Informatik 2 für Regenerative Energien

Klausur vom 27. Juli 2017

Jörn Loviscach

Versionsstand: 26. Juli 2017, 19:33

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 (auch nicht wearable), kein Handy und Ähnliches.

Name	Vorname	Matrikelnummer	E-Mail-Adresse

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. Die Methode `Test.Teste` des (korrigierten) Code aus dem Programmlisting im Anhang wird ausgeführt. Was steht am Ende in den Variablen `x`, `y` und `z`? Beschreiben Sie gegebenenfalls, wie Sie zu Ihrer Antwort kommen.

3. Jeder Anschluss soll höchstens einmal verbunden werden können. Stellen Sie das mit einer Exception sicher. Was ändern Sie dazu wie an dem (korrigierten) Code aus dem Programmlisting?

4. Leiten Sie von der Klasse `Batterie` des korrigierten Code aus dem Programmlisting eine Klasse `Blockbatterie` ab und schreiben Sie einen Konstruktor dafür, der die Spannung fest auf 9 Volt setzt.

5. Schreiben Sie für die Klasse `Anschluss` des korrigierten Code aus dem Programmlisting eine Methode `Löse`, die entgegengesetzt zur Methode `Verbinde` wirkt. Sind Änderungen an weiteren Stellen im Code nötig? Wenn ja, welche? Hinweis: Die Methode `Remove(x)` einer Liste entfernt daraus das erste Vorkommen des Elements `x`.

6. Schreiben Sie für die Klasse `Bauelement` aus dem korrigierten Code aus dem Programmlisting eine öffentliche Methode `PrüfeObVerbunden(Bauelement b)`, deren Rückgabewert angibt, ob das aktuelle Bauelement direkt mit dem Bauelement `b` verbunden ist.

7. Zeichnen Sie das UML-Klassendiagramm zu diesen drei Klassen. Kennzeichen Sie dabei Kursivschrift durch eine andere Farbe o. ä.

```
abstract class A
{
    public int b;
    abstract public double c(int x);
}
abstract class B : A
{
    public long d;
    int e(int x)
    {
        return 2 * x;
    }
}
class C : B
{
    public override double c(int x)
    {
        return 3.14 * x;
    }
}
```

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.

```
Stack<List<int>> s = new Stack<List<int>>();
List<int> v = new List<int>();
v.Add(1);
v.Add(2);
v.Add(3);
s.Push(v);
s.Push(v);
List<int> u = new List<int>();
s.Push(u);
u.Add(4);
List<int> w = u;
w.Add(5);
int x = s.Pop().Count;
int y = s.Pop()[0];
int z = s.Pop()[1];
```

Dieses Listing enthält 15 Fehler!

