrektheit soll zur Übersetzungszeit gewährleistet werden (Aufgabe des Compilers)
Nachteil: Weist auch nützliche, sinnvolle und typkorrekte Programme zurück
- dynamisch: Prüfung zur Laufzeit vor einer Variablenzuweisung, ob der zuzuweisende Wert den von der Variable geforderten Typ hat
- SMALLTALK: keine dynamische Typprüfung – Typfehler werden erst im letztmöglichen Moment offenbar, wenn auf einer Variable eine Methode aufgerufen werden soll, die für das Objekt, auf das die Variable verweist, nicht definiert ist

Subtyping

- Subtypenbeziehung = Rolle in einer Beziehung zwischen 2 Typen:
Supertyp ← Subtyp
- Subtyp ist mit seinem Supertyp per Definitionem zuweisungskompatibel
(Ersetzbarkeit liegt Definition von Subtypen zugrunde)
- Wo Werte des Typs eines Supertyps erwartet werden, dürfen Werte oder Objekte des Subtyps auftauchen

Subtyping vs. Subclassing vs. Vererbung vs. Delegation

- Subclassing: Von einer Klasse Ableitung einer spezielleren Klasse
Java: Bei Subclassing immer auch Subtyping und Vererbung (Ausnahme Interface)
- Vererbung: Mechanismus zur Übertragung von Eigenschaften und Methoden einer Generalisierung auf ihre Spezialisierung
- Subtyping: „Ist-ein-Beziehung“ zwischen Typen
- Delegation: Weg, bestehenden Code wiederzuverwenden, ohne Subtypenbeziehung zu begründen. Zu erledigende Aufgabe wird an anderes Objekt (delegate) durchgereicht.
Konzeptionell ist Vererbung spezieller Fall von Delegation (vom Subtyp an Supertyp)

Statische vs. Dynamische Bindung

Statische Bindung: Zum Übersetzungszeitpunkt definiert, welcher Code als Folge eines Prozeduraufrufs ausgeführt wird
→ Normalfall prozedurale Programmierung

Dynamische Bindung: Durch Auswertung eines Nachrichtenausdrucks erst zur Laufzeit wird Empfängerobjekt für Methodenaufruf festgestellt
→ Normalfall objektorientierte Programmierung

Polymorphie oder Polymorphismus

- ermöglicht Konstrukte so zu programmieren, dass sie mit verschiedenen Typen arbeiten können

Subtyp-Polymorphie (auch: Inklusionspolymorphie)

- Im wesentlichen dasselbe wie Subtyping (gebräuchlich als Abgrenzung zur parametrischen Polymorphie)
- Wo Objekte eines Typs erwartet werden, können Objekte eines anderen Typs stehen, weil der erste Typ den anderen subsumiert (inkludiert)
- ergibt sich aus der Subtypbeziehung (Substituierbarkeit)

parametrischer Polymorphismus

- Annotation mit Typparametern
- Elemente des betreffenden Typs sind erlaubt (bei gleichzeitiger Subtyppolymorphie auch Elemente der Subtypen)
- Bei Erzeugung eines Exemplars wird Typ festgelegt

beschränkter parametrischer Polymorphismus

- wie parametrische Polymorphie, jedoch Einschränkung im voraus durch Festlegung einer oder mehreren Schranken für den Typparameter (Bsp. Comparable)

Typen vs. Klassen

- Typdefinition vollkommen frei von Implementierungsaspekten
- sind abstrakte Spezifikationen
- schränken Wertebereich von Variablen ein
- geben das Protokoll von Objekten an

- Klasse legt Implementierung ihrer Objekte fest
- Subklassenbeziehung auf parallele Subtypenbeziehung nur übertragbar, da die meisten OOP Löschen oder inkompatible Redefinition von Methoden verbieten.

- Zu parametrischen Klassendefinitionen gehören beliebig viele Typen
- Symbiose zweier verschiedener Konzepte, die unterschiedlichen Zwecken dienen, deren strukturelle Ähnlichkeit sich aber durch eine syntaktische Zusammenlegung nutzen lässt

WildcardTypen

- Wunsch – z.B. für generische Collections Methode *sort* definieren
Parametertyp `ArrayList<Comparable>` - geht nicht!
- Integer und String Subtypen von Comparable, aber `ArrayList<Integer>` und `ArrayList<String>` nicht Subtyp von `ArrayList<Comparable>`

Lösung: Wildcards als Platzhalter, z.B. `List<?>`, definitionsgemäß Supertyp von `List<T>` (also auch `ArrayList<Integer>` und `ArrayList<String>`)

speziellerer Typ – Lösung: Beschränkte Wildcards – `List<? extends Comparable>`

- nach oben beschränkte Wildcards:
`extends y` – alles „Super y“ raus (lesender Zugriff erlaubt)
- nach unten beschränkte Wildcards:
`super x` – alles „Sub x“ rein (schreibender Zugriff erlaubt)

Bsp.: `Tier ← Vogel ← Spatz`
`<? super Vogel extends Tier>`
 → Vogel + Spatz rein
 → Tier raus

Probleme der OOP

6. Das Problem der Substituierbarkeit (Liskov-Substitutions-Problem)
7. Das Fragile-base-class-Problem
8. Das Problem der schlechten Tracebarkeit
9. Das Problem der eindimensionalen Strukturierung
10. Das Problem der mangelnden Kapselung
11. Das Problem der mangelnden Skalierbarkeit
12. Das Problem der mangelnden Eignung

Prüfung: Objektorientierte Programmierung (Modul 1814)

Prüfungsprotokoll: Jan 2012

Prüfer: Prof. Steimann

Beisitzerin: Dr. Keller

Note: 1,0

- wie schon oft in anderen Prüfungsprotokollen erwähnt, begann die Prüfung mit der Frage: „Was verstehen Sie unter objektorientierter Programmierung?“

→ darauf hin begann ich über die folgenden Punkte zu referieren:

- Weltbild der OOP (Objekte kapseln Daten/Funktionen, haben eine Identität, schicken einander Nachrichten, haben Beziehungen, entstehen und vergehen)
 - Variablen (Wertsemantik/Referenzsemantik, Aliase)
 - Ausdrücke (Zuweisungsausdrücke, Nachrichtenausdrücke)
 - dynamische Bindung und Abgrenzung zur prozeduralen Programmierung (Unterprogrammaufruf mit Verzweigung)
 - Objekterzeugung (prototypisch vs. klassenbasiert)
 - Klassifikation (Ist-Ein-Abstraktionsbeziehung zwischen Element und Allgemeinbegriff, Verallgemeinerung und Vereinfachung in der OOP, Elementsein-Beziehung)
 - Generalisierung (Ist-Ein-Abstraktionsbeziehung zwischen Allgemeinbegriffen, Teilmengenbeziehung)
 - Vererbung als Teilen der Klassendefinition und Mechanismus um Generalisierung umzusetzen (Ansichten herausgearbeitet: Generalisierung vs. Vererbung)
 - Polymorphismus (Vielgestaltigkeit, 5 Gründe der Typisierung, Idee des Subtyping)
 - vom Subtyping und dessen Probleme zur Substituierbarkeit übergeleitet und das LSP (Liskovsches Substitutionsprinzip) erläutert (Fokus auf Schwächen)
- anschließend sollte das Problem mit Aliasing erläutert werden (Problem der mangelnden Kapselung)

Die Prüfung verlief sehr angenehm. Wenn man sich mit den Themen intensiv auseinander setzt, hat man nichts zu befürchten. Den überwiegenden Teil der Prüfung habe ich mit meinen Auslegungen selbst bestimmen können. Dabei habe ich mich auf die oben erwähnten Themen konzentriert (dies sind auch die oft angesprochenen Prüfungsschwerpunkte auf dem Studientag und in anderen Prüfungsprotokollen gewesen).

