technische universität dortmund

fakultät für informatik

Lutz Oettershagen
Jurij Kuzmic

Dortmund, den 10. Januar 2019

Übung zur Vorlesung EidP (WS 2018/19)

Blatt 10

Block grün

Es können 4 Punkte erreicht werden.

Abgabedatum: 17. Januar 2019, 23:59 Uhr

Hinweise

- Bitte beachten Sie die aktuellen Hinweise unter

  `https://ls11-www.cs.tu-dortmund.de/teaching/ep1819uebung/`

- Stellen Sie sicher, dass alle von Ihnen abgegebene Dateien reine Textdateien im UTF-8-Format sind.
- Für die Abgabe sind die Dateien `Aufgabe_10_1.txt`, `Fahrzeug.h`, `Fahrzeug.cpp` und `FahrzeugTest.cpp` notwendig.
- Für die Kompilierung muss ebenfalls sichergestellt werden, dass sich Ihre Programme mit den Parametern `-pedantic` und `-Werror` kompilieren lassen.
- Für die Programmieraufgaben kopieren Sie immer die Ergebnisse als Block-Kommentar an das Ende der Datei, welche das jeweilige Hauptprogramm enthält.

Aufgaben

Aufgabe 1: Grundlagen (1 Punkt)

Legen Sie für Ihre Antworten eine Text-Datei `Aufgabe_10_1.txt` an.

a) Was gilt in Bezug auf Attribute und Methoden, wenn `A` eine Oberklasse von `B` ist? (0.2 Punkte)

b) Was versteht man unter Polymorphie? (0.2 Punkte)

c) Wie werden virtuelle Methoden gekennzeichnet und wann werden sie gebunden? (0.2 Punkte)

d) Wie werden rein virtuelle Methoden deklariert? (0.1 Punkte)

e) Wann nennt man eine Klasse abstrakt? (0.1 Punkte)

f) Wozu werden abstrakte Klassen eingesetzt? (0.2 Punkte)

Aufgabe 2: Vererbung (3 Punkte)

Legen Sie für diese Aufgabe die Dateien `Fahrzeug.h`, `Fahrzeug.cpp` und `FahrzeugTest.cpp` an.

a) Erstellen Sie ein Programm für die Speicherung von Fahrzeugdaten. Für die unterschiedlichen Fahrzeugarten (PKWs, Motorräder und LKWs) sind jeweils unterschiedliche Daten zu erfassen:

1. Für alle Fahrzeugtypen: Kennzeichen, Erstzulassung [Jahr], Hubraum [ccm]
2. PKWs, zusätzlich: Leistung [kW], Schadstoffklasse [Euro 1-6]
3. Motorräder, zusätzlich: Beiwagen [Ja/Nein]
4. LKWs, zusätzlich: Achsen [Anzahl], Zuladung [t]

Erstellen Sie geeignete `C++`-Klassen, um die oben aufgelisteten Daten **ohne Redundanz** abzuspeichern. Wählen Sie sinnvolle Datentypen für die jeweiligen Attribute. (1 Punkt)

b) Erweitern Sie Ihr Programm um die Delegation an die Basisklassenkonstruktoren und die Initialisierungsliste wo immer dies möglich ist. Schreiben Sie zusätzlich eine Methode `void print()` zur Ausgabe der gespeicherten Daten für die jeweiligen Klassen. Jede `void print()` Methode muss in der jeweiligen Klasse überschrieben werden. Diese gibt dann nur die in der Klasse gespeicherten Attribute aus. (1 Punkt)

c) Schreiben Sie ein Hauptprogramm `FahrzeugTest.cpp` und legen Sie pro Fahrzeug jeweils zwei Testobjekte an. Folgendes Beispiel muss nach Ihrer Implementierung möglich sein:

```
1 PKW vw("MK - JK 1111", 2006, 1980, 130, 2);
2 Fahrzeug *seat = new PKW(" K - KJ 1284", 2014, 1990, 150, 5);
3 vw.print();
4 seat->print();
```

Beachten Sie dabei, dass von der Basisklasse **keine Objekte** erzeugt werden sollen. Geben Sie anschließend die gespeicherten Daten mit der `print()` Methode aus. (1 Punkt)

Hinweis: Unterteilen Sie die Deklaration und die Definition der jeweiligen Klassen in die Dateien `Fahrzeug.h` und `Fahrzeug.cpp`. Handeln Sie ebenfalls nach dem „`information hiding`“ Prinzip.

Präsenzaufgabe 3: Konstruktoren und Destruktoren (0 Punkte)

Geben Sie die Ausgabe des folgenden Programms an.

```
1  /********** konstdest.h **********/
2  /* Klasse: Baz */
3  class Baz {
4  private:
5    char x;
6  public:
7    Baz();
8    Baz(char y);
9    ~Baz();
10 };
11
12 /* Klasse: Foo */
13 class Foo {
14 private:
15   char x;
16 public:
17   Foo();
18   Foo(char y);
19   Foo(char x, char y);
20   ~Foo();
21 };
22
23 /* Klasse: Bar */
24 class Bar : Foo, Baz {
25 private:
26   char x;
27 public:
28   Bar();
29   Bar(char y);
30   ~Bar();
31   void barbaz(char c);
32 };
33 /************* Ende **************/
```

```
1  /********* konstdest.cpp *********/
2  #include <iostream>
3  #include "konstdest.h"
4  using namespace std;
5  /********** Klasse: Baz **********/
6  Baz::Baz() : Baz('B') {
7    cout << "Baz\t" << x << endl;
8  }
9
10 Baz::Baz(char y) : x('Z') {
11   char t = x;
12   x = y;
13   cout << "Baz\t" << x << endl;
14   x = t;
```

```
15  }
16
17  Baz::~Baz() {
18    cout << "-Baz\t" << x << endl;
19  }
20
21  /********** Klasse: Foo **********/
22  Foo::Foo() : Foo('F') {
23    x++;
24    cout << "Foo\t" << x << endl;
25  }
26
27  Foo::Foo(char y) : x(y) {
28    cout << "Foo\t" << x << endl;
29  }
30
31  Foo::Foo(char x, char y) : x(x) {
32    cout << "Foo\t" << x << endl;
33    cout << "Foo\t" << y << endl;
34  }
35
36  Foo::~Foo() {
37    cout << "-Foo\t" << x << endl;
38  }
39
40  /********** Klasse: Bar **********/
41  Bar::Bar() : Bar('B') {
42    x = 'A';
43    cout << "Bar\t" << x << endl;
44  }
45
46  Bar::Bar(char y) : x(y) {
47    cout << "Bar\t" << x << endl;
48  }
49
50  Bar::~Bar() {
51    cout << "-Bar\t" << x << endl;
52  }
53
54  void Bar::barbaz(char c) {
55    Baz b(c);
56    Foo f(c+1, c+2);
57  }
58  /********* main-Funktion *********/
59  int main() {
60    Bar f;
61    f.barbaz('X');
62    return 0;
63  }
64  /************* Ende **************/
```