Einführung in die Programmierung

Tutorium 12

Wintersemester 2025/2026

Arbeitsgruppe Systemsoftware
Angewandte Informatik 12

technische universität dortmund

Arbeitsgruppe Systemsoftware

Aufgabe 1.a:

Was versteht man unter Polymorphie?

Polymorphie erlaubt es, dass zur Laufzeit basierend auf dem konkreten Typ eines Objekts entschieden wird, welche Implementierung einer Methode ausgeführt wird.

Aufgabe 1.b:

Was gilt in Bezug auf Attribute und Methoden, wenn A eine Oberklasse von B ist?

Die Unterklasse B erbt alle nicht-privaten Attribute und Methoden von A und kann diese nutzen, erweitern oder überschreiben.

Aufgabe 1.c:

Wie werden virtuelle Methoden gekennzeichnet und wann werden sie gebunden?

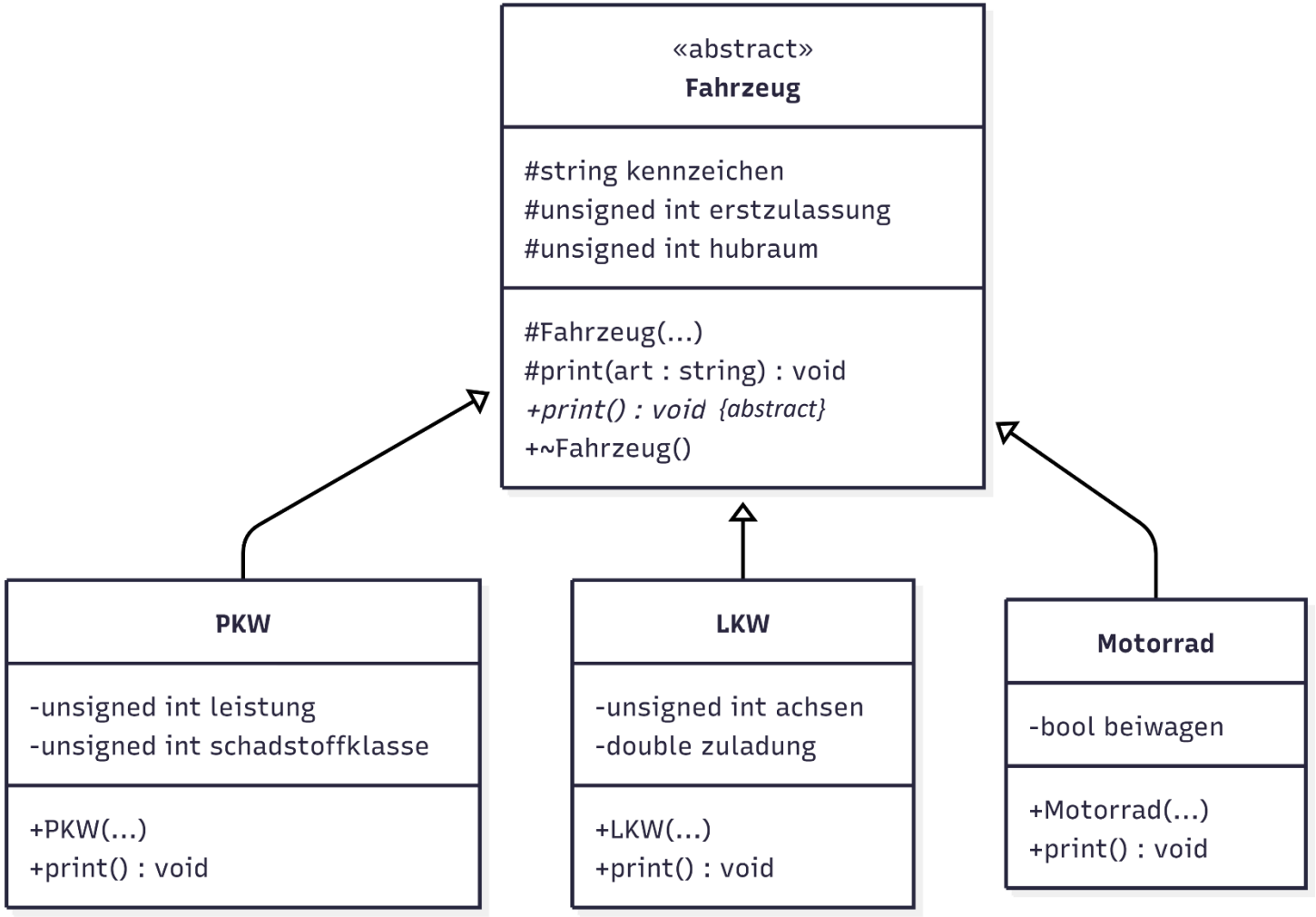
Sie werden mit dem Schlüsselwort virtual gekennzeichnet und dynamisch zur Laufzeit gebunden.

Aufgabe 1.d:
Wie werden rein virtuelle Methoden deklariert?

