

# Klausur Informatik II

8. August 2000

Name:

Matr.-Nr.:

Die Klausur ist komplett und geheftet abzugeben. Für die Vollständigkeit nicht gehefteter Klausuren wird keine Verantwortung übernommen. Achten Sie darauf, daß Ihr Name und Ihre Matrikelnummer auf allen Blättern stehen!

Aufg.	1	2	3	4	5	6	7	$\Sigma$	Note
Möglich	15	7	5	9	12	2	10	60	
Punkte									

**Musterlösung**

## Aufgabe 1: Verständnisfragen (15 Punkte)

a) Multiple-choice-Fragen. Nur richtige Zeilen ergeben Punkte. Keine Abzüge. (8 P.)

richtig	falsch	
X	<input type="radio"/>	In Breitensuchbäumen gibt es keine Vorwärtskanten.
X	<input type="radio"/>	Der symmetrische Nachfolger eines Knotens $k$ in einem natürlichen binären Suchbaum ist der Knoten, der den kleinsten Schlüssel im rechten Teilbaum von $k$ enthält.
X	<input type="radio"/>	Ein Objekt ist eine Ausprägung einer Klasse.
<input type="radio"/>	X	Nach einer Zuweisung $a = b$ mit Wertsemantik verweisen $a$ und $b$ auf dasselbe Objekt.
X	<input type="radio"/>	Ein Methodenaufruf ist polymorph, wenn der zu erbringende Dienst von der konkreten Klasse des Objekts abhängt.
X	<input type="radio"/>	Ereignisse realisieren asynchrone Kommunikation.
<input type="radio"/>	X	Beim <i>wp</i> -Kalkül versucht man die stärkste Vorbedingung für ein Programm aus einer gegebenen Nachbedingung zu berechnen.
<input type="radio"/>	X	Den Vorgang des Messens, welche Programmsegmente die größte Laufzeit haben, nennt man <i>symbolische Programmausführung</i> .
X	<input type="radio"/>	Wenn für zwei Prädikate $P$ und $Q$ die Beziehung $Q \rightarrow P$ gilt, dann heißt $P$ schwächer als $Q$ .
<input type="radio"/>	X	Eine Terminierungsfunktion ist streng monoton fallend und nach oben beschränkt.
X	<input type="radio"/>	Das Suchen eines oder aller Ausgänge aus einem Labyrinth kann durch einen rekursiven Algorithmus mit Rücksetzen gelöst werden.
<input type="radio"/>	X	Bei der Fourier-Transformation handelt es sich um eine schwer invertierbare Bereichstransformation.
<input type="radio"/>	X	Die Operationen Einfügen, Suchen und Löschen haben in binären natürlichen Suchbäumen immer den Aufwand $O(\log(n))$ ( $n =$ Anzahl der Knoten)
<input type="radio"/>	X	Hashfunktionen sind immer injektiv.
X	<input type="radio"/>	Ein Schlüssel identifiziert einen Datensatz eindeutig.
<input type="radio"/>	X	Das Klassenmodell beschreibt, wie die Objekte eines Systems zur Laufzeit miteinander interagieren, d. h., in welcher Reihenfolge Methodenaufrufe erfolgen.

**b)** Sortieren und Aufwand. (2 P.)

Für jede richtige Zeile gibt es 0.5 Punkte, für jede falsche Zeile werden 0.5 Punkte abgezogen. Die Aufgabe wird aber nie mit weniger als 0 Punkten bewertet.

Welche der folgenden Sortierverfahren haben einen garantierten asymptotischen Aufwand (also auch im schlechtesten Fall) von  $O(n \cdot \log(n))$ ? Dabei ist  $n$  der Eingabumfang.

