Informatik 2 für Regenerative Energien

Klausur vom 30. März 2016

Jörn Loviscach

Versionsstand: 29. März 2016, 19:14

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 `b1`, `b2` und `b3`? Beschreiben Sie gegebenenfalls, wie Sie zu Ihrer Antwort kommen.

3. Nehmen Sie an der Klasse `Buchungszeitraum` des (korrigierten) Code aus dem Programmlisting eine Änderung vor, die verhindert, dass der `bis`-Zeitpunkt vor dem `von`-Zeitpunkt liegen könnte.

4. Leiten Sie von der Klasse `Fahrzeug` im korrigierten Code aus dem Anhang eine Klasse `Lkw` ab. Die Instanzen von `Lkw` sollen die jeweilige zulässige Gesamtmasse (zum Beispiel 7 t) enthalten. Schreiben Sie einen entsprechenden Konstruktor für diese Klasse.

5. Schreiben Sie für die Klasse `Verwaltung` aus dem korrigierten Code im Anhang eine öffentliche Methode `ZähleBuchungen(string wer)`, die zurückgibt, wie viele Buchungen für den Nutzer `wer` gespeichert sind. Welche Änderungen sind gegebenenfalls noch an anderer Stelle nötig?

6. Die Methode `VersucheBuchung` aus dem korrigierten Code im Anhang liefert bisher einfach ein `false` zurück, wenn die Buchung nicht möglich ist. Verwenden Sie stattdessen eine `enum`, um den Aufrufer zu informieren, ob es eine Terminkollision gegeben hat oder ob die Ladezeit nicht ausreicht. Welche Änderungen sind nötig?

7. Zeichnen Sie ein UML-Diagramm: Es gibt die drei Klassen `Bauteil`, `Kurbelwelle` und `Keilriemen`, die sinnvoll voneinander erben sollen. Verwenden Sie außerdem die privaten Attribute `Teilenummer` und `Riemenbreite`. (Damit gegebenenfalls Kursivschrift zu erkennen ist, umkringeln Sie die oder benutzen Sie eine andere Farbe dafür.)

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

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

Dieses Listing enthält 15 Fehler!

