Programmierstarthilfe SS 2008
Fakultät für Ingenieurwissenschaften und Informatik

10. Blatt
Für den 30.6. und 1.7.2008

Organisatorisches

Um auf die Mailingliste aufgenommen zu werden schicke einfach eine Mail an `guido.de-melo@uni-ulm.de`.

Die Webseiten zur Veranstaltung sind unter `http://www.uni-ulm.de/in/mi/lehre/ss2008/programmierstarthilfe.html` zu finden.

1. Rekursion

Rekursion kommt eigentlich aus der Mathematik, und und ist eine andere Sichtweise auf Funktionen. Manche sagen auch sie dient hauptsächlich der Verwirrung von Studenten ;). Zum Beispiel

$$f(x) = \begin{cases} 0, & falls \quad x = 0 \\ f(x-1)+1, & sonst \end{cases} \quad \forall x \in \mathbb{N}$$

definiert einfach nur $f(x) = x$.

Wichtig, um die Rekursion zu verstehen ist der so genannte Basisfall (in diesem Fall $x = 0$), denn auf diesem baut die Rekursion auf. Oft wird der Basisfall auch Abbruchbedingung genannt, weil dies die Bedingung ist die erfüllt sein muss, damit die Rekursion „abgebrochen" wird. Den Basisfall/Abbruchbed. sollte man sich immer zuerst überlegen.

Dann kommt der Rekursionsfall (auch Rekursionsschritt genannt). In diesem Fall wird die gleiche Funktion f mit veränderten Parametern $(x - 1)$ nochmals aufgerufen und dann um eins erhöht.

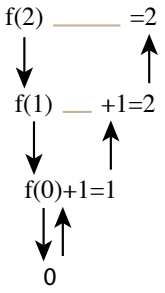
Die angegebene Formel lässt sich folgendermaßen als Methode implementieren:

```
1 int f(int x){
2         if(x == 0) {
3                 return 0;
4         } else {
5                 return f(x-1)+1;
6         }
7 }
```

Ein einfaches Beispiel: Wir möchten $f(2)$ berechnen.

$$f(2) = f(1) + 1 = (f(0) + 1) + 1 = (0 + 1) + 1 = 2$$

Anschaulich sieht das dann so aus:



Hier sieht man, dass zwei Rekursionsschritte bis zur Abbruchbedingung ausgeführt werden, und beim „wiederaufsteigen“ das Ergebnis um 1 erhöht wird.

2. Exceptions

In allen Computerprogrammen kommt es manchmal zu Fehlern und Ausnahmesituationen. In Java wird in einem solchen Fall eine sogenannte `Exception` ausgelöst. Wenn wir zum Beispiel mit einer for-Schleife Array-Werte auslesen und dabei nicht aufpassen, versuchen wir ein Feld zu viel zu lesen:

```
1        int[] arr = new int[15];
2        for (int i = 0 ; i <= arr.length; i++) {
3                System.out.print(arr[i]);
4        }
```

Deswegen meldet Java:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 15
at MeineKlasse.main(MeineKlasse.java:15)
```

`ArrayIndexOutOfBoundsException` zeigt uns dabei den Fehlertyp an, um den wir uns kümmern müssen. Danach steht die Zeile, in der der Fehler aufgetreten ist.

Oft können wir solche Fehler nicht so einfach wie hier im vorfeld korrigieren. Zum Beispiel gibt es viele Java-API-Methoden, die mitteilen, dass sie eventuell einen Fehler auslösen könnten. Nach dem Methodennamen steht dann in der Dokumentation `throws ...Exception`.

In solchen Fällen müssen wir uns darum kümmern eine Fehlerbehandlung zu schreiben. Dazu bietet uns Java das Konstrukt `try / catch`:

