Objektorientierte Programmierung

Kapitel 0: Organisatorisches

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Wintersemester 2018/19

http://www.informatik.uni-halle.de/~brass/oop18/

Inhalt

1. Vorlesungsinhalte
2. Organisatorisches
3. Hausaufgaben, Prüfung
4. Arbeitsmittel
5. Ratschläge

Lernziele

- Programmieren können

  Selbständige Erstellung eigener Programme für gegebene Aufgaben.
  Nur kleinere/einfachere Programme, z.B. werden interessantere Algorithmen und Datenstrukturen hier nicht behandelt (eigene Vorlesung).

- Rechner, Betriebssystem und Editor bzw. Entwicklungsumgebung ausreichend bedienen können.

- Gegebene Programme verstehen können.

  Z.B. Klausurfrage: Was gibt dieses Programm aus?

- Die syntaktische Korrektheit gegebener Programme beurteilen können ($\rightarrow$ Syntax-Diagramme).

- Grundkonzepte von Programmiersprachen kennen, sich leicht in neue Sprachen einarbeiten können.

Zur Programmiersprache (1)

- In dieser Vorlesung wird Programmierung anhand der Sprache "Java" gelehrt (+ Vergleich mit anderen Sprachen).
  - In der Praxis verbreitet.

    Z.B. Umfrage von iX bei 50 großen deutschen IT-Dienstleistern (2011): Alle verwenden auch Java (neben anderen Sprachen) und erwarten Java-Kenntnisse bei Bewerbern: [http://www.heise.de/developer/meldung/Umfrage-Java-ist-verbreitetste-Programmiersprache-bei-IT-Dienstleistern-1198249.html] Nach dem TIOBE Programming Community Index [https://www.tiobe.com/tiobe-index/] steht Java momentan auf Platz 1.
  - Gut mit Software-Werkzeugen und Literatur unterstützt.
  - Vermeidet bestimmte Probleme.

    Manche Fehler, die man in C++ machen kann, kann man in Java nicht machen. Dafür hardwarenahe Dinge nicht auszuprobieren.

Zur Programmiersprache (2)

Auf die genaue Sprache kommt es nicht an:

- Wenn man Programmiersprachen mit Fremdsprachen vergleicht, sind Wortschatz und Syntax viel einfacher.

  Der wesentliche Aufwand beim "Programmieren lernen" sind nicht so sehr die spezifischen Konstrukte einer Programmiersprache, als zu lernen, sich in Computer hineinzudenken, und einen Schatz von Mustern für bestimmte Aufgaben im Kopf zu haben, die man nach Bedarf aktivieren und anpassen kann. Diese Muster sind aber in weiten Grenzen unabhängig von der Programmiersprache, außer bei ganz anderen Programmierparadigmen, wie etwa logischer Programmierung. Viele Sprachen stammen wie Java aus der "C-Familie".

- Sie werden noch viele andere Programmiersprachen lernen müssen.

  Z.B. C++, C#, JavaScript, PHP, SQL, sh, Prolog, verschiedene Assembler.

Motivation (1)

Warum sollten Sie programmieren lernen?

- Grundlegende handwerkliche Fähigkeit, im Berufsleben eines Informatikers notwendig.

  Selbst wenn Sie nicht an der Entwicklung wirklich großer Programme mitwirken, ist Programmierung z.B. auch nötig, um Inhalte von Datenbanken schön aufbereitet ins Netz zu stellen.

- Komplexere Software-Werkzeuge bieten für Nicht-Standard-Aufgaben meist Möglichkeiten, die ähnlich zu einer Programmiersprache sind.

  Z.B. ist die Datenbanksprache SQL zwar keine allgemeine Programmiersprache, aber hat doch vieles gemeinsam mit Programmiersprachen. Programmierkenntnisse erleichtern das Erlernen von SQL.

Motivation (2)

Warum sollten Sie programmieren lernen? (Forts.)

- Vieles in der Informatik kann man besser verstehen, wenn man es im Prinzip auch selbst programmieren könnte.
- Es erleichtert Verhandlungen mit Programmierern, wenn man auch selber programmieren kann.
- Meine Kollegen und ich sind der Meinung, dass niemand ein Diplom/Bachelor-Abschluß in Informatik, Bioinformatik, oder Wirtschaftsinformatik bekommen sollte, der nicht programmieren kann.

Motivation (3)

Warum macht Programmieren Spaß?

- Man kann mit relativ wenig Aufwand komplexe Maschinen und virtuelle Welten (Spiele) konstruieren.

  Oder reale Maschinen / elektronische Schaltungen (Microcontroller).

- Man kann sich das Leben im Umgang mit dem Computer etwas erleichtern.

  Stupide manuelle Tätigkeiten durch kurzes Programm automatisieren.

- Den Computer beherrschen statt umgekehrt.

  Computer-Experte sein: Dazu gehören Programmierkenntnisse, aber auch Hardware- und Betriebssystem-Kenntnisse.

- Konkrete Anwendung von Mathematik.

Motivation (4)

Warum ist Programmieren eine Herausforderung?

- Computer befolgen genau die gegebenen Regeln (das Programm), aber können kein bißchen selbst denken.

  Nichts versteht sich von selbst.

- Man muss bei der Programmierung also an alles denken: Alle möglichen Fälle, auch Ausnahmen.

  Ein Programm ist wie ein mathematischer Beweis, vielleicht sogar hinsichtlich der Präzision noch anspruchsvoller. Man kann kein Glied in der Kette auslassen, oder sagen, “wie man leicht sieht” / “trivialerweise gilt”.

- Man muss sich in die Maschine hineindenken.

  Es sieht zwar wie ein Text aus, muss aber funktionieren wie Zahnräder, die ineinander greifen. Der Text wird durch eine Maschine verarbeitet.

- Hier muss man Perfektionist sein.

Inhalt

1 Vorlesungsinhalte

2 Organisatorisches

3 Hausaufgaben, Prüfung

4 Arbeitsmittel

5 Ratschläge

Ansprechpartner (1)

Dozent: Prof. Dr. Stefan Brass

- Email: brass@informatik.uni-halle.de

  Betreff-Zeile sollte mit [oop18] beginnen, möglichst aussagefähig.

  Falls keine Antwort: nochmal senden.

- Büro: Von-Seckendorff-Platz 1, Raum 313.
- Telefon: 0345/55-24740.
- Sprechstunde: Montags, 12^{00}–13^{00}.
- Frühere Unis: Braunschweig, Dortmund, Hannover, Hildesheim, Pittsburgh, Gießen, Clausthal.
- Spezialgebiet: Datenbanken, Wissensrepräsentation.

  Oracle8 Certified DBA. IBM Certified Advanced DBA (DB2 8.1).

Ansprechpartner (2)

Übung (Gruppen 2, 7, 8): Annett Thüring

- Büro: Von-Seckendorff-Platz 1, Raum 319.
- Telefon: 0345/55-24739.
- Email: thuering@informatik.uni-halle.de

Übung (Gruppen 1, 3): Mario Wenzel

- Büro: Von-Seckendorff-Platz 1, Raum 315.
- Telefon: 0345/55-24776.
- Email: mario.wenzel@informatik.uni-halle.de

Ansprechpartner (3)

Übung (Gruppen 4, 6): Vaibhav Kasturia

- Büro: Von-Seckendorff-Platz 1, Raum 211.
- Telefon: 0345/55-24718.
- Email: vaibhav.kasturia@informatik.uni-halle.de

Übung (Gruppe 5): Janek Bevendorff

- Büro: Von-Seckendorff-Platz 1, Raum ?.
- Telefon: 0345/55-24718.
- Email: janek.bevendorff@informatik.uni-halle.de

Ansprechpartner (4)

Sekretärin: Ramona Vahrenhold

- Büro: Von-Seckendorff-Platz 1, Raum 324.
- Telefon: 0345/55-24750, Fax: 0345/55-27333.
- Email: vahrenhold@informatik.uni-halle.de

Rechnerbetriebsgruppe:

- Poolaufsicht (studentische Hilfskräfte): Raum 318.
- Daniel Trull: Raum 320.