<input type="radio"/>	Quicksort
<input checked="" type="checkbox"/>	Heapsort (Baumsortieren)
<input type="radio"/>	Bubblesort (Blasensortieren)
<input checked="" type="checkbox"/>	Mergesort (Mischsortieren)

**c)** Polymorphie (2 P.)

Welche Ausgabe liefert das folgende Programm?

```
class Y {
    public int m(int x) {
        return x*x;
    }
}

class YY extends Y {
    public int m(int x) {
        return x+x;
    }
}

public class M {
    public static void main(String[] args) {
        Y y1 = new Y();
        YY yy1 = new YY();
        Y y2 = yy1;

        System.out.println(y1.m(3));
        System.out.println(y2.m(3));
        System.out.println(yy1.m(3));
    }
}
```

**Lösung:**

9  
6  
6

**d)** Parallelität. (1.5 P.)

Welche konzeptuellen Klassen von Parallelität wurden in der Vorlesung vorgestellt?

*Hinweis:* Diese Klassen wurden im Artikel von Carriero und Gelernter definiert.

**Antwort:** Ergebnisparallelität, Agendaparallelität und Spezialistenparallelität.

**e)** Datenstrukturen (0.5 P.)

Welchen Aufwand haben die Operationen Einfügen, Suchen und Löschen in Rot-Schwarz-Bäumen im schlechtesten Fall, wenn  $n$  die Anzahl der Knoten ist?

**Antwort:**  $O(\log(n))$ .

**f)** Aufwand (1 P.)

Gegeben sei folgende Rekurrenz:

$$T(n) = \begin{cases} 1 & , \text{ falls } n = 1 \\ 8 \cdot T(n/2) + n & , \text{ falls } n > 1 \end{cases}$$

Dabei sei  $n$  stets eine ganzzahlige Potenz von 2.

Geben Sie die minimale asymptotische Lösung im  $O$ -Kalkül für diese Rekurrenz an. Minimal heißt dabei, daß Sie von den oberen Schranken die kleinste angeben sollen, also beispielsweise  $O(\log(n))$  statt  $O(n)$ . Vereinfachen Sie außerdem  $O(n + c)$  mit  $c$  konstant zu  $O(n)$ , oder wenden Sie die anderen Rechenregeln des  $O$ -Kalküls zur Vereinfachung Ihres Ergebnisses an.

Verwenden Sie zur Lösung der Rekurrenz einen Satz der Vorlesung bzw. Übung, und geben Sie an, warum der von Ihnen gewählte Fall des Satzes zutrifft. Geben Sie dazu die entsprechende Ungleichung an.

**Lösung:**  $O(n^{\log_2(8)}) = O(n^3)$ , weil  $8 > 2$ .

## Aufgabe 2: Verifikation (2+3.5+1.5 = 7 Punkte)

a) Geben Sie eine Invariante  $p$  an, so daß die Prädikate in dem folgenden annotierten Programm beim Passieren der entsprechenden Programmstelle jeweils gültig sind:  
Machen Sie in Ihrer Invariante eine Aussage über den Wert von  $s$  und eine Aussage über mögliche Werte von  $i$ . Über die Zulässigkeit von  $a[i]$  brauchen Sie sich keine Gedanken machen. Außerdem dürfen Sie annehmen, daß  $N \geq 0$  gilt.

```
int i = N, s = 0;
// {i = N ∧ s = 0}
// {p}
while (i >= 0) {
  // {p ∧ (i ≥ 0)}
  s = s + a[i];
  i = i - 1;
  // {p}
}
// {p ∧ (i < 0)}
// {s = ∑k=0N a[k]}
```

### Lösung:

Als Invariante definiert man:

$$p := (-1 \leq i \leq N) \wedge (s = \sum_{k=i+1}^N a[k])$$

oder

$$p := (i \geq -1) \wedge (s = \sum_{k=i+1}^N a[k])$$

Dabei ist jeweils  $\sum_{k=N+1}^N a[k] := 0$ .

**b)** Beweisen Sie für die von Ihnen in Teilaufgabe 1 getroffene Wahl von  $p$  mit Hilfe des  $wp$ -Kalküls:

$$\{p \wedge (i \geq 0)\}$$

$$s = s + a[i];$$

$$i = i - 1;$$

$$\{p\}$$

Sie müssen also zeigen, daß gilt:  $(p \wedge i \geq 0) \Rightarrow wp("s = s + a[i]; i = i - 1;", p)$

Geben Sie dabei alle Zwischenschritte an.

