Informatik 2 für Regenerative Energien

Klausur vom 15. Juli 2015

Jörn Loviscach

Versionsstand: 15. Juli 2015, 09:50

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. Mit dem (korrigierten) Code aus dem Programmlisting im Anhang wird Folgendes ausgeführt:

```
Zelle f = new Fresszelle(640, 480);
Zelle b = new Bakterium(640, 480);
List<Zelle> zellen = new List<Zelle>();
zellen.Add(f);
zellen.Add(b);
int x = f.Vermehre().Count;
b.FrissWennMöglich(zellen);
int y = zellen.Count;
bool z = zellen[0].IstNahe(f);
```

   Was steht danach in den Variablen `x`, `y`, `z`? Beschreiben Sie gegebenenfalls, wie Sie zu Ihrer Antwort kommen.

3. Beschreiben Sie in ca. zwei Sätzen, an welcher Stelle im Programm und aus welchem Grund Folgendes mit dem (korrigierten) Code aus dem Programmlisting fehlschlägt:

```
Zelle g = new Fresszelle(640, 480);
g.IstNahe(null);
```

4. Jede Fresszelle soll sich merken, wie viele Bakterien sie gefressen hat. Diese Anzahl soll man öffentlich abfragen können. Welche Änderungen sind dazu an den (korrigierten) Klassen im Anhang nötig?

5. Erweitern Sie den Code der `for`-Schleife in den Zeilen 31 bis 34 des Listings im Anhang so, dass die Bakterien mit einem gewissen Mindestabstand zueinander platziert werden. Verwenden Sie dazu die Methode `IstNahe`.

6. Es soll eine Klasse für eine weitere Sorte an Bakterien geschrieben werden. Diese sollen als längliche Ovale dargestellt werden. Beschreiben Sie in ca. drei Sätzen, was dafür zu tun ist.

7. Zeichnen Sie das UML-Klassendiagramm für diese C#-Klassen:

```
abstract class X
{
    int a;
    public abstract int f(double x);
}

class Y : X
{
    double b;
    string g(int y)
    {
        return "bla";
    }
}

class Z : Y
{
    public override int f(double x)
    {
        return 42;
    }
}
```

   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.

```
Stack<List<int>> a = new Stack<List<int>>();
List<int> b = new List<int>();
List<int> c = new List<int>();
a.Push(b);
a.Push(b);
a.Push(c);
a.Push(c);
b.Add(16);
b.Add(23);
a.Pop().Add(8);
int x = a.Pop()[0];
int y = a.Pop()[0];
int z = a.Pop()[0];
```

Dieses Listing enthält 15 Fehler!

