technische universität
dortmund

Fakultät für Informatik
Lehrstuhl 14
Lars Hildebrand, Marcel Preuß,
Iman Kamehkhosh, Marc Bury, Diana Howey

Praktikum zu

Einführung in die Informatik für LogWilngs und WiMas

Wintersemester 2015/16

Übungsblatt 7

Besprechung:
14.–18.12.15
(KW 51)

Vorbereitende Aufgaben

Aufgabe 7.1: Selbsttest

Machen Sie sich mit den Grundlagen der ersten 6 Übungsblätter vertraut. Ihr Übungsgruppenleiter wird Ihnen zu Beginn der Praktikumsstunde eine Aufgabe zum Selbsttest stellen, in der Sie Ihr bisheriges Wissen auf die Probe stellen können.

Für diese Aufgabe haben Sie 45 Minuten Zeit.

Diese Aufgabe wird nicht benotet oder bewertet und dient zur reinen Überprüfung Ihres Wissensstandes

Eingabe und Importieren von Bibliotheken

Diese Seite soll Ihnen eine Übersicht über das Einlesen von **Eingaben** über die Tastatur und das **Importieren** von anderen Programmen in Ihr eigenes Programm geben.

Sie können mit der Anweisung **import java.util.Scanner;** ein bereits geschriebenes Programm zur Behandlung von Eingaben aus der **Java-Standard-Bibliothek** in Ihrem Programm verfügbar machen.

Ein **Scanner** muss vor der Verwendung **instanziiert** werden. Dies ist ein Vorgang, mit dem Sie sich im **objektorientierten** Teil der Veranstaltung noch genauer beschäftigen werden. Wenn Sie einen **Scanner** verwenden wollen, müssen Sie eine neue Variable vom Typ **Scanner** anlegen und mit der Anweisung **new Scanner(System.in)** instanziieren. Mit dieser Variablen können Sie anschließend die Standard**eingabe** auslesen.

Ein leerer Klassenrumpf, der in der **main**-Methode einen Scanner verwenden will, würde also folgendermaßen aussehen:

```
1  package blatt07;
2
3  import java.util.Scanner;
4
5  public class Input {
6      public static void main(String[] args) {
7          Scanner scanner = new Scanner(System.in);
8      }
9  }
```

Sie können anschließend mit folgenden Anweisungen Daten auslesen:

```
int number = scanner.nextInt();         /* liest einen Integer ein */
long number = scanner.nextLong();       /* liest einen Long ein */
float number = scanner.nextFloat();     /* liest einen Float ein */
double number = scanner.nextDouble();   /* liest einen Double ein */
String text = scanner.nextLine();       /* liest Text ein */
```

Führt ein Programm eine solche Programmzeile aus, **wartet** es, bis eine Eingabe durch das Betätigen der **Eingabetaste** in der Konsole an das Programm gesendet wird.

Sie sollten, bevor ein Programm terminiert, mit der Anweisung **scanner.close();** die Standardeingabe wieder freigeben.

Präsenzaufgaben

Aufgabe 7.2: Hallo Echo!

Sie sollen nun ein einfaches Programm schreiben, das eine Eingabe einliest und diese dann unmittelbar wieder ausgibt.

Erstellen Sie eine neue Klasse mit dem Namen **Echo** im Paket **blatt07**.

- Importieren Sie den **Scanner** und instanziieren Sie ihn in der **main**-Methode.
- Implementieren Sie eine Schleife, die die nächste Zeile aus der Standardeingabe ausliest und wieder ausgibt.
- Dies soll solange wiederholt werden bis der leere String (`""`) eingegeben wird.

Hinweis: Das **Vergleichen von Strings** funktioniert nicht mit dem „==“-Operator, sondern mit der Methode **equals**. In der Vorlesung (Kapitel 3.1) wurde Ihnen der Grund dafür schon einmal erklärt.