  Telefon: 0345/55-24717, EMail: daniel.trull@informatik.uni-halle.de
- Leiterin der Rechnerbetriebsgruppe ist Frau Thüring.

Webseiten

- http://www.informatik.uni-halle.de/~brass/oop18/
  - Folien (Kurzfassung in Vorlesung, Langfassung als Skript).

    Die Folien werden jeweils vor der Vorlesung ins Web gestellt. Es gibt auch eine erweiterte Fassung, u.a. mit selten benötigten Java-Konstrukten und Vergleichen zu C++. Diese Zusätze sind nicht prüfungsrelevant.
  - Alte Klausuren und Programmiertests.
  - Nützliche Links.
- StudIP: https://studip.uni-halle.de/dispatch.php/course/details?sem_id=5cf5e121ed5d8e5d635855734479bab
  - Anmeldung zu Vorlesung und Übungsgruppen.
  - Link zu Übungsplattform (Hausaufgaben).
  - Forum (z.B. auch Fragen zu Hausaufgaben).

Zeit und Ort (1)

Vorlesung (2 SWS):

- Dienstags, 10^{15}–11^{45}, Raum 3.28.

  302 Teilnehmer in StudIP, Raum hat 196 Plätze. Z.B. nochmal Di 18–20?

Übung am Rechner (2 SWS):

- Acht Gruppen, je max. 27 Teilnehmer, Beginn 22./23.10.:

Nr	Tag	Zeit	Raum	Leiter	Teiln.
1/8	Montag	12^{15}–13^{45}	3.34/3.02	MW/AT	71
2	Montag	14^{15}–15^{45}	3.34	AT	47
3	Montag	16^{15}–17^{45}	3.34	MW	43
4/5	Dienstag	12^{15}–13^{45}	3.34/3.02	VK/JB	85
6/7	Dienstag	14^{15}–15^{45}	3.34/3.02	VK/AT	62

9. Gruppe in Vorbereitung. Freiwillige Umverteilung. StudIP-Link unter Nr.

Zeit und Ort (2)

Anmeldung zu Übungsgruppen:

- Tragen Sie sich in StudIP für eine Gruppe ein.

  [http://studip.uni-halle.de/]. Direkte Links auf der Vorlesungs-Webseite:
  [http://www.informatik.uni-halle.de/~brass/oop18/termine.html].
  Falls noch kein StudIP-Zugang: Papierliste hier.

- Wir garantieren, dass Sie einen Platz in einer der Übungsgruppen bekommen.

  Wenn Sie zur Teilnahme an dieser Vorlesung verpflichtet sind.

- Wir können leider nicht garantieren, dass Sie einen Platz in einer bestimmten Gruppe bekommen.

  Bitte wechseln Sie möglichst aus den überfüllten Gruppen in eine andere.
  Wenn sich das Problem nicht freiwillig löst, werden wir umverteilen.

Zeit und Ort (3)

Anmeldung zu Übungsgruppen, Forts.:

- Für die Anmeldung zu einer Übungsgruppe über StudIP brauchen Sie einen Benutzernamen und ein Passwort, das Ihnen mit der Immatrikulationsbescheinigung zugegangen sein müsste.

  Der Benutzername identifiziert Sie eindeutig im System (Vor- und Nachname sind nicht immer eindeutig). Das Passwort sollten nur Sie wissen.
  Es dient als Ausweis, dass Sie auch wirklich Sie sind.

- Bitte tragen Sie sich bei StudIP sowohl
  - für die Übungsgruppe Ihrer Wahl, als auch
  - für die Vorlesung selbst ein.

    Gelegentlich werden wichtige Mitteilungen über diesen Verteiler versendet.

Linux

- Die Rechner in den Pools laufen unter Linux.

  Eventuell haben sie eine "Dual Boot" Möglichkeit (Windows oder Linux), aber es ist für uns einfacher, wenn alle in den Übungen Linux verwenden.

- Es ist zu empfehlen, Linux zu lernen.

  Früher oder später brauchen Sie es in Ihrem Studium (und im Beruf).
  Die Horizonterweiterung ("nicht alles ist Windows!") ist wichtig und hilfreich.
  Linux ist "open source" und man kann im Prinzip alle Interna anschauen.

- Die Programmierung in den Übungen geschieht über eine Webschnittstelle (YAPEX, mit automatischen Tests), Sie brauchen also sehr wenig Linux.

  Zu Hause können Sie natürlich unter Windows arbeiten. Die Hausaufgaben müssen über YAPEX angegeben werden, aber sie können die Programme vorher mit anderen Werkzeugen entwickeln und ausprobieren.

Tutorium / OOP-Sprechstunde

- Wenn Sie Hilfe brauchen, bieten wir folgenden optionalen Termin:

Tag	Zeit	Raum
Donnerstag	12^{15}–13^{45}	3.34

- Es sind hier zwei Tutoren anwesend, also Studierende in höheren Semestern.
- Sie können beliebige Fragen zu OOP stellen.
- Da es im Rechnerpool ist, ggf. auch mit mehr Hilfe üben.
- Das Tutorium ist nur gedacht für Studierende, die mit der Programmierung Schwierigkeiten haben.

Anmeldung: Übersicht / Checkliste

Um dieses Modul erfolgreich abzuschließen, müssen Sie

- sich bei StudIP für eine Übungsgruppe anmelden,
- sich zum Modul anmelden (Löwenportal, bis ca. 28.10.2018),
- die Vorlesung nacharbeiten (Skript, Buch, ausprobieren),
- Hausaufgaben bearbeiten und abgeben,
  und dabei 50% der "Theoriepunkte" sammeln,
- in den Übungen ausreichend aktiv mitarbeiten,
  und dabei durch Lösen praktischer Programmieraufgaben am Rechner mindestens 50% der "Praxispunkte" bekommen,
- sich zur Prüfung anmelden (Löwenportal),
- die Prüfung bestehen (voraussichtlich 26.02./26.03.2019).

Modulanmeldung

- Für fast alle Studiengänge ist die Modulanmeldung über das Löwenportal Pflicht.

  [http://loewenportal.uni-halle.de/]

  Anleitung im grünen Kasten rechts unter Weitere Einstellungen/Hilfe.

  Falls über Löwenportal nicht möglich, dann im Prüfungsamt.

- Die Modulanmeldung ist nur bis zum 28.10.2018 möglich.

  Im Notfall kann man im Prüfungsamt fragen, aber die Mitarbeiter dort freuen sich nicht über die Extra-Arbeit. Machen Sie es besser gleich.

- Die Modulanmeldung ist zwingende Voraussetzung für die spätere Anmeldung zur Prüfung.

  Ein Zweck der Modulanmeldung ist die Prüfung von Voraussetzungen, dieses Modul hat aber keine Voraussetzungen.