**Lösung:**

$$wp("s = s + a[i]; i = i - 1;", (-1 \leq i \leq N) \wedge (s = \sum_{k=i+1}^N a[k]))$$

$$\stackrel{\text{Komposition}}{=} wp("s = s + a[i];", wp("i = i - 1;", (-1 \leq i \leq N) \wedge (s = \sum_{k=i+1}^N a[k])))$$

$$\stackrel{\text{Zuweisung inkl. Subst.}}{=} wp("s = s + a[i];", (-1 \leq i - 1 \leq N) \wedge (s = \sum_{k=i}^N a[k]))$$

$$\stackrel{\text{alg. Umformung}}{=} wp("s = s + a[i];", (0 \leq i \leq N + 1) \wedge (s = \sum_{k=i}^N a[k]))$$

$$\stackrel{\text{Zuweisung inkl. Subst.}}{=} (0 \leq i \leq N + 1) \wedge (s + a[i] = \sum_{k=i}^N a[k]))$$

$$\stackrel{\text{alg. Umformung}}{=} (0 \leq i \leq N + 1) \wedge (s = \sum_{k=i+1}^N a[k]))$$

$$\Leftrightarrow (i \geq 0) \wedge (-1 \leq i \leq N) \wedge (s = \sum_{k=i+1}^N a[k])$$

**c)** Zeigen Sie:

$$(p \wedge (i < 0)) \Rightarrow s = \sum_{k=0}^N a[k]$$

**Lösung:**

$$\begin{aligned} & (-1 \leq i \leq N) \wedge (s = \sum_{k=i+1}^N a[k]) \wedge (i < 0) \\ \equiv & (i = -1) \wedge (s = \sum_{k=i+1}^N a[k]) \\ \equiv & s = \sum_{k=0}^N a[k] \end{aligned}$$

### Aufgabe 3: Dynamisches Programmieren (3+2 = 5 Punkte)

Gegeben sei die Adjazenzmatrix  $A$  des Graphen  $G = (E, K)$ . Die Kosten der Kante  $(e_i, e_j)$  sind dann durch  $a_{ij}$  gegeben, wobei  $a_{ij} \geq 0$  gelte. Für nicht adjazente Ecken  $e_i$  und  $e_j$  ist  $a_{ij} = +\infty$ .

a) Beschreiben Sie ein Verfahren, das mittels dynamischen Programmierens für alle Paare  $(i, j)$  die geringsten Kosten eines Weges von  $e_i$  nach  $e_j$  berechnet. Geben Sie dazu lediglich die rekursive Definition der Berechnung der Werte  $\omega_{ij}$  der Bewertungsfunktion als mathematische Formel an (kein Programm, kein Pseudocode).

**Lösung:**

$$\omega_{ij} = \begin{cases} 0, & \text{falls } i = j \\ \min(\{\omega_{ik} + \omega_{kj} \mid k \neq i \wedge k \neq j\} \cup \{a_{ij}\}), & \text{sonst} \end{cases}$$

Der Algorithmus findet  $\omega_{ij} \leq a_{ij}$ . Falls es einen Weg von  $e_i$  über  $e_k$  nach  $e_j$  mit geringeren Kosten als  $a_{ij}$  gibt, dann ist  $\omega_{ij}$  echt kleiner als  $a_{ij}$ .

b) Worauf müssen Sie achten, wenn Sie die Werte  $\omega_{ij}$  mittels dynamischen Programmierens statt durch Rekursion berechnen wollen?

**Lösung:**

Berechnung der  $\omega_{ij}$ : Man muß bei der Berechnung die Abhängigkeiten der  $\omega_{ij}$  berücksichtigen und alle Zwischenergebnisse in einer Tabelle abgespeichern, damit man sie nicht ständig erneut berechnen muß.

## Aufgabe 4: Schrittweises Ausschöpfen (greedy, gierige Algorithmen) (4+5 = 9 Punkte)

a) Jede natürliche Zahl  $n$  läßt sich als Summe verschiedener Fibonacci-Zahlen darstellen:

$$n = F_{k_1} + \dots + F_{k_r} \text{ mit } r > 0, F_{k_1} > 0, F_{k_1} < F_{k_2} < \dots < F_{k_r}$$

Dabei sind die  $F_{k_j}$  Fibonacci-Zahlen, die durch die folgende Rekursion definiert sind:

$$F_0 = 1, F_1 = 1, F_i = F_{i-2} + F_{i-1}$$

Formulieren Sie einen Algorithmus als Java-Methode, der nach der Methode der schrittweisen Ausschöpfung (gierig, greedy) eine solche Darstellung für eine natürliche Zahl  $n$  berechnet.

