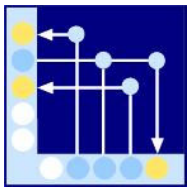
Fakultät Elektronik und Informatik
Studiengang Informatik



Programmieren in C++

Vorlesung im Wintersemester 2017/2018
Prof. Dr. habil. Christian Heinlein

4. Übungsblatt (16. November 2017)

Aufgabe 8: Abgeleitete Klassen

a) Definieren Sie Basisklassen `Person` (mit `string name`), `Male` (mit `bool bearded`) und `Female` (mit `bool pregnant`) mit sinnvollen Konstruktoren!

b) Definieren Sie abgeleitete Klassen `Man` und `Woman` als Kombination von `Person` mit `Male` bzw. `Female`, ebenfalls mit sinnvollen Konstruktoren!

c) Definieren Sie eine abgeleitete Klasse `Student` mit Basisklasse `Person` und Datenelement Matrikelnummer (`int number`) sowie `MaleStudent` und `FemaleStudent` als Kombination von `Student` mit `Man` bzw. `Woman`!
Achten Sie auf eine semantisch sinnvolle Verwendung virtueller Basisklassen!
In welcher Reihenfolge werden die Konstruktoren der einzelnen Teilobjekte ausgeführt? Welche Konstruktoraufrufe werden möglicherweise ignoriert?

d) Definieren Sie eine abgeleitete Klasse `DoubleStudent` mit einem `Person`- und zwei `Student`-Teilen zur Repräsentation von Studierenden mit zwei Studienfächern bzw. Matrikelnummern!
Objekte dieser Klasse sollen polymorph als `Student`-Objekte sowohl des ersten als auch des zweiten Studienfachs verwendbar sein, so dass ein Student, der z. B. Informatik und Elektrotechnik studiert, sowohl in einem Feld `Student* in []` (Liste aller Informatik-Studenten) als auch in einem Feld `Student* et []` (Liste aller Elektrotechnik-Studenten) mit der jeweils passenden Matrikelnummer auftreten kann, z. B.:

```
DoubleStudent* ds = new DoubleStudent("Maier", 123, 456);
Student* s1 = (Student1*)ds; in[0] = s1;
Student* s2 = (Student2*)ds; et[0] = s2;
assert(s1->number == 123);
assert(s2->number == 456);
assert((Person*)s1 == (Person*)s2);
```

e)* Definieren Sie, wenn möglich, abgeleitete Klassen `MaleDoubleStudent` und `FemaleDoubleStudent` als Kombination von `DoubleStudent` mit `MaleStudent` bzw. `FemaleStudent`!
Objekte dieser Klassen sollen genau die folgenden Teilobjekte enthalten und entsprechend polymorph verwendbar sein:

- 1 `Person`-Teilobjekt
- 1 `Male`- bzw. `Female`-Teilobjekt
- 1 `Man`- bzw. `Woman`-Teilobjekt
- 2 `Student`-Teilobjekte

- 1 `DoubleStudent`-Teilobjekt
- 2 `MaleStudent`- bzw. `FemaleStudent`-Teilobjekte
- ggf. leere Teilobjekte künstlicher Zwischenklassen

Insbesondere sollen die Objekte nicht mehr als zwei `Student`-Teilobjekte besitzen:

```
MaleDoubleStudent* mds = new MaleDoubleStudent("Maier", true, 123, 456);

DoubleStudent* ds = mds;
Student* s1 = (Student1*)ds;
Student* s2 = (Student2*)ds;

MaleStudent* ms1 = (MaleStudent1*)mds;
MaleStudent* ms2 = (MaleStudent2*)mds;
assert(s1 == (Student*)ms1);
assert(s2 == (Student*)ms2);
```

Die erste korrekte Lösung, die per E-Mail an `christian.heinlein@hs-aalen.de` gesandt wird, wird mit einer Flasche Champagner prämiert!

