Programmierkurs
Wintersemester 2020/21
Übungsblatt 10 (25.01.2021)

M.Sc. Laslo Hunhold
Dept. Math./Inf., Abt. Informatik
Universität zu Köln

Abgabe für Rückmeldungen bis zum 01.02.2021 (08:00 Uhr), Besprechung ab dem 01.02.2021.

Aufgabe 10-0 ((Sub-)Packages und Annotationen)
Erstellen Sie ein neues Eclipse-Projekt und darin vier Packages mit den Namen `main`, `werkzeuge`, `werkzeuge.zahlen` und `werkzeuge.math`. Wie in der Vorlesung bereits erklärt, spricht man zwar bei `werkzeuge.zahlen` und `werkzeuge.math` von „Subpackages“ des Packages `werkzeuge`, jedoch bezieht sich das nur auf die Struktur im Projektordner, wo z.B. die Klassen des Packages `werkzeuge.zahlen` nicht im Ordner „werkzeuge.zahlen“, sondern in einem Unterordner „zahlen“ des Ordners „werkzeuge“ abgelegt werden. Rein syntaktisch (d.h. aus Sicht des Compilers) betrachtet sind alle vier Packages gleichwertig und unabhängig voneinander, aber es ermöglicht eine semantische (d.h. bezogen auf die Bedeutung) Hierarchierung der Packages untereinander, welche auch beliebig tief gehen kann (z.B. mit `werkzeuge.math.folgen`).
Stellen Sie sich vor, daß Sie in den Packages `werkzeug` und seinen „Subpackages“ eine Sammlung von Klassen und Funktionen anlegen, welche allgemein nützlich sind. Es bietet sich hier an, die Ergebnisse vorheriger Aufgaben zu verwenden.
Fügen Sie dem Package `werkzeuge` eine Klasse `HalloWelt` hinzu, die zwei statische Prozeduren `hallo_welt()` und `hallo()` enthält, welche beide „Hallo Welt!“ in der Konsole ausgeben. Versehen Sie `hallo` mit der `Deprecated`-Annotation. Nehmen Sie die vollständige Klasse `Bruch` aus Aufgabe 06-1 und fügen Sie sie dem Package `werkzeuge.zahlen` hinzu. Erstellen Sie im Package `werkzeuge.math` zwei Klassen `Fibonacci` bzw. `Fakultaet`, die jeweils eine statische Funktion `berechne()` enthalten, die analog zu den Aufgaben 07-3 bzw. 07-4 einen `int`-Parameter `n` annimmt, daraus die `n`te Fakultät bzw. `n`te Fibonaccizahl berechnet und zurückgibt. Versehen Sie alle hinzugefügten Klassen und Methoden mit dem `public`-Schlüsselwort.
Gehen Sie nun über in das Package `main`, erstellen Sie eine Klasse `Main` mit `main`-Methode und lösen Sie folgende Aufgaben darin.

0. Rufen Sie die Prozeduren `hallo_welt()` und `hallo()` sowohl unter Angabe des Packagenamens (also z.B. `werkzeuge.HalloWelt.hallo_welt()`) als auch auf Basis eines Imports der `HalloWelt`-Klasse (also „`import werkzeuge.HalloWelt`“ und `HalloWelt.hallo_welt()`) auf. Was beobachten Sie?

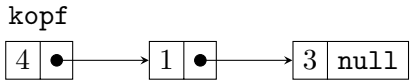
1. Gehen Sie analog vor, um danach die 12. Fibonaccizahl und die 6. Fakultät zu bestimmen und auf der Konsole auszugeben. Testen Sie beim Import statt zweier `import`-Ausdrücke (für die Klassen `Fibonacci` und `Fakultaet`) auch die Herangehensweise mit dem Wildcard-Platzhalter (also „`import werkzeuge.math.*`“).

2. Erstellen Sie im Package `main` eine Subklasse der Klasse `Bruch` mit dem Namen `Bruch2`, überschreiben Sie die Methode `print()` mit einer eigenen, die jeweils Leerzeichen vor und nach dem Schrägstrich ausgibt, und erweitern Sie die Klasse mit zwei Funktionen `kleiner_als(int f)` und `kleiner_als(Bruch f)`. Erzeugen Sie in Ihrer `main`-Methode jeweils eine Instanz der Klassen `Bruch` und `Bruch2` und testen Sie Ihre Änderungen.

Was beobachten Sie, wenn Sie das `public`-Schlüsselwort bei bestimmten Klassen oder Methoden weglassen?