Die Ausgabe des Algorithmus soll das Tupel  $(k_1, \dots, k_r)$  in genau dieser Reihenfolge sein (als **String**).

Verwenden Sie dazu das unten angegebene Rahmenprogramm und tragen Sie die fehlenden Anweisungen an der markierten Stelle ein. Sie dürfen annehmen, daß es in der Klasse **F** eine Funktion `int fib(int j)` gibt, so daß `fib(i)` die  $i$ -te Fibonacci-Zahl berechnet.

```
public class F {
    public static void main(String[] args) {
        if (args.length != 1) { System.exit(1); }
        int n = Integer.parseInt(args[0]);
        System.out.println("Fibonacci-Zerlegung von " + n + " = " + fibo(n));
    }

    public static int fib(int j) { ... }

    public static String fibo(int n) {
        // Schachtele n durch Fibonacci-Zahlen:
        int j = 1;
        while (fib(j) <= n) {
            j = j+1;
        }
        // Nun gilt: fib(j-1) <= n < fib(j)
        // Initialisieren des Lösungstupels
        String L = "";
        // Tragen Sie hier Ihren Lösungscode zum Ausschöpfen ein!!!

        while (n > 0) {
            j = j-1;
            if (fib(j) <= n) {
                n = n - fib(j);
                L = j + ", " + L;
            }
        }

        return L;
    }
}
```

**b)** Zeigen Sie durch vollständige Induktion, daß sich jede natürliche Zahl  $n$  als Summe verschiedener Fibonacci-Zahlen darstellen läßt:

$$n = F_{k_1} + \dots + F_{k_r}$$

mit  $r > 0$ ,  $F_{k_1} > 0$ ,  $F_{k_1} < F_{k_2} < \dots < F_{k_r}$ .

Dabei sind die Fibonacci-Zahlen wie in Teilaufgabe a) definiert. Verwenden Sie für Ihren Induktionsschritt die Tatsache, daß es eine eindeutig bestimmte natürliche Zahl  $j$  gibt, so daß sich jedes  $n$  folgendermaßen durch Fibonacci-Zahlen schachteln läßt:  $F_j \leq n < F_{j+1}$

### Lösung:

Die Behauptung wird durch vollständige Induktion über  $n$  gezeigt.

*Induktionsanfang:*  $n = 1$

1 ist selbst Fibonacci-Zahl und die Darstellung von  $n = 1$  ist:  $n = F_0$  oder  $n = F_1$ .

*Induktionsannahme:* Für alle  $n' < n$  existiert die gewünschte Zerlegung.

*Induktionsschluß:* Zu zeigen: Die Zerlegung existiert auch für  $n$ .

Sei  $j$  der eindeutig bestimmte Index, so daß gilt:  $F_j \leq n < F_{j+1}$ .

**1. Fall:**  $n = F_j$  ist die gesuchte Zerlegung.

**2. Fall:**  $F_j < n < F_{j+1}$

Dann gilt:

$$0 < n - F_j < F_{j+1} - F_j = F_{j-1}$$

Wegen  $F_j > 0$  ist  $n - F_j < n$ . Nach Induktionsannahme läßt sich also  $n' := n - F_j > 0$  als Summe verschiedener Fibonacci-Zahlen darstellen, deren größte wegen  $n - F_j < F_{j+1} - F_j = F_{j-1}$  echt kleiner als  $F_{j-1}$  ist. Also:

$$n' = n - F_j = F_{k_1} + \dots + F_{k_{r'}}, \text{ mit } F_{k_{r'}} < F_{j-1}$$

Damit:

$$n = F_{k_1} + \dots + F_{k_{r'}} + F_j$$

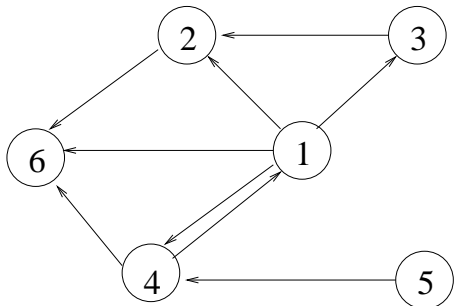
Wegen  $j > 1$  gilt:  $F_{j-1} < F_j$ . Damit ist gezeigt, daß in der so konstruierten Darstellung von  $n$  nur verschiedene Fibonacci-Zahlen vorkommen.

