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

Übungsblatt 12

Ausgegeben am:	24.01.2007
Abgabe bis:	02.02.2007

Thema: Abstrakte Datentypen, Objektorientierung, Java , Binäre Suchbäume, UML

Die Lösungen der Aufgaben auf diesem Blatt werden am 02.02.2007 im Stud.IP freigegeben.

Aufgabe 1 (Theorie: 15 Punkte):

Abstrakter Datentyp Queue

Aus der Vorlesung kennen Sie den abstrakten Datentyp `Stack` von Elementen des Typs `T`:

Operatoren:	Axiome:
• create: → Queue • enq: (Queue,T) → Queue • deq: Queue → Queue • first: Queue → T • is_empty: Queue → Bool	• (A) deq(enq(create,x)) = create • (B) deq(enq(enq(q,y),x)) = enq(deq(enq(q,y)),x) • (C) first(enq(create,x)) = x • (D) first(enq(enq(q,x),y)) = first(enq(q,x)) • (E) is_empty(create) = true • (F) is_empty(enq(q,x)) = false

Berechnen Sie die Normalform von:

1. first(enq(enq(create,1),2))
2. deq(enq(enq(enq(create,1),2),3))
3. is_empty(deq(enq(deq(enq(enq(create,1),2)),3)))
4. enq(enq(enq(create,1),2), first(enq(enq(create,3),4)))

Hinweis: die Musterlösung zur Aufgabe 1 der Saalübung 11 (im Stud.IP) enthält sämtliche Definitionen zur algebraischen Spezifikation von abstrakten Datentypen. Dazu gehören die ausführlichen Erläuterungen der Begriffe *Algebra*, *Axiom* und *Normalform*.

Aufgabe 2 (Praktisch: 25 Punkte):

Abstrakter Datentyp Queue

Gegeben ist die Klasse `intList` *, die eine Liste bereits implementiert.

Implementieren Sie die Klasse `intList` neu als doppelt verkettete Liste. Die neue Klasse `intDVList` soll dabei die Methoden/Funktionalitäten des abstrakten Datentyps Queue umsetzen.

Lassen Sie Ihre umgeschriebe Klasse `intDVList` testieren.