Inhalt

1 Vorlesungsinhalte

2 Organisatorisches

3 Hausaufgaben, Prüfung

4 Arbeitsmittel

5 Ratschläge

Studienleistung

- Die Studienleistung eines Moduls (einer Lehrveranstaltung) ist Voraussetzung für den erfolgreichen Abschluss des Moduls, hat aber selbst keinen Einfluss auf die Note.
- Die Studienleistung dieses Moduls besteht aus:
  - Erreichen von 50% der "Theoriepunkte" (Hausaufgaben),
  - Erreichen von 50% der Praxispunkte in den Übungen.

    Im Gegensatz zu den Hausaufgaben gibt es hier eine Zeitbeschränkung und weniger Gelegenheit zum Mogeln. Im Donnerstags-Tutorium können Sie noch 1 von 2 Punkten bekommen. Wenn Sie gegen Ende nicht mehrere Aufgaben ohne Hilfe schaffen, sind Sie nicht reif für die Klausur (wichtiger Selbsttest).
- Ein Teil nicht erreicht $\Rightarrow$ komplette Studienleistung nächstes Jahr neu (bei längerer Krankheit etc. Dozenten fragen).

Eigentlich sollte kein Druck nötig sein ...

- Programmieren sollte Ihnen Spaß machen, oder Sie sollten den Wunsch verspüren, es zu lernen.

  Sonst hätten Sie nicht Informatiker werden sollen!

  Lange Zeit nur programmieren oder unter großem Druck programmieren macht mir auch keinen Spaß, aber immer mal wieder finde ich bis heute schön.

- Unser Dilemma ist:
  - Wenn wir keinerlei Druck ausüben, fürchten wir, dass ein Teil der Studierenden nicht ausreichend Zeit in das Lernen investiert (praktische Fähigkeit lernt man durch Tun).

    Z.B. ist der Druck in anderen Lehrveranstaltungen größer. Oder Sie glauben, es würde reichen, drei Tage vor der Klausur zu beginnen.

  - Ohne ausreichend Vorbereitung wird das Klausurergebnis schlecht sein.

    Das fällt dann auch auf die Dozenten zurück.

Hausaufgaben (1)

- Es gibt wöchentlich Hausaufgaben.

  Zum größten Teil Programmieraufgaben. Die Aufgaben werden Dienstag abend ins StudIP gestellt. Sie sind dann bis zum Montag der nächsten Woche (11:00) über YAPEX bzw. die Übungsplattform im StudIP abzugeben. Programme bitte als Quellcode (`.java`), in ASCII codiert (ohne Umlaute). Das Programm muss durch den Complier von YAPEX laufen und alle Test bestehen (auch die versteckten). Quellcode nicht als Word-Datei abgeben! Wenn das Programm nicht durch den Compiler läuft, gibt es 0 Punkte. Für theoretische Aufgaben wird ASCII oder PDF akzeptiert.

- Leider können nicht alle Aufgaben korrigiert werden.

  Oft können Sie selbst feststellen, ob Sie die Aufgabe gelöst haben. Fragen Sie, wenn sie unsicher sind. Wird ein automatischer Test bestanden, gibt es die volle Punktzahl, sonst 0 Punkte. Wir machen dann nur stichprobenartige Kontrollen. Tricks/Täuschungsversuche können zu negativen Punkten führen.

Hausaufgaben (2)

- Die Aufgaben sind einzeln zu bearbeiten.

  Eigentlich wäre Gruppenarbeit gut, aber die Erfahrung zeigt, dass manche Teilnehmer dann nicht genug tun. Selbstverständlich ist es sozial, anderen Studierenden bei Problemen zu helfen. Wenn man aber nur die fertige Lösung zum Kopieren gibt, ist das keine echte Hilfe, sondern nimmt dem Betreffenden die Möglichkeit, das dringend benötigte Wissen und Können zu erwerben. Setzen Sie sich vielleicht an den Rechner dazu, und helfen Sie aber jedes Mal nur ein Stück weit. Lassen Sie dem schwächeren Studenten genügend Zeit, selbst zu denken, und lassen Sie ihn auch selbst tippen. Dann kommt nicht ein identisches Programm heraus.

- Zu ähnliche (abgeschriebene) Abgaben können mit 0 Punkten für alle Beteiligten bewertet werden.

  In nicht ganz klaren Fällen werden Sie Ihre Abgabe vor der ganzen Gruppe erklären müssen, oder zu einem mündlichen Test beim Professor bestellt.
  Es gibt Plagiats-Erkennungssoftware. Wiederholungsfall: ggf. Täuschungsversuch!

Hausaufgaben (3)

- Die Hausaufgaben werden von den Tutoren (Studenten aus höherem Semester) korrigiert.

  Bei Programmieraufgaben gibt es bei der Abgabe eine automatische Vorkontrolle durch die Übungsplattform: Besser nicht bis zur letzten Minute warten, damit ggf. noch Korrektur/Hilfe möglich.

- Falls Sie die Korrektur nicht verstehen, fragen Sie bitte.

  Fehler kommen leider gelegentlich vor. Selbst wenn sich am Ende herausstellt, dass der Punktabzug berechtigt war, haben bei der Diskussion beide Seiten etwas gelernt. "Wer nicht fragt, bleibt dumm."

- Sie müssen mindestens 50% der Punkte erreichen.

  Feilschen Sie nicht wegen einem Punkt. Wenn Sie eindeutig einen Fehler gemacht haben, lohnt es sich meist nicht, über einen einzelnen Punkt zu reden. Für die Note ist nur die Klausur wichtig. Wenn Sie am Ende 49% haben, können Sie dann noch mal über einzelne Punkte diskutieren.

Hausaufgaben (4)

- Übliche Regeln für guten Programmierstil müssen eingehalten werden, sonst Punktabzug. Z.B.:
  - Sinnvolle Formatierung.

    Zeilenumbrüche, Einrückungen ähnlich wie in der Vorlesung.
  - Ausreichend Kommentare, verständliche Bezeichner.

    Kann der Tutor Ihr Programm nach ca. 5 Minuten noch nicht verstehen, bekommen Sie 0 Punkte.
  - Keine üblen Programmiertricks.
- Wenn Ihre Abgabe nicht mit `javac` (JDK für Java SE 8) unter Linux compilierbar ist: 0 Punkte.

  Rechtzeitig anfangen, damit Sie bei Problemen noch fragen können (Forum).

  Da keine spezielle Ordnerstruktur: Keine `package`-Deklaration.
- Halten Sie sich genau an die Spezifikation aus der Aufgabe!

Wichtig: Vorlesung nacharbeiten (1)

- Obwohl es nicht explizit auf dem Aufgabenblatt steht, ist jede Woche eine Hausaufgabe, die Vorlesung nachzuarbeiten:
  - Lesen Sie die ausführliche Fassung der Folien.
  - Gehen Sie dabei auch Ihre Aufzeichnungen aus der Vorlesung nochmal durch, notieren Sie sich Fragen.

    Und sorgen Sie für die Klärung dieser Fragen. Z.B. in nächster Vorlesung, Übung oder Forum stellen. Oder selbst ausprobieren. Im Internet suchen.
  - Lesen Sie zusätzliche Literatur: Lehrbücher, Internet-Quellen.
  - Probieren Sie kritische Dinge am Rechner aus.
  - Schreiben Sie sich mit eigenen Worten das Wichtigste auf.

    Am Ende eigene "Quick Reference". Bei der Klausur sind 5 Blätter erlaubt.
  - Reden Sie mit anderen Studierenden (bilden Sie Lerngruppen).