## Aufgabe 5: Suchen und Hashen (5+3+4=12 Punkte)

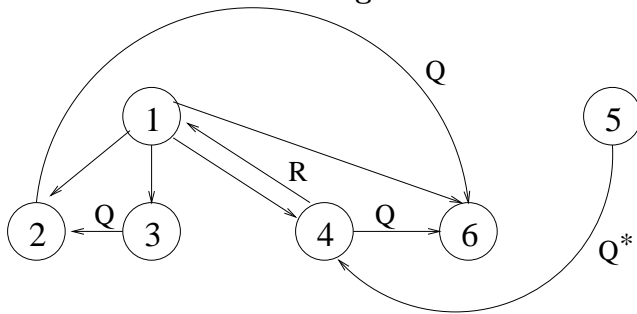
a) Breiten- und Tiefensuche (5 P.)

Konstruieren Sie einen Breiten- und einen Tiefensuchwald für den unten angegebenen Graphen. Wählen Sie die Ecke mit der kleinsten Nummer, wenn mehrere ausgewählt werden können. Starten Sie die Suchen jeweils bei Ecke 1.

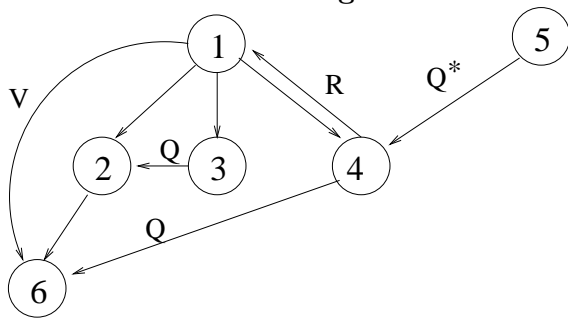
Geben Sie zusätzlich jeweils alle Vorwärts-, Rückwärts- und Querkanten an, wobei Sie Vorwärtskanten mit dem Buchstaben  $V$ , Rückwärtskanten mit dem Buchstaben  $R$ , Querkanten innerhalb eines Suchbaumes mit dem Buchstaben  $Q$  und Querkanten zwischen Suchbäumen mit  $Q^*$  markieren.



**Breitensuchwald: Lösung:**



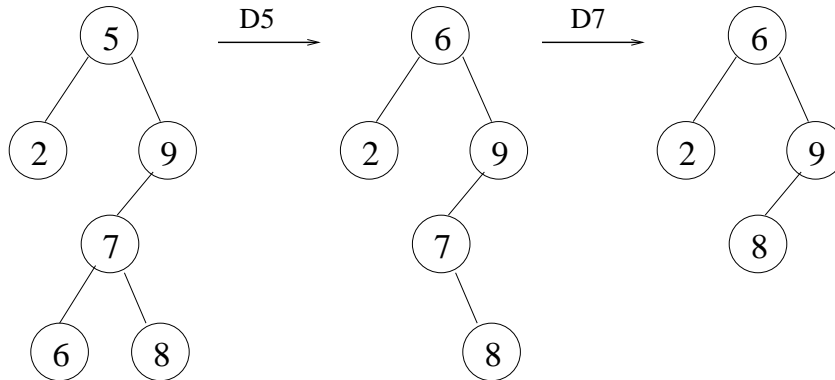
**Tiefensuchwald: Lösung:**



**b)** Binärer natürlicher Suchbaum: (3 P.)

Fügen Sie die Schlüssel 5, 2, 9, 7, 6, 8 in der angegebenen Reihenfolge in einen binären natürlichen Suchbaum ein. Es reicht, wenn Sie das Endergebnis des Einfügens angeben.

