Überblick und Wiederholung

Annabelle Klarl

Zentralübung zur Vorlesung
„Einführung in die Informatik: Programmierung und Softwareentwicklung“

`http://www.pst.ifi.lmu.de/Lehre/wise-15-16/infoeinf`

WS15/16

Klausurinformationen

- 6 ECTS: Klausur 06.02.2016 09:15 – 11:15 Uhr (120 Minuten)
  - Raum: M218, HGB, Geschw.-Scholl-Pl. 1
- 9 ECTS: Klausur 06.02.2016 09:15 – 11:**45** Uhr (150 Minuten)
  - Raum: B101, HGB, Geschw.-Scholl-Pl. 1
- Keine Hilfsmittel (weder gedruckt noch elektronisch)!
- Eine Anmeldung per UniWorX ist zwingend nötig.

Bitte seien Sie um 09:00 Uhr im jeweiligen Raum,
damit wir pünktlich beginnen können!

Nachholklausur

- Voraussichtlich Anfang/Mitte Juni 2016
  (Zeitpunkt und Raum werden auf der Website bekannt gegeben)

- Die Nachholklausur kann mitgeschrieben werden, egal ob Sie an der regulären Klausur angemeldet waren oder teilgenommen haben.

- Mitschreiben zur Notenverbesserung muss mit dem jeweiligen Prüfungsamt geklärt werden.

- Eine Anmeldung per UniWorX ist zwingend nötig.

Probeklausur

- Auf der Website steht eine Probeklausur mit Musterlösung für beiden Varianten der Vorlesung zu Verfügung.

- Die Probeklausur wird nicht in Vorlesung, Zentralübung oder Tutorien besprochen.

Inhaltsverzeichnis der Vorlesung

Kapitel 1: Java Übersicht

- Java ist eine imperative objekt-orientierte Programmiersprache.
- Die Programme sind plattformunabhängig, d.h. sie können ohne Änderungen z.B. unter Windows, OS X, Linux ausgeführt werden.
  - Java-Programme werden mit dem Compiler `javac` in Bytecode übersetzt.
  - Der Bytecode wird mit der Java Virtual Machine `java` ausgeführt.
- Geeignete Formatierung steigert die Verständlichkeit von Programmcode.
- Kommentare
  - `// KOMMENTAR` für einzeilige Kommentare
  - `/* KOMMENTAR */` für mehrzeilige Kommentare
  - `/** JAVADOC */` für javadoc-Kommentare

Kapitel 2: Syntax – EBNF-Grammatik, Ableitung

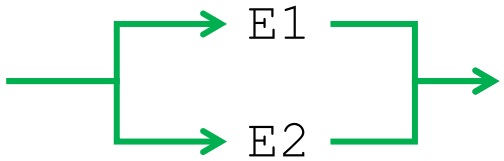
- Die Backus-Naur-Form ist eine Notation für Grammatiken.
- Aufbau einer Grammatik:
  - Startsymbol
  - Regeln der Form `Nichtterminal = Ausdruck`, wobei `Ausdruck` eine Kombination aus Nichtterminalen, Terminalen und Operatoren ist
    - `E1 E2`
    - `E1 | E2`
    - `[E1]`
    - `{E1}`
- Ableitung eines Worts: lang/kurz
  - Ersetzung von Nichtterminalen durch die rechte Seite der Regel oder/und
  - Ausführung von Operatoren

Kapitel 2: Syntax – Syntaxdiagramm

BNF-Grammatiken und Syntaxdiagramme sind äquivalent.

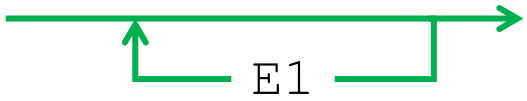
- Nichtterminale -> Rechtecke
- Terminale -> Ovale
- Operatoren -> Pfeile bzw. Verzweigungsstruktur
  - `E1 E2`

  - `E1 | E2`



  - `[E1]`

  - Zusätzlich: `{E1}`



Kapitel 3: Grunddatentypen

- Zahlen:
  - Ganze Zahlen:
    `byte` (2^7-1) < **`short`** $(2^{15}-1)$ < **`int`** $(2^{31}-1)$ < **`long`** $(2^{63}-1)$
  - Gleitkommazahlen: **`float`** (7 Stellen) < **`double`** (15 Stellen)
  - Operationen: `+ - * / %` (`< <= > >= ==`)

Typecasting = Erzwingen einer Typkonversion zum Typ `type`
durch Voranstellen von `(type)`
! Automatische Typkonversion zum größeren Typ !

