

### Aufgabe 1 (10 Punkte)

Die zweite Version des `LinearBuffer` aus 3.1.2 ist so konzipiert, dass Sender und Empfänger überlappend tätig werden können (wenngleich nicht immer fehlerfrei).

a) (2 P.) Die folgende Variante von `send` birgt eine weitere Fehlerquelle. Zeigen Sie dies durch Angabe eines Beispiels mit entsprechenden Zeitschnitten!

```
public void send(M m) throws Overflow {
    if(count()==size) throw new Overflow();
    synchronized(this) { // sender exclusion
        cell[rear] = m;
        ...
    }
}
```

b) (3 P.) Die folgende Variante von `recv` ist auch fehlerhaft; warum?

```
public M recv() throws Underflow {
    synchronized(cell) { // receiver exclusion
        if(count()==0) throw new Underflow();
        M m = cell[front];
        front = (front+1)%(size+1); }
    return m; }
}
```

Kann der Fehler dadurch behoben werden, dass `m` vor dem `synchronized` vereinbart wird?

c) (5 P.) Erweitern Sie die Klasse um eine zusätzliche Methode `urgent`, die eine dringliche Nachricht *vorn im Puffer* ablegt! (Achtung: die richtige Ausschluss-Synchronisation ist hier etwas kniffliger.)

## Aufgabe 2 (3 Punkte)

Wir betrachten die folgende Klasse:

```
class Funny {
    private static int n;
    private int i;
    void incr() { n += i++; }
}
```

Wenn zwei Funny-Objekte von mehreren Fäden nebenläufig benutzt werden, droht offenbar Unheil. Wie können Sie dies durch den Einsatz des Schlüsselworts `synchronized` vermeiden?

## Aufgabe 3 (4 Punkte)

`java.util.Collections` stellt Klassenmethoden für das *Nachrüsten von Ausschluss* zur Verfügung. Vergleichen Sie die dafür verwendete Technik mit den in der Vorlesung vorgestellten Techniken (Vererbung bzw. Delegation)! Sehen Sie Vorteile/Nachteile? Was passiert, wenn die Nachrüstung auf eine Klasse bzw. ein Objekt angewendet wird, die/das *schon synchronisiert war*?

## Aufgabe 4 (4 Punkte)

In der Dokumentation zu `java.util.Collections.synchronizedSet` findet sich die folgende Passage:

*„It is imperative that the user manually synchronize on the returned set when iterating over it:*

```
Set s = Collections.synchronizedSet(new HashSet());
...
synchronized(s) {
    Iterator i = s.iterator();
    while (i.hasNext())
        foo(i.next());
}
```

*Failure to follow this advice may result in non-deterministic behavior.“*

Was steckt wohl dahinter?