Löschen Sie anschließend die Schlüssel 5 und 7 (in dieser Reihenfolge) aus dem von Ihnen aufgebauten Baum. Falls notwendig, plazieren Sie den symmetrischen Nachfolger des gelöschten Schlüssels an der passenden Position. Zeichnen Sie für jeden Schritt beim Löschen von Schlüsseln den entstehenden Suchbaum.



**c)** Hashen mit quadratischer Sondierung. (4 P.)

Geben Sie die Belegung einer Hashtabelle der Größe 7 an, wenn die Schlüssel 24, 3, 9, 17 in die anfangs leere Tabelle in dieser Reihenfolge eingetragen werden.

Dabei sei  $h(k) = k \bmod 7$  und Kollisionsauflösung soll durch quadratisches Sondieren mit Sondierungsfunktion  $s(k, i) = \lceil i/2 \rceil^2 (-1)^i$  erfolgen.

**Achtung:** Wählen Sie beim Sondieren die Variante des *Subtrahierens* der Sondierungsfunktion von der Hashfunktion.

Geben Sie bei jedem eingetragenen Schlüssel an, nach wievielen Kollisionen der Schlüssel auf seinem Tabelleneintrag gelandet ist.

**Lösung:**

Der Tabellenindex für den Schlüssel  $k$  berechnet sich aus:  $(h(k) - s(k, i)) \bmod 7$  für  $i = 0, 1, \dots$

Position:	0	1	2	3	4	5	6
Schlüssel:	17		9	24	3		
# Kollisionen:	3		0	0	1		

## Aufgabe 6: Übersetzung von Ausdrücken (2 Punkte)

Geben Sie für den arithmetischen Ausdruck

$$a + b * (c + d)$$

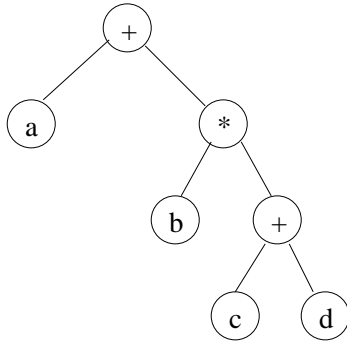
den Kantorowitsch-Baum an, erzeugen Sie aus diesem Baum durch Postorder-Durchlauf eine Postfixdarstellung des arithmetischen Ausdrucks und geben Sie eine Anweisungsfolge an, die den arithmetischen Ausdruck unter Zuhilfenahme eines Kellers auswertet. Erlaubte Anweisungen sind:

**load v** Legt den Wert v auf den Keller.

**add** Nimmt die beiden obersten Kellerelemente vom Keller herunter, addiert sie und legt das Ergebnis der Addition auf dem Keller ab.

**mul** Nimmt die beiden obersten Kellerelemente vom Keller herunter, multipliziert sie und legt das Ergebnis der Multiplikation auf dem Keller ab.

