Übungen OOSWE/Progr. 2 (C++) – KE 11

Der C/C++-Coding Styleguide ist einzuhalten.
Folgende Einstellungen sind für Debug und Release (All Configurations) vorzunehmen:

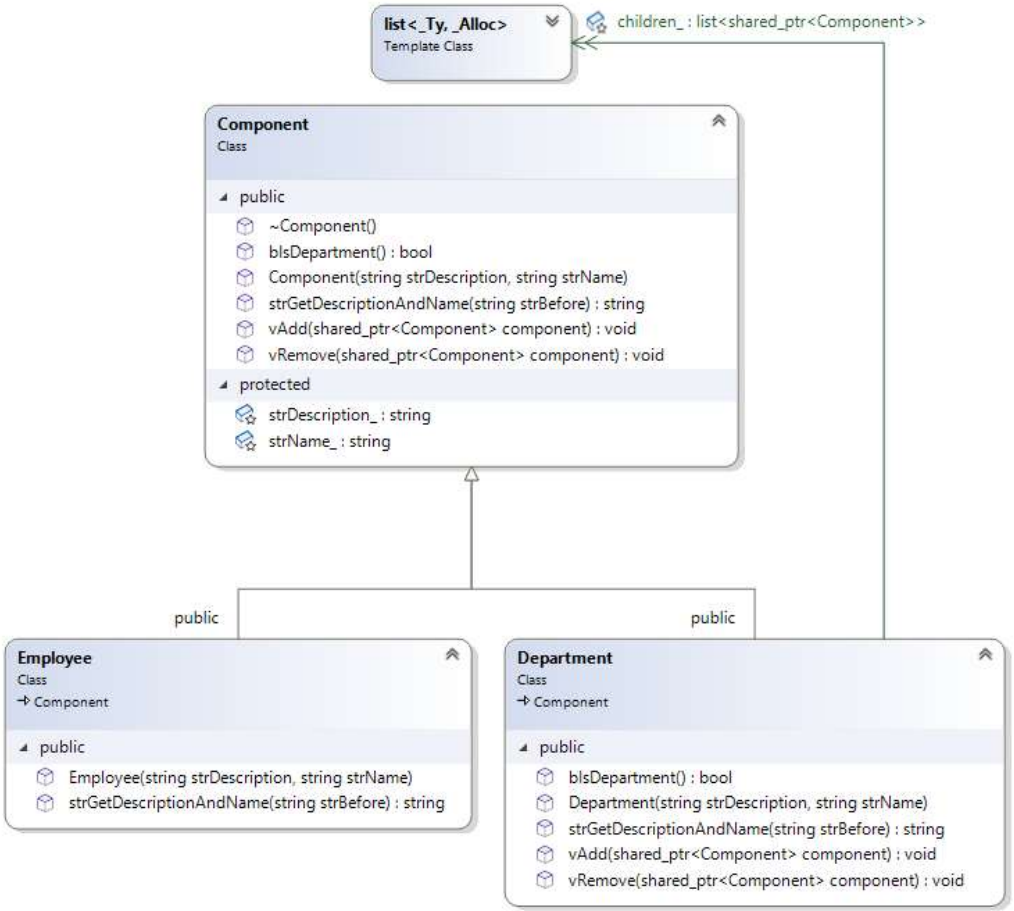
Einstellung	**Wert**
Solution Platform	x86
Properties->Conf. Properties->C/C++->General->Warning Level	Level4 (/W4)
Properties->Conf. Properties->C/C++->General->Treat Warnings As Errors	Yes (/WX)
Properties->Conf. Properties->C/C++->General->SDL checks	Yes (/sdl)
Properties->Conf. Properties->C/C++->Code Generation->Basic Runtime Checks	Default
Properties->Conf. Properties->C/C++->Code Generation->Security Checks	Enable Security Checks (/GS)
Properties->Conf. Properties->C/C++->Language->C++ Language Standard	ISO C++ 20 Standard

Die geforderten Kommentare sind in Englisch zu hinterlegen.

Aufgabe 1: Composite Pattern

Arbeiten Sie sich in die Thematik Composite Pattern ein. Das Composite Pattern wird häufig auch als „große Schwester“ des Decorator Pattern bezeichnet. Mit dem Pattern lassen sich Strukturen aufbauen, die Teil-Ganzes-Hierarchien enthalten. Es sind darin primitive Objekte als auch Kompositionen enthalten, die wiederum primitive Objekte und Kompositionen enthalten (rekursiv). In dieser Aufgabe soll eine Firma mit Abteilungen, Unterabteilungen und Teams abgebildet werden.

