ÜBUNG 11.1 VIRTUELLE METHODEN

Betrachten Sie folgende Hierarchie von drei Klassen.

```
class A {
public:
    void a();
    virtual void va() = 0;
    virtual void a(int n);
private:
    void c();
};
```

```
class B: public A {
public:
    void b();
    virtual void vb();
    void a(double d);
    void a(int i);
    virtual void va();
};
```

```
class C: public B {
public:
    virtual void c();
    void a(float);
    virtual void a();
    virtual void va();
};
```

Eine Implementierung aller deklarierten Methoden existiert. Was passiert, wenn Sie den folgenden Programmteil compilieren wollen? Welche Ausdrücke sind nicht korrekt, und warum?

```
A a;  B b;   C c;  A* pa=&b;  B* pb=&c; float x = 1.2;
pa->a();  pa->va();  pa->a(1);  pa->c();  pa->b();  pa->vb();  pa->a(x);
pb->a();  pb->va();  pb->a(1);  pb->c();  pb->b();  pb->vb();  pb->a(x);
pa = &c;
pa->a();  pa->va();  pa->a(1);  pa->c();  pa->b();  pa->vb();  pa->a(x);
```

Wenn Sie die falschen Ausdrücke entfernt haben und das Programm ausführen, von welcher Klasse werden dann die Methoden aufgerufen? Geben Sie für jeden Methodenaufruf an, ob die Methodenimplementierung von Klasse `A`, `B` oder `C` verwendet wird. *5 Punkte*

ÜBUNG 11.2 STREICHELZOO

Häufig wird Vererbung dazu eingesetzt, eine "ist ein"-Beziehung zwischen verschiedenen Klassen darzustellen. Funktionen, die mit vielen verschiedenen Klassen funktionieren sollen, werden für eine abstrakte Basisklasse (Schnittstelle) definiert und anschließend mit Objekten benutzt, deren Klassen von dieser Basisklasse abgeleitet wurden.

Ziel dieser Aufgabe ist die Darstellung der Vorteile dieser Trennung in Schnittstelle und Implementierung an einem konkreten Beispiel. Stellen Sie sich einen Zoo vor, in dem es eine Vielzahl verschiedener Tiere gibt. Für jedes Tier soll vereinfachend gelten, dass es sich um ein Säugetier, einen Vogel oder einen Fisch handelt. Desweiteren

- ist es den Besuchern möglich, einen Teil der Tiere zu füttern (z.B. Ziegen, Gänse, Goldfische), andere nicht (Pinguine, Bären, Haie).
- ist es den Besuchern möglich, einen Teil der Säugetiere zu streicheln (Ziegen), andere nicht (Bären). Vögel und Fische können nicht gestreichelt werden.
- machen Säugetiere und Vögel charakteristische Geräusche. Wir nehmen an, dass Fische stets stumm sind.
- haben alle Tiere (ja, auch die Goldfische!) einen Spitznamen, der ihnen im Konstruktor übergeben wird.

Implementieren Sie eine Klassenhierarchie, so dass es möglich ist, die Funktionen

- `void fuettern ( Tier& tier )`
- `void streicheln ( Saeugetier& saeugetier )`
- `void geraeusch ( Saeugetier& saeugetier )`
- `void geraeusch ( Vogel& vogel )`

zu nutzen. Legen Sie außerdem mindestens zehn verschiedene Tierarten an, davon mindestens zwei aus jeder der drei Kategorien. Ein Aufruf der oben genannten Funktionen soll das Tier fragen, ob die gewünschte Operation möglich ist, und dann Ausgaben z.B. der folgenden Form erzeugen:

```
Die Ziege Billy lässt sich füttern.
Der Bär Bruno möchte nicht gestreichelt werden!
Die Gans Herta schnattert.
```

Da die Eigenschaften der Tiere für eine ganze Art gelten und somit von der Instanz unabhängig sind (alle Ziegen lassen sich füttern, nicht nur Billy), können Sie sie mit dem Schlüsselwort "`static`" als Klassenvariable deklarieren:

```
static const bool fuetterbar = <Wert>;
```

Dadurch wird die Variable nicht für jede Instanz erneut angelegt, außerdem wird deutlich, dass es sich um eine Eigenschaft handelt, die alle Tiere dieser Art teilen.

Benutzen Sie für diese Aufgabe abstrakte Basisklassen und rein virtuelle Methoden. Sorgen Sie dafür, dass die einzelnen Tierarten genau jene Variablen und Methoden erhalten, die sie zum Funktionieren benötigen, z.B. sollte es kein `bool streichelbar` für Fische geben. Geben Sie zusätzlich zu Ihrem Programm ein Diagramm ab, in dem Sie die durch die Vererbung gegebene Baumstruktur darstellen.

10 Punkte

Übung 11.3 Polynomschablone

In der Vorlesung (Kapitel 9) wurde die Klasse `Polynomial` durch Vererbung aus `SimpleFloatArray` erzeugt. Man kann diese Konstruktion durch die Verwendung von Templates vom unterliegenden Datentyp `float` lösen und dadurch flexibler machen; dadurch wird es zum Beispiel möglich `double`-Arithmetik zu verwenden. Grundlage der neuen Hierarchie ist die Template-Klasse `SimpleArray` aus Kapitel 11 der Vorlesung. Davon kann man eine templatisierte Klasse `Polynomial` ableiten. Die Klassendefinition sieht dann so aus:

```
template <class T>
class Polynomial :
    public SimpleArray<T> {
public:
    Polynomial (int n);
    // ...
};
```

Implementieren Sie die Klasse, indem Sie sich die entsprechenden Dateien für die Klassen `Polynomial` und `SimpleArray` von der Vorlesungs-Website auf Ihren Computer laden und diese dann verändern. Die Klasse `Polynomial` soll im Anschluss eine Schablone sein, die für verschiedene unterliegende numerische Datentypen funktioniert. Testen Sie Ihr Programm für mindestens zwei verschiedene Datentypen, z.B. `double` und `float`, wie in diesem Programmfragment:

```
Polynomial<float>  p(2);
p[0] = 1.0;
p[1] = 1.0;
p.print();
p = p*p;
p.print();

Polynomial<double> q(3);
q[0] = 2.0;
q[1] = -1.0;
q[3] = 4.0;
q.print();
```

5 Punkte