6. Übung

Abgabe bis 04.12.2017, 10:00 Uhr

Einzelaufgabe 6.1: Einfache Laufzeitaufwandsanalyse — 8 EP

Bestimmen Sie für die folgenden Quellcode-Fragmente die *kleinste obere Schranke* für den Laufzeitaufwand im $\mathcal{O}$-Kalkül in Abhängigkeit von den Parametern. Vereinfachen Sie die Funktionsklasse so weit wie möglich und verwenden Sie bevorzugt die aus der Vorlesung bekannten „typischen Laufzeitklassen“. Begründen Sie Ihre Lösung *kurz* und geben Sie sie als `Laufzeitaufwand.pdf` im EST ab.

a)

```
public int methA(int n) {
    int r = 0;
    while (n > 0) {
        r++;
        n = n / 3;
    }
    return r;
}
```

b)

```
public int methB(int n) {
    int ret = 0;
    for (int k = 15; k <= n / 2; k++) {
        ret += methA(k); // Siehe a)
    }
    return ret;
}
```

c)

```
public int methC(int n) {
    int sum = 0;
    for (int k = 15; k * (k - 5) <= 4 * n; k++) {
        sum = (sum + k) % 43;
    }
    return sum;
}
```

d)

```
public int methD(int n) {
    int fak = 1;
    for (int k = 0; k <= 4 * n; k++) {
        while (n != 0) {
            fak *= n--;
        }
    }
    return fak;
}
```

e)

```
public int methE(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= i; j++) {
            for (int k = 1; k <= j; k++) {
                sum += i * k * j;
            }
        }
    }
    return sum;
}
```

Einzelaufgabe 6.2: Objekte und Referenzen — 10 EP

Betrachten Sie folgenden Java-Code:

```
1  class Vec3 {
2      float x, y, z;
3
4      void add(Vec3 b) {
5          x += b.x;
6          y += b.y;
7          z += b.z;
8          // *** 2 ***
9      }
10
11     float sum() {
12         float sum = x + y + z;
13         // *** 3 ***
14         return sum;
15     }
16 }
```

```
18 public class Scope {
19     static int m(int a, int b) {
20         int sum = a + b;
21         // *** 1 ***
22         return sum;
23     }
24
25     public static void main(String args[]) {
26         args = null;
27         int sum = 0;
28         for (int i = 0; i < 4; i++)
29             sum += i;
30         // *** 4 ***
31         if (sum > 3) {
32             int b = 3 * 7;
33             sum = m(sum, b);
34             // *** 5 ***
35         }
36         Vec3 p = new Vec3();
37         p.x = p.y = -(p.z = 1);
38         float x = p.x;
39         // *** 6 ***
40         Vec3 q = p;
41         q.y = 2;
42         // *** 7 ***
43         q = new Vec3();
44         q.x = q.y = q.z = 2;
45         q.add(p);
46         // *** 8 ***
47     }
48 }
```

Welche Variablen sind bei der Ausführung des Programms (beginnend bei `main`) jeweils an den mit "`*** x ***`" markierten Stellen gültig und welche Datentypen sowie Werte haben diese Variablen? Geben Sie dabei auch an, welche Objekte dort jeweils existieren und auf welche Objekte etwaige Referenzvariablen verweisen. Nummerieren Sie die Objekte in der Reihenfolge ihrer Erzeugung mit $O_1, O_2, \ldots$ Geben Sie Ihre Antwort tabellarisch nach folgendem Beispiel an:

```
1  class Foo { // das ist nur ein Beispiel!
2      float b, a, r;
3
4      Foo baz(Foo other) {
5          Foo uTurn = new Foo();
6          // *** 1 ***
7          // ...
8          return uTurn;
9      }
10
11     // ...
12     public static void main(String[] args) {
13         Foo foo = new Foo();
14         Foo baz = foo.baz(foo);
15     }
16 }
```

Position	Variable/Objekt	Typ	Art	Wert
`*** 1 ***`	O_0	String[]	Objekt (args)	unbekannt
	O_1	Foo	Objekt	{0,0,0}
	other	Referenz auf Foo	Parameter	$\rightarrow O_1$
	b	float	Objektvariable (von O_1)	0
	a	float	Objektvariable (von O_1)	0
	r	float	Objektvariable (von O_1)	0
	O_2	Foo	Objekt	{0,0,0}
	uTurn	Referenz auf Foo	lokale Variable	$\rightarrow O_2$

Geben Sie Ihre Lösung als `Ref.pdf` über EST ab.