Dies ist der Anfang eines Programms, das elektronische Schaltungen modelliert. Bauelemente haben Anschlüsse. Anschlüsse werden mittels Verbindungen verbunden; die Information darüber ist dann sowohl im Anschluss und in der Verbindung vorhanden. Die Methode `Teste` der Klasse `Test` macht die Benutzung der Klassen vor. Dies ist der Programmcode der Klassen:

```
1  class Test
2  {
3      public static void Teste()
4      {
5          Batterie b = new Batterie(9.0);
6          Transistor t = new Transistor(false);
7          Widerstand w1 = new Widerstand(10000.0);
8          Widerstand w2 = new Widerstand(1000.0);
9          Widerstand w3 = new Widerstand(470.0);
10         Widerstand w4 = w1;
11         Verbindung vPlus = new Verbindung();
12         Verbindung vMasse = new Verbindung();
13         Verbindung vBasis = new Verbindung();
14         Verbindung vKollektor = new Verbindung();
15         b.GibAnschluss("Pluspol").Verbinde(vPlus);
16         b.GibAnschluss("Minuspol").Verbinde(vMasse);
17         w1.GibAnschluss("A").Verbinde(vPlus);
18         w1.GibAnschluss("B").Verbinde(vBasis);
19         w2.GibAnschluss("A").Verbinde(vBasis);
20         w2.GibAnschluss("B").Verbinde(vMasse);
21         t.GibAnschluss("Basis").Verbinde(vBasis);
22         t.GibAnschluss("Emitter").Verbinde(vMasse);
23         t.GibAnschluss("Kollektor").Verbinde(vKollektor);
24         w3.GibAnschluss("A").Verbinde(vPlus);
25         w3.GibAnschluss("B").Verbinde(vKollektor);
26
27         bool x = t.PrüfeObAlleAnschlüsseVerbunden();
28         bool y = w4.PrüfeObAlleAnschlüsseVerbunden();
29         string z = t.FindeVerbundeneBauelemente()[1].Name;
30     }
31 }
32
33 class Anschluss
34 {
35     string name;
36     public string Name { get { return name; } }
37
38     Bauelement bauelement;
39     public Bauelement Element { get { return bauelement; } }
40
41     Verbindung verbundenMit;
42
43     public Anschluss(string name, Bauelement bauelement)
44     {
```

```
45          this.name = name;
46          this.bauelement = bauelement;
47      }
48
49      public void Verbinde(v)
50      {
51          verbundenMit = v;
52          v.FügeHinzu(this);
53      }
54
55      public bool PrüfeObVerbunden()
56      {
57          return verbundenMit != null;
58      }
59
60      override public List<Anschluss> FindeVerbundeneAnschlüsse()
61      {
62          List<Anschluss> ergebnis = new List<Anschluss>;
63          if (verbundenMit != null)
64          {
65              List<Anschluss> alle = verbundenMit.GibVerbundeneAnschlüsse();
66              for (int i = 0; i < Count; i++)
67              {
68                  if (alle[i] != this)
69                  {
70                      ergebnis.Add(alle);
71                  }
72              }
73          }
74          return;
75      }
76  }
77
78  class Verbindung
79  {
80      List<Anschluss> verbundeneAnschlüsse = new List<Anschluss>();
81
82      public void FügeHinzu(Anschluss a)
83      {
84          verbundeneAnschlüsse.Add(a);
85      }
86
87      public List<Anschluss> GibVerbundeneAnschlüsse()
88      {
89          // Die nächste Zeile ist korrekt. Sie liefert eine Kopie der Liste.
90          return new List<Anschluss>(verbundeneAnschlüsse);
91      }
92  }
93
94  abstract class Bauelement
95  {
```

```
96      Anschluss[] anschlüsse;
97
98      string name;
99      public string Name { get { return name; } }
100
101     public Bauelement(string name)
102     {
103         this.name = name;
104     }
105
106     public GibAnschluss(string name)
107     {
108         for (int i = 0; i < anschlüsse.Length; i++)
109         {
110             if(anschlüsse[i].Name != name)
111             {
112                 return anschlüsse[i];
113             }
114         }
115         return;
116     }
117
118     abstract public bool PrüfeObAlleAnschlüsseVerbunden()
119     {
120         for (int i = 0; i < anschlüsse.Length; i++)
121         {
122             if(!anschlüsse[i].PrüfeObVerbunden())
123             {
124                 return false;
125             }
126         }
127         return true;
128     }
129
130     public List<Bauelement> FindeVerbundeneBauelemente()
131     {
132         b = new List<Bauelement>();
133         for (int i = 0; i < anschlüsse.Length; i++)
134         {
135             List<Anschluss> a = anschlüsse[i].FindeVerbundeneAnschlüsse();
136             for (int j = 0; j < a.Count; j++)
137             {
138                 Bauelement be = a[j].Element();
139                 if(!b.Contains(be))
140                 {
141                     b.Add(be);
142                 }
143             }
144         }
145         return b;
146     }
```

```
147 }
148
149 class Widerstand : Bauelement
150 {
151     double wertInOhm;
152
153     public Widerstand(double wertInOhm)
154         : base("R " + wertInOhm + " ?")
155     {
156         this.wertInOhm = wertInOhm;
157         anschlüsse = new Anschluss[] { new Anschluss("A", this),
158                                        new Anschluss("B", this) };
159     }
160 }
161
162 class Transistor : Bauelement
163 {
164     bool istPNP;
165
166     public Transistor(double istPNP)
167         : base(istPNP ? "BC 177A", "BC 107A")
168     {
169         this.istPNP = istPNP;
170         anschlüsse = new Anschluss[] { new Anschluss("Emitter", this),
171                                        new Anschluss("Basis", this),
172                                        new Anschluss("Kollektor", this) };
173     }
174 }
175
176 class Batterie : Bauelement
177 {
178     double spannung;
179
180     public Batterie(double spannung)
181         : base("Batterie mit " + spannung + " Volt")
182     {
183         this.spannung = spannung;
184         anschlüsse = new Anschluss[] { new Anschluss("Pluspol", this),
185                                        new Anschluss("Minuspol", this) };
186     }
187 }
```