Wichtig: Vorlesung nacharbeiten (2)

- Wenn Sie die Vorlesung nicht nacharbeiten (und nicht bereits über gute Java-Programmierkenntnisse verfügen), werden Sie in dieser Vorlesung untergehen.

  In der nächsten Woche kommt schon wieder neuer Stoff.

- In der Modulbeschreibung stehen 90 Stunden Arbeitszeit (im Semester) für Hausaufgaben und Selbststudium.

  Durchschnittliche Arbeitszeit: Studenten ohne Vorkenntnisse eher mehr, mit Vorkenntnissen eher weniger (hängt aber auch von anderen Faktoren ab).

  - Z.B. 2 Stunden pro Woche für Hausaufgaben.

    Wenn Sie wesentlich mehr brauchen, stimmt etwas nicht.

  - Z.B. 4 Stunden pro Woche zur Nacharbeit.

    Ggf. müssen Sie aus Zeitgründen einen Teil auf die Semesterpause verschieben (Klausurvorbereitung), aber nicht so viel, das Sie im Semester abgehängt werden.

Übung

- In der Übung werden die Hausaufgaben besprochen.

  Nutzen Sie die Gelegenheit, Alternativen zu Ihrer Lösung anzuschauen. Auch interessante Fehler werden besprochen. Bringen Sie interessante Aspekte Ihrer Lösung aktiv ein, nicht nur nach Aufforderung. Seien Sie bereit, Ihre Lösung auch vorzuführen und zu erklären. Die Diskussion der nicht korrigierten Aufgaben ist besonders wichtig. Wenn niemand seine Lösung vorführen will, wird die Aufgabe möglicherweise nicht besprochen.

- Sie können Fragen in einer kleineren Gruppe stellen.

  Sie können Fragen selbstverständlich auch in der Vorlesung stellen!

- Lösen einer praktischen Programmieraufgabe am Rechner, für die es Praxispunkte gibt.

  Sie müssen 50% der Praxispunkte erwerben, um das Modul erfolgreich abschließen zu können. Das impliziert eine gewisse Anwesenheitspflicht.

Praxispunkte (1)

- Normale Übungstermine sind aufgeteilt in
  - Besprechung von Fragen und Hausaufgaben (ca. 45 min),
  - Praktische Programmieraufgabe (ca. 45 min).
- Jede zweite Woche (ab 5./6. November, 6 Termine) gibt es stattdessen ein "Programmiertestat":
  - Sie müssen eine Programmieraufgabe am Rechner innerhalb von 60 min so lösen, dass sie durch den Compiler läuft und alle Tests besteht.

    Fünf DIN A4 Blätter (Vorder- und Rückseite) mit Notizen / "Quick Reference" sind erlaubt. Aber keine Lösung aus anderer Gruppe!
  - Wenn das Programm rechtzeitig vor Ende der Zeit vollständig funktioniert, gibt es 2 "Praxispunkte".

    Das Programm muss über YAPEX eingereicht werden (wie HA).

Praxispunkte (2)

- Wenn es nicht geklappt hat, können Sie beim Tutorium an den beiden folgenden Donnerstagen noch einen Praxispunkt bekommen, unter folgenden Bedingungen:
  - Bei Ihrem letzten Versuch während des Programmiertestats waren wenigstens Ansätze erkennbar.

    Sie müssen also teilgenommen haben.
  - Sie führen im Tutorium eine korrigierte Lösung vor.

    Sie müssen persönlich vorbeikommen.
  - Sie können auch den Fehler/das Problem von Ihrer letzten Version im Praxistest erklären.

    Wenn Sie den Fehler nicht selbst finden, können Sie auch Hilfe bekommen.

Praxispunkte (3)

- Es ist nicht ausgeschlossen, dass einzelne Teilnehmer versuchen, zu mogeln.

  Dazu gehört auch, sich bei den Teilnehmern von früheren Übungsgruppen nach den Aufgaben zu erkundigen. Wir behalten uns allerdings das Recht vor, in unterschiedlichen Gruppen auch unterschiedliche Aufgaben zu verwenden.

- Wenn wir Sie erwischen, bekommen Sie 0 Punkte.

  Im Wiederholungsfall können Sie von der Vorlesung ausgeschlossen werden.

- Bedenken Sie, dass die praktischen Übungen
  - ein wichtiger Beitrag zum Lernerfolg sind,
  - wertvolles Feedback geben.

    Wir gehen davon aus, dass ein erfolgreicher Vorlesungsteilnehmer diese Aufgaben ohne Hilfe lösen kann (außer in den ersten ca. 3 Übungen, wenn man noch praktische Tipps braucht, und bei seltenen Problemen).

Praxispunkte (4)

- 2013 hatten wir ein "Programmiertestat" (zwei Versuche, keine Möglichkeit für nachträglichen Punkt).
- Von den 8 Studenten, die trotz nicht-bestandenem Programmiertestat an der Klausur teilgenommen haben, sind 7 durchgefallen (88%).

  Der achte hatte eine 3.0 in der Klausur (und Programmiertestat im 3. Versuch).
- Dieses Jahr sollten 50% Praxispunkte einfach sein.

  Wenn Sie immer zur Übung kommen, können Sie spätestens am Donnerstag 1/2 Punkten bekommen. Wir zwingen Sie, das Feedback abzuholen, aber es ist kein ernster Filter mehr: Die Konsequenzen müssen Sie selber ziehen.
- Wenn Sie die Programme nie alleine zum Laufen bekommen, wäre es sehr riskant, zur Klausur zu gehen!

Studienleistung und Klausur

- Sie können zwar an der Klausur teilnehmen, auch wenn Sie die Studienleistung nicht erfolgreich abschliessen.

  Das wäre aber normalerweise nicht klug: Die Klausur testet ähnliche Fähigkeiten wie Hausaufgaben und die Programmiertestate in den Übungen.

- Sie müssen dann aber im nächsten Jahr nochmals Hausaufgaben machen und Praxispunkte erwerben.

  Mit dieser zusätzlichen Übung wären Sie auch für die Klausur besser gerüstet und hätten eine bessere Note bekommen. Bestandene Klausuren können aber nicht wiederholt werden.

- Falls Sie die Studienleistung aufgrund von besonderen Belastungen (längerer Krankheit etc.) nicht erreicht haben, wenden Sie sich an den Dozenten (ggf. Ersatzleistung).

  Da das zusätzliche Arbeit macht, gibt es das nur in klaren Fällen.

Prüfung (1)

- Die Prüfung erfolgt voraussichtlich elektronisch (am Rechner).
- Termin: 26.02.2019, 10^{00}-12^{00}, Raum 5.09 u.a.

  Der Termin ist noch nicht ganz endgültig, achten Sie auf weitere Ankündigungen. Melden Sie ggf. Konflikte möglichst frühzeitig.
- Man muss sich spätestens zwei Wochen vorher zur Klausur anmelden (im Löwenportal).
- Man kann sich bis eine Woche vorher ohne Angabe von Gründen wieder abmelden.

  Alle Angaben zur Prüfungsordnung sind ohne Gewähr (möglicherweise auch abhängig vom Studiengang).
- Wenn man angemeldet ist und nicht zur Klausur erscheint, bekommt man eine 5 (bei Krankheit zum Arzt gehen!).

Prüfung (2)

- Es sind fünf DIN A4-Blätter mit beliebigen Notizen erlaubt.

  Sie dürfen die Vorder- und Rückseite bedrucken oder beschreiben. Es ist eine gute Übung, sich selbst eine "Quick Reference" zu machen, und das Wichtigste aus der Vorlesung nochmal auf begrenztem Platz mit eigener Strukturierung zusammenzustellen.

