Lehrstuhl für Angewandte Informatik I
Prof. Dr. Bernhard Westfechtel
Sandra Greiner

WS 2017/18
22. Dezember 2017

Konzepte der Programmierung

Kleine Probeklausur

Bearbeitungszeit: 80 Minuten

Diese Probeklausur deckt in etwa die Vorlesungsinhalte bis Weihnachten ab.

Insgesamt können 80 Punkte auf vier Aufgaben verteilt erreicht werden und die Bearbeitungszeit beträgt 80 Minuten; ein Punkt entspricht also einer Minute Bearbeitungszeit. Beachten Sie dies bei der Bearbeitung! In der echten Klausur gibt es 120 Punkte bei 120 Minuten Bearbeitungszeit.

Sämtliche Aufgaben sind ausdrücklich **handschriftlich** zu bearbeiten! Alle Programmieraufgaben sind in der Sprache **Java** zu lösen!

Alle Blätter der Abgabe sind mit **Namen und Matrikelnummer** zu versehen! Abgaben sind geheftet per Postkasten oder eingescannt bzw. abfotografiert per E-Mail[1] oder e-Learning möglich.

In dieser Klausur bezeichnet $\mathbb{N} = \{0, 1, 2, \dots\}$ die Menge der natürlichen Zahlen.

Aufgabe 1 (Codeverständnis)

(a) Gegeben ist folgender Quellcode mit zwei Methoden.

```
1  public class Fehlersuche {
2
3      public int berechneSumme(int n) {
4          int res = 0;
5          int i = 0;
6          while (n < i) {
7              Res = i++;
8              return res;
9          }
10         return i;
11     }
12
13     public static void main(int n) {
14         Out.println("Geben Sie eine Zahl ein!")
15         Out.println("Die Summe lautet " + berechneSumme(n));
16     }
17
18 }
```

Die Methode `berechneSumme(...)` soll die Summe der Zahlen von `1` bis `n` zurückgeben. Im Hauptprogramm soll eine Zahl über die Standardeingabe eingelesen werden und als Übergabewert für `berechneSumme(...)` dienen. Sie können davon ausgehen, dass die Klassen `In` und `Out` im selben Paket sind.
Finden und korrigieren Sie **alle semantischen Fehler unter Angabe von Zeilennummern**! Fügen Sie keine neuen Zeilen ein und versuchen Sie, mit möglichst wenigen Änderungen auszukommen. Kennzeichnen Sie darüber hinaus jene Fehler, die eine

[1] vorrangig Philipp Dennerlein (*s1phdenn@uni-bayreuth.de*) oder Johannes Schröpfer (*s1joschr@uni-bayreuth.de*)

Übersetzung des Codes verhindern (Syntax-Fehler). Begründungen sind nicht erforderlich.

(b) Sind folgende Aussagen wahr oder falsch? Begründen Sie Ihre Antwort jeweils in **maximal zwei Sätzen**!

(1) Die Sichtbarkeit des Parameters `n` der Methode `berechneSumme(...)` im obigen Code ist gleich seiner Lebensdauer.

(2) Die Sichtbarkeit des Parameters `n` der Methode `berechneSumme(...)` im obigen Code ist gleich der Lebensdauer der Variable `i`.

(3) Will man Instanzen einer Klasse erzeugen, muss man einen Konstruktor definieren.

(4) Angenommen, folgender Code befindet sich in einer main-Methode. Dann lässt er sich übersetzen:

```
1 boolean istAn = false;
2 if (istAn = true) Out.println("Geraet ist an");
```

(5) Angenommen, folgender Code befindet sich in einer main-Methode. Dann lässt er sich übersetzen:

```
1 int x = 5;
2 if (!x == 0) Out.println("x = " + x);
```

(6) Gegeben ist folgender arithmetischer Ausdruck: `(float) a * b + c / d - e`.
Wenn eine der Variablen vom Typ `double` ist, wird bereits der gesamte Ausdruck zum Typ `double` ausgewertet.

(c) Deklarieren Sie ein Klassenfeld mit dem Wert `42`, den auch andere Klassen abfragen können, der jedoch nicht einmal in der eigenen Klasse geändert werden kann.

(d) Für ein Projekt müssen Sie eine frei verfügbare Programmbibliothek verwenden, die von einem fremden Anbieter geschrieben wurde. Dieser hat die **Java-Konventionen für Bezeichner** eingehalten. Beim Überfliegen der Beispielcodes stoßen Sie auf die folgenden Bezeichner:
`Name, ONLINE, Chat, istVerfuegbar, gibName, alter`
Um was könnte es sich dabei jeweils handeln? Ordnen Sie jedem Bezeichner **alle passenden Begriffe** zu:
Klasse, Variable, Konstante, Methode, Operator, Literal

(8 + 8 + 2 + 3 = 21 Punkte)