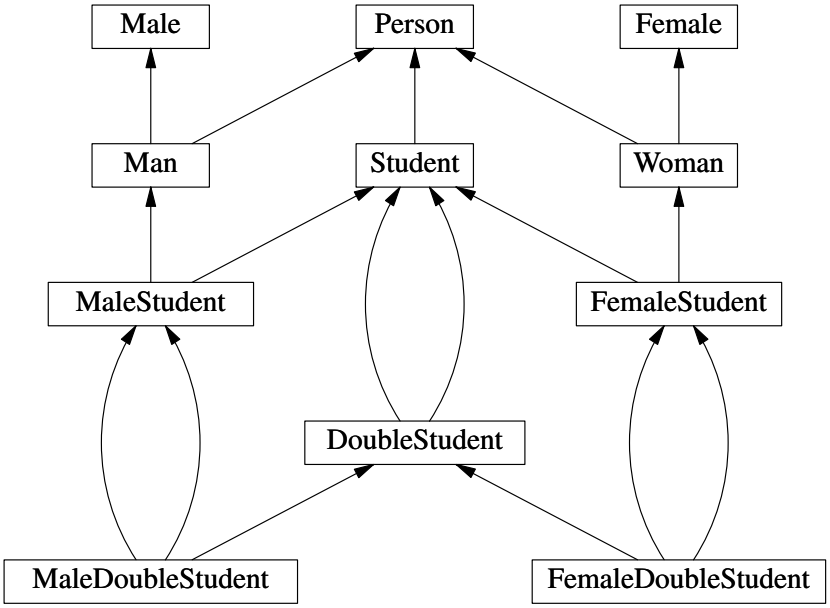
f) Erweitern Sie alle bisher definierten Klassen (außer eventuell benötigten Hilfsklassen) um ein Datenelement `long id`, das in den zugehörigen Konstruktoren initialisiert wird!
Für die Basisklassen `Person`, `Male` und `Female` soll für jedes Objekt eine global eindeutige Nummer vergeben werden, während in allen abgeleiteten Klassen die Summe der `id`-Werte der direkten Basisklassen zur Initialisierung des eigenen `id`-Werts verwendet werden soll.
Geben Sie alle `id`-Werte eines `MaleStudent`-, `FemaleStudent`- und `DoubleStudent`-Objekts aus!

g) Implementieren Sie eine Funktion

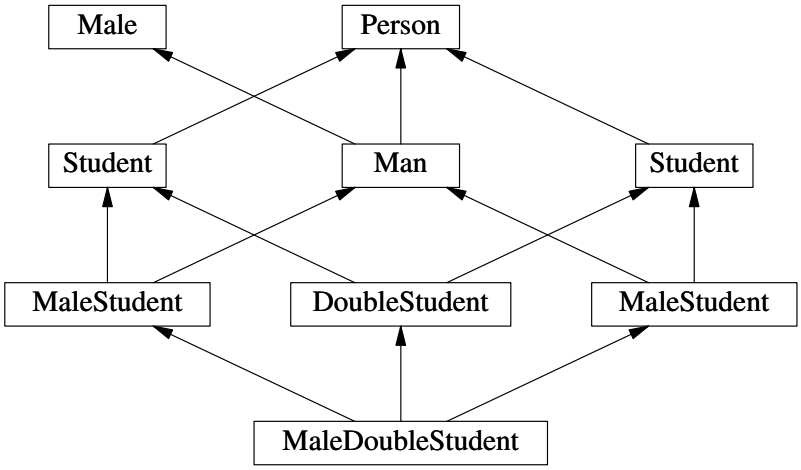
```
void print (Student* s, int d = 0);
```

die den Namen und die Matrikelnummer des Studenten `s` ausgibt!
Wenn der optionale Parameter `d` den Wert `1` bzw. `2` hat, soll die Funktion davon ausgehen, dass `s` auf das erste bzw. zweite `Student`-Teilobjekt eines `DoubleStudent`-Objekts zeigt, und zusätzlich die Matrikelnummer des anderen `Student`-Teilobjekts ausgeben.
Wie sehen entsprechende Aufrufe von `print` konkret aus? Mit welchen anderen Objekten kann `print` aufgerufen werden?

Typhierarchie (ohne künstliche Zwischenklassen)



Struktur von `MaleDoubleStudent`-Objekten



Anmerkung: Wenn man die Struktur dreidimensional auffasst, liegt die Raute mit den Typen `Person`, `Student` und `DoubleStudent` über der Raute mit den entsprechenden männlichen Typen `Man`, `MaleStudent` und `MaleDoubleStudent`.