Den Kurs kann ich sehr empfehlen, wenn man sich mit den Konzepten der OOP auseinander setzen möchten. Auch die Vorstellung zu den Sprachen C#, Eiffel und C++ sind sehr informativ. Der Kurs versetzt einen in die Lage auch weitere Programmiersprachen anhand der beschriebenen Kriterien bewerten zu können.

Gedächtnisprotokoll zum Kurs Objektorientierte Programmierung
(1814)

in Hagen im März 2011

Zur Vorbereitung auf die mündliche Prüfung habe ich an der angebotenen Klausur zu diesem Kurs teilgenommen, dies hat mir meine vorhandenen Lücken sehr deutlich aufgezeigt. Für die mündliche Prüfung selbst habe ich einen Vortrag zum Einstieg sowie jeweils einen Kurzvortrag über die Kernpunkte zu allen Kapiteln ausgearbeitet.

Da ich in anderen Protokollen gelesen hatte, dass Herr Prof. Steimann gerne mit der Frage "Was bedeutet objektorientierte Programmierung?" einsteigt hatte ich meinen Einstieg auf diese Frage ausgerichtet. Hierzu habe ich die Definition aus dem Vorwort zum Kurs genommen,

Objekte die eine Identität haben senden einander Nachrichten. Auf den Erhalt einer Nachricht ändern sie ihren Zustand, die Anweisungen dazu sind in Methoden hinterlegt.

Welche Nachrichten ein Objekt versteht zählt zu seinen Eigenschaften Objekte können entstehen und wieder vergehen (dynamisches Weltbild).

und die einzelnen Punkte näher erläutert.

Abgerundet wurde der Vortrag mit dem Problem der Substituierbarkeit (mit LSP) und den Problemen der objektorientierten Programmierung.

Nachdem ich meine Vortrag beendet hatte, stellte mir Herr Prof. Steimann noch zwei Fragen.

Frage: Was ist parametrischer Polymorphismus?

Antwort: Hier habe ich mit dem vorbereiteten Vortrag zu diesem Thema antworten können.

- Idee, aus einer Typdefinition durch Parametrisierung viele zu machen
- Collections als Standardanwendungsfall
- Parametrischer Polymorphismus und Inklusionspolymorphie
- Beschränkter parametrischer Polymorphismus (wann lesender, wann schreibender Zugriff)

Frage: Wer profitiert vom parametrischen Polymorphismus?

Antwort: Der Typ selbst.

Fazit: Ich kann sowohl den Kurs als auch den Prüfer uneingeschränkt weiterempfehlen. Mein Eindruck ist, dass Herr Prof. Steimann einen Prüfling in seinen Ausführungen nicht unterbricht, auch wenn diese, wie in meinem Fall, etwas umfassender sind und somit viel Zeit in Anspruch nehmen. Dass nach meinem "Einstiegs-vortrag" Raum für nur noch zwei Fragen war, hat sich nicht negativ auf meine Note, mit der ich sehr zufrieden bin, ausgewirkt.

Datum: 25.03.2011

Note: 1,0

Prüfer: Prof. Steimann

Mal wieder ein etwas aktuelleres Prüfungsprtokoll.

Vorbereitung

Zur Vorbereitung auf den Kurs habe ich während dem Semester im Wesentlichen das Skript durchgearbeitet und die Einsendaufgaben bearbeitet. Bei der letzten ist mir allerdings ein wenig die Zeit davongelaufen, so dass ich sie nicht mehr zur Korrektur einschicken konnte. Die bereitgestellten Musterlösungen sind allerdings so detailliert, dass es sich sehr gut damit lernen lässt.

Weiterhin kann ich den Besuch des angebotenen Studientages nur empfehlen. Idealerweise hat man zuvor das Skript schon einmal durchgelesen.

Vor dem eigentlichen Prüfungstermin habe ich mir noch einmal zwei Wochen Urlaub genommen. Ein paar Tage weniger hätten sicher auch gereicht. In der Zeit habe ich das Skript nochmal detailliert durchgearbeitet und versucht alle zentralen Themen des Kurses (Konzept OOP, Klassen, Typsysteme, Polymorphie und die Unterschiede der einzelnen Programmiersprachen) frei Vortragen zu können.

Prüfung

- Eröffnungsfrage (klassisch): Was ist OOP? Hierzu habe ich die zentralen Konzepte aufgezählt (Objekte als zentrales Element, Abgrenzung zur imperativen Programmierung, Polymorphie, Vererbung, Generalisierung, Kapselung, Klassenbasierte vs. prototypenbasierte Objekterzeugung.
- Was ist der alternative Ansatz zum dynamischen Binden? Meine Antwort: Statisches Binden. Allerdings in Smalltalk praktisch nicht möglich wg. mangelnder Informationen. Notwendige Informationen liefern z.B. Typsystem.
- Was macht ein Typsystem aus?
- Was sind Wildcards? Wie werden diese verwendet?
Hier habe ich etwas weiter ausgeholt (parametrischer Polymorphismus, keine Übertragung der Subtypen Beziehung von Parametertypen auf den parametrisierten Datentyp).

Die Prüfung ist nun schon ein paar Tage her - ich habe sicherlich das ein oder andere Detail vergessen.

Fazit

Die Prüfungsatmosphäre war überraschend angenehm. Insbesondere Frau Dr. Keller gibt sich große Mühe dem Prüfling die Angst zu nehmen. Prof. Steimann gibt die Möglichkeit zu ausführlichen Antworten, unterbrach mich allerdings 1-2 mal um zu einem anderen Thema zu springen mit Verweis auf die begrenzten Prüfungszeit. Das hat mich zu Beginn ein wenig verunsichert.

Prüfungsprotokoll zur mündlichen Prüfung von:

Kurs 01814 - Objektorientierte Programmierung

Prof. Dr. F. Steimann

April 2010

Vorab möchte ich erwähnen, dass sich meine Prüfung auch nach Aussage von Prof. Dr. Steimann wohl von den meisten anderen gängigen Prüfungen unterscheidet: Üblich wären in der Regel detailliertere Fragen zum parametrischen Polymorphismus, die in meiner Prüfung nicht gestellt worden sind, da der Schwerpunkt dieser Prüfung auf der Substituierbarkeit lag mit einem Exkurs zu Eiffel.