Das zu erstellende C++ Programm enthält nur das Composite Pattern – weitere Klassen sind hier nicht notwendig. **Department** repräsentiert hier das **Composite**. **Employee** wäre das **Leaf.**



Übungen OOSWE/Progr. 2 (C++) – KE 11

1.1 Legen Sie ein neues Projekt an. Deklarieren und implementieren Sie eine Funktion `static void vClientCode (void)`, welche von main aufgerufen wird.

```
#include <iostream>
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>
#include "Composite.h"

static void vClientCode(void);

int main(void)
{
   vClientCode();
   _CrtDumpMemoryLeaks();

   return 0;
}

static void vClientCode(void)
{
   // Creation of an hierarchical structure of HS OG
   // rectors, faculties, study programs, deans
}
```

1.2 Das komplette Composite Pattern ist nur in einer h-Datei zu realisieren (Composite.h). Dies erleichtert den Einstieg deutlich – in der Praxis würde dieses Pattern allerdings in verschiedene cpp- und h-Dateien aufgeteilt werden. Alle Klassen des Composite Pattern befinden sich im namespace MyDesignPattern.

Kern des Patterns bildet die abstrakte Klasse Component.

```
class Component
{
public:
   Component(std::string strDescription, std::string strName) :
             strDescription_(strDescription), strName_(strName)
   {
   }

   virtual ~Component()
   {
   }
   virtual void vAdd(std::shared_ptr <MyDesignPattern::Component> component)
   {
   }
   virtual void vRemove(std::shared_ptr <MyDesignPattern::Component> component)
   {
   }
   virtual bool bIsDepartment() const
   {
      return false;
   }

   virtual std::string strGetDescriptionAndName(std::string strBefore) const = 0;

protected:
   std::string strName_;
   std::string strDescription_;
};
```

Die Klasse Component enthält vier virtuelle Funktionen und eine pure virtual Funktion.

Machen Sie sich nochmals die Folgen klar, falls eine abgeleitete Klasse eine virtuelle und eine pure virtuelle Funktion **nicht** überschreibt. Schreiben Sie die Antwort als Kommentar über main.

Die konkreten Klassen Department und Employee bekommen im Konstruktor zwei Strings (strDescription und strName) übergeben und **leiten diese an den Konstruktor von Component weiter.** Ein Konstruktor der Kindklasse ruft immer nur automatisch den leeren Konstruktor der Elternklasse auf. Soll ein anderer Konstruktor der Elternklasse aufgerufen werden, so muss dies explizit so implementiert werden (Vergisst man schnell!).

```
Department(std::string strDescription, std::string strName)
           : Component(strDescription, strName)
{
}
```

Department enthält noch zusätzlich eine Liste mit shared Component-Zeigern (children_). Mit vAdd und vRemove können der Liste shared Component-Zeiger hinzugefügt oder weggenommen werden. Wählen Sie die passenden Funktionen von list aus.

Die überschriebene Funktion bIsDepartment liefert für ein Objekt der Klasse Department ein true zurück. Nur in diesem Fall steht die Liste children_ zur Verfügung.

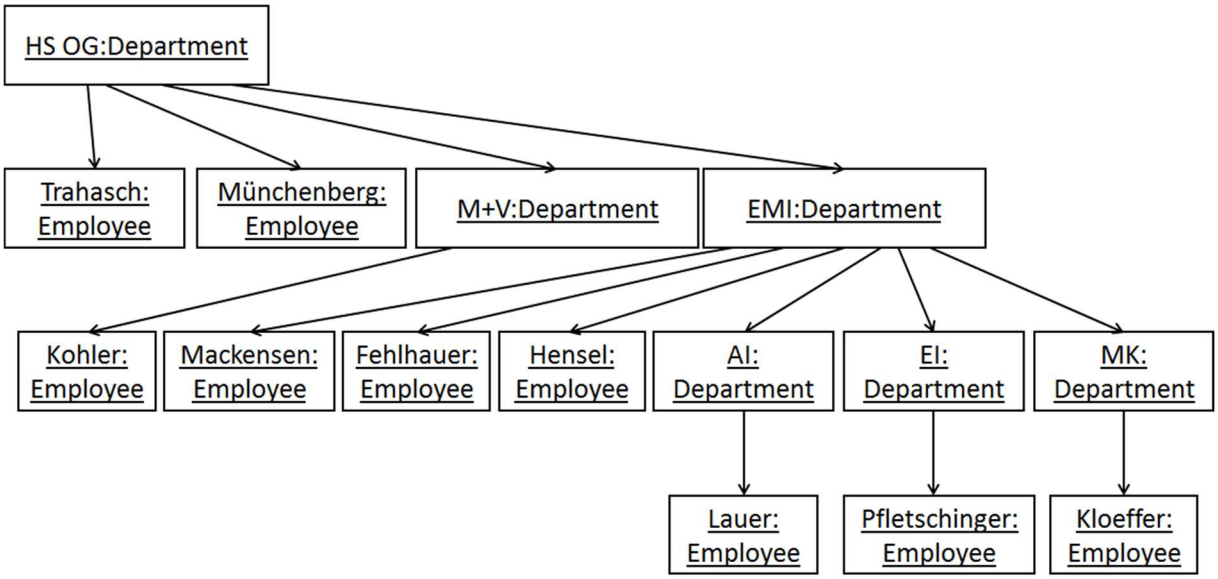
Die wichtigste Funktion stellt die pure virtual Funktion strGetDescriptionAndName dar. Bei Employee sieht die Implementierung wie folgt aus:

```
std::string strGetDescriptionAndName(std::string strBefore) const override
{
   return (strBefore + strDescription_ + "-" + strName_ + "\n");
}
```