Aufgabe 2 (Syntax von Datenanfragen)
Im Rahmen dieser Aufgabe beschäftigen Sie sich mit Anfragen auf Datenbanken. Syntaktisch verwendet wird dabei die **Prädikatenlogik erster Stufe** (vereinfacht):
Wesentlicher Bestandteil von Anfragen sind **Variablen** – erlaubt sind der Einfachheit halber die Literale x, y und z – sowie **Konstanten** – a, b und c. Ein **Atom** enthält ein Relationssymbol – möglich sind R, S und T – gefolgt von einem Tupel. Ein Tupel enthält eine durch Kommata getrennte (nicht-leere) Sequenz von Variablen und Konstanten (beliebig gemischt), welche durch ein Paar runder Klammern umschlossen wird. Jedes Atom ist für sich eine Anfrage.
Atome können auch verknüpft werden: Sind w_1 und w_2 Anfragen, so ist auch der Ausdruck $(w_1 \wedge w_2)$ (**Konjunktion**), der Ausdruck $(w_1 \vee w_2)$ (**Disjunktion**) sowie der Ausdruck $\neg w_1$ (**Negation**) jeweils eine Anfrage. Des Weiteren kann einer Anfrage eine **Quantifizierung** voranstehen, sodass wieder eine Anfrage resultiert. Eine Quantifizierung beginnt mit einem Quantor – $\exists$ oder $\forall$ –, welchem genau eine Variable folgt.
Folgende Wörter stellen beispielsweise syntaktisch korrekte Anfragen dar:

- $(R(a, b) \vee \forall x \neg T(x))$
- $\exists x S(x, b)$

(a) Geben Sie eine **kontextfreie Grammatik (BNF)** für syntaktisch korrekte Datenanfragen an.

(b) Konstruieren Sie einen **Ableitungsbaum** für das zweite Beispielwort.

In der praktischen Anwendung spielt insbesondere eine einfacher Teilmenge von Anfragen, die **konjunktiven Anfragen**, eine wesentliche Rolle. Diese bestehen jeweils aus einer (eventuell leeren) Sequenz von Quantifizierungen, welcher eine große (nicht-leere) Konjunktion von Atomen folgt; als Quantor ist nur $\exists$ erlaubt. Wir verwenden für die Konjunktion keine Klammern.
Folgendes Wort stellt beispielsweise eine syntaktisch korrekte konjunktive Anfrage dar:
$\exists x \exists y \; R(x, a) \wedge S(y, b) \wedge T(c)$

(c) Geben Sie einen Satz von **EBNF-Regeln** an, um eine syntaktisch korrekte konjunktive Anfrage zu formulieren; es gilt (nur für diese Teilaufgabe) die zusätzliche Bedingung, dass **mindestens drei Atome** vorkommen müssen.

(d) Geben Sie einen Satz von **EBNF-Regeln** an, um eine syntaktisch korrekte konjunktive Anfrage zu formulieren; es gilt (nur für diese Teilaufgabe) die zusätzliche Bedingung, dass **höchstens drei Atome** vorkommen dürfen.

(8 + 3 + 3 + 1 = 15 Punkte)

Aufgabe 3 (Programmier-Grundlagen: Kontrollstrukturen, Arrays und Rekursion)
Die Collatz-Folge ist eine Folge $(n_i \mid i = 0, \ldots, n)$ von natürlichen Zahlen, die sich aus folgenden Regeln ergibt:

- Ein Element n_{i+1} $(i \geq 0)$ der Folge ergibt sich nach folgender Vorschrift:

$$n_{i+1} = \begin{cases} \frac{n_i}{2} & \text{, falls } n_i = 2k \ (k \geq 1) \\ 3 \cdot n_i + 1 & \text{, falls } n_i = 2k - 1 \ (k \geq 1) \end{cases}$$

- Das letzte Element n_n der Folge gilt: $n_n = 1$

Für $n_0 = 5$ sieht die Collatz-Folge z. B. folgendermaßen aus:

$$(5, 16, 8, 4, 2, 1)$$

Sie haben folgende Vorlage für eine Klasse gegeben:

```
1  public class Collatz {
2
3      // Dieses Array bietet genug Platz fuer Folgen
4      // bis zu einem Startwert von 1000.
5      public static int[] sequence = new int[200];
6
7      public static void collatzIter(...) {
8          ...
9      }
10
11     public static void collatzRek(...) {
12         ...
13     }
14
15 }
```

Die Klassenmethoden `collatzIter()` bzw. `collatzRek()` sollen jeweils für ein $n_0 \leq 1000$, das sie übergeben bekommen, die dazugehörige Collatz-Folge in das Array *sequence* sequenziell bei Index 0 beginnend speichern.

(a) Implementieren Sie die Methode `collatzIter(...)` **iterativ**. Überlegen Sie sich, welche Parameter die Methode benötigt und geben Sie die ganze Methode, nicht nur ihren Rumpf an.

(b) Implementieren Sie die Methode `collatzRek(...)` **rekursiv**. Überlegen Sie sich, welche Parameter die Methode benötigt und geben Sie die ganze Methode, nicht nur ihren Rumpf an.