- Eröffnet wurde die Prüfung mit der Frage, was denn Objektorientierte Programmierung für mich bedeutet: Darauf hatte ich mich vorab vorbereitet und konnte daher meinen Vortrag abspulen (Stichworte: Was ist ein Objekt, was ist eine Klasse, was ist die Intension und Extension einer Klasse, Beziehungen, Zustand, Nachrichtenversand, dynamisches Binden, statisches Binden, Objekterzeugung, Aliasing).
- Bezüglich Aliasing wurde detailliert nachgefragt, was denn der Unterschied zwischen Referenz- und Wertsemantik im Vergleich zum Call by Value bzw. Call by Reference sei.
- Anschließend wurde gefragt, was Vererbung für mich bedeutet (Stichwort: Vererbung ungleich Subtyping, dazu Unterschiede in Java erklärt, Ko- und Kontravarianz der Eingabe- und Rückgabeparameter genannt, damit die Zuweisungskompatibilität bestehen bleibt).
- Dazu danach die Frage, was Substituierbarkeit bedeutet (Stichwort: Ziel des LSP, die fünf Bedingungen, Praxistauglichkeit).
- Danach folgte ein Exkurs zu Eiffel, da bisher das Meiste zu Ko- und Kontravarianz und Substituierbarkeit gefragt worden ist (Stichwort: Alleinstellungsmerkmal von Eiffel ist die kovariante Redefinition, Grenzfälle dazu, d.h. wann die Substituierbarkeit verloren geht → siehe auch ArrayStoreException).
- Schließlich wurden die sieben Probleme der OOP verlangt, jeweils mit kurzer Erläuterung (Ausführlicher wurde ausschließlich die Antwort zum Problem der mangelnden Kapselung verlangt, dazu etwas von Repräsentationsobjekten erzählt).
- Law of Demeter (Ziel, Ausnahmen etc)

Alles in allem war es eine sehr angenehme Prüfung. Man kann nur dazu raten, vorab sich einen Vortrag zu OOP zu überlegen, das hilft dann auch die Nervosität zu reduzieren. Die Prüfung wurde sehr gut bewertet.

Kurs und Prüfer empfehle ich weiter.

Prüfungsprotokoll Kurs 01814 – Objektorientierte Programmierung

Prüfer: Prof. Dr. Steimann
Februar 2010

Auch wenn sich der Prüfungsverlauf nicht wesentlich von den anderen Protokollen unterscheidet, ist es vermutlich dennoch hilfreich, mal wieder ein aktuelles Beispiel zu sehen.

Für den Beginn hatte ich einen kleinen Vortrag als Einstieg in das Thema OOP vorbereitet, der ca. 6-7 Minuten dauerte.

Inhalt (Stichworte): Unterschiede zw. prozeduraler und OO-Programmierung; Geflecht interagierender Objekte, Attribute und Methoden, Zustand, Nachrichtenversand, Instanzvariablen (benannte und indizierte), Assoziationen (:1-Beziehung, :n-Beziehung, Aggregation/Komposition, wie kann man Komposition realisieren), Möglichkeiten der Objekterzeugung (Klonen -> prototypenbasierte Form der OOP; Instanzen von bestimmten Vorlagen -> klassenbasierte Form der OOP), Klasse, Klassifikation, Generalisierung

Während des Vortrags kamen keine Zwischenfragen, anschließend ging es dann mit den Fragen direkt zu einem neuen Thema weiter:

- Was versteht man unter Subtyping?
- Welche Bedingungen sind an die Substituierbarkeit geknüpft?
-> 5 Bedingungen des LSP aufgezählt
- Welche Bedingungen kann (praktisch gesehen) ein Compiler überprüfen, welche nicht?
- Soviel zur Inklusionspolymorphie, Sie ahnen, was jetzt kommt?
-> Ja, parametrischer Polymorphismus
- Was ist das? Wozu wird er eingesetzt?
-> Erklärt, was es ist; Beispiel mit Collections erläutert, insbes. auch den Zusammenhang zw. parametr. Polymorphismus und Inklusionspolymorphie (vgl. Kurstext 3.12.3)
- Wo werden dennoch Down Casts benötigt?
-> heterogene Collections
- Was versteht man unter dem beschränkten parametrischen Polymorphismus? Wofür ist er da?
- Ist die Beschränkung eine Ober- oder Unterschanke?
-> Oberschranke (Achtung: hier wird lt. Prof. Steimann sehr häufig die Situation mit den Wildcards in Java verwechselt und dementsprechend behauptet, es gibt hier beim beschränkten parametrischen Polymorphismus beides, nämlich Ober- und Unterschanke, was aber falsch ist!)
- Wo gibt es denn die Möglichkeit, Ober- oder Unterschanken zu verwenden?
-> Wildcards in Java
- Wofür sind die Wildcards da?
- Warum funktionieren in einem Fall nur lesende und im anderen Fall nur schreibende Zugriffe?
- Themawechsel: Erzählen Sie etwas zum Problem der mangelnden Kapselung.
- Was kann man dagegen tun?
-> Ansatz in Eiffel mit „Repräsentationsobjekten“ mit Wertsemantik erläutert

- Erzählen Sie etwas zum Gesetz von Demeter: was versteht man darunter? Ist es sinnvoll? Kann man es immer verhindern? Was kann man tun, wenn das Gesetz verletzt wird?
-> Erklärt, was es ist; sinnvoll: prinzipiell schon, da es hilft, die Kopplung zwischen den Klassen gering zu halten; immer verhindern nein bspw. bei Collections; bei Verletzung/Missachtung kann man sog. Vermittlermethoden einführen

Fazit:

Die Prüfungsatmosphäre war sehr angenehm. Einen kleinen Vortrag zu Beginn zu halten kann ich nur jedem empfehlen, da es hilft, die Nervosität (wer hat die nicht?) zu Beginn zu senken. Als Note erhielt ich eine 1.0, obwohl ich bei den Wildcards und den lesenden/schreibenden Zugriffen doch ein wenig „rumgeschwommen“ bin.

=> Kurs und Prüfer kann ich uneingeschränkt weiterempfehlen!

Kurzprotokoll Prüfung Objektorientierte Programmierung (1814)

Bei Professor Steimann

Am 28.02.2008

- Was ist objektorientierte Programmierung?
- Sind Klassen auch Objekte?
- Wovon sind Klassen in Smalltalk Instanzen?
- Wovon sind Metaklassen Instanzen etc.?
- Welche Beziehungen gibt es noch außer der Klassifikations- und Generalisierungsbeziehung? -> Assoziationen zwischen Objekten
- Wie werden diese implementiert? -> über Instanzattribute
- Stellen Attribute immer Beziehungen dar?
- Welche Beziehungen gibt es noch? -> 1:n-Beziehungen; werden über Zwischenobjekte modelliert
- Werden alle 1:n-Beziehungen über Zwischenobjekte modelliert?
- Welche Beziehungen gibt es noch? -> Komposition, Aggregation
- Wie werden Kompositionen modelliert? -> Attribut mit Wertsemantik
- Welche Sprachen bieten auch Wertsemantik an?
- Was bedeutet Subtyping?
- Welche Probleme gibt es beim Subtyping unter Wertsemantik?
- Was macht ein Typsystem?
- Was ist beschränkter parametrischer Polymorphismus?
- Was ist eine ArrayStoreException?
- Warum werden solche Zuweisungen zunächst überhaupt zugelassen? -> Flexibilität
- Wie wird dies beim parametrischen Polymorphismus gelöst? -> Wildcards
- Können Wildcards auch beschränkt werden?
- Wie kann man dann Listen sortieren (dabei muss man ja lesen und schreiben)?

Viel Glück!

Kurs 01814, OOP Prüfungsprotokoll, 24.01.2008, Prof. Dr. Steinmann

Was haben Sie für sich aus dem Kurs mitgenommen?

- Als C++ Entwickler einen ganz neuen Blick auf Objektorientierung bekommen
- Unterscheidung Prozedural vs Objektorientierung

Was macht die Objektorientierung aus?