Lösung

```
#include <string>
#include <iostream>
using namespace std;

// Nächste zu vergebende Nummer.
long nextId = 1;
```

```
struct Person {
    string name;
    long id;
    Person (const string& n) : name(n), id(nextId++) {}
};

struct Male {
    bool bearded;
    long id;
    Male (bool b) : bearded(b), id(nextId++) {}
};

struct Female {
    bool pregnant;
    long id;
    Female (bool p) : pregnant(p), id(nextId++) {}
};

struct Man : virtual Person, Male {
    long id;
    Man (const string& n, bool b)
                : Person(n), Male(b), id(Person::id + Male::id) {}
};

struct Woman : virtual Person, Female {
    long id;
    Woman (const string& n, bool p)
                : Person(n), Female(p), id(Person::id + Female::id) {}
};

struct Student : virtual Person {
    int number;
    long id;
    Student (const string& n, int m)
                : Person(n), number(m), id(Person::id) {}
};

struct MaleStudent : Student, Man {
    long id;
    MaleStudent (const string& n, bool b, int m)
                : Person(n), Student(n, m), Man(n, b)
                , id(Student::id + Man::id) {}
};

struct FemaleStudent : Student, Woman {
    long id;
    FemaleStudent (const string& n, bool p, int m)
                : Person(n), Student(n, m), Woman(n, p)
                , id(Student::id + Woman::id) {}
};
```

```
struct Student1 : Student {
    Student1 (const string& n, int m) : Person(n), Student(n, m) {}
};

struct Student2 : Student {
    Student2 (const string& n, int m) : Person(n), Student(n, m) {}
};

struct DoubleStudent : Student1, Student2 {
    long id;
    DoubleStudent (const string& n, int m1, int m2)
                : Person(n), Student1(n, m1), Student2(n, m2)
                , id(Student1::id + Student2::id) {}
};

void print (Student* s, int d = 0) {
    cout << "Name: " << s->name << endl;
    cout << "Matrikelnummer: " << s->number << endl;

    switch (d) {
    case 1:
        cout << "Andere Matrikelnummer: "
          << ((DoubleStudent*)(Student1*)s)->Student2::number << endl;
        break;
    case 2:
        cout << "Andere Matrikelnummer: "
          << ((DoubleStudent*)(Student2*)s)->Student1::number << endl;
        break;
    }
}

int main () {
    MaleStudent ms("Maier", true, 123);
    print(&ms);
    cout << ms.Person::id << endl;
    cout << ms.Male::id << endl;
    cout << ms.Man::id << endl;
    cout << ms.Student::id << endl;
    cout << ms.id << endl;

    FemaleStudent fs("Müller", true, 456);
    print(&fs);
    cout << fs.Person::id << endl;
    cout << fs.Female::id << endl;
    cout << fs.Woman::id << endl;
    cout << fs.Student::id << endl;
    cout << fs.id << endl;
```

```
    DoubleStudent dds("Schulze", 123, 456);
    print((Student1*)&dds, 1);
    print((Student2*)&dds, 2);
    cout << dds.Person::id << endl;
    cout << dds.Student1::id << endl;
    cout << dds.Student2::id << endl;
    cout << dds.id << endl;
}
```

Lösungsversuch zu Teilaufgabe e

Annahme: `Man` ist eine virtuelle Basisklasse von `MaleStudent`.

```
struct MaleStudent1 : MaleStudent {
    MaleStudent1 (const string& n, bool b, int m)
                    : Person(n), Man(n, b), MaleStudent(n, b, m) {}
};

struct MaleStudent2 : MaleStudent {
    MaleStudent2 (const string& n, bool b, int m)
                    : Person(n), Man(n, b), MaleStudent(n, b, m) {}
};

struct MaleDoubleStudent : DoubleStudent, MaleStudent1, MaleStudent2 {
    MaleDoubleStudent (const string& n, bool b, int m1, int m2)
                : Person(n), Man(n, b), DoubleStudent(n, m1, m2)
                , MaleStudent1(n, b, m1), MaleStudent2(n, b, m2) {}
};

// Analog für FemaleDoubleStudent.
```

Fehler: Ein so definiertes `MaleDoubleStudent`-Objekt enthält *vier* `Student`-Teilobjekte.