Christian-Albrechts-Universität zu Kiel
Sommersemester 2000
Fortgeschrittenenpraktikum (Java)

Übung 1:

Ausgabetermin: 17. April 2000

Abgabe: 2. Mail 2000

In diesem Übungszettel geht nicht um die Benutzung der API, sondern um die Definition und Manipulation von *Datenstrukturen*, wobei die Verwendung der *Konstruktoren*, *Methodenaufrufe* und *Vererbung* im Mittelpunkt stehen sollen. Die folgenden kleineren Aufgaben bauen teilweise aufeinander auf. Später werden wir (vielleicht) diese Klassen noch erweitern.

Hinweis: alle folgenden Aufgaben sollen lauffähig (nicht nur Kompilierbar) sein. Das bedeutet, es sollte zu jeder Aufgabe noch eine kleine **Main**-Klasse programmiert werden, die durch Aufruf von ein Paar Methoden und das Ausdrucken der Ergebnisse die "Korrektheit" plausibel macht. Ein einfaches `System.out.println` genügt, Graphik kommt das nächste Mal.

Aufgabe 1: [Pizza]

Ziel ist es, den zentralen Datentyp für eine *Pizzabäckerei* objektorientiert darzustellen. Dieser Datentyp ist die *Pizza*.

Wir stellen uns vor, eine Pizza bestehe aus einem *Pizzaboden*, und als Belag stehen Zwiebeln, Salami, Knoblauch, Oliven, Thunfisch und Lammfleisch zur Verfügung. Präziser: ein `Boden`) ist eine Pizza, und belegt man eine Pizza mit beispielsweise `Zwiebel`, ist es immer noch eine Pizza.[1]

- **Programmieren sie also den Datentyp** `Pizza_D`.[2]
  Eine bestimmte Pizza soll man beispielsweise mittels

  ```
  Pizza_D p = new Zwiebel(new Olive (new Boden()));
  ```

  bekommen können.
- Um die Diagnose des Kommenden zu erleichtern, brauchen wir eine **Methode zur Ausgabe** der Daten. Für eine gegebene Pizza wollen wir also eine Auflistung der Zutaten inklusive des Boden. Hinweis: das Einfachste wird sein, `System.out.println` zu verwenden, wobei `println`, wie in der Vorlesung erwähnt, nach einer Methode `toString` sucht.

Aufgabe 2: [Pizza: lesender Zugriff]

[1] Eine induktive Definition von Pizzas, sozusagen.
[2] das `D` soll an „Daten" erinnern.

Neben dem Konstruieren von Daten wird man auf die Daten lesend zugreifen wollen[3] Wir wollen zwei Methode

1. `istVegetarisch()`
2. `nurZwiebeln()`

wobei die erste Methode bei einer gegebenen Pizza überprüft, ob sie kein Fleisch enthält, die zweite, ob sie nur aus Zwiebeln besteht. Beispielsweise sei für das weiter oben definierte `p` der Wert `p.istVegetarisch()` gleich `true`, der für `p.nurZwiebeln()` gleich `false`.

Aufgabe 3: [Punkte]

Nehmen wir an, wir wollten Punkte der zweidimensionalen Ebene modellieren. Punkte existieren für uns nicht unabhängig von einer Metrik, das heißt, die Klasse der Punkte (`Point`) sei abstrakt. Im konkreten benutzen wir zwei Metriken, die *Euklidische* und die aus *Manhattan*.[4] Programmieren Sie die Klassen `CartesianPt` und `MannhattanPt`, beides seien Punkte die die Methoden `distanceToO` und `closerToO` verwenden. `distanceToO` bestimmt den Abstand zum Nullpunkt, `closerToO` vergleicht, ob das Objekt näher am Ursprung ist als das Argument. Achten Sie darauf, daß `closerToO` zwei *Punkte* (`Point`) vergleicht, d.h. finden Sie eine möglichst sparsame Lösung, bei der auch kartesische Punkte mit Manhattan-Punkten verglichen werden können und umgekehrt.

Wie sieht es mit der Erweiterbarkeit Ihrer Lösung aus, d.h., beantworten Sie, was mit Ihren Klassen passiert, wenn eine dritte Art von Punkten erfunden wird.[5]

Aufgabe 4: [Nochmal Pizza]

Nehmen Sie die Definition der Pizzas aus einer der vorangegangenen Aufgaben. Schreiben Sie zwei Methoden, die aus Pizzas arbeiten und die

- aus einer Pizza allen Knoblauch entfernen
- in einer Pizza Zwiebeln durch Lamm ersetzen

Fügen Sie den Pizza-Klassen die Methoden `remKnoblauch` und `subPZdL` hinzu.

Aufgabe 5: [Wiederum Pizza: Trennung von Daten und Prozeduren]

[3] Im Grunde war das Ausdrucken in der vorangegangenen Aufgabe bereits ein lesender Zugriff.

[4] Der Abstand eines Punktes vom Ursprung mit euklidischer Norm oder kartesischen Koordinaten ist $\sqrt{x^2+y^2}$, in Manhattan wäre er $|x|+|y|$.

[5] mit anderen Worten: ein Teil der Klassenbibliothek an der Sie gearbeitet hatten, nämlich der Punkte-Teil, muß erweitert werden und Sie waren mal für `ManhattanPt` und `CartesianPointPt` zuständig.

In dieser Aufgabe soll wieder mit den Pizzas gearbeitet werden, allerdings wird sich die Definition nun ändern. Wir hatten ein bestimmtes Datum in Klassen organisiert dargestellt und bestimmte Funktionen darauf definiert, in dem wir entsprechende Methoden in die Klassen gesteckt hatten. Um Grunde kann man die Aktion "*entferne-die-Zwiebeln*" oder "*nur-Zwiebeln?*" als eine gedankliche Einheit betrachten und nach allem was wir über OO-Design wissen, gehören zusammengehörige „Dinge“ in eine Klasse oder in eine gemeinsame Klassenhierarchie. Das soll nun passieren.

Programmieren Sie eine Klasse `NurZ_V` für „*Nur-Zwiebeln?*“[6] Das Gerüst der Klasse soll so aussehen:

```
class NurZ_V {
  boolean forBoden() {
    return true;
  };
  .........
};
```

Das Problem besteht vor allem darin, sich zu Überlegen, welchen Nutzen man aus einer solchen Klasse machen kann, also: was bekommt man wenn man die Klasse instanziiert, und wer benötigt die Instanzen wo und mit welchen Parametern (`Boden` ist der einfache Fall).

Wenn Sie `NurZ_V` programmiert habe, erstellen Sie analoge Klassen `IstVeg_V` sowie `Sub_ZdL_V` und `rem_Knob_V` anstelle der anderen bereits definierten Methoden.

Aufgabe 6: [8stündig]

Für die 8stündigen Teilnehmer: In den Pizzaaufgaben 1, 2, 4 soll die Interaktion mit den Pizzas graphisch möglich sein, z.B. über ein Applet, d.h. die Konstruktion, der lesende Zugriff und die verändernden Methoden sollen mit den im awt-package zur Verfügung gestellten Möglichkeiten realisiert werden. Die Gestaltung der "Benutzeroberfläche" (Knöpfe, Menues oder ähnliches) bleibt Ihnen überlassen.

[6] Das *V* steht für *Visitor*, das wird vielleicht später erklärt.