- Programm nicht mehr lineare Folge von Befehlen wie bei prozeduralen Sprachen, sondern Geflecht von interagierenden Objekten
- vor allem Funktionen und Daten nicht mehr getrennt, sondern in Objekten gekapselt
- Objekte haben Zustand und Verhalten

Was macht den Zustand eines Objektes aus?

- Attribute (Beziehungen zu anderen Objekten und Eigenschaften)

Wie ist das Typsystem von C++ aufgebaut?

- primitive Typen mit Wertsymantik (long, int, etc.)
- Von Klassen (implizieren Typen) können Wertobjekte, als auch Referenzobjekte erzeugt werden

Was versteht man unter parametrischen Polymorphismus?

- Methodendefinition enthält Platzhalter. Erst bei der Verwendung Angabe des Typs.

Was ist der Unterschied von Generischen Typen in Java zu C++?

- da bin ich etwas "rumgeschwommen", in C++ sind es reine Vorlagen die zu einem Typ kompiliert werden... siehe Kurstext

Wofür taugen die Wildcards in Java?

- Zuweisungskompatibilität...

Wieso gibt es Extends und Super bei den Wildcards in JAva?

- Ober und Unterschranke...

Nennen Sie mir die Probleme der Objektorientierung.

- alle aus Kurstext aufgezählt

Erläutern Sie das Problem der mangelnden Dekomposition.

- es fehlt sowas wie ein Komponentn

Erläutern Sie das Problem der mangelnden Kapselung.

- Was kann man dagegen tun? -> in C++ bspw. Wertobjekte (aber meiner Meinung nach auch nicht sicher, da auch auf diese Pointer nach "außen" gegeben werden können)

Erläutern Sie das Gesetze von Demeter.

- "sprich nicht mit Fremden"
- nur Funktionen von Objekten in direkter Beziehung, ggf. Vermittlermethode einführen

Fazit

Bei Prof. Steinmann muss man wohl nicht alles auswendig können. Ich erhielt als Note 1.0, obwohl ich nicht alles bis ins Detail wußte. Ich habe vieles aus meiner Erfahrung mit C++ heraus erklärt, das hat ihm wohl gefallen (C++ allerdings weniger ;-)) .

Fachprüfung in Objektorientierte Programmierung, 1814 (Version vom WS 06/07 mit Smalltalk)

Prüfer: Prof. Dr. Steinmann

Beisitzer: Frau Dr. Keller

Prof. Steinmann ist sehr nett und locker, man kommt leicht ins Gespräch. Nach der Prüfung haben wir uns noch ein bisschen über Gott und die Welt unterhalten.

In den Prüfungsprotokollen hatte ich gelesen, dass eine Prüfung bei ihm am besten ein Vortrag vom Studenten sein sollte. Daher hatte ich mir im Voraus einen Ablauf meiner Prüfung überlegt. Ich informierte Prof. Steinmann und legte los.

Zuerst erzählte ich ein wenig über die Grundlagen der Objektorientierung; was Objekte, Klassen, Metaklassen und Typen sind. Leider bin ich dann total aus dem Konzept gekommen und ab da stellte Prof. Steinmann mehr Fragen als ich den Vortrag hielt. Prof. Steinmann fragte was ich so alles ueber Beziehungen in der OO wuesste. Dabei zählte ich auf:

*Spezialisierung / Generalisierung

*Klassifikation / Instanziierung

*1:1, 1:n, n:n-Beziehungen zwischen Objekten

*Sub- / Supertypenbeziehungen (dabei kamen wir auf das Substitutionsprinzip von Liskov zu sprechen) Zum Schluss redeten wir über parametrischen Polymorphismus.

Als Note habe ich eine 2,0 erhalten. Kritikpunkte:

*Die Konzepte in der OO habe ich größtenteils richtig erklärt, aber die meisten Schlüsselwörter nicht gekonnt (z. B. "parametrischer Polymorphismus"). Als Konsequenz hat es dann immer gedauert bis ich einen Sachverhalt erklärt hatte. [Zurückblickend haette ich dies durch eine konsequentere Bearbeitung der Einsendeaufgaben verbessern können] *Ich habe die Vortragsstruktur nicht durchgehalten. Dies ist nicht grundsätzlich schlecht, aber er war irritiert dass ich einen Vortrag angekündigt hatte und dann doch keinen gehalten hatte. [Meine Vorbereitung auf einen eigenständigen Vortrag war einfach nicht gut genug]

Insgesamt bin ich mit der glatten zwei zufrieden; Prof. Steinmann war sehr fair (ich habe wirklich manchmal Mist erzählt) und hat mir (und den anderen Studenten) zu Gute gehalten, dass wir mit einem ganz neuen Kurstext arbeiten mussten.

Prüfungs-Gedächtnis-Protokoll
Objektorientierte Programmierung (111)
(neue Version)
Wintersemester 0000
Prüfer Prof. Dr. F. Steimann
Beisitzerin Frau Dr. D. Keller
0 März 00

Vorkenntnisse □Vorbereitung□

Ich programmiere seit ca. 10 Jahren in Java, habe zwischendurch Bekanntschaft mit Smalltalk und C++ gemacht. Mein Schwerpunkt liegt jedoch in der Web-Entwicklung mit Java und dem Einsatz von Frameworks.

Als explizite Vorbereitung habe ich das Online-Skript durchgearbeitet. Einmal während des Semesters mit Bearbeitung der Aufgaben, welche ich auch fast immer eingesendet habe. Dies gibt immer ein gutes Feedback, damit man weiß wo man steht.

Vor der Prüfung habe ich das Skript noch einmal durchgearbeitet.

Prüfungsverlauf□

Wie in den anderen Protokollen bereits angekündigt, hatte ich einen gewissen Ablauf vorbereitet, den ich frei erzählen konnte. Ich begann mit der Motivation, warum man gerne OOP möchte, was das Besondere ist im Vergleich zu anderen Programmierparadigmen, was die Idee hinter dem Konzept ist (das zugrundeliegende Weltbild der Objekte und Nachrichten). Danach habe ich erzählt, welche Arten es gibt, wie man zu Objekten kommen kann, was man unter Instanziierung und Klassifikation versteht.

Als ich dynamisches Binden erklärt habe, hat Prof. Dr. Steimann nachgefragt, wie das konkret funktioniert, wonach entschieden wird, welche Methode verwendet wird und ob diese Bindung dann ausschließlich dynamisch stattfindet. (Kommt auf die Sprache□bei Smalltalk ausschließlich, bei Java kann auch statisch gebunden werden, bei final Klassen, wenn der Compiler das unterstützt.)

Vom dynamischen Binden kamen wir dann über Zuweisungskompatibilität auf Typensysteme, was das ist und wozu das gut ist. Dazu habe ich erklärt was Typkonformität ist und dass dies notwendige Bedingung für Zuweisungskompatibilität ist, aber für Ersetzbarkeit nicht unbedingt ausreicht.

□err Prof. Dr. Steimann hat dann direkt das Thema gewechselt und wollte wissen, was beschränkter, parametrischer Polymorphismus ist. Zu der Beschränkung, von der ich sagte, dass es eine Schranke nach oben gibt über einen Supertyp, wollte er auch wissen, ob es eine untere Schranke gibt. Die gibt es wohl, viel mir aber nicht ein und er sagte es sei auch nicht so ganz einfach ein Beispiel dafür zu finden.

Am Schluss wollte er noch etwas über Probleme hören, die in der OOP vorkommen. Ich habe ein das Fragile-Base-Class Problem genannt, es aber leider falsch verstanden und daher zwar das Beispiel noch aus der Erinnerung wiedergeben können, aber die eigentliche Erklärung, was da warum gebrochen wird, nicht richtig wiedergeben können.