- Einzelne Zeichen: **`char`** z.B. `'a'`
- (Zeichenketten: `String`)
- Wahrheitswerte: **`boolean`** mit Operationen `!, &&, &, ||, |`

Kapitel 3: Korrektheit und Auswertung

- Syntaktische Korrektheit von Ausdrücken
  = Ableitbarkeit in der EBNF-Grammatik
- Typkorrektheit von Ausdrücken
  = Zuordbarkeit eines Typen
  ! Achtung: Präzedenzen

Operation	Präzedenz
`!, unäres +-`	14
`(type)`	13
`*, /, %`	12
binäres `+, -`	11
`>, >=, <, <=`	9
`==, !=`	8
`&`	7
`|`	6
`&&`	4
`||`	3

- Auswertung von Ausdrücken
  - Vollständige Klammerung gemäß *Präzedenzen*
  - Auswertung unter Berücksichtigung der Klammern
    - Der Wert von Variablen ist durch den Zustand (σ,η) bestimmt.
    - Operationen, Methoden, Attributzugriff, Arrayzugriff… sind auszuwerten.

Kapitel 4: „Praktisches Programmieren"

- `main`-Methode
- Lokale Variablendeklaration z.B. `int i = 1;`
- Zuweisung z.B. `i = 3;`
- Block -> Gültigkeitsbereiche
- `if`-Anweisung
- `for`-Anweisung
- `while`-Anweisung

Übungsaufgaben anschauen und *selbst* programmieren!

Kapitel 5: Klassen

- Aufbau einer Klassendeklaration (UML-Notation)
  - Attribute
  - Konstruktoren
  - Methoden
- Initialisierung und Verwendung von Objekten

  ➡ Unterschied Klasse-Objekt
- Objekte im Speicher (Stack und Heap)
- Statische Methoden vs. Instanzmethoden

Übungsaufgaben anschauen und *selbst* programmieren!

Kapitel 6: Vererbung

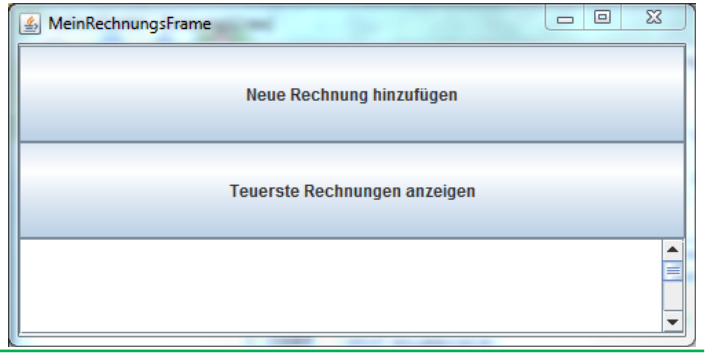
- Subtyping: Oberklasse – Unterklasse mit `extends`
  - Vererbung von Attributen
    ➡ Achtung: auf geerbte private Attribute kein Zugriff mit `.`
  - Keine Vererbung von Konstruktoren
    ➡ Aber Aufruf mit `super(…)`
  - Vererbung von Methoden
    ➡ Überschreiben in der Unterklasse möglich

  Übungsaufgaben anschauen und *selbst* programmieren!

- (Abstrakte Klassen und) Interfaces
  (Benutzung siehe Grafische Benutzeroberflächen)

Kapitel 7: Grafische Benutzeroberflächen

- Grafische Benutzeroberflächen bieten eine benutzerfreundliche Kommunikationsmöglichkeit mit Programmen.
- Vorgehensweise:
  1. Erstellung des strukturellen Aufbaus der GUI:
     Aufbau aus den Übungen (`JFrame`, `JButton`, `JTextArea`, `JPanel`…)
     - `ContentPane`
     - `GridLayout` mit Zeilen und Spalten
  2. Verbindung der GUI mit den inhaltlichen Objekten der Anwendung:
     Stichwort "Modell einer GUI"
  3. Ereignisgesteuerte Behandlung von Benutzereingaben (z.B. Knopfdruck):
     - `actionPerformed(ActionEvent e)`
     - `JOptionPane.showInputDialog(…)`