Betrachten Sie folgendes Beispielprogramm:

```
1  public class StringCompare {
2      public static void main(String[] args) {
3          String input = "Hallo!";
4          String password = "geheim";
5          if(input.equals(password)) {
6              System.out.println("Access granted!");
7          }
8          else {
9              System.out.println("Access denied!");
10         }
11     }
12 }
```

In diesem Fall würde das Programm `Access denied!` ausgeben.

Aufgabe 7.3: Fakultät rekursiv

In dieser Aufgabe sollen Sie ein iteratives Programm in ein rekursives Programm umwandeln.

Auf Aufgabenblatt 5 haben Sie die Fakultät einer Zahl iterativ berechnet. Diese Aufgabe eignet sich auch sehr gut als Übung für rekursive Funktionen.

a) Überlegen Sie sich eine rekursive Definition zur Berechnung der Fakultät.

b) Schreiben Sie ein Programm namens **RecursiveFactorial** im Paket **blatt07**. Es soll eine Zahl n mit einem **Scanner** einlesen und die Fakultät von n berechnen und ausgeben.

Hinweis: Hier sollten Sie keine Ausgabe in den rekursiven Funktionen verwenden.

Aufgabe 7.4: „ganz einfache“ Rekursion

In dieser Aufgabe sollen Sie sich tiefer mit dem Verhalten von rekursiven Programmen beschäftigen.

a) Schreiben Sie eine rekursive Funktion **descendingPrint**, die die Zahlen von 1 bis **n** in **umgekehrter** Reihenfolge (von **n** bis **1**) ausgibt. **n** ist der Parameter der Funktion.

b) Kopieren und verändern Sie die Funktion so, dass die Zahlen **aufsteigend** (von **1** bis **n**) ausgegeben werden. Nennen Sie die neue Funktion **ascendingPrint**.

Aufgabe 7.5: Potenzen

In dieser Aufgabe sollen Sie lernen, eine rekursive mathematische Definition zu implementieren.

Um die Potenz b^e auszurechnen, kann man folgende Funktion benutzen ($b \in \mathbb{R}$ und $e \in \mathbb{N}$, mit b als Basis und e als Exponent):

$$\text{pow}(b, e) = \begin{cases} 1 & \text{falls } e = 0 \\ (\text{pow}(b, e/2))^2 & \text{falls } e \text{ gerade} \\ b \cdot \text{pow}(b, e-1) & \text{sonst} \end{cases}$$

Machen Sie sich am Beispiel 3^5 klar, dass diese Funktion tatsächlich funktioniert.

Implementieren Sie diese Funktion in der Klasse **Pow** im Paket **blatt07**. Ignorieren Sie bei Ihrer Implementierung den Fall, dass der Exponent auch negativ sein kann.

Ergänzende Aufgaben

Aufgabe 7.6: Euklidscher Algorithmus

In dieser Aufgabe sollen Sie lernen, ein rekursives Programm zu analysieren und zu optimieren.

Der **größte gemeinsame Teiler** (ggT) zweier natürlicher Zahlen m und n ist die größte Zahl, durch die sowohl m als auch n teilbar ist.

Ein Algorithmus zur Berechnung des ggT ist der **euklidsche Algorithmus**. In diesem wird immer abwechselnd die kleinere Zahl von der größeren abgezogen, bis eine von beiden 0 ergibt. Die andere Zahl ist dann der ggT.

Eine rekursive Implementierung könnte so aussehen:

```
public static int euclid(int m, int n) {
    if(m == 0)
        return n;
    else if(n == 0)
        return m;
    else if(m > n)
        return euclid(m - n, n);
    else
        return euclid(m, n - m);
}
```

Diese Implementierung funktioniert, ist aber ineffizient. Im Laufe dieser Veranstaltung haben Sie eine Möglichkeit kennengelernt, die Rechenschritte dieses Algorithmus zu verkürzen.

Berechnen Sie per Hand `euclid(15, 42)`.

Was fällt Ihnen auf? Angenommen m sei im letzten Rekursionsschritt größer n, was haben Sie berechnet, wenn in diesem Rekursionsschritt m nicht mehr größer als n ist?

Geben Sie eine rekursive Implementierung des euklidschen Algorithmus an, der sich diese Erkenntnis zu Nutze macht.