

Algorithmen und Programmierung IV

SS 2006

26. Juni 2006

Musterlösung zum Aufgabenblatt 6

Aufgabe 1

- a) Wenn je ein Faden mit dem Objekt hinter `one` bzw. `two` operiert, erfolgt kein gegenseitiger Ausschluss, da die beiden Objekte jeweils sich selbst zur Kennzeichnung der kritischen Abschnitte verwenden. Leider verwenden beide Objekte ein anderes Objekt gemeinsam: die komplexe Zahl.

```
// Initialisierung der komplexen Zahl mit (0f, 1f)
-----
// Faden 1 führt aus:           // Faden 2 führt aus:
// one.update("new complex", 2f, 3f); // System.out.print(two.get());
-----
if (s != null) text = "new complex";
co.re = 2f;
-----
                                System.out.print(text + " " + co.re
                                                + " " + co.im);
                                // Ausgabe: "new complex 2.0 1.0"
-----
co.re = 3f;
```

Man würde von der Methode `get` die Zeichenketten `"complex 0.0 1.0"` oder `"new complex 2.0 3.0"` erwarten, stattdessen erhalten wir aber die Ausgabe `"new complex 2.0 1.0"`.

- b) Das Problem besteht in dem Code-Fragment darin, dass die Exemplare des Monitors als Attribut ein Objekt verwenden, das sie nicht selbst erstellt haben, sondern zu dem sie jediglich einen Verweis erhalten haben. Es wird somit nicht gewährleistet, dass auf ein solches Objekte nur von einem Exemplar des Monitors aus zugegriffen wird.

Es ist daher empfehlenswert, dass ein Monitor als Attribute nur Werttypen und *selbsterstellte* Exemplare von Verweistypen verwendet. Wird einem Monitor ein Verweistyp übergeben, dann sollte ein Monitor eine Kopie davon

anfertigen, z.B. mittels `clone`, falls die Schnittstelle `Cloneable` implementiert wird (Achtung: Wenn die Implementierung von `clone` nur eine flache Kopie erstellt, ist dies immer noch nicht sicher).

Anmerkung: In bestimmten Fällen wäre es wünschenswert, wenn man auch zusammengesetzte Datentypen auch als Werttypen verwenden könnte. Diese Möglichkeit besteht bei Java nicht, aber z.B. in C#. Dort gibt es neben Klassen (Schlüsselwort `class`) auch Strukturen (Schlüsselwort `struct`). Sie haben ähnliche Eigenschaften wie Klassen, aber Variablen von einem Strukturtyp repräsentieren keinen Verweis, sondern den Wert. Bei einer Parameterübergabe wird somit auch der Wert und nicht nur der Verweis kopiert. Sie bieten sich besonders für kleinere zusammengesetzte Datentypen an, bei denen eine Kopie der Attribute bei jeder Zuweisung oder Parameterübergabe keinen großen Aufwand darstellt, z.B. komplexe Zahlen oder Punkte (Attribute `x` und `y`).

Aufgabe 2

Implementierung von `ReentrantLock` mit `AWAIT`:

```
MONITOR ReentrantLock {  
  
    private volatile Thread owner;  
    private volatile int locks;  
  
    public void lock() {  
        if (owner != Thread.currentThread()) {  
            AWAIT owner == null;  
            owner = Thread.currentThread();  
        }  
        locks++;  
    }  
  
    public void unlock() {  
        if (owner != Thread.currentThread())  
            throw new IllegalMonitorStateException();  
        locks--;  
        if (locks == 0) owner = null;  
    }  
}
```

Anmerkung: Bei beiden Operationen `lock` und `unlock` wird überprüft, ob der aktuelle Faden im Besitz der Sperre ist. Ist dies der Fall, so muss kein kritischer Abschnitt betreten werden, da der aktuelle Faden nicht gleichzeitig einen weiteren Aufruf einer Methode der Sperre durchführen kann.