Einzelaufgabe 6.3: OOP – Profiling — 11 EP

Beim Profiling wird u.a. gezählt, wie oft bestimmte Methoden während einer Programmausführung aufgerufen werden. Beachten Sie in dieser Aufgabe *unbedingt* folgende Anforderungen:

- Setzen Sie bei a) *genau* die Sichtbarkeiten aus dem UML-Diagramm (Abb. 1) in Code um.
- Die ab Teilaufgabe b) geforderten Methoden *müssen öffentlich* sein.
- Falls Sie weitere Attribute oder Methoden benötigen, *müssen* diese `private` sein.
- Die ab Teilaufgabe b) umzusetzende Zählung bezieht sich *ausschließlich* auf die Konstruktoren und Methoden im UML-Klassendiagramm (Abbildung 1), d.h. u.a. z.B. auch, dass der Aufruf von `getCallsTo` *nicht* mitgezählt werden soll.

a) Implementieren Sie die öffentliche Klasse `FooBarBazQux` *exakt* gemäß Abbildung 1. Beachten Sie die Unterscheidung zwischen statischen und nicht-statischen Attributen bzw. Methoden. Initialisieren Sie die Attribute wie angegeben und beachten Sie den Rumpf/Rückgabewert jeder Methode, der jeweils darunter als *„{documentation = '...'}“* steht:

b) Ergänzen Sie `FooBarBazQux` um eine Klassenmethode `long getInstantiations()`, die jeweils zurückgibt, wie viele Instanzen der Klasse bislang erzeugt wurden.

c) Ergänzen Sie nun eine Instanzmethode `long getCallsTo(String methodName)`, die jeweils zurückgibt, wie oft die Methode `methodName` (nur die im UML-Diagramm!) bislang aufgerufen wurde. Falls `methodName` eine Instanzmethode bezeichnet, dann soll die Anzahl der Aufrufe *pro Instanz* zurückgegeben werden. Falls es keine Methode mit dem angegebenen Namen gibt, dann muss `getCallsTo` den Wert -1 zurückgeben.

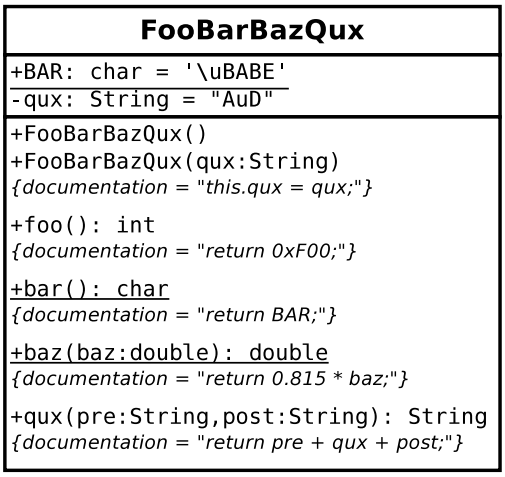

Abbildung 1: UML-Klassendiagramm für Klasse `FooBarBazQux`

d) Schließlich soll die Klassenmethode `long getTotalCallsTo(String methodName)` jeweils zurückgeben, wie oft die Methode `methodName` (nur die im UML-Diagramm!) bislang insgesamt aufgerufen wurde. Falls `methodName` eine Instanzmethode bezeichnet, dann soll die Anzahl *aller* Aufrufe (also in *allen* Instanzen) zurückgegeben werden. Falls es keine Methode `methodName` gibt, dann muss `getTotalCallsTo` den Wert -1 zurückgeben.

Geben Sie Ihre Lösung als `FooBarBazQux.java` über EST ab.