Außerdem nannte ich noch das Problem der schlechten Tracebarkeit, eindimensionale Strukturierungsmöglichkeit und das Problem der mangelnden Kapselung.

Fazit□

Die Prüfung war sehr angenehm und die Erklärung für den Abzug in der Note total fair. (War trotzdem noch im Bereich sehr gut).

Prüfungsprotokoll

Kurs: 1814, Objektorientierte Programmierung (Version WS 2006/07)

Prüfer: Prof. Dr. Steimann

Beisitzer: Dr. Keller

Datum: 23.02.2007

Note: 1,0

- Was versteht man unter Objektorientierung?
- Grundkonzepte der Objektorientierten Programmierung
 - Modularisierung erklären
 - Generalisierung, Spezialisierung und Vererbung erklären
- Typsysteme definieren und Nutzen erklären
- Subtyping erklären
 - Probleme des Subtyping
 - Substitution und LSW erklären
- Weitere Probleme der Objektorientierten Programmierung erklären
 - Fragile-Base-Class-Problem
 - Schlechte Tracebarkeit
 - Mangelnde Kapselung
 - Aliasing erklären
 - Schutz vor Aliasing (explizite Kopie, dennoch bleibt Unsicherheit)

Die Atmosphäre der Prüfung war sehr angenehm und entspannt, die Prüfung war mehr ein Gespräch über die Thematik. Aufkommende Unsicherheiten durch meine Nervosität wurden gelöst, indem Prof. Steimann bei unpräzisen Formulierungen nachfragte.

Generell ist dieser Kurs bzw. Kurse bei Prof. Steimann sehr zu empfehlen.

Prüfungs-Gedächtnis-Protokoll
Objektorientierte Programmierung (1814)
Sommersemester 2006
Prüfer: Prof. Dr. F. Steinmann
Beisitzerin: Frau Dr. D. Keller

29. August 2006

Abstract In dem Fach 1814 **Objektorientierte Programmierung** habe ich meine erste Fachprüfung an der Fernuni abgelegt. Anbei ein paar Notizen, die dem ein oder anderen Studenten hilfreich sein können. Dabei lege ich das Augenmerk mehr auf allgemeine Themen, als direkte Fragen. Wie die Verfolgung der Newsgroup gezeigt hat ist - logischerweise - der Stoff der gleiche, aber die Fragen haben in dem Sinne kein direktes Muster, die Prof. Steinmann stellt.

Vorkenntnisse habe ich bereits sowohl in Java, als auch ganz positive Nebeneffekte vom Softwareengineering 1 bei Prof. Six erwerben können. Ich habe bereits einen Informatikstudiengang absolviert. In Summe kann ich sagen, dass ich Java an sich kenne, mich nicht wirklich mit den Hintergründen ausgekannt habe (bis ich sie mal in dem Kurs gelernt habe ;-)) und zähle mich auch nicht zu Superhackern.

Ich habe die Aufgabenzettel aus 1618 abgeben und bis auf 2 Ausnahmen immer über 83% gehabt, was mir nicht wirklich schwer viel.

Wenn man hier noch nie was zur OOP oder Java gehört hat, ist ein stärkeres Eintauchen in die Materie nötig.

Vorbereitungen habe ich schon früh begonnen, als Material stehen prinzipiell folgende Möglichkeiten zur Verfügung:

- Kursmaterial des Kurses 1618
- Der Artikel *Flexible Alias Protection*
- Der Artikel *A Behavioral Notion of Subtyping*
- Das Buch Konzepte Objektorientierter Programmierung

Ich habe mir das Buch bestellt gehabt und dann zusätzlich den Kurs 1618 belegt, den ich sonst nicht weiter für mein Studium benötige. Später als die Kursunterlagen da waren und ich sie mit dem Buch verglichen habe, stellte ich aber fest, dass es mehr oder weniger das Gleiche ist. Frau Dr. Keller hatte die Unterschiede einmal in der Newsgroup zum Kurs beschrieben. Es gibt an einigen Stellen Differenzen, die aber in Summe nicht erheblich sind. Die beiden Artikel können im Internet bezogen werden und sind auf der Seite des Kurses zu finden.

Während des Semesters habe ich mich dann auf den Kurs konzentriert und überwiegend die KE nachgearbeitet, wobei für die Übersicht in meinen Augen die KE1 und die KE7 besonders wichtig sind, weil da die Themen umfassend und in einer Zusammenfassung dargelegt sind.

Ich habe mir für zwei Prüfungen (eine FP und ein LN) in Summe noch zwei Wochen Urlaub genommen. Für die FP viel im Endeffekt "nur" zwei bis drei Tage konkret ab, da ich dort mehr Verwendung für die Aufgabenzettel hatte und die zur Bearbeitung besser geeignet waren.

Hauptbestandteil meiner Vorbereitungen war eine Sichtung der KE und die Notierung der wichtigsten Punkte. Aus diesen Informationen habe ich mir dann ein großes Mindmap erstellt, welches den ganzen Kurs grob skizziert hat. Auf Basis dessen habe ich mir an dieser Stelle einen Vortrag auf Basis des MM erstellt, nicht auswendig gelernt, aber schon zu Hause versucht frei vorzutragen (mach ich immer so, um einfach

mal ein Gefühl dafür zu bekommen, wie lange man reden kann). Das besteht ganz grob aus folgenden Komponenten:

- OOP
 - Objektkonzept
 - * Was ist ein Programm?
 - * Was ist eine Programmausführung?
 - * Vergleiche zu anderen Programmierparadigmen
 - * einfach mal überlegen, wie man die Thematik ansprechen würde um es jemandem zu erklären, der davon noch nie was gehört hat
 - Klassifizierung
 - * ganz grob, was ermöglicht das für die OOP?
 - * welche beiden Stichworte gehören in diesem Zusammenhang genannt? (Abstraktion, Spezialisierung)
 - Sprachliche Realisierung dieser Konzepte
 - * Klassenkonzept
 - Klassenkonzept erklären, Vergleich zu anderen Programmiersprachen sind immer gut (nicht alle Klassenkonzept → Prototypenkonzept)
 - * Vererbung
 - In dem Zusammenhang kann man durchaus auch Themen wie Subclassing etc. nennen und wie es in Java oder einer anderen Sprache aussieht (der Kurstext und das Buch geben vielen Beispiele)
 - * Subtyping / meth. Methodenauswahl
 - Klare Thematik und Übergang zum Subtyping Artikel, mit Typen, "wann ist ein Subtyp, Typ von einem anderen", usw. hier finden sich viele Beispiele

Obiges Gerippe kann man gut auskleiden. Für die Artikel bin ich hergegangen und habe den FAP Artikel komplett ins Deutsche übersetzt um dann beim Durchlesen zu merken, dass das Englisch doch einfach ist.