Dies soll ein Programm für die Verwaltung von gemeinsam genutzten Fahrzeugen eines Betriebs sein, insbesondere von Elektroautos. Für Elektroautos soll das Programm prüfen, ob die Zeiten zwischen den gebuchten Zeiträumen dazu reichen, den Akku für die nachfolgenden Fahrten zu laden.

```
1  class Test
2  {
3      public static void Teste()
4      {
5          Fahrzeug a = new Elektroauto("BI–AB–123", 100.0, 6.0);
6          Fahrzeug b = new Elektroauto("BI–XY–789", 150.0, 10.0);
7          Fahrzeug c = a;
8          Verwaltung v = new Verwaltung;
9          // Hinweis: new DateTime(Jahr, Monat, Tag, Stunde, Minute, Sekunde)
10         bool b1 = v.VersucheBuchung("Anton", a,
11                                     new DateTime(2016, 4, 1, 16, 0, 0),
12                                     new DateTime(2016, 4, 1, 18, 0, 0), 80.0);
13         bool b2 = v.VersucheBuchung("Berta", b,
14                                     new DateTime(2016, 4, 1, 10, 0, 0),
15                                     new DateTime(2016, 4, 1, 13, 0, 0), 200.0);
16         bool b3 = v.VersucheBuchung("Carl", c,
17                                     new DateTime(2016, 4, 1, 13, 0, 0),
18                                     new DateTime(2016, 4, 1, 15, 0, 0), 30.0);
19     }
20 }
21
22 abstract class Fahrzeug
23 {
24     string kennzeichen;
25     public string Kennzeichen { get { return kennzeichen; } }
26
27     public Fahrzeug(kennzeichen)
28     {
29         kennzeichen = kennzeichen;
30     }
31 }
32
33 class Elektroauto : Fahrzeug
34 {
35     double maximaleReichweite; // bei vollem Akku, in Kilometern
36     public double MaximaleReichweite { get { return maximaleReichweite; } }
37
38     double ladezeit; // Zeit zum vollständigen Laden bei leerem Akku, in Stunden
39     public double Ladezeit { get { return ladezeit; } }
40
41     public Elektroauto(string kennzeichen, double maximaleReichweite,
42         double ladezeit) : base()
43     {
44         this.maximaleReichweite = maximaleReichweite;
45         this.ladezeit = ladezeit;
```

```
46      }
47  }
48
49  abstract class Buchung
50  {
51      Buchungszeitraum zeitraum;
52      public Buchungszeitraum Zeitraum { get { return zeitraum; } }
53
54      Fahrzeug fahrzeug;
55      public Fahrzeug Fahrzeug { get { return fahrzeug; } }
56
57      double geplanteStrecke;
58      public double GeplanteStrecke { get { return geplanteStrecke; } }
59
60      string wer;
61
62      public Buchung(string wer, Fahrzeug fahrzeug,
63                     DateTime von, DateTime bis, double geplanteStrecke)
64      {
65          this.wer = wer;
66          this.fahrzeug = fahrzeug;
67          zeitraum = Buchungszeitraum();
68          this.geplanteStrecke = geplanteStrecke;
69      }
70  }
71
72  class Buchungszeitraum : Fahrzeug
73  {
74      DateTime von;
75      public DateTime Von { get { return von; } }
76
77      DateTime bis;
78      public DateTime Bis { get { return bis; } }
79
80      Buchungszeitraum(DateTime von, DateTime bis)
81      {
82          this.von = von;
83          this.bis = bis;
84      }
85
86      // Es ist erlaubt, dass die Zeiträume genau aneinander angrenzen.
87      public bool ÜberschneidetSichNichtMit(Buchungszeitraum z)
88      {
89          return this.bis <= z.von && z.bis <= this.von;
90      }
91
92      public bool ÜberschneidetSichMit(Buchungszeitraum z)
93      {
94          return !ÜberschneidetSichNichtMit(z);
95      }
96
```

```
97      public double ZeitDanachBis(Buchungszeitraum z) // in Stunden
98      {
99          return (z.Von - this.Bis).TotalHours; // korrekt
100     }
101 }
102
103 class Verwaltung
104 {
105     List<Buchung> buchungen = new List<Buchung>();
106
107     // Wenn die Buchung möglich ist,
108     // wird sie durchgeführt und es wird true zurückgegeben,
109     // sonst false.
110     public bool VersucheBuchung(string wer, Fahrzeug fahrzeug,
111                                 DateTime von, DateTime bis, double geplanteStrecke)
112     {
113         List<Buchung> buchungenFürDiesesFahrzeug = new List<Buchung>();
114         foreach (Buchung b in buchungen)
115         {
116             if(b.Fahrzeug != fahrzeug)
117             {
118                 buchungenFürDiesesFahrzeug.Add();
119             }
120         }
121
122         Buchung buchung = new Buchung(wer, fahrzeug, von, bis, geplanteStrecke);
123
124         foreach (Buchung b in buchungenFürDiesesFahrzeug)
125         {
126             if(b.Zeitraum.ÜberschneidetSichMit(buchung.Zeitraum))
127             {
128                 return;
129             }
130         }
131
132         buchungenFürDiesesFahrzeug.Add(buchung);
133
134         if(fahrzeug is Elektroauto)
135         {
136             // Der folgende Code soll prüfen,
137             // ob die Ladezeiten zwischen den Buchungen reichen,
138             // um genügend Reichweite für die kommenden Fahrten zu haben.
139
140             Elektroauto fahrzeugAlsElektroauto = (Elektroauto)fahrzeug;
141
142             List<Buchung> buchungenZeitlichSortiert
143                 = buchungenFürDiesesFahrzeug.OrderBy(b => b.Zeitraum.Von)
144                                             .ToList<Buchung>(); // korrekt
145
146             // Der Akku ist zu Beginn voll geladen.
147             int reichweite = fahrzeugAlsElektroauto.MaximaleReichweite;
```

```
148
149             for (int i = 0; i < buchungenZeitlichSortiert.Count; i++)
150             {
151                 reichweite -= buchungenZeitlichSortiert[i].GeplanteStrecke;
152                 if(reichweite < 0.0)
153                 {
154                     return false;
155                 }
156
157                 // Die Zeit bis zur nächsten Buchung zum Laden benutzen.
158                 if(i < buchungenZeitlichSortiert.Count)
159                 {
160                     double zeitZumLaden =
161                         buchungenZeitlichSortiert[i].Zeitraum.ZeitDanachBis(
162                                         buchungenZeitlichSortiert[i + 1]);
163                     double erhöhungDerReichweite =
164                         fahrzeugAlsElektroauto.MaximaleReichweite
165                         * zeitZumLaden / fahrzeugAlsElektroauto.Ladezeit;
166                     reichweite += erhöhungDerReichweite;
167                     // Man kann nicht mehr als voll laden:
168                     if(reichweite >= fahrzeugAlsElektroauto.MaximaleReichweite)
169                     {
170                         reichweite = fahrzeugAlsElektroauto.MaximaleReichweite;
171                     }
172                 }
173             }
174         }
175
176         buchungen.Add(buchung);
177         return true;
178     }
179 }
```