*: http://user.informatik.uni-goettingen.de/~damm/Informatik-I/java/

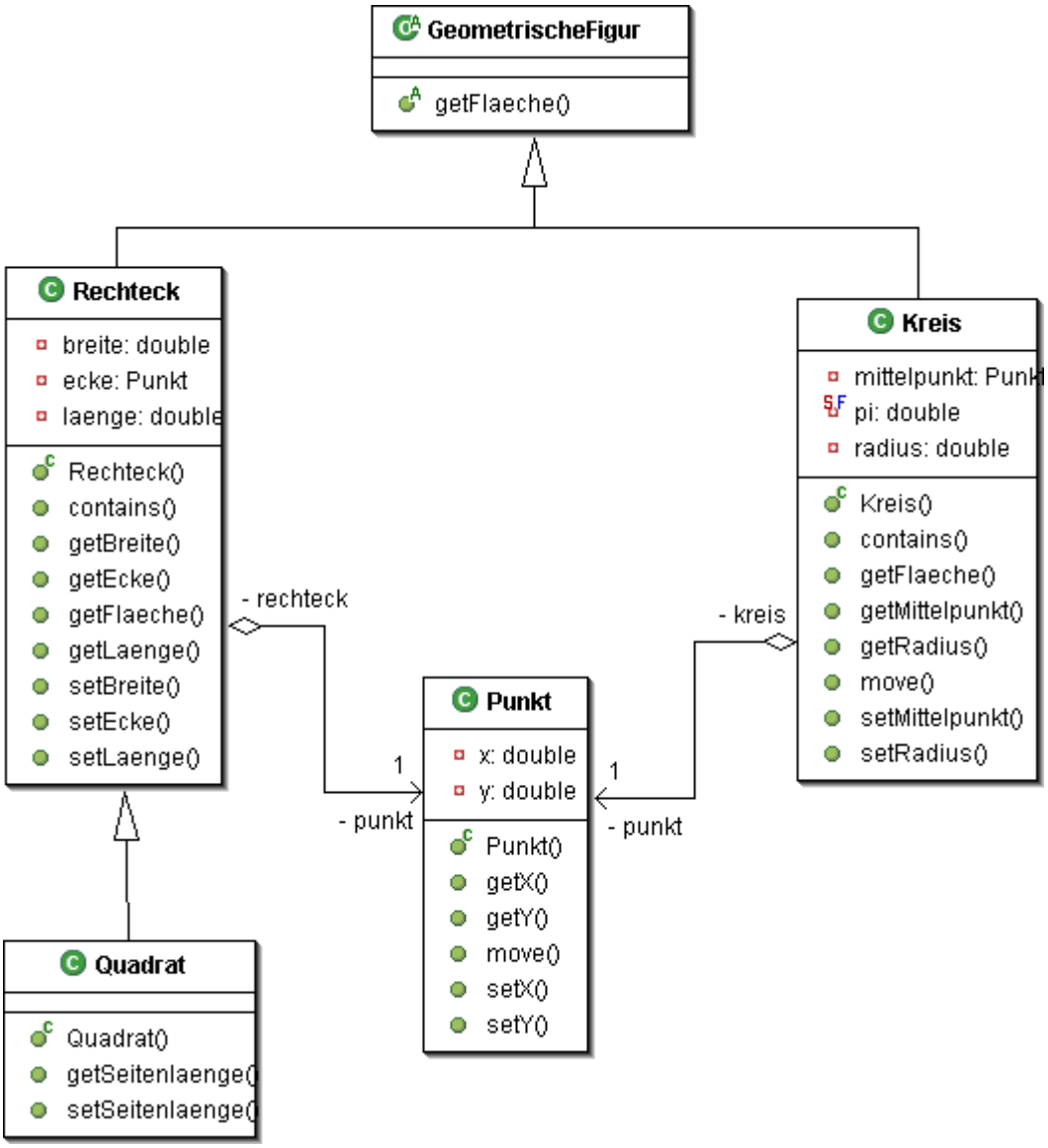
Aufgabe 3 (Theorie: 20 Punkte):

Binärbaum, Pre-Order, In-Order

1. Sie erhalten eine Liste von Zahlen, von der Ihnen gesagt wird, es handele sich um die (eindeutigen) Knotenschlüssel eines Binärbaums in Preorder- Reihenfolge. Können Sie daraus den ursprünglichen Baum eindeutig rekonstruieren?
2. Sie erhalten nun zwei Listen von Zahlen, einmal die Preorder- und einmal die Inorder- Reihenfolge der Knoten desselben Binärbaums. Ist hier eine eindeutige Rekonstruktion möglich?

Aufgabe 4 (Praktisch: 40 Punkte):

Punkte und Kreise



1. <u>Teilaufgabe</u>

   Schreiben Sie die Java-Klassen "Punkt", "GeometrischeFigur", "Kreis","Rechteck" und "Quadrat" gemäß dem gegebenen UML-Diagramm. Beachten Sie folgendes:

   Ein **Punkt** hat:
   - eine Position, bestehend aus x- und y-Koordinate (als private deklarieren)
   - get- und set-Methoden für x- und y-Koordinate

- eine Methode `move(double deltaX, double deltaY)`, mit der man die Position des Punktes um deltaX bzw. deltaY verschieben kann
- und einen Konstruktor, der mit x- und y-Koordinate aufgerufen wird.

Eine **Geometrische Figur** hat:
- eine *abstrakte* Methode `getFlaeche()`, die einen double-Wert zurück liefert.

Ein **Kreis**
- ist ein Spezialfall einer Geometrischen Figur
- hat einen Radius (private)
- hat eine Methode `move(...)`, die die Methode `Punkt.move(...)` verwendet
- implementiert die Methode `getFlaeche()`, die den Flächeninhalt des Kreises zurück liefert
- hat eine Methode `boolean contains(...)`, die angibt, ob ein bestimmter Punkt innerhalb des Kreises liegt
- hat get- und set-Methoden für den Radius
- und einen Konstruktor, der mit dem einem Objekt vom Typ `Punkt` als Mittelpunkt und einem Radius aufgerufen wird.

