# Objektorientiertes Programmieren - SWB2 & TIB2
# Labor 5

## Aufgabe 1: Methoden - virtuell oder nicht virtuell?

Passen Sie das nachfolgende Programm so an, dass die gegebene Ausgabe erzeugt wird. Verändern Sie dabei das Hauptprogramm *nicht*.

Ausgabe:

```
A::f1()
B::f1()
C::f1()
D::f2()
D::f2()
D::f2()
A::f3()
B::f3()
D::f3()
A::f4() D::f2()
C::f4() D::f2()
C::f4() D::f2()
```

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

void p(string s, bool nl = true) {
    cout << s << " ";
    if (nl) { cout << endl; }
}

class A {
public:
    void f1() { p("A::f1()"); }
    void f2() { p("A::f2()"); }
    void f3() { p("A::f3()"); }
    void f4() { p("A::f4()", false); f2(); }
};

class B : public A {
public:
    void f1() { p("B::f1()"); }
    void f2() { p("B::f2()"); }
    void f3() { p("B::f3()"); }
    void f4() { p("B::f4()", false); f2(); }
};
```

```
class C : public B {
public:
    void f1() { p("C::f1()"); }
    void f2() { p("C::f2()"); }
    void f3() { p("C::f3()"); }
    void f4() { p("C::f4()", false); f2(); }
};

class D : public C {
public:
    void f1() { p("D::f1()"); }
    void f2() { p("D::f2()"); }
    void f3() { p("D::f3()"); }
};

int main() {
    D d;
    A * aptr = &d;
    B * bptr = &d;
    C * cptr = &d;
    aptr->f1();
    bptr->f1();
    cptr->f1();
    aptr->f2();
    bptr->f2();
    cptr->f2();
    aptr->f3();
    bptr->f3();
    cptr->f3();
    aptr->f4();
    bptr->f4();
    cptr->f4();
}
```

## Aufgabe 2: Virtuelle Methoden in der Bibliothek

Vereinfachen Sie Ihr Programm für eine Bibliothek (Labor 4, Aufgabe 1) durch den Gebrauch von virtuellen Methoden. Entfernen Sie dabei auch die nun unnötige Instanzvariable `typ` in der Klasse `Medium`.

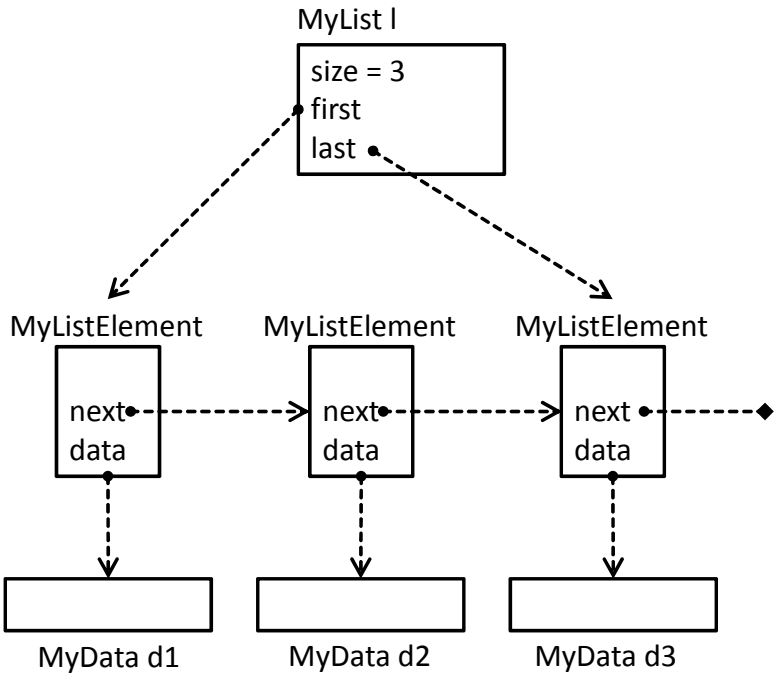
Ersetzen Sie, wenn möglich, die beiden Methoden **`void`** `mediumBeschaffen(Buch &)` und **`void`** `mediumBeschaffen(DVD &)` aus der Klasse `Bibliothek` durch eine Methode **`void`** `mediumBeschaffen(Medium &)`.

## Aufgabe 3: Eine eigene Klasse für einfach verkettete Listen

In Hausaufgabe 5 haben Sie ein Programm geschrieben, um fast beliebige Datentypen in einem Vektor speichern zu können. In Labor 2 haben Sie einen Linienzug (Klasse `Polygonline`) als eine verkettete Liste realisiert. Das war eine Speziallösung. Daher sollen Sie hier eine Klasse `MyList` schreiben, die ähnlich zu der Klasse `MyVector` jegliche Objekte der Klasse `MyData`

speichern kann.

Die folgende Skizze zeigt schematisch die Datenstruktur. Ein Objekt der Klasse `MyList` hat zwei Zeiger (`first` bzw. `last`), die auf den Anfang bzw. das Ende der verketteten Liste verweisen. Die Elemente der verketteten Liste sind Objekte der Klasse `MyListElement`. Diese zeigen jeweils mit dem Zeiger `next` auf den Nachfolger in der Liste. Der Zeiger `data` verweist auf ein Objekt der Klasse `MyData`. Die Objekte der Klasse `MyData` sind die eigentlichen Daten, die wir speichern wollen.




Kopieren Sie die Klasse `MyData` aus dem `MyVector`-Projekt aus der Hausaufgabe. Schreiben Sie dann die Klassen `MyList` und `MyListElement`. In diesem Fall soll aber die Klasse `MyListElement` innerhalb der Klasse `MyList` deklariert werden.

Ergänzen Sie dann die Klasse `MyList` um die folgenden zusätzlich zu den oben genannten Eigenschaften:

a) Die Methode `void push_back(const MyData &)` hängt ein Element am Ende der Liste an.
b) Die Methode `void pop_back()` löscht das Element am Ende der Liste.
c) Die Methode `front()` liefert das erste und die Methode `back()` das letzte Element zurück.
d) Mit der Methode `clear()` wird die Liste geleert.
e) Die Methode `empty()` prüft, ob die Liste leer ist.
f) Die Methode `size()` liefert die Anzahl der Elemente in der Liste zurück.
g) Ein eigener Zuweisungsoperator `MyList operator=(const MyList &)` und ein Konkatenationsoperator `MyList operator+(const MyList &)` sind nützlich.

Zum Testen Ihrer Klasse `MyList` nutzen Sie Ihre Klassen `Point` und `Circle`. Nutzen Sie dazu das folgende Hauptprogramm.

