

---

Name:

Matr.Nr.:

Studiengang:

---

**Vorbemerkungen:**

- Die Bearbeitungszeit beträgt 60 Minuten. Sie können maximal 50 Punkte erreichen. Hilfsmittel sind nicht zugelassen!
  - Die Klausur umfasst insgesamt 5 Seiten. Überprüfen Sie bitte *sofort* die Vollständigkeit!
  - Geben Sie die Lösung auf dem jeweiligen Aufgabenblatt an. Bei Platzmangel können Sie auch die dazugehörige Rückseite verwenden.
  - Lesen Sie die Aufgabenstellung jeweils gut und in Ruhe durch. An sich richtige Lösungen, die nicht der Aufgabenstellung entsprechen, werden nicht gewertet.
  - Viel Erfolg!
- 

**Punktzahl:**

Aufgabe 1:	/16=	/5+	/5+	/6		
Aufgabe 2:	/34=	/8+	/8+	/6 +	/6 +	/6

---

Summe: /50

Note:

## Aufgabe 1: Logikprogrammierung und Funktionale Programmierung

a) Angenommen, ein Prolog-Programm enthält bereits eine Menge von Fakten der Form `elternteil(hans, karin)` (Hans ist Vater oder Mutter von Karin). Schreiben Sie Prolog Regeln auf, die definieren, welche Vorfahren man hat (Vorfahren sind die Eltern und deren Vorfahren)..

```
vorfahr(Person, Eltern):- elternteil(Eltern, Person).
vorfahr(Person, Vorfahr):-
    elternteil(Eltern, Person),
    vorfahr(Eltern, Vorfahr).
```

b) Im Unterschied zur mathematischen Logik hat Prolog ein festes Reihenfolge-Schema bei der Beantwortung einer Frage. Wie sieht das aus? (Es sind 2 Reihenfolgeregeln!)

- 1. Prolog resolviert die Teilziele einer Anfrage von links nach rechts (goal order)*
- 2. Prolog wählt brauchbare Regeln für die Resolution gemäß der textuellen Reihenfolge (von oben nach unten) (rule order).*

c) Wandeln Sie das folgende Scheme-Programm in endrekursive Form um. (Hinweis: Definieren Sie neben der rekursiven Funktion eine weitere Funktion für den Aufruf von „außen“).

```
(define summe
  (lambda (liste)
    (if (null? liste)
        0
        (+ (car liste) (summe (cdr liste))) )))
```

```
(define summe
  (lambda (liste)
    (summe* 0 liste) )))
```

```
(define summe*
  (lambda (sum liste)
    (if (null? liste)
        sum
        (summe* (+ sum (car liste)) (cdr liste)) )))
```

## Aufgabe 2: Generische Typen und Nebenläufigkeit

Wir gehen von der folgenden Klasse aus.

```
class IntStack {
    private int top = 0;
    private int[] data = new int[10];

    public void push(int x) {
        data[top++] = x;
    }
    public int pop() {
        return data[--top];
    }
    public int size() {
        return top;
    }
}
```

a) Schreiben Sie die Klasse so um, dass Sie mit einem Typparameter versehen ist und damit für beliebige Objekttypen verwendbar ist. Geben Sie auch an, welches Problem dabei durch die Typlöschung (was ist das?) entsteht und finden Sie dafür eine brauchbare Lösung.

```
class IntStack<T> {
    private int top = 0;
    private T[] data = (T[]) new Object[10];

    public void push(T x) {
        data[top++] = x;
    }

    public T pop() {
        return data[--top];
    }

    public int size() {
        return top;
    }
}
```

b) Schreiben Sie die Klasse (in `int` oder in generischer Form) so um, dass sie threadsicher ist und dass bei leerer Queue gewartet wird.

```
class Stack<T> {
    private int top = 0;
    private T[] data = (T[]) new Object[10];

    public synchronized void push(T x) {
        data[top++];
        notifyAll();
    }

    public synchronized T pop() {
        while (size() == 0) wait();
        return data[--top];
    }

    public synchronized int size() {
        return top;
    }
}
```

c) Warum können bei rein funktionaler Programmierung (z.B. in Java) keine Wettlaufbedingungen auftreten?

**Da die funktionale Programmierung zustandslos ist (es gibt keine nichtlokalen Variablen, Variablen sind zudem unveränderlich).**

d) Unter welchen Bedingungen können bei der Verwendung von Sperrmechanismen für Objekte Deadlocks auftreten?

**Voraussetzung: mindestens 2 Threads, die beide mindestens 2 Objektsperren benötigen und durch den Zugriff auf die Objektsperren entsteht eine zyklische Abhängigkeit.**

e) Erläutern Sie drei Fehler bzgl. Nebenläufigkeit und Threadsicherheit im folgenden Programmbeispiel. Geben Sie jeweils eine kurze Erklärung.

```
01 class Unsinn {
02     Set<Konto> set = new HashSet<Konto>();
03
04     synchronized void methode1(Konto k) throws Exception {
05         while (k.istLeer()) k.wait();
06         set.add(k);
07     }
08     boolean methode2(Konto k) {
09         return set.contains(k);
10     }
11     synchronized void methode3(Konto k) throws Exception{
12         while(true) {
13             if (methode2(k)) return;
14             Thread.sleep(1000);
15         }
16     }
17     void methode4(Konto k) {
18         synchronized(k) {
19             set.remove(k);
20         }
21     }
22 }
```

**Zeile 5:** `k.wait()` besitzt nicht die Objektsperre von `k` (nur die von `this`)

**Zeile 8:** der Zugriff auf `contains` ist unsicher, da nicht `synchronized`.

**Zeile 14:** `sleep` in `Synchronized Block` sperrt Methoden, da die Sperre nicht freigegeben wird.

**Zeile 19:** Es ist unsinnig an verschiedenen Stellen verschiedene Sperrungen zu nutzen um das Objekt in `set` zu schützen (mal `this`, hier `k`).

Es ist auch verkehrt, dass dem `k.wait()` kein `notifyAll()` entspricht.