Aufgabe 10-1 (Einfach verkettete Liste aus einem Array)
Bisher haben wir nur den Array als „serielle“ Datenstruktur kennengelernt. Hier zu notieren

ist vor allen Dingen, daß alle Einträge eines Arrays im Speicher aufeinanderfolgen und wir nur einen einzigen Zeiger haben, der auf den ersten Eintrag des Arrays zeigt (i.e. der die Adresse des ersten Eintrags enthält). Es liegt nahe, daß ein Array bei Iterationen sehr schnell ist. Schwieriger wird es in dem Fall, wenn man diesen verändern möchte (z.B. Hinzufügen oder Entfernen eines Eintrags zwischen zwei Einträgen). Aus diesem Grund gibt es mehr als nur Arrays zum Zwecke der „seriellen“ Speicherung von Daten.
Ein Beispiel für eine alternative „serielle“ Datenstruktur ist die einfach verkettete Liste. Statt eines Blocks einer Reihe von Einträgen sind die einzelnen Einträge im Allgemeinen an verschiedenen Stellen im Speicher verteilt. Um sinnvoll damit arbeiten zu können, speichert jeder Eintrag neben seinem Wert noch zusätzliche Informationen, die eine „Verkettung“ der Daten ermöglicht. Im Fall der einfach verketteten Liste speichert jeder Eintrag neben seinem Wert noch einen Zeiger `next` auf seinen Folgeeintrag; der letzte Eintrag gibt den Nullzeiger als seinen Folgeeintrag an. Da ein Eintrag nicht nur aus seinem Wert an sich besteht, spricht man in diesem Kontext nicht mehr von „Einträgen“, sondern von „Knoten“. Analog zum „Startzeiger“ beim Array speichert man bei einer einfach verketteten Liste nur den Zeiger auf den ersten Knoten, den wir als „Kopf“ (englisch „head“) bezeichnen. Der Vorteil ist, daß das Einfügen eines Knotens sehr effizient ist, weil man lediglich den neuen Knoten erzeugen und zwei Zeiger neu setzen muß. Die folgende Abbildung zeigt ein Beispiel mit den Einträgen 4, 1 und 3.



Eine beispielhafte Implementierung sehen Sie im folgenden (lückenhaften) Listing.

```
1  class Knoten {
2      int wert;
3      Knoten next;
4
5      Knoten(int wert, Knoten next) {
6          this.wert = wert;
7          this.next = next;
8      }
9  }
```

```
1  public class Einfach_verkettete_Liste {
2      Knoten kopf;
3
4      public Einfach_verkettete_Liste(int w) {
5          kopf = new Knoten(w, null);
6      }
7      public Einfach_verkettete_Liste(int[] arr) { ... }
8
9      public void print() {
10         for (Knoten k = kopf; k != null; k = k.next) {
11             System.out.print(k.wert + " ");
12         }
13         System.out.print("\n");
14     }
15
16     public Knoten fuege_ein_nach(Knoten k, int w) {
17         return (k.next = new Knoten(w, k.next));
18     }
19 }
```

Implementieren Sie den Konstruktor `Einfach_verkettete_Liste(int[] arr)`, der die Liste mit den Werten aus dem `int`-Array `arr` füllen soll, und testen Sie Ihr Programm mit folgendem Listing.

```
1  int[] arr = { 1, 4, 2, 4, 6,  1, 4, 5, 6 };
2
3  for (int i = 0; i < arr.length; i++) {
4          System.out.print(arr[i] + " ");
5  }
6  System.out.print("\n");
7
8  Einfach_verkettete_Liste liste = new Einfach_verkettete_Liste(arr);
9  liste.print();
```

Präsenzaufgabe 10-2 (Zeitmessung (Array versus einfach verkettete Liste))

Nehmen Sie Ihre Methode `rand_million()` aus Präsenzaufgabe 09-3 und Ihre Ergebnisse aus Aufgabe 10-1 und betrachten Sie die Funktion `System.currentTimeMillis()` (siehe Handbuch). Messen Sie damit die Zeit, die jeweils die Ausführung folgender Listings benötigt, und geben Sie diese in der Kommandozeile mit der korrekten Einheit aus.

0.
```
int[] arr = rand_million();
```

1.
```
Einfach_verkettete_Liste liste = new Einfach_verkettete_Liste(arr);
```

2.
```
for (int i = 0; i < arr.length; i++) {
        arr[i] = 0;
}
```

3.
```
for (Knoten k = liste.kopf; k != null; k = k.next) {
        k.wert = 0;
}
```