- Bücher/Aktenordner sind nicht erlaubt.

- Da die Programmieraufgaben von YAPEX korrigiert werden, sollte mann durch Hausaufgaben und Programmiertestate gut vorbereitet sein.

- Es gibt auch einige Ankreuzaufgaben etc. (ILIAS), auch das wird vorher geübt.

  Eventuell gibt es auch eine spezielle Probeklausur.

Prüfung (3)

- Es gibt Beispiele alter Klausuren auf der Webseite zur Vorlesung.
- Im Rahmen Ihrer Prüfungsvorbereitung sollten Sie diese Klausuren durchrechnen.

  Da es keine elektronischen Klausuren waren, kann es natürlich Unterschiede geben. Die Programmieraufgaben sollten aber ähnlich sein, auch ein Teil der anderen Aufgaben. Manche alten Aufgaben funktionieren elektronisch nicht (Sie haben ja mit YAPEX einen Compiler).
- Voraussichtlich gibt es in der Semesterpause ein ganztägiges Treffen mit Frau Thüring, das der Prüfungsvorbereitung und Besprechung von Fragen vor der Klausur dient.

Prüfung (4)

- Die Note für das Modul basiert nur auf der Klausur.

  Der Dozent behält sich das Recht vor, in seltenen Ausnahmefällen Extrapunkte (bis max. 5% der Klausurpunkte) zu vergeben für das Finden von Fehlern im Skript, außergewöhnlich gute Hausaufgabenergebnisse verbunden mit entsprechend aktiver Übungsteilnahme, oder eventuell andere Aktivitäten, die allen Teilnehmern zugute kommen.

- Es wäre nicht klug, mit den Hausaufgaben vorzeitig aufzuhören (wenn Sie 50% sicher erreicht haben).

  Die Hausaufgaben sind eine wichtige Prüfungsvorbereitung.

- Falls Sie 60% der Klausurpunkte erreicht haben, haben Sie garantiert bestanden.

  Mit 50% ist das Bestehen noch nicht garantiert. Ab 95% gibt es die bestmögliche Zensur (1.0). Beide Grenzen werden möglicherweise gesenkt.

Prüfung (5)

- Falls Sie an der Klausur teilnehmen und bestehen, gibt es keine Möglichkeit zur Zensur-Verbesserung.
- Falls Sie bei der Klausur durchfallen, gibt es noch eine Wiederholungsmöglichkeit.

  Wenn Sie nicht erscheinen, obwohl Sie angemeldet sind, und keine Krankschreibung vorlegen, sind Sie automatisch durchgefallen. Es ist auch möglich, sich erst zum 2. Termin anzumelden. Dann gibt es aber keine Wiederholungsmöglichkeit im Rahmen dieser Vorlesung mehr.
  (Bei Krankheit oder anderen sehr guten Gründen fragen Sie den Dozenten möglichst umgehend nach einer mündlichen Prüfung — aber kein Anspruch).
- Nachholtermin: voraussichtlich 26.03.2019, 10–12.

  Bitte informieren Sie sich über Ihre Prüfungsordnung, inwieweit eine extra Anmeldung nötig ist, oder Sie automatisch angemeldet sind, falls Sie die erste Klausur nicht bestanden haben. Achten Sie auf Termin-Verschiebungen!

Prüfung (6)

- Falls Sie die Nachholklausur auch nicht bestehen, müssen Sie das Modul ein Jahr später neu belegen.

  Meines Wissens können Sie den dritten Prüfungsversuch erst machen, wenn Sie das Modul nochmals belegt haben (informieren Sie sich ggf. über die genauen Regeln, ob Sie z.B. auch die Studienleistung neu machen müssen).

- Es gibt maximal drei Prüfungsversuche.

  Nach dem dritten nicht erfolgreichen Prüfungsversuch für diese Vorlesung ist für Informatiker und Wirtschaftsinformatiker das Studium beendet.
  Im Laufe des Studiums sind nur 7 bzw. 8 zweite Wiederholungsprüfungen erlaubt, eventuell nur mit Antrag. Endgültiges Durchfallen (nach drei Prüfungsversuchen) reduziert Ihre Optionen zum Studienfachwechsel deutlich (z.B. Wirtschaftsinformatik $\rightarrow$ Wirtschaftswissenschaften ist dann nicht mehr möglich). Lassen Sie sich vor dem letzten Prüfungsversuch unbedingt kompetent beraten.

Statistik (1)

Note	2012/13		2013/14		2. Termin	
1.0	16	(14%)	22	(24%)	1	(6%)
1.3	13	(12%)	13	(14%)	1	(6%)
1.7	5	(4%)	12	(13%)	0	(0%)
2.0	8	(7%)	8	(9%)	4	(22%)
2.3	12	(11%)	6	(7%)	1	(6%)
2.7	12	(11%)	4	(4%)	0	(0%)
3.0	7	(6%)	4	(4%)	3	(17%)
3.3	7	(6%)	1	(1%)	1	(6%)
3.7	6	(5%)	0	(0%)	0	(0%)
4.0	7	(6%)	0	(0%)	0	(0%)
5.0	19	(17%)	21	(23%)	7	(39%)

2013/14 haben sich nur 51% der in StudIP für das Modul eingetragenen Studenten zur ersten Klausur angemeldet (2012/13: 71%).

Statistik (2)

- 2013/14 haben 7 der Teilnehmer, die bei der ersten Klausur durchgefallen waren, an der zweiten erneut teilgenommen.

  Davon ist einer wieder durchgefallen, zwei hatten eine 2.0.

- Bei der Klausur 2013/14 haben wir nach Vorkenntnissen ("Programmierkenntnisse vor der Vorlesung") gefragt:
  - Keine Vorkenntnisse: 34%

    Durchschnittsnote 2.3, sehr gut (1.0/1.3): 22%, nicht bestanden: 11%.
  - Etwas Vorkenntnisse: 41%

    Durchschnittsnote 2.4, sehr gut (1.0/1.3): 42%, nicht bestanden: 24%.

    Verteilung stark an den beiden Enden: Manche haben aus dem Vorwissen etwas gemacht, andere haben sich zu sehr darauf verlassen.
  - Viel Vorkenntnisse: 12%

    Durchschnittsnote 1.1, sehr gut (1.0/1.3): 100%, nicht bestanden: 0%.

Geben Sie nicht auf!

- Die bisherige Erfahrung zeigt, dass ein nicht unwesentlicher Teil der Studierenden, die diese Vorlesung anfangen, nicht bis zum Ende dabei bleibt (auch nicht zur Klausur kommt).
- Zum Teil lag es am Programmiertest (jetzt verändert).

  Sie bekommen früher und häufiger sanfteres Feedback, mit weniger Druck.

  Spätestens in der Klausur werden aber die gleichen Fähigkeiten verlangt.

  Tatsächlich war der Programmiertest ein guter Indikator für den Klausur-Erfolg.
- Zum Teil merken die Studierenden aber zu spät, dass sie abgehängt sind, oder es zu viel Arbeit ist.

  Studium in der Regelstudienzeit verlangt eher mehr als 40 Stunden/Woche (zumindest während der Vorlesungszeit).
- Bedenken Sie Modul-Abhängigkeiten.

  Sie können z.B. “Datenstrukturen und effiziente Algorithmen I” im 2. Semester nur besuchen, wenn Sie diese Vorlesung erfolgreich abgeschlossen haben.