Ein **Rechteck**:
- ist ein Spezialfall einer geometrischen Figur
- hat eine Breite (private)
- hat eine Länge (private)
- hat eine Methode `move(...)`, die die Methode `Punkt.move(...)` verwendet
- implementiert die Methode `getFlaeche()`, die den Flächeninhalt des Rechtecks zurück liefert
- eine Methode `boolean contains(...)`, die angibt, ob ein bestimmter Punkt innerhalb des Rechtecks liegt
- get- und set-Methoden für Länge und Breite.
- und einen Konstruktor, der mit dem einem Objekt vom Typ `Punkt` (als linke, obere Ecke) und mit Breite und Länge aufgerufen wird.

Ein **Quadrat**
- ist ein Spezialfall eines Rechtecks
- hat eine zusätzliche Methode `setSeitenlaenge(double s)`, die Höhe und Breite des Rechtecks auf den selben Wert s setzt.
- hat eine Methode `double getSeitenlaenge()`, die `getLaenge` aufruft.
- und einen Konstruktor, der mit dem einem Objekt vom Typ `Punkt` (als linke, obere Ecke) und mit der Seitenlänge des Quadrats aufgerufen wird.

Testen Sie die Lösung bitte mit dem unter
http://user.informatik.uni-goettingen.de/~info1/Java/GeometrieTest.java bereit liegenden Testprogramm.
Für x- und y-Koordinaten, Radius oder sonstige skalare Werte verwenden Sie bitte den Datentyp "double". Setzen Sie der Einfachheit halber π auf den Wert 3.1415927.

---

2. Teilaufgabe:
   Implementieren Sie für Objekte p vom Typ `Punkt` eine Methode `equals(Object q)`. `p.equals(q)` soll *genau dann* ein "true" liefern, wenn *alle* folgenden Punkte erfüllt sind:
   - Objekt q ist vom Typ "Punkt"
   - der Punkt q zeigt auf die selben x- und y-Koordinaten wie der Punkt p.

---

3. Teilaufgabe:
   Implementieren Sie für Objekte k vom Typ `Kreis` eine Methode `equals(Object q)`. `k.equals(q)` soll *genau dann* ein "true" liefern, wenn *alle* folgenden Punkte erfüllt sind:

- Objekt q ist vom Typ "Kreis"
- die Mittelpunkte von q und p sind identisch
- q und p haben den selben Radius.

---

4. Teilaufgabe:
   Implementieren Sie für Objekte p vom Typ `GeometrischeFigur` eine Methode `int compareTo(Object q)`. `p.compareTo(q)` soll:
   - **-1** liefern, falls die Fläche von Figur p *kleiner* ist als die von q
   - **+1** liefern, falls die Fläche von Figur p *größer* ist als die von q
   - **-1** liefern, falls die Flächen von Figur p und Figur q *gleich* sind.

Hinweis: Interfaces werden verwendet, um Eigenschaften auszudrücken, die auf Klassen aus unterschiedlichen Klassenhierarchien zutreffen können. Das erkennt man auch daran, daß ihre Namen oft (substantivierte) Eigenschaftswörter sind. Ein bekanntes Beispiel ist das Interface `Comparable` des Pakets `java.lang` :

```
public interface Comparable { public
        int compareTo(Object o);
}
```

Dieses Interface kann von Klassen implementiert werden, deren Objekte paarweise *vergleichbar* sind.

Mit Hilfe von `Comparable` kann die Reihenfolge der Objekte einer Klasse ermittelt werden. Aus dem paarweisen Vergleich läßt sich eine (nicht notwendigerweise eindeutige) implizite Ordnung der Elemente ableiten, denn für alle aufeinanderfolgenden Objekte `a` und `b` muß `a.compareTo(b) <= 0` gelten. Damit ist es möglich, Methoden zu schreiben, die das kleinste oder größte Element einer Menge von Objekten ermitteln oder diese sortiere.
Auch Objekte unterschiedlicher Klassen können problemlos miteinander verglichen werden, sofern `compareTo` dazu in der Lage ist. So ist es leicht vorstellbar, daß sowohl Autos als auch Fußballplätze und Papierblätter (und eventuell noch Äpfel und Birnen) miteinander verglichen werden