```
1        try {
2                //Code von oben
3        }
4        catch (Exception e) {
5                System.out.println("Das war zu viel");
6        }
```

Tritt im try-Block eine Exception auf, wechselt das Programm in den catch-Teil, wo dann steht, was im Fehlerfall zu tun ist.

3. Aufgaben

3.1. Summenbildung rekursiv

Es soll eine Methode `summiere(int n)` geschrieben werden, die den Wert $\sum_{i=1}^{n} i$ berechnen und zurückgeben soll.

a) Implementiere zunächst die Methode iterativ, also mit einer Schleife.

b) Überlege nun eine rekursive Lösung für das Problem. Mache dir zunächst den Basisfall für eine Abbruchbedingung klar. Wie muss nun der Rekursionsfall aussehen? Schreibe auf einem Blatt Papier auf, wie die Summe rekursiv entsteht. Gibt es mehrere Möglichkeiten, aufzusummieren?

c) Implementiere nun die rekursive Methode in Java und teste sie.

3.2. Fakultät rekursiv

Es soll eine Methode `fakultaet(int n)` geschrieben werden, die den Wert $n!$, also $1 \cdot 2 \cdot \ldots \cdot n$, berechnen und zurückgeben soll.

a) Überlege dir eine rekursive Lösung für das Problem. Mache dir zunächst den Basisfall für eine Abbruchbedingung klar. Wie muss nun der Rekursionsfall aussehen? Schreibe auf einem Blatt Papier auf, wie die Fakultät rekursiv entsteht. Was ist dir lieber: $1 \cdot 2 \cdot \ldots$, oder $n \cdot n-1 \cdot \ldots$?

b) Implementiere nun die rekursive Methode in Java und teste sie.

c) Implementiere zum Schluss die Methode iterativ, also mit einer Schleife.

3.3. Rekursiver Palindromtest

Es soll eine Methode `istPalindrom(String wort)` geschrieben werden, die den übergebenen String rekursiv auf Symmetrie testet und zurückgeben soll, ob es sich um ein Palindrom handelt oder nicht.

a) Mache dir zunächst Gedanken darüber, wie du bisher auf Symmetrie getestet hast und wie du rekursiv vorgehen könntest. Wichtig sind ein korrekter Basisfall sowie Rekursionsfälle. Wie lange musst du rekursiv absteigen? Wann kann abgebrochen werden? Wie transportierst du das Zwischenergebnis?

b) Implementiere die Methode nun in Java. Hierfür wird es nützlich sein, eine überladene Methode `istPalindrom(...)` zu schreiben, in der du mehr Parameter mitgeben kannst. Die Methode `istPalindrom(String wort)` ruft dann die überladene Methode mit Startwerten auf.

3.4. Array Index Out Of Bounds Exception

Diese Ausnahme ist dir sicherlich bei deinen ersten Programmen mit Arrays begegnet. Eigentlich sollte sie in einem guten Programm gar nicht erst auftreten und sollte somit auch nicht mit try-catch gefangen werden.

a) Erstelle ein Array mit zwei Integer-Werten und prüfe was passiert, wenn du auf den dritten Wert zugreifen möchtest.

b) Schreibe nun einen try-catch-Block um die Befehlszeile, in der auf das nicht existierende Element zugegriffen wird.

c) Benutze nun Math.random(), um einen zufälligen Index zwischen 0 und 2 zu erzeugen und greife auf das Element mit diesem Index zu. Dieses soll ausgegeben werden, sofern es existiert. Ansonsten soll eine entsprechende Fehlermeldung ausgegeben werden.

4. Bonusaufgaben

4.1. Produkt rekursiv

Du kannst diese Aufgabe auch überspringen, wenn du denkst dass du das Prinzip von Rekursion verstanden hast
Es soll eine Methode `produkt(int a, int b)` geschrieben werden, die den Wert $\prod_{i=a}^{b} i$, berechnen und zurückgeben soll.

a) Überlege dir eine rekursive Lösung für das Problem. Mache dir zunächst den Basisfall für eine Abbruchbedingung klar. Wie muss nun der Rekursionsfall aussehen? Schreibe auf einem Blatt Papier auf, wie das Produkt rekursiv entsteht.

b) Implementiere nun die rekursive Methode in Java und teste sie.

c) Implementiere zum Schluss die Methode iterativ, also mit einer Schleife.

4.2. Potenzierung rekursiv

Es soll eine Methode `potenziere(int a, int b)` geschrieben werden, die den Wert a^b berechnen und zurückgeben soll.

a) Implementiere zunächst die Methode iterativ, also mit einer Schleife.

b) Implementiere die Methode mithilfe der Methode `Math.pow(double a, double b)`. Schau hierfür in die API-Dokumentation, wie diese Methode verwendet wird.

c) Überlege nun eine rekursive Lösung für das Problem. Mache dir zunächst den Basisfall für eine Abbruchbedingung klar. Wie muss nun der Rekursionsfall aussehen? Skizziere den Ablauf der rekursiven Berechnung auf einem Blatt Pappier.

d) Implementiere nun die rekursive Methode in Java und teste sie.

e) Es gilt $a^{2n} = a^n \cdot a^n$. Kannst du damit deinen Algorithmus noch optimieren?

4.3. Rekursiver Würfel

Nun soll eine Methode `wuerfeln(int anzahl)` programmiert werden, die alle möglichen Kombinationen bei einem Würfel mit sechs Seiten und einer gegeben Anzahl von Würfelnerzeugemnund ausgeben soll. Bei der Anzahl drei sollen zum Beispiel alle Ergebnisse 1-1-1, 1-1-2, ... 6-6-5, 6-6-6 ausgegeben werden.

a) Sicher kennst du aus der Schule noch Baumdiagramme für Wahrscheinlichkeiten. Erstelle auf Papier ein solches Diagramm für zweimaliges Würfeln. Ignoriere die Wahrscheinlichkeiten und beschrifte die Blätter mit der entstandenen Kombination.

b) Implementiere nun die Methode `wuerfeln(int anzahl)` rekursiv. Wie oft musst du pro Rekursionschritt verzweigen?

4.4. Primfaktorzerlegung

a) Schreibe ein Programm, welches eine benutzerspezifizierte natürliche Zahl rekursiv in ihre Primfaktoren zerlegt und diese ausgibt. Überlege dir dazu zunächst, wie man prüfen kann, ob eine Zahl durch eine andere teilbar ist. Wie sieht dann der Rekursionschritt aus? Was ist der Basisfall?

b) Vollziehe auf dem Papier nach, was bei der Primfaktorzerlegung von 42 passiert und überprüfe deine Überlegungen durch Ausgaben in deinem Programm.