**Kantorowitsch-Baum:**



**Postfixdarstellung:**  $a b c d + * +$

**Anweisungsfolge:**

```
load a
load b
load c
load d
add
mul
add
```

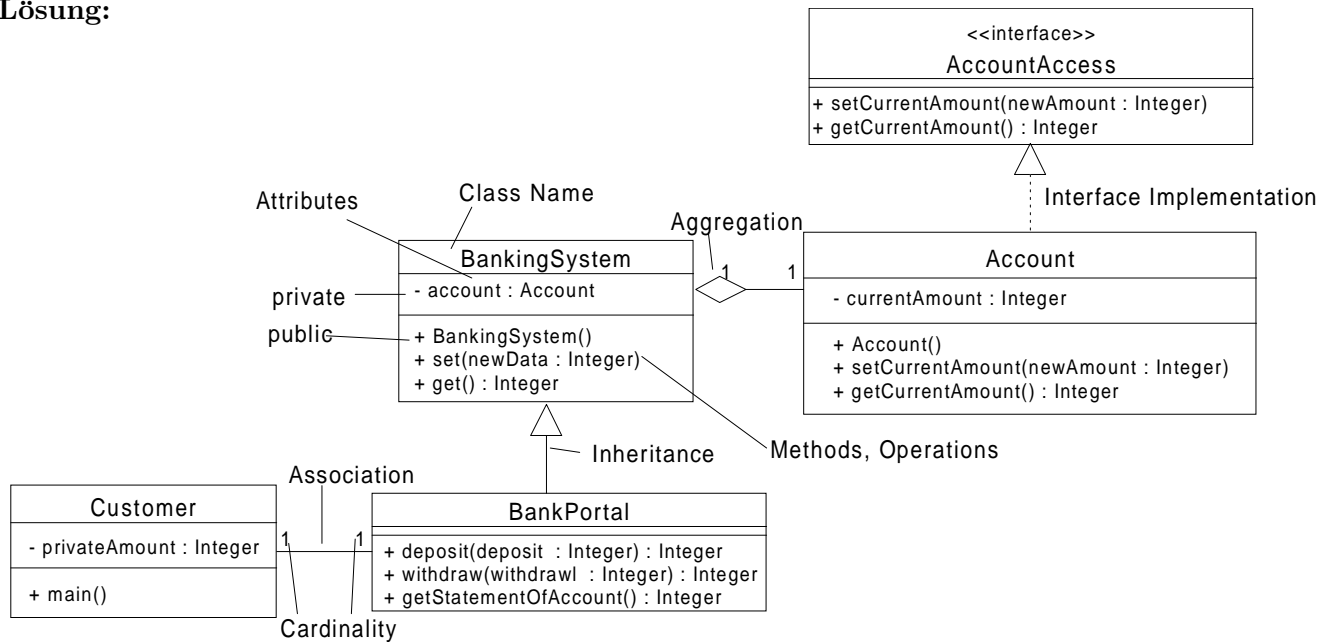
## Aufgabe 7: Objektorientierte Modellierung (6+4 = 10 Punkte)

Firma X hat undokumentierte Software im Einsatz, wobei die Entwickler nicht mehr in der Firma tätig sind. Da neue Funktionalität ergänzt werden muß (Wartung), muß ein Sanierungsprojekt (Reengineering) gestartet werden. Hierzu sollen zur vorhandenen Software zunächst UML-Diagramme erstellt werden.

Bei den folgenden Aufgaben sind alle Java-Systemklassen und Aufrufe bzw. Beziehungen zu diesen zu vernachlässigen!

a) Erstellen Sie ein Klassendiagramm (Notation: UML) aus dem auf den Seiten 15 und 16 angegebenen Java-Code. Geben Sie dabei alle Attribute und Methoden der jeweiligen Klassen in Ihrem Diagramm an. Kennzeichnen Sie dabei private Attribute und Methoden jeweils mit einem vorangestellten Minuszeichen (-) und öffentliche (public) Attribute und Methoden jeweils mit einem Pluszeichen (+). Kardinalitäten und Rollen an den Beziehungen sind optional. Attribute, die schon als Assoziationen, Kompositionen oder Aggregationen eingezeichnet sind, können weggelassen werden.