Gruppenaufgabe 6.4: $\mathcal{O}$-Kalkül für Funktionen **10 GP**

Seien $a > 0$, $b > 1$, $c \geq 2$ positive Konstanten sowie $x > 0$, $y > 0$ „freie“ Parameter (Eingabe-/Problemgrößen) der folgenden Funktionen ($a, b, c, x, y \in \mathbb{R}$). Geben Sie zu jeder Funktion die *kleinste* obere Schranke im $\mathcal{O}$-Kalkül so an, dass sich das Ergebnis nicht mehr weiter vereinfachen lässt. Verwenden Sie dazu nach Möglichkeit eine der Funktionsklassen aus der Vorlesung.

a) $f_{a,c}(x) = c \cdot x^2 + a \cdot x^5$

b) $f_{b,c}(x) = x^b \cdot x^c$

c) $f_{a,b,c}(x) = log_a(x) + log_b(x) + log_c(x)$

d) $f_{a,b,c}(x) = (c^a + x) \cdot (log_b(x) + b)$

e) $f_{b,c}(x) = \sqrt{c^b \cdot x}$

f) $f_{a,b,c}(x) = a \cdot b^x + \sqrt{x^c}$

g) $f_c(x) = \frac{x!}{c \cdot x} + (c^x)^2$

h) $f_{a,c}(x, y) = c \cdot x^2 + a \cdot y^5$

i) $f_b(x, y) = log_b(x^{b \cdot y})$

j) $f_{a,b,c}(x, y) = log_a(x^b \cdot c^y)$

Geben Sie Ihre Lösung als `OKFun.pdf` über EST ab.

Gruppenaufgabe 6.5: $\mathcal{O}$-Kalkül in Java — 9 GP

Gegeben seien die folgenden Methoden, in denen bestimmte Stellen mit Kommentaren der Form `/* **X** */` markiert sind. Geben Sie zunächst für jede solche Stelle einer Methode eine *geschlossene* Formel an, die *exakt* berechnet, wie oft die jeweiligen Stellen, in Abhängigkeit von den Eingangsgrößen n und m, durchlaufen werden. Geben Sie anschließend noch den asymptotischen Aufwand in $\mathcal{O}$-Notation für die jeweiligen Methoden an. Geben Sie Ihre Lösung als `OKalkJava.pdf` ab.

a)

```
static int[] matrixVectorMul(int[][] mat, int[] vec) {
    int hoehe = mat.length; /* -> m */
    if (hoehe <= 0) {
        return new int[0];
    }
    int breite = mat[0].length; /* -> n */
    int[] erg = new int[hoehe];
    for (int zeile = 0; zeile < hoehe; ++zeile) {
        /* **1** */
        for (int spalte = 0; spalte < breite /* **2** */; ++spalte) {
            erg[zeile] = mat[zeile][spalte] * vec[spalte];
            /* **3** */
        }
    }
    return erg;
}
```

b)

```
static int bitsZaehlen(int n) {
    int count = 0;
    while (n > 0) {
        if (n % 2 == 1) {
            ++count;
        }
        n /= 2;
        /* **1** */
    }
    return count;
}
```

c) Skizzieren Sie für folgende Methode in **wenigen** Sätzen, wie Sie die Aufwandsklasse abgeschätzt haben. Eine Beweisführung ist **nicht** verlangt.

```
static int f(int n) {
    if (n <= 0) {
        /* **1** */
        return 0;
    } else {
        for (int i = 0; i < n; ++i) {
            /* Operationen mit konstantem Aufwand */
        }
        return f(n / 2);
    }
}
```

Gruppenaufgabe 6.6: Lineare Algebra — 12 GP

In dieser Aufgabe sollen Sie schrittweise eine Klasse `Matrix3x3` für (3×3)-Matrizen objektorientiert implementieren. Vermeiden Sie dabei Code-Duplikation und verwenden Sie stattdessen bereits implementierte Funktionalität aus anderen Methoden wieder, indem Sie diese geeignet aufrufen. Zusätzlich zum öffentlichen Test gibt es diesmal auch eine Animation, die einen $3D$-Würfel dreht, sobald Ihre Matrix-Klasse korrekt implementiert ist.