Man deklariert sie mit dem Schlüsselwort virtual und fügt am Ende der Signatur = 0; anstelle eines Funktionsrumpfes an.

Aufgabe 1.e:
Wann nennt man eine Klasse abstrakt?

Eine Klasse gilt als abstrakt, sobald sie mindestens eine rein virtuelle Methode enthält.


«abstract»
Fahrzeug
#string kennzeichen
#unsigned int erstzulassung
#unsigned int hubraum
#Fahrzeug(...)
#print(art : string) : void
+print() : void {abstract}
+~Fahrzeug()
PKW
-unsigned int leistung
-unsigned int schadstoffklasse
+PKW(...)
+print() : void
LKW
-unsigned int achsen
-double zuladung
+LKW(...)
+print() : void
Motorrad
-bool beiwagen
+Motorrad(...)
+print() : void

Aufgabe 2.a:

Erstellen Sie geeignete C++-Klassen für PKWs, Motorräder und LKWs, um die spezifischen Daten redundanzfrei in einer abstrakten Basisklasse zu speichern, und unterteilen Sie den Code in die Dateien Aufgabe_2.h und Aufgabe_2.cpp.

Aufgabe_2.h

```
…
class Fahrzeug {
    protected:


        string kennzeichen;
        unsigned int erstzulassung;
        unsigned int hubraum;

    public:

};
```

Aufgabe_2.h

```
…
class Motorrad : public Fahrzeug {
    private:
        bool beiwagen;

    public:


};
```

Aufgabe_2.h

```
…
43  class PKW : public Fahrzeug {
44      private:
45          unsigned int leistung;
46          unsigned int schadstoffklasse;
47
48      public:
49
50
51
52
53
54
55
56
57  };
```

Aufgabe_2.h

```
…
62  class LKW : public Fahrzeug {
63      private:
64          unsigned int achsen;
65          double zuladung;
66
67      public:
68
69
70
71
72
73
74
75
76  };
```

Aufgabe 2.b:

Erweitern Sie Ihr Programm um die Delegation an die Basisklassenkonstruktoren und schreiben Sie eine print()-Methode, die in der jeweiligen Klasse überschrieben wird.

Aufgabe_2.h

```
…
class Fahrzeug {
    protected:
        Fahrzeug(string const kennzeichen,
                 unsigned int const erstzulassung,
                 unsigned int const hubraum) :
                 kennzeichen(kennzeichen), erstzulassung(erstzulassung),
                 hubraum(hubraum) {};
        string kennzeichen;
        unsigned int erstzulassung;
        unsigned int hubraum;
        void print(string const &art);
    public:
        virtual void print() = 0;        // fuer Ausgabe in der main
        virtual ~Fahrzeug() {}
};
```

Aufgabe_2.h

```
…
26  class Motorrad : public Fahrzeug {
27      private:
28          bool beiwagen;
29
30      public:
31          Motorrad(string const kennzeichen,
32                   unsigned int const erstzulassung,
33                   unsigned int const hubraum,
34                   bool const beiwagen) :
35                   Fahrzeug(kennzeichen, erstzulassung, hubraum),
36                   beiwagen(beiwagen) {};
37          void print();
38  };
```

Aufgabe_2.h

```
…
43  class PKW : public Fahrzeug {
44      private:
45          unsigned int leistung;
46          unsigned int schadstoffklasse;
47
48      public:
49          PKW(string const kennzeichen,
50              unsigned int const erstzulassung,
51              unsigned int const hubraum,
52              unsigned int const leistung,
53              unsigned int const schadstoffklasse) :
54              Fahrzeug(kennzeichen, erstzulassung, hubraum),
55              leistung(leistung), schadstoffklasse(schadstoffklasse) {};
56          void print();
57  };
```

Aufgabe_2.h

```
…
62  class LKW : public Fahrzeug {
63      private:
64          unsigned int achsen;
65          double zuladung;
66
67      public:
68          LKW(string const kennzeichen,
69              unsigned int const erstzulassung,
70              unsigned int const hubraum,
71              unsigned int const achsen,
72              double const zuladung) :
73              Fahrzeug(kennzeichen, erstzulassung, hubraum),
74              achsen(achsen), zuladung(zuladung) {};
75          void print();
76  };
```

Aufgabe_2.cpp

```
1   #include "Aufgabe_2.h"
2
3   /* Klasse: Fahrzeug */
4   void Fahrzeug::print(string const &art) {
5       cout << "********************* " << art;
6       cout << " *********************" << endl;
7       cout << "Kennzeichen:       " << kennzeichen << endl;
8       cout << "Erstzulassung:     " << erstzulassung << endl;
9       cout << "Hubraum(ccm):      " << hubraum << endl;
10  }
```

In C++ dürfen auch rein virtuelle Funktionen eine Implementierung besitzen, um gemeinsame Funktionalität zentral in der Basisklasse bereitzustellen. Diese dient als Default-Implementierung, die von abgeleiteten Klassen zwar überschrieben werden muss, aber dennoch explizit (z. B. via Fahrzeug::print) aufgerufen werden kann.