Den Subtypingartikel habe ich dann gar nicht mehr übersetzt. Aus beiden Artikeln habe ich dann wieder zwei Mindmaps erstellt, die ich auch bei der Prüfung vor Augen hatte (aber wie man die Daten aufbereitet ist ja unterschiedlich)

Die Prüfung an sich fand in einer wirklich netten Umgebung statt. Frau Dr. Keller hatte ich auf dem Flur getroffen, nachdem ich mich bei der Sekräterin angemeldet hatte. Nachdem wir über alles mögliche geschnackt hatten, sind wir dann zu Prof. Steinmann ins Zimmer gegangen. Begrüßung, Personalien überprüft und los gehts. Ein Block mit Stift vor mir auf dem Tisch machte mich neugierig was da wohl kommen würde. Frau Dr. Keller führt Protokoll und schreibt fleißig mit. Eröffnungsfrage von Prof. Steinmann war, womit ich beginne möchte, was mich kurzzeitig irgendwie kurz irritierte, aber ich entschied mich dann für eine Übersicht über die OOP. Zustimmungendes Nicken und los gings. Die Beschreibung orientierte sich weitestgehend

an meinem obige Gerippe. Es dauerte einen Moment um die Gesichtszüge einzuschätzen, und abzuwägen ob man auf dem richtigen Weg ist. War schwer, er hat sich sehr konzentriert und genau alles angehört, aber auch keine Zwischenfragen gestellt solange man den Vortrag erzählt.

Es gab zwischendrin zwar eine Diskussion über zwei, drei Punkte, ich kann mich aber gar nicht wirklich mehr erinnern wann die genau war :-/ auf jedenfall ging es hier explizit nochmal um Parallelität und Subtyping in Form von Vergleichen zur prozeduralen Programmierung

Ich habe nach der Übersicht eine Pause gemacht und bewußt die selbst gefragt, wie es weiter geht. Prof. Steinmann hat sich dann dafür entschieden (nachdem ich den Hinweis in der Einführung selber gebracht hatte) auf den Subtyping Artikel überzugehen. Fand ich eigentlich nicht gut, weil der Aliasingartikel besser vorbereitet (in meinen Augen) war, aber ich hätte mich auch anders entscheiden können, ich denke auch das hätte er akzeptiert. Bei der Übersicht habe ich viel aus dem Kurstext mit dem Artikel kombiniert und versucht die Zusammenhängen darzustellen. Um es bildlich zu untermalen habe ich mir ein Beispiel ausgedacht und dann anhand dieses Beispiel erklärt warum ein Subtyp nicht immer konform sein muss. In dem Zusammenhang gab es natürlich auch Ko und Kontravarianz Verweise meinerseits inklusive der Erklärung. Der Artikel bietet im Endeffekt zwei Ansätze einen Subtypen direkt zu bestimmen. Ein Weg führt über die Klasseninvariante (siehe auch Generalisierung/Konformität Kurs 1793) ein anderer Vorschlag vergleicht die Typen auf Basis von zusätzlichen Methoden zu bereits vorhanden Methoden. Das muss/sollte man frei vortragen können und natürlich auch die entsprechende Begriff zwischendrin einwerfen um auch zu zeigen das man es verstanden hat.

Dann kam ganz plötzlich, "ok warten sie bitte kurz draußen". Hatte mich fast etwas "geschockt" weil ich an den FAP Artikel gedacht hatte :-) Die Zeit war aber wohl schon lange rum, ich kann mich auch nicht wirklich mehr erinnern.

Das Warten hatte nicht lange gedauert, vielleicht eine Minute. Dann erfolgt die Mitteilung der Note, kurzer privater Plausch nach Werdegang etc. was ich durchaus als positiv empfanden habe. Ich habe auch noch ein paar Dinge zwecks Prüfungsamt gefragt und dann ging es praktisch schon wieder auf die Autobahn.

Zusammenfassung Für meine erste Prüfung war das Erlebnis positiv. Ich kann Prof. Steinmann absolut empfehlen, genauso Frau Dr. Keller. Die interessante Thematik an dem Fach fand ich, dass man sich in gewissen Dingen eingraben konnte und ganz frei mal erzählen kann, was man denkt und man das verstanden hat. Ich selber hätte mich wahrscheinlich um 0.3 schlechter bewertet, da wir eine Thematik hatten, die mir nicht ganz klar war und die auch nicht entsprechend 'richtig' im Kurstext stand (Thematik inhärent paralleles Ausführungsmodell). Da dieses der Kurstext aber so bot, spielte das in die Bewertung nicht hinein. Wer also gewissen Aufwand nicht schäut, ohne direkten Kurstext sich anhand von Buch (oder 1618) + englischen Artikel ein breites Spektrum erarbeiten möchte ist in dem Kurs definitiv richtig. Der Freiraum der einem in der Prüfung gelassen wird ist gut. Das in dem Protokoll keine direkten Fragen stehen hat folgende Bewandnis, Prof. Steinmann macht den Eindruck als ob er keinen "Standardkatalog" an Fragen hat. Es kommt auf die Vorbereitung an, wer sich gut überlegt was er erzählt und dementsprechend

Sattelfest ist, sollte hier wenig Probleme bekommen.

Ich denke durch die eigene Vortragsmöglichkeit hat man es auch ein wenig in der Hand zu welchen Themen man tiefer einsteigen möchte.

Für eure Prüfungen wünsche ich ganz viel Glück!

Prüfungsprotokoll Fachprüfung Master of Computer Science

Datum: 06.04.2006

Kurs: 1814

Prüfer: Herr Prof. Dr. Steimann

Beisitzerin: Frau Dr. Keller

Dauer: ca. 20 Min

Note: 1.0

Vorwort:

Als ich mich auf die Prüfung vorbereitet habe, war ich echt froh, dass es zumindest einige Protokolle zur Prüfung 1814 gab. Ich habe beschlossen, nach der Prüfung gegebenenfalls ein Prüfungsprotokoll zu erstellen und den Ablauf für Neulinge (wie für mich) zu beschreiben.

- *Zuerst zu FernUni:* Das Finden des Gebäudes und des Raumes ist einfach. Die Uni sieht modern und schön aus. Das Gebäude ist verglast und überall mit Alustall bearbeitet. Der weite Blick auf einen grünen Wald gibt innere Ruhe und Zuversicht vor der Prüfung.
- *Begrüßung:* Frau Dr. Keller lädt ins Büro ein. Man begrüßt sich per Handschlag. Der Professor wirkt sehr ruhig und gelassen. Er lächelt leicht. Zuerst wurde Matrikelnummer und Personalien aufgeschrieben. Man sitzt mit den beiden an einem kleinen halbrunden dunklen Tisch. Auf dem Tisch liegen Block und Kugelschreiber. Nachdem die offiziellen Sachen erledigt wurden, legte dann der Professor los.
- *Wahrnehmung:* Ich hatte die ganze Zeit nicht das Gefühl gehabt, dass er etwas suchte, was ich eventuell nicht wusste. Mit anderen Worten er will nicht reinlegen, sondern will erfahren, ob man die wichtigsten Themen beherrschen bzw. verstanden hat. Er gibt einem die Möglichkeit zu reden und seine Vorstellung zu einem oder dem anderen Thema zu äußern. Bei Unklarheiten und zur Vertiefung kann er etwas genauer nachfragen.
- *Deutsch ist nicht meine Muttersprache,* sie ist Russisch. Ich hatte etwas Zweifel gehabt, ob man mit ein wenig falschem und leicht akzentbehaftetem Deutsch gute Note bekommen kann. Es ist also möglich. Es zählen nur gute Vorbereitung zur Prüfung, Verständnis der Kursthemen und die Fähigkeit das Gelernte vorzutragen.
- *Vorbereitungsphase:* Ich habe zwei Wochen für die Übersetzungen der Forschungsbereiche und die Ausarbeitung der Kurzzusammenfassungen gebraucht. Zu den Forschungsarbeiten kann man einiges im Internet finden. So kann man sehen, ob man das Thema richtig aufgefasst hat. Des weiteren habe ich eine Kurzzusammenfassung geschrieben und danach ca. 2,5 volle Wochen gelernt. Vorhin habe ich die Klausur zum Kurs 1618 geschrieben und mit 1.3 bestanden. Ich war also mit dem Kurs vor der Vorbereitung zur mündlichen Prüfung gut vertraut. Für die Vorbereitung zur Prüfung habe ich mir Urlaub genommen.