a) Erstellen Sie die Klasse `Matrix3x3` mit einem einzigen Attribut: Einem Array vom Typ `Vector3D` mit genau drei Einträgen, die die Spalten einer Matrix M wie folgt repräsentieren:

$$M = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{bmatrix} \rightsquigarrow \texttt{Matrix3x3} = \left[\begin{pmatrix} m_{1,1} \\ m_{2,1} \\ m_{3,1} \end{pmatrix} \begin{pmatrix} m_{1,2} \\ m_{2,2} \\ m_{3,2} \end{pmatrix} \begin{pmatrix} m_{1,3} \\ m_{2,3} \\ m_{3,3} \end{pmatrix} \right]$$

b) Implementieren Sie nun die Methoden `get` und `set`. Die `get`-Methode erwartet die beiden Parameter `int row` und `int col` (in dieser Reihenfolge, 0-basierte Indizes!) und gibt den Wert des jeweiligen Koeffizienten $m_{row+1,col+1}$ der Matrix zurück.
Die `set`-Methode nimmt als dritten Parameter einen `double`-Wert entgegen und setzt den durch `row` und `col` beschriebenen Koeffizienten der Matrix auf den übergebenen Wert. Die `set`-Methode hat keinen Rückgabewert.

c) Ergänzen Sie eine parameterlose Instanzmethode ohne Rückgabewert namens `toIdentity`. Diese setzt die aktuelle Matrix auf die sogenannte Identitätsmatrix Id:

$$Id = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

d) Implementieren Sie die Methode `deepCopyFrom`. Diese nimmt ein `Matrix3x3`-Objekt Q entgegen und setzt die Koeffizienten (*nicht* die Spalten!) der aktuellen Matrix auf die Koeffizienten der übergebenen Matrix Q.

e) Implementieren Sie drei Methoden `setRotX`, `setRotY` und `setRotZ`. Diese nehmen einen Winkel θ im Bogenmaß vom Typ `double` entgegen und setzen die Matrix auf eine Rotationsmatrix, die einen $3D$-Punkt um die entsprechende Achse gegen den Uhrzeigersinn rotiert. Die Koeffizienten der Rotationsmatrizen errechnen sich wie folgt:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

f) Implementieren Sie die Methode `immutMul`. Diese nimmt eine Matrix B entgegen und multipliziert die aktuelle Matrix A mit der übergebenen Matrix B. Es wird eine neue Matrix C zurückgegeben, die das Ergebnis der Matrixmultiplikation enthält. Die aktuelle Matrix A selbst sowie die übergebene Matrix B bleiben *unverändert*! **Achtung:** Die Operation ist **nicht** kommutativ! Es muss gelten: $C = A \circ B =$ `a.immutMul(b)`.

g) Implementieren Sie die Methode `mutMul`. Diese führt wie `immutMul` eine Matrixmultiplikation aus, speichert aber das Ergebnis in der aktuellen Matrix und hat keinen Rückgabewert.

h) Implementieren Sie schließlich die Methode `mulVec`. Diese multipliziert die aktuelle Matrix M mit einem `Vector3D`-Objekt $\vec{v}$ und gibt das Ergebnis in einem neuen Vektor $\vec{v}'$ zurück. Eine Matrix-Vektor-Multiplikation wird wie folgt durchgeführt:

$$\vec{v}' = M \cdot \vec{v} = \begin{bmatrix} M_{1,1} \cdot v_x + M_{1,2} \cdot v_y + M_{1,3} \cdot v_z \\ M_{2,1} \cdot v_x + M_{2,2} \cdot v_y + M_{2,3} \cdot v_z \\ M_{3,1} \cdot v_x + M_{3,2} \cdot v_y + M_{3,3} \cdot v_z \end{bmatrix}, \text{ mit } \vec{v} = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Geben Sie die Datei `Matrix3x3.java` über EST ab. Vergessen Sie nicht, Ihre Lösung sowohl mit dem öffentlichen Test als auch mit `Matrix3x3MainTestGUI` ☺ zu testen.

29 EP + 31 GP = 60 Punkte