Aufgabe_2.cpp

```
…
13  /* Klasse: PKW */
14  void PKW::print() {
15      Fahrzeug::print("PKW");
16      cout << "Leistung(kW):     " << leistung << endl;
17      cout << "Schadstoffklasse: " << schadstoffklasse << endl << endl;
18  }
```

Aufgabe_2.cpp

```
…
21  /* Klasse: LKW */
22  void LKW::print() {
23      Fahrzeug::print("LKW");
24      cout << "Achsen(n):          " << achsen << endl;
25      cout << "Zuladung(t):        " << zuladung << endl << endl;
26  }
```

Aufgabe_2.cpp

```
…
21  /* Klasse: Motorrad */
22  void Motorrad::print() {
23      Fahrzeug::print("Motorrad");
24      cout << "Beiwagen:          " << ((beiwagen) ? "Ja" : "Nein");
25      cout << endl << endl;
26  }
```

Aufgabe 2.c:

Schreiben Sie das Hauptprogramm Aufgabe_2_test.cpp, legen Sie pro Fahrzeugart zwei Testobjekte an und geben Sie die Daten mittels print() aus, sodass auch das gezeigte Beispiel mit Basisklassenzeigern funktioniert.

Aufgabe_2_test.cpp

```
1   #include "Aufgabe_2.h"
2
3   int main() {
4
5       // zwei testobjekte fuer alle Fahrzeuge
6       PKW vw("MK - JK 1111", 2006, 1980, 130, 2);
7       PKW audi("MK - SJ 2405", 2010, 2990, 245, 1);
8       LKW man("D - HD 8373", 1998, 12000, 8, 12.0);
9       LKW mercedes("DO - JD 4578", 2015, 16000, 6, 7.5);
10      Motorrad bmw("MK - AF 1578", 2010, 750, true);
11      Motorrad suzuki("DO - BR 7487", 2016, 1000, false);
```

Aufgabe_2_test.cpp

```
...
13      // Zeiger
14      Fahrzeug *seat = new PKW("K - KJ 1284", 2014, 1990, 150, 5);
15      Fahrzeug *daf = new LKW("HSK - FR 7844", 2003, 14500, 8, 40.0);
```

Aufgabe_2_test.cpp

```
…
17      // Ausgaben ohne Zeiger
18      vw.print();
19      audi.print();
20      man.print();
21      mercedes.print();
22      bmw.print();
23      suzuki.print();
```

Aufgabe_2_test.cpp

```
...
25      // Ausgaben mit Zeiger
26      seat->print();
27      daf->print();
28
29      delete daf;
30      delete seat;
```

Aufgabe 3

Schreiben Sie Aufgabe_3.cpp auf Exception-Handling um. Definieren Sie Fehlerklassen für negative Summen und Nulldivision, werfen Sie diese mittels throw und fangen Sie sie im Hauptprogramm mit try und catch ab.

```
…
27  class NegSum {
28      public:
29          void print() const {
30              cout << "Error: Negative Summe!" << endl;
31          }
32  };
33
34  class NulDiv {
35      public:
36          void print() const {
37              cout << "Error: Division durch Null!" << endl;
38          }
39  };
```

Vorher

```
…
6  double Formel(double x, double y, int &fehler) {
7      double sum = x + y;
8      if (sum < 0) {
9          fehler = 1;
10     } else if (sum == 0) {
11         fehler = 2;
12     } else {
13         fehler = 0;
14         return sum / sqrt(sum);
15     }
16     return 0;
17 }
```

Nachher

```
…
41 double Formel(double x, double y) {
42     double sum = x + y;
43     if (sum < 0)
44         throw NegSum();
45
46     if (sum == 0)
47         throw NulDiv();
48
49     return sum / sqrt(sum);
50 }
```

Vorher

```
...
int main() {
    int fehler;
    double x[3] = { -2, -4, 0 };
    double y[5] = { 1, 2, 0, 3, 1 };
    for (unsigned int i = 0; i < 3; ++i) {
        for (unsigned int j = 0; j < 5; ++j){
            double z = Formel(x[i], y[j], fehler);
            switch (fehler) {
            case 1:
                cout << "Error: Negative Summe" << endl; break;
            case 2:
                cout << "Error: Division durch Null" << endl; break;
            default:
                cout << "Wert : " << z << endl;
            }
        }
    }
    return 0;
}
```

Nachher

```
...
int main() {
    double x[3] = { -2, -4, 0 };
    double y[5] = { 1, 2, 0, 3, 1 };
    for (unsigned int i = 0; i < 3; ++i) {
        for (unsigned int j = 0; j < 5; ++j){
            try {
                double z = Formel(x[i], y[j]);
                cout << "Wert : " << z << endl;
            }
            catch (NegSum const &e) {
                e.print();
            }
            catch (NulDiv const &e) {
                e.print();
            }
        }
    }
    return 0;
}
```