Prüfungsfragen:

Die Fragestellungen habe ich nicht genau gemerkt. Aber thematisch ging es um die Themen, die ich unten aufgeschrieben habe.

- Was ist OOP?
 - Antwort findet man in KE1 und KE7
- Beispiel für Abstraktion?
 - Ich habe Beispiel für Einzelhandelgeschäft und konkrete Läden (z.B. Buchladen) gemacht.
- Was unterscheidet Subtyping und Subclassing?
 - Antwort findet man in KE4
- Was bedeutet ein Subtyp zu sein?
 - Zuerst habe ich die formalen Bedingungen aufgezählt (Subtyping-Ordnung und -Regel) und dann nachgefragt, ob ich über den Artikel berichten sollte? Der Professor hörte dann meinen Vortrag.
- Dann haben wir uns über die Vorbedingungen des Subtyps Albino-Elefant (Elefanten-Typhierarchien) unterhalten.

Fazit:

Mir ging die Zeit zu schnell rum. Ich hätte mich mit dem Professor gerne noch länger unterhalten. Leider wollte er nichts weiter wissen. So musste ich aus dem Büro kurz rausgehen und nach 5 bis 10 Sekunden dürfte ich wieder rein. Der Professor hat mich für die gute Vorbereitung gelobt und die Note mitgeteilt.

Es ist aber wie ich es mitbekommen habe, nicht bei allen so gut gelaufen. Zwei andere Studierenden, die an dem Tag dieselbe Prüfung hatten, haben mit 3.0 bestanden.

Wer also gut abschneiden möchte, soll die Artikeln zum Herzen nehmen und die wichtigsten Kapiteln (KE1-KE4+Zusammenfassung KE7) verstanden haben.

Ich wünsche Euch viel Erfolg!

Den Professor kann ich als Prüfer uneingeschränkt empfehlen! Die Beisitzerin, Frau Dr. Keller, ist auch eine nette, freundliche und aufgeschlossene Person.

Prüfungsprotokoll

Fachprüfung Master of comp sc im Kurs 1814 bei Prof Steimann (S) Juli 2005, Unterlagen: Kurs 1618 + 2 Paper

S: Ja, 1814, was ist das?

- Äh, der Prüfungskurs, er handelt von Objektorientierter Programmierung...

S: Hm, was ist das?

Kurzer Einwurf: Prof Steimann stellt sehr offene Fragen. Das gibt einem die Möglichkeit, frei vor sich hin zu referieren. Anfangs bin ich - unter anderem aus Nervosität - nicht so richtig rein gekommen. Nach obiger Frage ging's ein bisschen hin und her zu Objektorientierten Konzepten. Beim Subtyping gings dann in die Tiefe:

S: Was muss man dabei beachten?

- Ko-/Kontravarianz

S: Warum?

- Beispiele, kein praktisches Beispiel für Kontravarianz

S: Wie siehts bei Java aus?

- seit 1.5 Kovarianz bei Ergebnis, bei Parametern Gleichheit

S: Warum keine Kontravarianz?

- Überladene Methoden

S: Warum noch?

?? (Wollte irgendwie was mit JavaBean Spezifikation und set-er/get-er Methoden)

S: Macht ja nix, was gibt's semantisch zum Subtyping zu sagen?

- konformes Verhalten, Vortrag über Subtyping Artikel

S: Was machen wir jetzt, hm, kennen sie auch Aliasing?

- Vortrag Aliasing Artikel

Ende

Die Prüfung hat in netter, entspannter Atmosphäre stattgefunden. Es hilft sehr, wenn man zu den Themen vortragen kann. Will sagen über ein Themengebiet frei, umfassend referieren kann. So kann man mehrere Minuten am Stück reden und der Prof fragt nur gelegentlich nach. Für diese Prüfung lohnt es sich, das vorher zu üben. Kann Prüfer und Fach empfehlen.

Gedächtnisprotokoll zur Fachprüfung 1814 (Objektorientierte Programmierung)

Termin: 10.05.2004

Ort: Videoprüfung im Studienzentrum Karlsruhe

Dauer: 20 min

Note: 1.0

1) Lieblingsthema?

Nein.

2) Was versteht man unter OOP?

OO-Grundmodell erläutert, OOP ist Programmierung mit OO-Konzepten.

3) Nebenläufigkeit in Java, aktive vs. passive Objekte?

In Java sind die meisten Objekte passiv, nur die Thread-Objekte sind aktiv. Grundmodell wurde nicht umgesetzt, lässt sich lediglich nachbilden.

4) Unterschied OO-Sprachen zu imperativen Sprachen?

Anderes Ausführungsmodell (Parallelität) und andere Programmstruktur (Objekt-konzept = Bündelung von Daten, Zustand und Operationen zu einer kleineren Einheit)

Weitere Konzepte: Klassifikation und Subtyping

Sprachlich umgesetzt durch: Vererbung und dynamisches Binden.

5) Dynamisches Binden näher erläutern?

Statischer Anteil: Compiler (u.a. „most-specific“-Ansatz erläutern)

Dynamischer Anteil: Binder

Unterschiedliche Behandlung zwischen virtuellem und non-virtuellem (statische Methoden, private Methoden, Aufruf der Methoden der Superklasse) Aufrufmodus.

6) Unterstützen imperative Sprachen dynamisches Binden?

Nein.

7) Vererbung?

Subtyping: Typspezifikation

Vererbung: Übernahme von Implementierungsteilen anderer Klassen

Subclassing := Vererbung + Subtyping, Beispiele Java und C++

8) Voraussetzungen für Subtyping?

Sprachlich: - Sprache muß Subtypordnung auf die Menge ihrer Typen einführen.

- Das Substitutionsprinzip muss gelten

- Subtyping ist nicht auf OO-Sprachen beschränkt

Syntaktisch: - Kovarianz der Rückgabe- und Ausnahmentypen

- Kontravarianz der Parametertypen

Semantisch: - Konformes Verhalten der Subtypen

9) Erklärung Kovarianz der Rückgabetypen und warum das so sein muß?

Eigenes Beispiel vorgetragen. Begründung durch das Substitutionsprinzip.

10) Kontravarianz der Parametertypen und warum das so sein muß?

Eigenes Beispiel vorgetragen.

11) Sinnvolles praktisches Beispiel zur Kontravarianz der Parametertypen?

Gibt es keines.

12) Inhalt des Liskov-Aufsatzes?

Syntaktische Bedingungen sind hilfreich aber nicht ausreichend. Subtypen müssen sich konform zum Supertyp verhalten, müssen die Spezifikationen des Supertyps einhalten.

13) Wie wird dies erreicht?

Allg.: Das Verhalten von Methoden wird durch die Formulierung von Vor- und Nachbedingungen beschrieben.