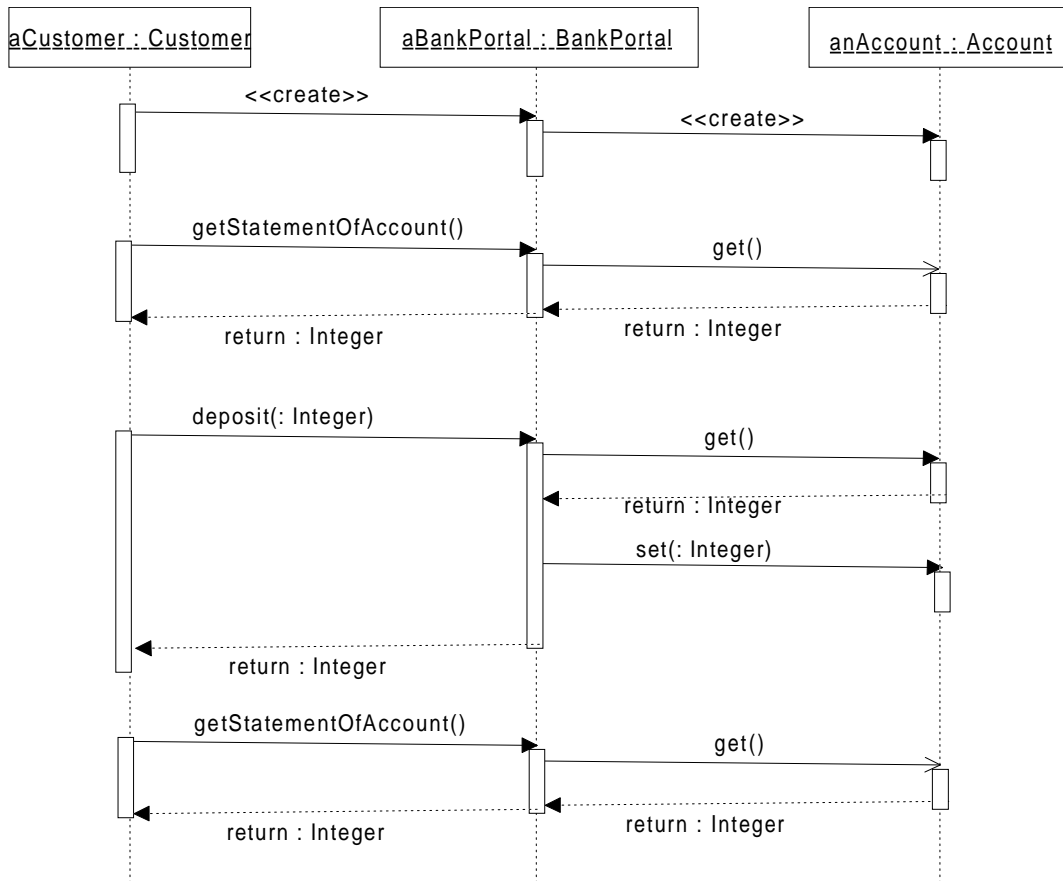
**Lösung:**



Booch, Rumbaugh and Jacobson. The Unified Modeling Language User Guide. Addison-Wesley, 1999

b) Erstellen Sie ein Reihenfolgediagramm (*Sequence Diagram*) ausgehend von `Customer.main` in UML-Notation.

Lösung:



Der zu wartende Quellcode:

### Klasse Customer:

```
import java.io.*;

public class Customer extends java.lang.Object {

    private static BankPortal bankportal;
    private int privateAmount;

    public static void main(java.lang.String[] args) {
        int i = 0;
        bankportal = new BankPortal ();
        i = bankportal.getStateOfAccount();
        java.lang.System.out.println("current Statement of Account: " + i);
        i = 200;
        java.lang.System.out.println("deposit: " + i);
        bankportal.deposit(i);
        java.lang.System.out.println("current Statement of Account: "
            + bankportal.getStateOfAccount());
    }
}
```

### Klasse BankingSystem:

```
public class BankingSystem {

    public BankingSystem(){
        account = new Account();
    }
    public void set(int newData){
        account.setCurrentAmount(newData);
    }
    public int get(){
        return account.getCurrentAmount();
    }

    private Account account;    // This relation is an aggregation
}
```

### Klasse BankPortal:

```
public class BankPortal extends BankingSystem {

    public int deposit(int deposit){
        int currentAmount = 0;

        currentAmount = get();

        currentAmount = currentAmount + deposit;

        set(currentAmount);

        return currentAmount;
    }

    public int withdraw(int withdrawl){
        int currentAmount = 0;

        currentAmount = get();

        currentAmount = currentAmount - withdrawl;

        set(currentAmount);

        return currentAmount;
    }

    public int getStatementOfAccount() {

        return get();
    }

}
```

### Interface AccountAccess:

```
public interface AccountAccess {
    public void setCurrentAmount(int newAmount);
    public int getCurrentAmount();
}
```

### Klasse Account:

```
public class Account implements AccountAccess {

    private int currentAmount;

    public Account(){
        currentAmount = 0;
    }

    public int getCurrentAmount(){
        return currentAmount;
    }

    public void setCurrentAmount(int newAmount){
        currentAmount = newAmount;
    }

}
```