Inhalt

1 Vorlesungsinhalte

2 Organisatorisches

3 Hausaufgaben, Prüfung

4 Arbeitsmittel

5 Ratschläge

Rechnernutzung, Software (1)

Rechnerpools:

- PC-Pool: Raum 3.34
- Multimedia-Pool: Raum 3.02
- Siehe: [http://www.informatik.uni-halle.de/studium/pools/]

  Sie können die Pools auch außerhalb Ihrer Übungszeiten nutzen, wenn sie nicht belegt sind (falls dort eine kleine Übung läuft, die nicht alle Rechner braucht, können Sie zumindest still auf der Empore arbeiten).

- Damit Sie auf den Rechnern in den Pools 3.34/3.02 arbeiten können, muss ein Benutzerkonto eingerichtet werden.

  Die Zugangsdaten entsprechen zu Beginn des Studiums dem Account, welchen Sie mit Ihren Immatrikulationsunterlagen erhalten haben.
  Bitte prüfen Sie vor der ersten Übung, ob Sie sich mit Ihren Zugangsdaten in den Pools anmelden können. Falls eine Anmeldung nicht möglich ist, so melden Sie sich bitte bei der Rechnerbetriebsgruppe (Räume: 318, 319, 320).

Rechnernutzung, Software (2)

Falls Sie einen eigenen PC haben:

- Sie brauchen mindestens das JDK ("Java Development Kit") für Java SE 8 von Sun/Oracle:
  [http://www.oracle.com/technetwork/java/javase/downloads/]

  Sun hat Java entwickelt, wurde aber inzwischen von Oracle gekauft.
  SE steht für "Standard Edition". Die Unterschiede zwischen den Versionen 6 bis 11 der Sprache sind für diese Vorlesung nicht wichtig. Java 8 wird noch unterstützt, Java 10 ist aktuelle "Feature Release", Java 11 ist erste "Long Term Support" Release (das nur frei zur Entwicklung, oder GPL).
  Das JDK enthält das JRE (Java Runtime Environment), das zur Ausführung von Java-Programmen nötig ist. Das JRE ist auf vielen Rechnern schon installiert (spätestens mit dem ersten Java-Programm). Zur Programmentwicklung brauchen Sie die zusätzlichen Bestandteile des JDK (insbesondere den Compiler).
- Entwicklungsumgebungen (z.B. Eclipse, Netbeans, BlueJ) enthalten dies oft schon.

Rechnernutzung, Software (3)

Falls Sie einen eigenen PC haben (Forts.):

- Nach der Installation unter Windows müssen Sie den Suchpfad für Programme so erweitern, das die Programme `java` und `javac` auch gefunden werden.

  Öffnen Sie die Systemsteuerung (Control Panel), System und Sicherheit, System, Erweiterte Systemeinstellungen (Change settings), Erweitert (Advanced), Umgebungsvariablen (Environment Variables).
  Dort die Variable "`Path`" ändern und an den aktuellen Wert nach einem Semikolon das bin-Verzeichnis der JDK-Installation anhängen (z.B. `C:\Program Files\Java\jdk1.8.0_181\bin`).

- Anschliessend starten Sie unter "Zubehör" die "Eingabeaufforderung" ("Command Prompt") und geben z.B. "`java -version`" und "`javac`" ein.

  Nach der Änderung von "`Path`" die Eingabeaufforderung neu starten.

Rechnernutzung, Software (4)

Falls Sie einen eigenen PC haben (Forts.):

- Sie brauchen außerdem einen Editor.

  Ein Editor ist ein Programm zum Eingeben und Ändern von Text-Dateien, z.B. Java-Programmen.

- `notepad` wäre möglich, ist aber wenig komfortabel.

  Word ist nicht für Programm-Dokumente gedacht!

- `notepad++` wird häufig empfohlen (für Windows).

  [http://notepad-plus-plus.org/]

- Bei Linux sind ausreichende Editoren schon dabei.

  Z.B. `gedit`.

- Weitere (mächtige) Editoren sind z.B. `gvim`, `emacs`.

  [http://www.vim.org/], [http://www.gnu.org/software/emacs/]

Rechnernutzung, Software (5)

Falls Sie einen eigenen PC haben (Forts.):

- Es gibt aber auch umfangreiche Programmpakete mit vielen weiteren Werkzeugen zur Programmentwicklung, sogenannte IDEs (Integrated Development Environment).
  - Eclipse [http://www.eclipse.org/] ist sehr bekannt.
  - NetBeans [http://netbeans.org/] kommt von Sun/Oracle.
  - BlueJ [http://www.bluej.org/] gilt als einfacher und wird gern in Schulen verwendet.
- Selbstverständlich werden Sie früher oder später mit einer IDE arbeiten wollen (u.a. wegen der graphischen Oberfläche).

  Man sollte beides (Compiler/Editor mit Kommandoschnittstelle und IDE) ausprobieren, auch schon in dieser Vorlesung.

Rechnernutzung, Software (6)

Kommandoschnittstelle (Compiler/Editor einzeln) vs. IDE:

- Dem in der Klausur verlangten "Programmieren auf Papier" würde ein gewöhnlicher Editor noch am nächsten kommen.

  Die IDE erkennt Syntaxfehler während Sie tippen und bietet automatische Vervollständigungen/Korrekturen an. Das ist nützlich (z.B. auch für die Praxispunkte in den Übungen), aber Sie sollten davon nicht abhängig werden.

- Die Vielzahl der Funktionen einer IDE könnte Anfänger überfordern (Eclipse hat auch noch viele Plugins).

  Mit den Einzelwerkzeugen sieht man eher, was genau passiert.

  Auch verstecken sich die Dateien nicht in einer tiefen Ordnerstruktur.

- Die schrittweise Ausführung eines Programms im Debugger einer IDE ist hilfreich zum Verständnis.

  Es gibt auch Kommandozeilen-Debugger (aber umständlicher zu bedienen).

Beschaffen Sie sich ein Lehrbuch!

- Es gibt verschiedene Lerntypen.
  Meine Vorlesung ist nicht für jeden optimal.

  Mir ist Präzision wichtig. Ich versuche alles systematisch zu definieren, inklusive der aus meiner Sicht nötigen Details. Manche Aussagen wären ohne die Details strenggenommen falsch. Im Laufe der Zeit habe ich auch viele Missverständnisse und mögliche Fehler gesehen, und häufig mein Skript erweitert, um den jeweiligen Punkt noch genauer zu klären (so wird es lang).

- Am Ende ist die Hauptsache, dass Sie programmieren können.

  Wenn Sie eine "1" wollen, sind allerdings auch Details wichtig.

- Sie können sich die Kenntnisse auch auf anderem Weg aneignen, z.B. aus einem Buch, das Ihrem Lerntyp entspricht.

- Außerdem können Sie selbstbewusster und kritischer mitdenken, wenn Sie auch andere Quellen haben.

Internet-Kurse

- Es gibt im Internet viele Java-Kurse.
  - Wenn man z.B. bei Google "Video Tutorial Java" eingibt, erhält man 1.7 Millionen Webseiten.
  - Es gibt aufgezeichnete Vorlesungen anderer Professoren.
  - Es gibt interaktive Tutorials (z.B. mit Ankreuzaufgaben).
- Eine Auswahl steht auch unter [http://users.informatik.uni-halle.de/~brass/oop18/links.html]
- Fragen Sie mich nicht, welches das beste Tutorial ist.