Konkret:

- (1) $Inv[Sub] \implies Inv[Super]$*
- (2) $Pre[Super] \implies Pre[Sub]$*
- (3) $Post[Sub] \implies Post[Super]$*
- (4) $Constraint[Sub] \implies Constraint[Super]$*

Oben genannte Implikationen noch mit eigenen Worten erläutert. Zu (1): Der Subtyp muss die Invarianten des Supertyps einhalten, er darf noch weitere Invarianten deklarieren...

14) Umsetzung des Aufsatzes bzw. praktische Relevanz?

Bis dato keine geeignete Werkzeugunterstützung.

Stärkt lediglich die Disziplin bzw. das Verständnis des Programmierers.

Formulierung durch Prädikatenlogik für die meisten Programmierer ohnehin zu aufwendig, zu schwierig.

Fazit:

Zu ersten Male hatte ich eine mündliche Prüfung in Form einer Videoprüfung abgelegt. Das ging eigentlich recht gut, man gewöhnt sich schnell daran.

Herr Prof. Steimann ging es am Anfang recht oberflächlich an. Im Laufe der Prüfung ging es, wie es das Protokoll zeigt, immer weiter in die Tiefen des Subtypings.

Ich selbst war erstaunt, wie schnell die 20 Minuten vorbei waren.

Herr Prof. Steimann kann ich als Prüfer weiterempfehlen.

Bericht über die Fachprüfung im Fach „Objektorientierte Programmierung (01814)“ zu Master of Computer Science

Termin: 23.12.2004, 12.10 – 12.40 Uhr

Prüfer: Prof. Dr. Steimann

Ort: Hagen

Prof. Steimann ist seit Oktober 2004 am Institut in Hagen. Die Atmosphäre war sehr entspannt und angenehm.

Prüfungsinhalt waren die Kursinhalte des Kurses 1618 sowie die beiden Fachaufsätze „A Behavioral Notation of Subtyping“ und „Flexible Alias Protection“, welche im Internet auffindbar sind.

Meine Vorbildung: Mathematiker mit Abschluss. Ich bin in einer Funktion tätig, welche am ehesten auf das Profil eines Wirtschaftsinformatikers passt. Vor 20 Jahren habe ich 2 Jahre lang als Programmierer gearbeitet, danach auch 2 Jahre als Projektleiter für eine Software-Entwicklung, jedoch nicht mit OO-Sprachen.

1987 Vordiplom Informatik in Hagen.

SS2004: Kurs 1618 im Studiengang „Master of Computer Science“.

Zur Einleitung fragte mich Prof. Steimann, ob ich ein Lieblingsthema aus den Prüfungsinhalten habe, was ich allerdings verneinte. So begannen wir mit der „Frage aller Fragen“:

1. Welche Eigenschaften haben oo-Programmiersprachen im Vergleich zu den klassischen imperativen Sprachen?
Abstraktion, Klassifikation, Subtyping, Vererbung, Polymorphismus
2. Was bedeutet Klassifikation? Ordnung, Kategorien, Struktur...
3. Wie kann man ohne Klassen (in anderen Programmiersprachen als JAVA) noch Objekte erzeugen?
Klone (z. B. in Sather)
4. Was ist Subtyping und Vererbung, welche Unterschiede bestehen zwischen diesen Eigenschaften, welcher Zusammenhang?
Dyn. Und statische Bindung, Interfaces vs. Klassen
Subclassing = Subtyping + Vererbung
5. Übergang zum Artikel zu Subtyping: Wer sollte Subtyp von wem sein: Pair, Triple?
Aus Programmierer-Sicht: Pair ist Subtyp von Triple; aber mathematisch ist die Menge der Triple keine Untermenge der Pairs!
6. Wie löst man dieses Problem? Keine Subtyping-Relation untereinander, aber zu einer abstrakten Superklasse definieren (vgl. Number in Java)
7. Was ist der Unterschied zwischen einem Interface und einer abstrakten Klasse? Abstrakte Klassen dürfen Implementationen enthalten, Interfaces nicht
8. Welcher Art sind Attribute in Interfaces? „static“
9. Gibt es auch in anderen oo-Sprachen Interfaces? Eher nicht, z.B. nicht in C++, dafür dort Mehrfachvererbung
10. Artikel zur Alias-Protection: Was ist das Problem, das dieser Artikel anspricht? Aliasing erklären
11. Welcher Unterschied besteht bei Zuweisungen von Werten an Variablen, wenn
 - a) der Wert ein primitiver Datentyp ist
 - b) der Wert eine Objektreferenz ist?Von einem primitiven Datentyp wird immer eine Kopie erzeugt, eine Objektreferenz ist ein Pointer. Eine „Hausaufgabe“ wäre: Schreibe eine Methode, die 2 Objekte als Argumente hat, welche in der Methode vertauscht werden. Geht das überhaupt in JAVA?
12. Was ist ein Container, was eine Repräsentation, was ein Argument?
13. Welche Schlüsselwörter führt der Artikel neu ein und welche Bedeutung haben sie?
14. Welche Methoden sind mit dem Modifier „free“ auszuzeichnen? Konstruktoren
15. Mit welchen Modifizieren lassen sich die FAP-Modifier vergleichen? Sichtbarkeitsmodifizierer
16. Welche Eigenschaften hat die Prüfung der FAP-Modifier?
Findet im Compiler statt
Keinen Einfluss auf das Binary und damit auf das Laufzeitverhalten

Modular, d.h. auf Programmteile anwendbar
Orthogonal zu anderen Modifiern

17. Was ist eine Komponente? Komponenten sind für einen bestimmten Zweck bereitgestellte Klassen. Schon eine Klasse in einem .class-file ist unter diesem Gesichtspunkt eine „Komponente“. Das AWT stellt solche Komponenten bereit: elementare Komponenten und Behälterkomponenten. Komponenten sind oft Modelle, aus denen durch Subtyping Anwendungsbausteine entstehen.
18. Wie kann Subtyping zu Anomalien führen?
Beispiel: equal() auf Super- und Subtyp angewandt kann verschiedene Ergebnisse haben. Grundsätzlich kann man in einem Subtyp jede nicht-finale Methode überschreiben und damit solche Anomalien erzeugen.
19. Wie könnte man diese „Fälle“ in JAVA vermeiden? Indem alle JAVA-Auslieferungsklassen als abstrakt oder als final erklärt werden.

Jetzt noch schnell ein weiteres Thema (die Zeit wird knapp):

20. Threads: JAVA kennt nur Threads als „passive“ Objekte, wünschenswert sind „aktive“ Objekte als Threads. Wie sehen diese aus?
Unter UNIX verwandte Prozesse, dort in C mit „fork()“ erzeugt, Kommunikation über shared memory, message queues oder Semaphoren
(dies ist kein Inhalt des Kurses 1618, sondern aus meiner Berufspraxis!)
21. Welcher dieser Kommunikationsmethoden entspricht der Kommunikation unter JAVA-Threads?
Die Thread-Kommunikation über gemeinsamen Speicher mit Monitor-Sperren entspricht den Semaphoren.

Note: „gut“ – mir hat meine Berufserfahrung (vor 20 Jahren „C“/UNIX-Programmierer) sowie die intensive Bearbeitung des Kurses 1618 im SS 04 geholfen. Einige Fragen konnte ich nur zögerlich/„schwammig“ beantworten. Wenn nicht 90% aller Prüfungen „sehr gut“ benotet werden, bin ich mit meiner Benotung zufrieden ;-)

Ich kann Prof. Steimann uneingeschränkt als Prüfer empfehlen.