Informatik 2 für Regenerative Energien

Klausur vom 23. September 2014

Jörn Loviscach

Versionsstand: 23. September 2014, 16:47

15 Punkte für die erste Aufgabe; 3 Punkte für alle weiteren Aufgaben. Mindestpunktzahl zum Bestehen: 20 Punkte. Hilfsmittel: maximal vier einseitig oder zwei beidseitig beschriftete DIN-A4-Spickzettel beliebigen Inhalts, möglichst selbst verfasst oder zusammengestellt; kein Skript, keine anderen Texte, kein Taschenrechner, kein Computer, kein Handy und Ähnliches.

Name	Vorname	Matrikelnummer	E-Mail-Adresse, falls nicht in ILIAS

1. Im C#-Programmlisting im Anhang sind 15 Fehler, darunter keine Tippfehler und höchstens ein Fehler pro Zeile. Erstellen Sie eine Liste mit 15 Zeilen aus den Fehlern und ihren jeweiligen Korrekturen, nach dem folgenden Muster:

Zeile	korrekter Programmtext
123	`public void foo()`
543	`int a = 42;`

2. Mit dem (korrigierten) Code aus dem Programmlisting im Anhang wird die Methode `Test.Teste()` ausgeführt. Was steht in den Variablen `empfA`, `empfB`, `empfC` in den Zeilen 18 bis 20? Beschreiben Sie gegebenenfalls, wie Sie zu Ihrer Antwort kommen.

3. Wenn derselbe Benutzer zum zweiten Mal versucht, dieselbe Musik zu bewerten, soll eine Exception geworfen werden. Welche Änderungen nehmen Sie dazu im (korrigierten) Code aus dem Programmlisting im Anhang vor?

4. Erweitern Sie in die Klasse `Empfehlungsgenerator` aus dem (korrigierten) Listing im Anhang um eine öffentliche Methode `int BestimmeNotenhäufigkeit(Note n)`, die als Ergebnis liefert, wie häufig die übergebene Note insgesamt vergeben worden ist.

5. Es sollen nicht nur einzelne Stücke bewertet werden können, sondern auch Sammlungen von Stücken, insbesondere Plattenalben. Leiten Sie dazu im (korrigierten) Listing im Anhang von der Klasse `Musik` eine Klasse `Album` ab. Sehen Sie in dieser Klasse ein Datenelement für die Sammlung vor und geben Sie ihr einen eigenen Konstruktor.

6. Jemand möchte ein anderes Verfahren ausprobieren, um Empfehlungen zu generieren. Das soll so eingebaut werden, dass man zu Beginn nicht den Konstruktor von `Empfehlungsgenerator` aufruft, sondern den einer neuen Klasse `Empfehlungsgenerator2`. Am Code, der den `Empfehlungsgenerator` nutzt, soll sich nichts ändern müssen. Beschreiben Sie in wenigen Sätzen, was im Sinne objektorientierter Programmierung nötig ist, um das zu verwirklichen.

7. Stellen Sie das Folgende C#-Programmfragment mit Klassen, Methoden, Attributen und Vererbung als UML-Diagramm dar:

```
class A : B
{
    int a;
}

class B
{
    public virtual string f(int x)
    {
        return "foo";
    }
}

class C : A
{
    public override string f(int x)
    {
        return "bar";
    }
}
```

   Damit Kursivschrift (falls nötig) zu erkennen ist, umkringeln Sie diese oder benutzen Sie eine andere Farbe dafür.

8. Welche Zahlen stehen nach Ausführung dieses C#-Programmfragments in den Variablen `x`, `y` und `z`? Geben Sie möglichst auch Zwischenschritte an, damit Ihr Gedankengang nachvollziehbar ist.

```
Queue<List<int>> a = new Queue<List<int>>();
List<int> b = new List<int>();
List<int> c = new List<int>();
a.Enqueue(b);
a.Enqueue(b);
a.Enqueue(b);
a.Enqueue(c);
b.Add(1);
b.Add(2);
b.Add(3);
c.Add(4);
int x = a.Dequeue().Count;
a.Dequeue().Add(5);
int y = a.Dequeue()[2];
int z = a.Count;
```

Dieses Listing enthält 15 Fehler!