  Ich halte naturgemäß diese Vorlesung für die beste Möglichkeit, Java zu lernen.
  Wenn Sie gute Tutorials finden, stelle ich sie gerne auf meine Webseite.
  Wenn Sie Tutorials auf meiner Seite für schlecht halten, teilen Sie mir das mit.

Lehrbücher für Anfänger

- Peter Pepper:
  Programmieren lernen: Eine grundlegende Einführung mit Java, 3. Auflage.

  Springer, Sept. 2007, ISBN: 3-540-72363-3, 604 Seiten, 27.99 €.

- Hans-Peter Habelitz:
  Programmieren lernen mit Java.

  "Der leichte Einstieg für Programmieranfänger."
  Rheinwerk Computing, Okt. 2017, ISBN: 3-8362-5605-3, 552 Seiten, 19.90 €.

- Dirk Louis, Peter Müller:
  Java: Eine Einführung in die Programmierung.

  Carl Hanser Verlag, April 2018, ISBN: 3-446-45194-3, 389 Seiten, mit DVD, 20.00 €.

RRZN-Handbücher

- Dr. Thomas Krökertskothen, Dr. Peter Heusch: Java (1. Band).

  RRZN, ca. 248 Seiten, ca. 5.00 €.

  [http://www.rrzn.uni-hannover.de/buch.html?&no_cache=1&titel=java6]

  Im normalen Buchhandel nicht zu haben (wegen öffentlicher Förderung), nur über das Rechenzentrum zu beziehen.

  Wir haben eine Sammelbestellung über 100 Exemplare ausgelöst. Sie werden in der Übung verkauft, sobald sie eingetroffen sind.

- Fortgeschrittene Techniken und APIs (2. Band)

  U.a. Graphische Benutzeroberflächen, Netzwerkprogrammierung, Threads.

  Ca. 174 Seiten. Ca. 5.00 €. Wir haben 50 Exemplare bestellt.

- Es gibt viele weitere RRZN Handbücher, z.B. zu C, C++, C#, Unix (gut und sehr preiswert).

  [http://www.urz.uni-halle.de/dienstleistungen/handbuecher/]

Lehrbücher für Fortgeschrittene (1)

- Guido Krüger, Heiko Hansen:
  Java-Programmierung - Das Handbuch zu Java 8.

  O'Reilly Verlag, 8. Aufl., 2014, ISBN 3-95561-514-6, 1104 Seiten, 49.90 €.
  HTML-Version der 7. Aufl. kostenlos:
  [http://www.javabuch.de/download.html]

- Christian Ullenboom:
  Java ist auch eine Insel: Insel 1: Das umfassende Handbuch.

  Galileo Computing, 11. Aufl., 2014, ISBN 3-8362-2873-4, 1306 Seiten, 49.90 €.
  10. Auflage kostenlos: [http://openbook.galileocomputing.de/javainsel/]
  Zweiter Band: Java SE 8 Standard-Bibliothek: Insel 2: Das Handbuch für Java-Entwickler (ISBN: 3-8362-2874-2).
  Version für Java 7: [http://openbook.galileocomputing.de/java7/]

Lehrbücher für Fortgeschrittene (2)

- Ken Arnold, James Gosling, David Holmes:
  The Java Programming Language, 4th Edition.

  Addison Wesley, 2005, ISBN 0-321-34980-6, 891 pages, ca. 40.95 €.

  5th Ed. soll 2012/13 bei Prentice Hall erschienen sein: ISBN 0-13-276168-8.

- Cay S. Horstmann, Gary Cornell:
  Core Java, Volume 1: Fundamentals, 9th Edition.

  Prentice Hall, ISBN: 0-13-708189-8, 2012, 974 Seiten, ca. 41.95 €.

  Es gibt auch "Core Java, Volume II - Advanced Topics".

- Joshua Bloch:
  Effective Java: A Programming Language Guide, 2nd Edition.

  Addison Wesley, 2008, ISBN 0-321-35668-3, 384 pages, ca. 35.95 €.

  Keine Java-Einführung, aber viele wichtige Tipps über guten Stil und Fehlervermeidung.

Spezialthemen

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley: The Java Language Specification, 5th Ed..

  Addison Wesley, 2014, ISBN 0-13-390069-X, 758 pages, ca. 42.31 €.

  Kostenlos online: [http://docs.oracle.com/javase/specs/]

- Arnd Poetzsch-Heffter: Konzepte objektorientierter Programmierung (Mit einer Einführung in Java).

  Springer, 2009, ISBN 3540894705, 352 Seiten, 29,95 €.

- Kathy Sierra, Bert Bates: Sun Certified Programmer for Java 6 Study Guide.

  McGraw-Hill/Osborne, 2008, ISBN-13: 978-0-07-159106-5, 851 Seiten, mit CD-ROM, ca. 33.95 €.

- [http://bibliothek.uni-halle.de/]

Online-Dokumentation

- Dokumentation zu Java SE 8:
  [http://docs.oracle.com/javase/]
- Insbesondere die API ("Application Program Interface")
  Dokumentation ist fast unverzichtbar:
  [http://docs.oracle.com/javase/8/docs/api/]

  Sie dokumentiert die vordefinierten Klassen, also Programmcode von den Java-Entwicklern, den Sie in Ihren Programmen aufrufen können.
- Java SE 7: [http://docs.oracle.com/javase/7/docs/]
- Online Tutorials:
  [http://docs.oracle.com/javase/tutorial/]
- Spezifikationen:
  [http://docs.oracle.com/javase/specs/]

Inhalt

1 Vorlesungsinhalte

2 Organisatorisches

3 Hausaufgaben, Prüfung

4 Arbeitsmittel

5 Ratschläge

Vorkenntnisse (1)

- Diese Vorlesung wird von Studierenden mit sehr unterschiedlichen Vorkenntnissen besucht.
  - Einige können schon programmieren, ggf. sogar in Java.

    Bedenken Sie aber bitte, dass diese Vorlesung auch Konzepte vermitteln soll, die in einer rein praktischen Ausbildung fehlen.

    Achtung: Man kann den Punkt verpassen, wo es dann doch neu wird.
  - Für andere Teilnehmer ist die Programmierung ein völliges Neuland.

    Wir haben leider keine Lehrkapazität für zwei getrennte Vorlesungen.

    Es ist auch nicht klar, ob das wirklich helfen würde: Vorkenntnisse und Aufnahmegeschwindigkeit bleiben unterschiedlich.
- Mein Anspruch ist, dass auch die zweite Gruppe der Vorlesung folgen kann: Ich möchte den Stoff "von Grund auf" logisch und vollständig (aber zügig) präsentieren.

Vorkenntnisse (2)

- Stellen Sie Fragen, wenn Sie etwas nicht verstehen, gerne auch während der Vorlesung.

  Wer schon so lange programmiert wie ich, übersieht manchmal, dass bestimmte Dinge nicht selbstverständlich sind. Ich bin dankbar für Hinweise, die es mir erlauben, mein Skript (Foliensatz) noch zu verbessern. Andere Studierende werden Ihnen auch dankbar sein.

- Lassen Sie sich nicht einschüchtern, wenn andere mehr als Sie wissen.

  Offiziell fordert diese Vorlesung keine Vorkenntnisse, es ist also ok, wenn Sie keine haben. Es ist Ihr gutes Recht, zu fragen. Es ist auch völlig unklar, wie es gegen Ende der Vorlesung mit dem Wissen aussieht: Manche Programmier-Neulinge können sich gut in Maschinen und Formalismen hineindenken und lernen schnell. Mancher Teilnehmer mit Vorwissen hat über viele Details noch nie nachgedacht und wüßte die auch nicht.