Kapitel 8: Arrays

- Ein Array ist ein **Tupel** von Elementen **gleichen** Typs
  z.B. Grunddatentyp, Klassentyp, Arraytyp.
  - **Feste** Länge, d.h. Vergrößerung nur durch Kopieren möglich (O(n))
    z.B. `char[] charArray = new char[5];`
    z.B. `int[] intArray = {1,2,3};`
  - Direkter Zugriff in **O(1)** z.B. `int a = intArray[1];`
  - Veränderung über Index z.B. `intArray[0] = 10;`
- Arrays im Speicher (Stack und Heap)

Algorithmen müssen meist durch das komplette Array laufen:

`for`-Schleife

Kapitel 9: Komplexität von Algorithmen

- Zeitbedarf und Speicherplatzbedarf: O-Notation
  - Worst-case
  - Average-case
  - Best-case
- Algorithmen
  - Binäre Suche in einem sortierten Array:
    Zeitkomplexität O(log n), Speicherplatzbedarf O(n)
  - Bubble-Sort:
    Zeitkomplexität $O(n^2)$, Speicherplatzbedarf O(n)
  - Selection-Sort:
    Zeitkomplexität $O(n^2)$, Speicherplatzbedarf O(n)

ENDE 6 ECTS

Kapitel 10: Rekursion

- Eine Methode ist rekursiv, wenn in ihrem Rumpf die Methode selbst wieder aufgerufen wird.
- Implementierung mit Hilfe einer `if`-Anweisung:
  - Basisfall: sofortige Terminierung z.B. `return 1;`
  - „`else`“-Fall: rekursiver Aufruf

Meist geben Aufgaben ein rekursives Konzept vor!

- Jeder rekursive Algorithmus kann auch iterativ gelöst werden, aber rekursive Algorithmen sind oft „schöner“.
  z.B. Quicksort

Kapitel 11: Ausnahmen

- Ein Programm heißt **robust**, falls es für jede (auch fehlerhafte) Eingabe eine sinnvolle Reaktion produziert.
- checked exceptions…
  - …erben von **`Exception`**
  - …müssen behandelt werden
- unchecked exceptions…
  - …erben von **`RuntimeException`**
  - …müssen **nicht** behandelt werden
- Ausnahmesituation erkennen
  - Objekte: Methodenaufruf auf Objekt mit Wert `null`
  - Arrays/Listen: Arrayzugriff außerhalb der Länge des Arrays
- Ausnahme auslösen: `throw`-Anweisung
  Achtung: `throws`-Deklaration für checked exceptions nötig!
- Ausnahme behandeln: `try-catch-finally`-Block

Kapitel 12: Listen

- Eine Liste ist eine endliche **Folge** von Elementen **gleichen** Typs.
  - **Dynamische** Länge, d.h. Vergrößern möglich
  - Zugriff je nach Implementierung
  - Veränderung je nach Implementierung
- Implementierung: Einfach-verkettete Liste
  - Direkter Zugriff in **O(n)**,
    da intern sequentiell durch die Liste gelaufen werden muss.
  - Veränderung über Vorne/Hinten-Anhängen bzw. über Index
- Implementierung: Doppelt-verkettete Liste

  ➡ Hinzufügen/Löschen am Ende in konstanter Zeit möglich

Praxis: `LinkedList<E>` und `Iterator<E>`

Kapitel 13: Bäume

- Ein Baum ist eine **hierarchische** Struktur von Elementen **gleichen** Typs.
  - Knoten speichern Elemente.
  - Knoten sind durch Kanten zu einer hierarchischen Struktur verbunden.
  - Jeder Knoten kann mehrere Nachfolger haben.
- Implementierung: Analog zu verketteten Listen, aber mit zwei oder mehr „nächsten“ Elementen (=Nachfolgern)
  - Zugriff auf ein Element in **O(log n)** für binäre Suchebäume
  - Einfügen eines neuen Elements in **O(log n)** für binäre Suchebäume

Operationen auf Bäumen sind meist rekursiv!
(siehe Tiefen- oder Breitendurchlauf)

Kapitel 14: Systemarchitektur

Größere Software-Systeme…

- …sind in Paketen strukturiert
- …bedürfen einer Gesamtarchitektur der beitragenden Klassen

Viel Erfolg bei der Klausur!