Dies soll ein Programm werden, das Musik empfiehlt, die einem wahrscheinlich gefällt. – Und zwar so: Jeder Benutzer hat Bewertungen für diverse Musikstücke abgegeben. Wenn ein Benutzer eine Empfehlung erhalten soll, sucht das Programm nach einem anderen Benutzer, der möglichst gleiche Bewertungen abgegeben hat. Dann empfiehlt es Musik, die dieser andere Benutzer gut findet, der erste Benutzer aber noch nicht bewertet hat. Die Methode `Teste` macht die Handhabung der Klassen vor.

```
1  class Test
2  {
3      public static void Teste()
4      {
5          Empfehlungsgenerator em = new Empfehlungsgenerator;
6          Benutzer a = new Benutzer("Anna");
7          Benutzer b = new Benutzer("Bertram");
8          Benutzer c = new Benutzer("Carla");
9          Musik m1 = new Musik("Yesterday", "Beatles");
10         Musik m2 = new Musik("Respect", "Aretha␣Franklin");
11         Musik m3 = new Musik("Jailhouse␣Rock", "Elvis");
12         em.FügeBewertungHinzu(a, m1, Note.Schlecht);
13         em.FügeBewertungHinzu(a, m2, Note.Gut);
14         em.FügeBewertungHinzu(b, m1, Note.Gut);
```

```
15          em.FügeBewertungHinzu(b, m3, Note.Gut);
16          em.FügeBewertungHinzu(c, m2, Note.Gut);
17          em.FügeBewertungHinzu(c, m3, Note.SehrGut);
18          Musik empfA = em.GeneriereEmpfehlung(a);
19          Musik empfB = em.GeneriereEmpfehlung(b);
20          Musik empfC = em.GeneriereEmpfehlung(c);
21      }
22  }
23
24
25  class Benutzer
26  {
27      string name;
28      public string Name
29      { get { name; } }
30      public Benutzer(string name)
31      {
32          this.name = name;
33      }
34  }
35
36  class Musik
37  {
38      string titel;
39      string künstler;
40      Musik(string titel, string künstler)
41      {
42          this.titel = titel;
43          this.künstler = künstler;
44      }
45  }
46
47  enum Note { Fehlt, SehrSchlecht, Schlecht, Neutral, Gut, SehrGut }
48
49  abstract class Bewertung
50  {
51      public Benutzer Wer;
52      public Musik Was;
53      public Note Wie;
54      public void Bewertung(Benutzer wer, Musik was, Note wie)
55      {
56          Wer = wer;
57          Was = was;
58          Wie = wie;
59      }
60  }
61
62  class Empfehlungsgenerator
63  {
64      List<Bewertung> bewertungen = new List<Bewertung>();
65
```

```
66      public FügeBewertungHinzu(Benutzer wer, Musik was, Note wie)
67      {
68          bewertungen.Add(Bewertung(wer, was, wie));
69      }
70
71      public Musik GeneriereEmpfehlung(Benutzer)
72      {
73          Benutzer ähnlich = FindeÄhnlichstenBenutzer(b);
74          return FindeUngehörteGuteMusik(ähnlich, b);
75      }
76
77      private Benutzer FindeÄhnlichstenBenutzer(Benutzer b)
78      {
79          // Sammle alle Musik ein.
80          // Sammle alle anderen Benutzer ein.
81          List<Musik> eigeneMusik = new List<Musik>();
82          List<Benutzer> andereBenutzer = new List<Benutzer>();
83          foreach (Bewertung bew in bewertungen)
84          {
85              if (bew.Wer == b && !eigeneMusik.Contains(bew.Was))
86              {
87                  eigeneMusik.Add(bew.Was);
88              }
89              if (bew.Wer != b && andereBenutzer.Contains(bew.Wer))
90              {
91                  andereBenutzer.Add(bew.Wer);
92              }
93          }
94
95          // Sammle die vergebenen Noten ein.
96          Note[] eigeneNoten = new Note[eigeneMusik.Count];
97          Note[,] fremdeNoten = new Note[eigeneMusik.Count,
98                                         andereBenutzer.Count];
99          for (int i = 0; i < eigeneMusik.Count; i++)
100         {
101             Bewertung eigeneBewertung
102                 = bewertungen.Find(x => x.Was == eigeneMusik[i]
103                                          && x.Wer == b);
104             eigeneNoten[i] = eigeneBewertung.Wie;
105
106             for (int j = 0; j < andereBenutzer.Count; j++)
107             {
108                 foreach (Bewertung bew in bewertungen)
109                 {
110                     if (bew.Was == eigeneMusik[i]
111                         && bew.Wer == andereBenutzer)
112                     {
113                         fremdeNoten[i, j] = bew.Wie;
114                         break; // Wir nehmen den ersten Treffer.
115                     }
116                 }
```

```
117                 }
118             }
119 
120             // Finde anderen Benutzer mit ähnlichsten Bewertungen.
121             Benutzer ähnlichsterBenutzer = null;
122             int größteZahlDerÜbereinstimmungen = 0;
123             for (int j = 0; j < andereBenutzer.Count; j++)
124             {
125                 int übereinstimmungen = 0;
126                 for (int i = 0; i < eigeneMusik.Count; i++)
127                 {
128                     if (eigeneNoten[i] != fremdeNoten[i, j])
129                     {
130                         übereinstimmungen++;
131                     }
132                 }
133                 if(übereinstimmungen > größteZahlDerÜbereinstimmungen)
134                 {
135                     ähnlichsterBenutzer = andereBenutzer[j];
136                     größteZahlDerÜbereinstimmungen = übereinstimmungen;
137                 }
138             }
139             return ähnlichsterBenutzer;
140         }
141 
142         private Musik FindeUngehörteGuteMusik(Benutzer vonWem, Benutzer fürWen)
143         {
144             foreach (bew in bewertungen)
145             {
146                 if (bew.Wer == vonWem
147                     && (bew.Wie == Note.Gut && bew.Wie == Note.SehrGut))
148                 {
149                     bool schonGehört;
150                     foreach (Bewertung bew2 in bewertungen)
151                     {
152                         if (bew2.Was == bew.Was && bew2.Wer == fürWen)
153                         {
154                             schonGehört = true;
155                         }
156                     }
157                     if(!schonGehört)
158                     {
159                         return Was;
160                     }
161                 }
162             }
163             return null;
164         }
165 }
```