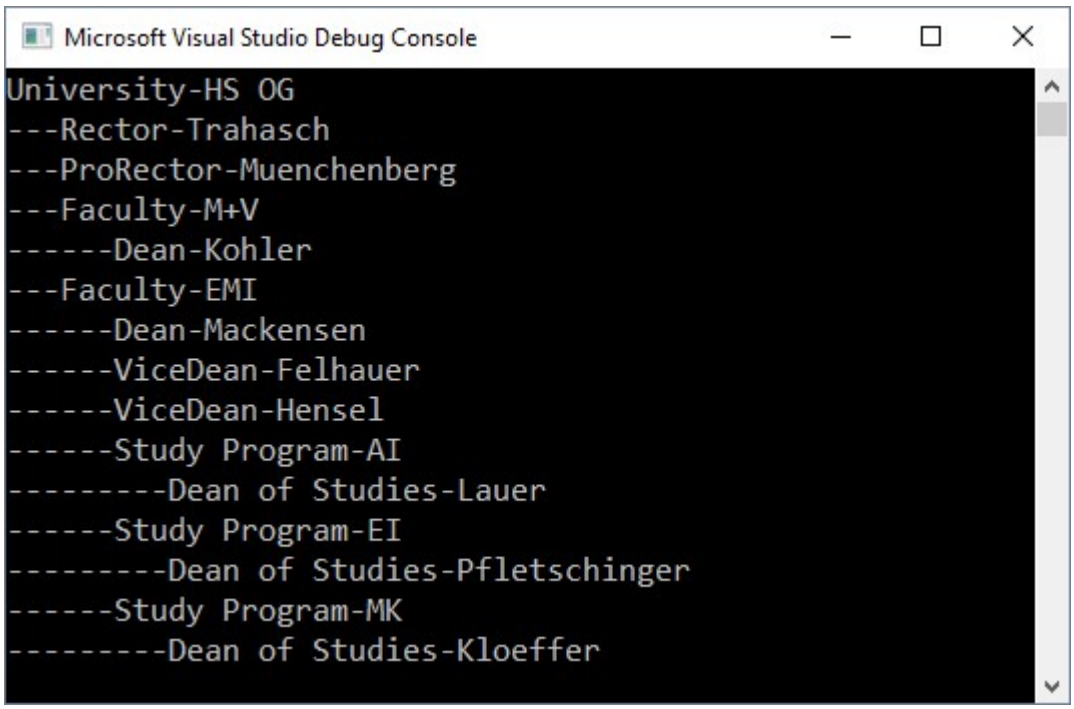
strBefore soll ein String sein, der am Anfang des zurückgegebenen Resultstrings steht (siehe später).

Bei Department sieht dies Funktion etwas komplexer aus. Dort ist noch zusätzlich über die Liste zu iterieren und deren strGetDescriptionAndName Funktionen sind aufzurufen. Da sich diese um eine Hierarchie noch unten versetzt befinden, ist dem strBefore noch ein konstanter String (z.B. „---“) hinzuzufügen.

1.3 Legen Sie sich in Clientcode die folgende Objektstruktur in Clientcode an.



Rufen Sie danach über den Zeiger auf HS OG die Funktion strGetDescriptionAndName auf und geben Sie das Ergebnis auf dem Bildschirm aus. Dabei ist ein leerer String zu übergeben. Das Ergebnis sieht dann wie folgt aus.



Da überall shared Pointer verwendet wurden, dürften keine Memory Leaks entstehen.