

# Paradigmen der Programmierung

## Beispiele zu Klausurfragen

### Logikprogrammierung

Formulieren Sie die folgenden Tür-Sachverhalte als Fakten. Wir haben 5 Räume (a,b,c,d,e,f). Es gibt in ihnen Türen a-b, a-c, b-d, e-f, a-f. Schreiben Sie die angegebenen (gerichteten) Türverbindungen auf. Und schreiben Sie eine Regel, dass man eine Tür in beiden Richtungen passieren kann.

#### Lösung:

Schreiben Sie eine Regel auf, die feststellt, ob man von einem Startpunkt aus einen bestimmten Raum erreichen kann. Sie brauchen nicht darauf zu achten, ob Prolog die Lösung in endlicher Zeit findet.

#### Lösung:

#### Zusatzfragen:

Warum findet Prolog die Lösung nicht?

#### Lösung:

Wie kann man erreichen, dass das Ziel von Prolog gefunden wird?

#### Lösung:

### Funktionale Programmierung

Wie unterscheidet sich der Funktionsbegriff der funktionalen Programmierung von dem Funktionsbegriff in der prozeduralen Sprache C?

#### Lösung:

Warum kennt verwendet man in der funktionalen Programmierung Rekursion anstelle der Iteration?

#### Lösung:

Definieren Sie in (Scala oder Scheme) eine endrekursive Funktion, die die Reihenfolge der Elemente einer Liste umdreht?

#### Lösung:

Angenommen, wir haben eine Funktion

```
def filter[T](liste: List[T], f: (T)=>Boolean): List[T] // Scala
(filter liste boolescheFunktion) // Scheme
```

Die zu einer Liste die Teilliste bestimmt, für die die boolesche Funktion (f) erfüllt ist. Wie sieht der Ausdruck aus, der aus einer Liste zahlenListe die positiven Zahlen herausfiltert?

**Lösung:**

Schreiben Sie (in Scala oder Scheme) eine Funktion höherer Ordnung, die die Teilliste derjenigen Elemente zurückgibt, die eine bestimmte Bedingung erfüllen.

**Lösung**

## Typparameter

Überlegen Sie genau! Eine Prioritätswarteschlange (priority queue) ist für einen Typparameter T definiert. Damit die richtige Reihenfolge festgelegt werden kann, wird dem Konstruktor ein brauchbares Comparator-Objekte übergeben. Wie lautet in Java die korrekte Typdefinition für den Comparator. Entscheiden Sie sich zwischen Invarianz, Kovarianz und Kontravarianz.

```
class PriorityQueue<T> {  
    PriorityQueue(c: Comparator<...>) {           // welcher Parameter steht hier?
```

**Lösung**

## Nebenläufigkeit

Schreiben Sie ein kleines Codefragment, in dem neben dem aktuell laufenden Thread eine Anweisung gestartet wird, die in 10 Sekunden (`Thread.sleep(10000)`) das Programm abbricht.

**Lösung**

Schreiben Sie (Implementierung mit einer einfach verketteten Liste) eine Klasse `IntStack`, die threadsicher ist und deren Methode `pop()` bei leerem Stack blockiert, bis ein Element verfügbar ist.

**Vorgabe:**

```
public class IntStack {  
    private static class Node {  
        int value;  
        Node next;  
        Node(int value, Node next) {  
            this.value = value;  
            this.next = next;  
        }  
    }  
    private Node first = null;  
  
    public boolean isEmpty() {  
        return first == null;  
    }  
}
```

```

public void push(int value) {
    first = new Node(value, first);
}

public int pop() {
    int result = first.value;
    first = first.next;
    return result;
}
}

```

### Lösung:

Die Thread-Methode `yield()` kann bewirken, dass der laufenden Thread vom Zustand „*running*“ in den Zustand „*ready*“ versetzt wird. Warum ist diese Methode nicht geeignet, eine gewünschte Ausführungsreihenfolge von Threads zu garantieren?

### Lösung:

Die Klasse `Semaphore` realisiert eine zählende Semaphore. Sie hat die Operationen `require` und `release`. Der Aufruf von `release` erhöht den Zähler um 1. Der Aufruf von `require` blockiert, wenn der Zähler kleiner oder gleich 0 ist. Anschließend wird der Zähler um 1 erniedrigt. Ergänzen Sie die folgende Klasse `ParkingDeck` mit einer Semaphore, so dass sie threadsicher ist und bei gefülltem Parkhaus anstehende Fahrzeuge warten lässt.

```

public class ParkingDeck {
    private Semaphore sem;

    public ParkingDeck(int capacity) {
        sem = ...
    }
    public void enter() {
        ...
        System.out.println("a car enters");
    }

    public void leave() {
        ...
        System.out.println("a car leaves");
    }
}

```

### Lösung: