Informatik I WS 06/07
Prof. Dr. C. Damm
Dipl.-Inform. Marc Njoku

Übungsblatt 11

Ausgegeben am:	17.01.2007
Abgabe bis:	26.01.2007

Thema: Abstrakte Datentypen, Objektorientierung, Java

Die Theorieaufgaben auf diesem Blatt sind bis zum Freitag, 26. Januar um 14.00 Uhr in die Info-I-Briefkästen im Erdgeschoss der NAM einzuwerfen. Die Kästen sind nach Übungsgruppen geordnet. Achten Sie bitte darauf dass Sie Ihren Zettel in den richtigen Kasten werfen, und dass **Name und Gruppe** auf **jedem Blatt** stehen! Falls Ihre Lösung mehrere Blätter umfasst, heften Sie diese bitte zusammen. Die Praxis-Aufgaben bearbeiten Sie bitte am Rechner und führen Sie Ihrem Tutor am Rechner vor. Eine Bearbeitung in Zweier-Teams innerhalb Ihrer Übungsgruppe ist möglich.

Aufgabe 1 (Theorie: 20 Punkte):

Abstrakte Datentypen

1. Was ist ein Abstrakter Datentyp (ADT) ?
2. Was ist eine Algebra ?
3. Was ist ein Interface?
4. Was ist der Unterschied zwischen abstrakten Klassen und Interfaces?
5. Was bedeutet es Methoden zu überladen? Was heißt Methoden überschreiben? Können Methoden überschrieben und überladen sein?
6. Welche Methoden aus Aufgabe 2 werden überladen, welche überschrieben?

Aufgabe 2 (Praktisch: 44 Punkte):

Vererbung, Polymorphie

1. Implementieren Sie die Klassen *Gebäude*, *Haus*, *Stall*, *Reihenhaus* und *Mehr-Familien-Haus*. Dabei soll eine Klasse als abstrakte Klasse implementiert werden. Implementieren Sie die Attribute *Hausnummer* und *Jahr_der_Erbauung* mit den zugehörigen *get* und *set*-Methoden nur in einer Klasse, aber so, dass alle Klassen sie nutzen können.
   Implementieren Sie das Attribut *Anzahl_Kinderzimmer* mit seinen Methoden in nur einer Klasse, aber so, dass es nur von den Klassen *Haus*, *Reihenhaus* und *Mehr-Familien-Haus* genutzt werden kann.
2. Schreiben Sie eine Testklasse *Gebäudeinspektor*, die Objekte von allen Klassen mit
   `Objekt = new Klasse (Hausnummer, Jahr_der_Erbauung)`
   erzeugt.
3. In der Klasse *Stall* sollen nun maximal drei verschiedene Tierarten untergebracht werden. Fügen Sie der Klasse hierfür ein geeignetes Attribut hinzu, sowie die Methoden *getTierart* und *setTierart(String Tierart, [String Tierart], [String Tierart])*.
   (Hinweis: Die Klammern *[]* bedeuten: optional)
4. Fügen Sie der Klasse *Mehr-Familien-Haus* ein Attribut *Anzahl_Wohnungen* und die zugehörigen *get* und *set*-Methoden hinzu. Erlauben Sie die Eingaben *setAnzahlWohnungen(3)* und *setAnzahlWohnungen("drei")*.
   (Hinweis: ein Mehr-Familien-Haus soll aus mindestens 2 und höchstens 10 verschiedenen Wohnungen bestehen.).
5. **Nur** in der Klasse *Mehr-Familien-Haus* soll nun in der Methode *getAnzahlKinderzimmer()* die

Anzahl der Kinderzimmer durch die Anzahl der Wohnungen dividiert werden.

6. Fügen Sie jeder Klasse von der Sie Objekte erzeugen können eine Methode *toString()* hinzu, die ausgibt um was es sich handelt, (z.B."Mehr-Familien-Haus").
7. Erweitern Sie die Testklasse *Gebäudeinspektor* indem Sie alle implementierten Funktionen testen.
8. Rufen Sie in der Testklasse *Gebäudeinspektor* insbesondere auch die Methoden ihrer Objekte auf, die nicht in der zugehörigen Klasse, sondern in der Oberklasse implementiert wurden!

Aufgabe 3 (Theorie: 12 Punkte):

ADT Stack, ADT Queue

1. ADT Stack wird oft unter dem Namen LIFO (Last-in-First-out) und ADT Queue häufig unter dem Begriff FIFO (First-in-First-out) geführt. Erläutern Sie warum.
2. Was wäre dann ein "LILO (Last-in-Last-out)" und was ein "FILO (First-in-Last-out)" ?

Aufgabe 4 (Theorie: 24 Punkte):

Objektorientierte Programmierung

In folgendem Java-Codefragment sind einige Fehler enthalten. Die Zeilen des Aufretens sind durch Kommentare markiert. Lokalisieren Sie die Fehler näher, indem Sie die genauen Fehlerstellen im Programmtext markieren. Geben Sie für jeden Fehler eine kurze Fehlerbeschreibung an, die eine mögliche Meldung des Java-Systems widerspiegelt. Machen Sie insbesondere eine Aussage dazu, ob der Fehler zur Übersetzungszeit oder zur Laufzeit auftritt. Schlagen Sie vor, wie der Fehler möglichst einfach im Sinne der beabsichtigten Wirkungsweise des Programmes korrigiert werden kann.

```
class List {
        Element ListHead;

        public int zeroCnt(int x) {

        // Zaehlt die Anzahl der Nullen vor dem ersten Auftreten von x.
        // Falls x nicht vohanden ist, wird 0 zurueckgegeben.

                int cnt;
                Element p = ListHead;

                while (p.dat != x && p != null)                 // Fehlerzeile 1
                        if (p.dat = 0)                          // Fehlerzeile 2
                                cnt++;                          // Fehlerzeile 3(a)
                        else
                                p = p.next;
                return cnt;                                     // Fehlerzeile 3(b)
        }

        private class Element {
                int dat;
                Element next;
        }

        public static void main(String[] args) {
                List lst = new List;                            // Fehlerzeile 4
                        :
                // Aufbau von lst. Dieser Code ist fehlerfrei.
                        :
                lst.zeroCnt(3);
        }
}
```