Aufgabe 3

a) Implementierung mit der Wache (WHEN):

```
MONITOR TwoPrinters {

    private volatile boolean normalbusy;
    private volatile boolean specialbusy;

    public boolean request(boolean special)
        WHEN (!normalbusy && !special) || !specialbusy {
        if (special) {
            specialbusy = true;
            return true; // special
        } else /* normal */ {
            if (!normalbusy) {
                normalbusy = true;
                return false; // normal
            } else {
                specialbusy = true;
                return true; // special
            }
        }
    }

    public release(boolean special) {
        if (special) {
            specialbusy = false;
        } else /* normal */ {
            normalbusy = false;
        }
    }
}
```

Im Gegensatz zu Ereignissen führt die Wache keine Wartenschlangen. Wird der normale Drucker freigegeben, haben alle Fäden, die auf einen beliebigen Drucker warten, die gleiche Chance den Zuschlag für diesen Drucker zu erhalten. Wird der Spezialdrucker freigegeben, haben alle wartenden Fäden die gleiche Chance, den Zuschlag für den Spezialdrucker zu erhalten; insbesondere werden nicht die Fäden bevorzugt, die explizit den Spezialdrucker angefordert haben.

b) Implementierung mit der Warteanweisung (AWAIT):

```
MONITOR TwoPrinters {

    private volatile boolean normalbusy;
    private volatile boolean specialbusy;

    public boolean request(boolean special) {
        if (special) {
            AWAIT !specialbusy;
            specialbusy = true;
            return true; // special
        } else /* normal */ {
            AWAIT !normalbusy || !specialbusy;
            if (!normalbusy) {
                normalbusy = true;
                return false; // normal
            } else {
                specialbusy = true;
                return true; // special
            }
        }
    }

    public release(boolean special) {
        if (special) {
            specialbusy = false;
        } else /* normal */ {
            normalbusy = false;
        }
    }
}
```

Die Implementierung mit der Warteanweisung verwendet etwas weniger Ausschluss als jene mit Ereignissen oder Wachen, da die Warteanweisung nicht die erste Operation ist und daher Auswertungen vorgenommen werden, die sich nur auf den Übergabeparameter beziehen. Ansonsten entspricht diese Implementierung der Implementierung mit Wachen.

c) Die Implementierung mit Ereignissen ist weitestgehend fair, da die Fäden nach dem *first come first serve*-Prinzip bedient werden. Eine Ausnahme besteht darin, dass Fäden, die den Spezialdrucker anfordern, bevorzugt werden und den Zuschlag somit früher erhalten können als Fäden, die einen beliebigen Drucker angefordert haben. Diese Strategie kann man als Kompensation dafür ansehen, dass Fäden, die den Spezialdrucker anfordern, dadurch benachteiligt sind, dass sie auf *eine* spezielle Ressource angewiesen sind.

Bei der Implementierung mit der Wache und der Warteanweisung haben alle wartenden Fäden die gleiche Chance, den Zuschlag für einen *beliebigen* Drucker zu halten. Fordert ein Faden einen bestimmten Drucker, nämlich den Spezialdrucker, an, hat er daher eine geringere Chance den Zuschlag zu erhalten.

- d) Die Implementierung mit Ereignissen ist am effizientesten. Die Abfrage, ob ein Drucker zugeteilt werden kann, findet immer genau einmal statt. Ist diese Bedingung nicht erfüllt, wartet der Faden, bis er gezielt aufgeweckt wird.

Bei der Implementierung mit der Wache bzw. der Warteanweisung müssen alle blockierten Fäden jedesmal, wenn ein Faden den Monitor verlässt, die Bedingung erneut überprüfen. Wurde z.B. der normale Drucker freigegeben, so würden überflüssigerweise auch Fäden, die den Spezialdrucker anfordern, die Bedingung zum Betreten des kritischen Abschnitt auswerten.

Aufgabe 4

(keine Musterlösung)