(c) Klassifizieren Sie ihre rekursive Methode hinsichtlich der drei aus der Vorlesung bekannten Kriterien (*kurze Begründung!*).

Im Folgenden arbeiten Sie mit **zweidimensionalen int-Arrays**. Hierbei handelt es sich um kodierte Informationen; jede Ganzzahl repräsentiert ein Zeichen entsprechend der üblichen **Integer-Character-Konvertierung** aus der Vorlesung. Jedes innere Array steht für eine **Zeile**.

(d) Für jede **Spalte** soll zwecks Datensicherheit eine Prüfsumme ermittelt werden. Schreiben Sie eine Klassenmethode `sumCols(...)`, die ein zweidimensionales int-Array **mit gleich langen Zeilen** erwartet, für jede Spalte die darin befindlichen Zahlen aufsummiert und ein

int-Array zurückgibt, das für jede Spalte deren Summe enthält. Überlegen Sie sich, welche Signatur die Methode hat.

Die Eingabe

$$\{\{1,\ 2,\ 3,\ 4\},\\ \{5,\ 6,\ 7,\ 8\},\\ \{9, 10, 11, 12\}\}$$

soll bspw. folgendes Ergebnis liefern:

$$\{15, 18, 21, 24\}$$

(e) Nun sollen Sie sich um die Dekodierung der Werte kümmern. Schreiben Sie eine Klassenmethode `decode(...)`, die ein zweidimensionales int-Array **mit gleich langen Zeilen** erwartet. Erzeugen Sie zunächst ein **char-Array mit unterschiedlich langen Zeilen**, wovon die i-te Zeile diejenigen Buchstaben unter Beibehaltung der Reihenfolge beinhaltet, deren korrespondierende int-Werte in der i-ten Zeile des übergebenen Arrays gespeichert sind. Beachten Sie, dass im übergebenen Array auch char-Werte gespeichert sein können, die keine Buchstaben repräsentieren, sodass diese im neuen Array nicht mitgespeichert werden sollen; sie können davon ausgehen, dass es in derselben Klasse eine Klassenmethode `boolean istBuchstabe(char c)` für diesen Zweck gibt. Geben Sie anschließend einen String zurück, welcher die Buchstaben **formatiert** enthält; Buchstaben einer Zeile werden mit einfachem Leerzeichen getrennt, Zeilen im Array sollen analog zu Zeilen im String führen.

(6 + 5 + 3 + 4 + 7 = 25 Punkte)

Aufgabe 4 (Linien und Punkte in Java)
Ihr Team ist gerade dabei, ein Geometrie-Programm zu entwickeln. Sie bekommen zur Bearbeitung die folgenden Aufgaben zugeteilt.

(a) Erstellen Sie eine Klasse `Punkt`, welche einen `Namen` enthält und zwei Koordinaten `x` und `y`. Erstellen Sie zudem einen passenden Konstruktor, der als Parameter den `Namen` und die beiden Koordinaten übergeben bekommt. Verwenden Sie geeignete Datentypen und Sichtbarkeiten. Beachten Sie dabei, dass die Koordinaten auch außerhalb der Klasse lesend sichtbar sein müssen.
Tipp: Getter und Setter.

(b) Welche Sichtbarkeiten haben Sie für die Variablen der Punkt-Klasse verwendet und aus welchem Grund?

(c) Fügen Sie zur Klasse `Punkt` ein enum `PunktTyp` mit den Werten NotAPoint (NaP) und NormalPoint(NP) hinzu. Außerdem soll es einen neuen Konstruktor in der Klasse Punkt geben, der als Parameter einen PunktTyp übergeben bekommt und diesen speichert. Im Fall von NaP wird x und y auf 42 gesetzt und der Name auf *Ganz viele Punkte*. Im Fall von NP wird x und y auf 0 gesetzt und der Name *Ursprung* eingefügt.

(d) Erstellen Sie eine Klasse `Gerade`. Eine `Gerade` soll zwei Punkte speichern sowie einen `Namen` haben. Die zwei Punkte und der `Name` sollen der Klasse über den Konstruktor als Parameter übergeben werden. Die Variablen sollen alle auch von außen sichtbar sein.

(e) Implementieren Sie nun in der Klasse `Gerade` eine Methode `getSteigung(...)`, welche die Steigung der Geraden zurückliefert. Achten Sie darauf, dass die Punkte auch in einer anderen Reihenfolge übergeben worden sein können. Sollte die Steigung unendlich sein, geben sie `Integer.MAX_VALUE` zurück.

(f) Implementieren Sie in der Klasse `Gerade` eine Funktion `hatSchnittpunkt(...)`, welche eine `Gerade` übergeben bekommt und `true` zurückgibt, falls sich die aktuelle Gerade und die übergebene schneiden.
Tipp: Die vorherige Teilaufgabe könnte dabei hilfreich sein. Sie können die Information daraus auch verwenden, wenn Sie diese nicht bearbeitet haben.

(3 + 1 + 4 + 3 + 4 + 4 = 19 Punkte)