```
#include <iostream>
#include "MyList.hpp"
```

```
#include "Circle.hpp"
using namespace std;

int main() {
    Point p1(1,1);
    Point p2(2,2);
    Point p3(3,3);
    Point p4(4,4);
    cout << "Liste v1 erstellen ..." << endl;
    MyList v1;
    v1.push_back(p1);
    v1.push_back(p2);
    v1.push_back(p3);
    cout << "Liste v1 ausgeben ..." << endl;
    v1.print();
    cout << "Liste v1 in v2 kopieren ..." << endl;
    MyList v2(v1);
    cout << "Liste v2 ausgeben ..." << endl;
    v2.print();
    cout << "Punkt am Beginn der Liste v2 verschieben ..." << endl;
    dynamic_cast<Point&>(v2.front()).move(10,10);
    cout << "Liste v1 ausgeben ..." << endl;
    v1.print();
    cout << "Liste v2 ausgeben ..." << endl;
    v2.print();
    cout << "Groesse von v1: " << v1.size() << endl;
    if (!v1.empty()) {
        v1.clear();
    }
    cout << "Groesse von v1: " << v1.size() << endl;
    v1.print();
    v2.print();
    cout << "Groesse von v2: " << v2.size() << endl;
    v2.push_back(p1);
    cout << "Groesse von v2: " << v2.size() << endl;
    v2.print();
    cout << "Punkt (4,4) hinten an v1 anhaengen ..." << endl;
    v1.push_back(p4);
    cout << "Liste v1 an v2 anhaengen ..." << endl;
    v2 = v2+v1;
    cout << "Liste v1 ausgeben ..." << endl;
    v1.print();
    cout << "Liste v2 ausgeben ..." << endl;
    v2.print();
    cout << "Liste mit Kreisen ..." << endl;
    Circle c1(p1, 1);
    Circle c2(p2, 2);
    MyList v3;
```

```
    v3.push_back(c1);
    v3.push_back(c2);
    v3.print();
    return 0;
}
```

## Aufgabe 4: Grafische Objekte - Teil 5 - Polymorphie

Diese Aufgabe des Labors wird am sechsten Labortermin als Teil einer Gesamtaufgabe testiert. Sie müssen im folgenden Labor die Lösung von diesem Labor erweitern.

Erweitern Sie Ihr Programm für grafische Objekte von Labor 4 auf folgende Weise:

a) Machen Sie die Klasse `DrawingObject` zu einer abstrakten Klasse, indem Sie die Funktion `void print(bool=false) const` als rein virtuelle Funktion deklarieren.

b) Führen Sie die Klasse `Rectangle` als abgeleitete Klasse der Klasse `OneDimObject` ein. Die Klasse soll eine Komposition von zwei Punkten sein. Schreiben Sie einen Konstruktor, der zwei diagonal gegenüber liegende Punkte des Rechtecks als Parameter nimmt und ansonsten mit den Punkten (0,0) und (1,1) als Defaultwerte arbeitet.

c) Schreiben Sie für die Klasse `Rectangle` die Instanzfunktion `print(bool) const`, die zwei gespeicherten Eckpunkte des Rechtecks in der Form `[(x1,y1), (x2,y2)]` am Bildschirm ausgibt.

d) Ergänzen Sie das unten gegebene Hauptprogramm. Lesen Sie wie in Labor 3 Aufgabe 3 die Objekte von der Tastatur ein. Erkennen Sie anhand des eingegebenen Strings, was für ein Objekt eingegeben wurde und rufen Sie dann den entsprechenden Konstruktor auf. Zum Schluss geben Sie mit `print` alle Objekte auf der Konsole aus.

```
int main()
{
    DrawingObject * objects[20];
    int anzahl = 0;
    cout << "Wieviele Objekte wollen Sie einlesen?" << endl;
    cout << "Anzahl: ";
    cin >> anzahl;
    // Objekte einlesen
    ....
    for (int i=0; i<anzahl; i++)
    {
        // einzelnes Objekt einlesen
        ....
        objects[i]= ....
    }
    // Objekte ausgeben
    for (int i=0; i<anzahl; i++)
    {
        // einzelnes Objekt ausgeben
        ....
    }
    return 0;
}
```