Vorkenntnisse (3)

- Auch bei der Sprache Java gibt es einen Unterschied zwischen "aktivem" und "passivem Wortschatz":
  - Wenn Sie selbst programmieren, brauchen Sie keineswegs sämtliche Sprachkonstrukte von Java.

    Häufig gibt es mehrere Möglichkeiten, das Gleiche auszudrücken.
  - Diese Vorlesung strebt aber eine gewisse Vollständigkeit an, will also fast alle Sprachkonstrukte vorstellen (manche aber nur in der erweiterten Version der Folien).

    Durch die zusätzlichen Konstrukte werden Programme eleganter, teils auch objektiv besser. Sie können manchmal auch Fehlermeldungen besser verstehen, wenn Sie mehr von der Sprache wissen.
  - Eigenes Programmieren ist ein großer Teil der Klausur.

    Teilweise muss man aber auch gegebene Programme verstehen (natürlich ohne Konstrukte aus der erweiterten Skript-Version).

Vorkenntnisse (4)

- Manchmal gibt es kurze Exkurse (in der "extended version" der Folien und gelegentlich in der Vorlesung).
  - Z.B. nenne ich Unterschiede zu C++ oder skizziere die interne Implementierung der Sprachkonstrukte.
  - Diese Dinge sind nicht (direkt) prüfungsrelevant.

    Ich glaube, dass man durch den Kontrast auch die Sprache Java besser verstehen kann. Wenn das auf Sie zutrifft, würde es indirekt auch die Klausurergebnisse verbessern. In der Klausur wird es aber keine Aufgabe der Art "Und wie ist das in C++?" geben.
  - Vermutlich hilft diese Zusatzinformationen nur einem Teil der Studierenden.
  - Wenn Sie nicht dazu gehören, können Sie ohne Schaden eine kurze gedankliche Pause machen.

Vorkenntnisse (5)

Markierung von Folien mit Inhalt für Fortgeschrittene:

- Folien, deren Inhalt "freiwilliger Zusatzstoff" ist (nur in "extended Version") sind so wie diese markiert.
- Es ist ein Experiment: Sie können mir später sagen, ob Sie diese Trennung nützlich fanden.

  Ich bin mir auch nicht sicher, ob es immer so funktioniert: Wenn es nur eine kurze Anmerkung ist, scheint eine extra Folie Verschwendung zu sein.
- Für einen Teil der Hörer sind diese Anmerkungen vermutlich interessant, und bringen ein tieferes Verständnis.

  Das müssen nicht nur die Hörer mit den Vorkenntnissen sein.
- Die übrigen können kurz abschalten und brauchen keine Panik zu bekommen, wenn sie nicht folgen können.

Theorie vs. Praxis (1)

- Praktische Fähigkeiten wie die Programmierung erwirbt man neben dem
  - notwendigen theoretischen Wissen auch durch

    Präzision des Denkens, die sich aus theoretischem Wissen speist, ist gerade in der Programmierung wichtig.
  - praktisches Tun und Lernen aus Fehlern.
- Oft können auch Lösungen, die im Prinzip korrekt sind (funktionieren), noch verbessert werden.

  Z.B. einfacher, kürzer, übersichtlicher, leichter änderbar, resourcen-sparender (schneller/weniger CPU-Zeit oder weniger Speicherplatz). Nutzen Sie die Übung zum Austausch mit anderen Studierenden und dem Tutor (natürlich möglichst nach Abgabe der Hausaufgaben).
- Wie viel Sie lernen, hängt an der investierten Zeit.

Theorie vs. Praxis (2)

- Wenn praktische Fähigkeiten auch notwendig sind, ist dies doch eine Vorlesung an einer Universität:
  - Man darf über Java auch kritisch nachdenken.

    Mir gefällt auch nicht alles in Java. Überlegen Sie sich, was Sie als Programmiersprachen-Entwerfer anders machen würden.
  - Ziel ist auch ein möglichst leichter Übergang zu anderen Sprachen (allgemeine Konzepte!).
  - Auswendiglernen ("pauken") sollte nicht nötig sein.

    Wichtige Dinge prägen sich bei ausreichender theoretischer und praktischer Beschäftigung mit der Programmierung automatisch ein, selten benötigte Details kann man bei Bedarf nachschlagen. Das meiste ist logisch aufgebaut, und man kann die Gründe dafür verstehen.
  - Ihre Mitwirkung (Fragen, Vorschläge) ist wichtig!

Vorlesungs-Etikette

- Vermeiden Sie Verhalten, das Ihre Mitstudenten oder den Professor ablenkt:
  - Vermeiden Sie Gespräche während der Vorlesung.

    Glauben Sie nicht, dass Sie in der Masse untergehen: Der Professor kann durchaus sehen, wer sich unterhält. Wenn Sie Ihren Nachbarn etwas zur Vorlesung fragen müssen, machen Sie es leise und kurz. Wenn die Frage möglicherweise auch für andere interessant ist, stellen Sie sie offiziell (melden, ggf. rufen).
  - Wenn Sie zu spät kommen oder früher gehen müssen, setzen Sie sich möglichst an den Rand.
  - Notebooks sollten während der Vorlesung nur die Folien anzeigen (eventuell Editor, Compiler).
- Nur anwesend zu sein, reicht nicht. Sie müssen mitdenken.

  Und ggf. Fragen stellen. Sie sind erwachsen. Verschwenden Sie Ihre Zeit nicht.

Ratschläge zum Studium (1)

- Professoren freuen sich über Fragen!
- Scheuen Sie sich nicht vor anderen Studierenden, die den Eindruck machen, schon mehr zu wissen.

  Gegen Ende des Semesters kann es schon ganz anders aussehen. Es ist keine Schande, Informatik nicht schon in der Schule gehabt zu haben.
- Lesen Sie ein Buch zum Thema der Vorlesung, nicht nur das Skript. Benutzen Sie unterschiedliche Quellen.

  Seien Sie selbständig und etwas kritisch. Glauben Sie nicht etwas, nur weil Sie es zufällig gehört haben. Versuchen Sie zu verstehen, nicht auswendig zu lernen.
- Nehmen Sie das Studium ernst.

  Es gibt mehr Freiheiten als an der Schule, und die negativen Auswirkungen eines Missbrauchs dieser Freiheit zeigt sich erst mit etwas Verzögerung.
  Auch hier gilt: Ohne Fleiss kein Preis. Soviel Zeit, wie Sie jetzt haben, sich fortzubilden und Bücher zu lesen, haben Sie später vermutlich nie wieder.

Ratschläge zum Studium (2)

- Lernen Sie programmieren (und wirklich gut).
  Es ist das Handwerk, auf dem alles beruht.
- Lernen Sie mathematisches Denken (Formalisieren, Beweisen), und Techniken wie z.B. vollständige Induktion, Grundlagen der Algebra und Logik.
- Arbeiten Sie in Gruppen zusammen, aber sorgen Sie dafür, dass jedes Gruppenmitglied alles lernt.
- Falls Frage nach Sinn des Lebens: Ich bin Christ.

  Das bedeutet nicht, das jeder am Ende eine 1.0 bekommt. Das wäre Untreue meinem Arbeitgeber gegenüber und würde auch Ihnen letztendlich wenig nützen. Sie können aber davon ausgehen, dass mir wichtig ist, dass Sie etwas lernen, und ich Ihnen helfen will. Leider sind meine Zeit und Fähigkeiten begrenzt.