

### Aufgabe 1 (8 Punkte)

Die folgenden Code-Fragmente sind vorgegeben:

```
class X {
    private String text;
    private Complex co; // well-known complex numbers
X(String s, Complex c) { text=s; co=c; }
synchronized void update(String s, float re, float im){
    if(s!=null) text = s;
    co.re = re;
    co.im = im;
}
synchronized String get() {
    return text+" "+co.re+" "+co.im;
}
}

String text = "complex";
Complex com = new Complex(0f,1f);
X one = new X(text,com);
X two = new X(text,com);
```

Wir nehmen an, dass nach dieser Initialisierung mehrere Fäden aktiv werden, die teils auf `one` und teils auf `two` operieren. Da es sich hierbei um Monitore handelt, sollten keine unerwünschten Effekte auftreten.

a) (4 P.) Zeigen Sie mit einem Beispiel, dass diese Hoffnung trügt!

b) (4 P.) Erläutern Sie, woran das liegt, und leiten Sie daraus eine allgemeine Empfehlung für den Entwurf von Monitoren ab!

### Aufgabe 2 (6 Punkte)

Entwickeln Sie eine Klasse `ReentrantLock` mit Methoden `lock`, `unlock`, die sich wie die gleichnamigen Methoden der gleichnamigen Java-Bibliotheksklasse verhalten (vgl. 3.1.5). Benutzen Sie dabei die Warteangweisung **AWAIT**, programmieren Sie aber ansonsten in korrektem Java!

### Aufgabe 3 (16 Punkte)

In der Vorlesung wurde ein **MONITOR TwoPrinters** vorgestellt (3.2.2.2), der die Bedingungssynchronisation über Ereignisse realisiert. Gesucht sind zwei alternative Implementierungen, die mit **WHEN** bzw. **AWAIT** arbeiten und dabei dem Verhalten des Originals möglichst nahe kommen:

- a) (4 P.) Wie lautet die Lösung mit **WHEN**? Welche Abweichungen vom Original gibt es?
- b) (4 P.) Wie lautet die Lösung mit **AWAIT**? Welche Abweichungen vom Original gibt es?
- c) (4 P.) Vergleichen Sie die Fairness-Eigenschaften der drei Versionen!
- d) (4 P.) Vergleichen Sie die Effizienz der drei Versionen!

### Aufgabe 4

(Programmieraufgabe mit Java – folgt nächste Woche)