Dies soll ein Programm für eine biologische Simulation sein: Fresszellen und Bakterien bewegen sich langsam über eine Fläche. Die Bakterien vermehren sich. Die Fresszellen fressen Bakterien, die in ihre Nähe kommen.

```
1  abstract class Simulation
2  {
3      protected double fensterbreite;
4      protected double fensterhöhe;
5      protected List<Zelle> zellen = new List<Zelle>;
6
7      public Simulation(double fensterbreite, double fensterhöhe)
8      {
9          this.fensterbreite = fensterbreite;
10         this.fensterhöhe = fensterhöhe;
11     }
12
13     public void MacheSimulationsSchritt();
14
15     public void Zeichne(Canvas c)
16     {
17         c.Children.Clear();
18         foreach (var z in Zelle)
19         {
20             Zeichne(c);
21         }
22     }
23 }
24
```

```
25 class Bakteriensimulation
26 {
27     public Bakteriensimulation(int zahlBakterien, int zahlFresszellen,
28         double fensterbreite, double fensterhöhe)
29         : base(fensterbreite, fensterhöhe)
30     {
31         for (int i = 0; i < zahlBakterien; i++)
32         {
33             zellen.Add(new Bakterium(fensterbreite, fensterhöhe));
34         }
35         for (int i = 0; i < zahlFresszellen; i++)
36         {
37             zellen.Add(new Fresszelle(fensterbreite, fensterhöhe));
38         }
39     }
40
41     public override void MacheSimulationsSchritt()
42     {
43         int i = 0;
44         while (i < zellen.Count)
45         {
46             Zelle z = zellen[i];
47             z.FrissWennMöglich(zellen);
48
49             // z kann Zellen gefressen haben, die in der Liste
50             // vor z stehen; das bisherige i stimmt dann nicht mehr.
51             // Deshalb:
52             i = zellen.IndexOf(z) + 1;
53         }
54
55         List<Zelle> tochterzellen = new List<Zelle>();
56         foreach (var z in zellen)
57         {
58             z.Bewege(fensterbreite, fensterhöhe);
59             // AddRange heißt: alles aus einer anderen Liste hinzufügen
60             tochterzellen.AddRange(z.Vermehre);
61         }
62         zellen.AddRange(tochterzellen);
63     }
64 }
65
66 abstract class Zelle
67 {
68     static Random würfel = new Random();
69     protected Point ort;
70     protected double geschwindigkeit = 5.0;
71     protected Brush farbe = Brushes.Red;
72     protected double radius = 2.0;
73
74
75
```

```
76      public Zelle(Point p)
77      {
78          ort = p;
79      }
80
81      public Zelle(double fensterbreite, double fensterhöhe)
82      {
83          ort = Point(fensterbreite * würfel.NextDouble(),
84                      fensterhöhe * würfel.NextDouble());
85      }
86
87      public void Zeichne(Canvas canvas)
88      {
89          Ellipse ellipse = new Ellipse();
90          ellipse.Width = 2.0 * radius;
91          ellipse.Height = 2.0 * radius;
92          ellipse.Fill = farbe;
93          Canvas.SetLeft(ellipse, ort.X - radius); // korrekt!
94          Canvas.SetTop(ellipse, ort.Y - radius);
95          canvas.Children.Add(ellipse);
96      }
97
98      public virtual List<Zelle> Vermehre()
99      {
100         return new List<Zelle>();
101     }
102
103     public virtual void FrissWennMöglich(List<Zelle> zellen)
104     { }
105
106     public void Bewege(double fensterbreite, double fensterhöhe)
107     {
108         double x = ort.X + geschwindigkeit * (würfel.NextDouble() - 0.5);
109         double y = ort.Y + geschwindigkeit * (würfel.NextDouble() - 0.5);
110
111         // Ins Fenster zurückbringen:
112         if (x > fensterbreite) { x -= fensterbreite; }
113         else if (x < 0) { x += fensterbreite; }
114
115         if (y > fensterhöhe) { y -= fensterhöhe; }
116         else if (y < 0) { y += fensterhöhe; }
117
118         ort = new Point();
119     }
120
121     public bool IstNahe(Zelle z)
122     {
123         double dx = ort.X - z.ort.X;
124         double dy = ort.Y - z.Y;
125         // Pythagoras!
126         return dx * dx + dy * dy < 7.0 * 7.0;
```

```
127     }
128 }
129
130 class Fresszelle : Zelle
131 {
132     public override Fresszelle(double fensterbreite, double fensterhöhe)
133         : base(fensterbreite, fensterhöhe)
134     {
135         geschwindigkeit = 3.0;
136         farbe = Brushes.Green;
137         radius = 5.0;
138     }
139
140     public override void FrissWennMöglich(List<Zelle> zellen)
141     {
142         int i = 0;
143         while (i < zellen)
144         {
145             if (this != zellen[i])
146             {
147                 if (zellen[i] is Bakterium && !IstNahe(zellen[i]))
148                 {
149                     zellen.RemoveAt(i);
150                     continue;
151                 }
152             }
153             i++;
154         }
155     }
156 }
157
158 class Bakterium : Zelle
159 {
160     public Bakterium(Point p)
161         : base(Point p)
162     { }
163
164     public Bakterium(double fensterbreite, double fensterhöhe)
165         : base(fensterbreite, fensterhöhe)
166     { }
167
168     public List<Zelle> Vermehre()
169     {
170         List<Zelle> tochterzellen = new List<Zelle>();
171         if (würfel.NextDouble() < 0.001)
172         {
173             tochterzellen.Add(new Bakterium(ort));
174         }
175         return tochterzellen;
176     }
177 }
```