

✍ Übungen OOSWE/Progr. 2 (C++) – KE 12

Der C/C++-Coding Styleguide ist einzuhalten.

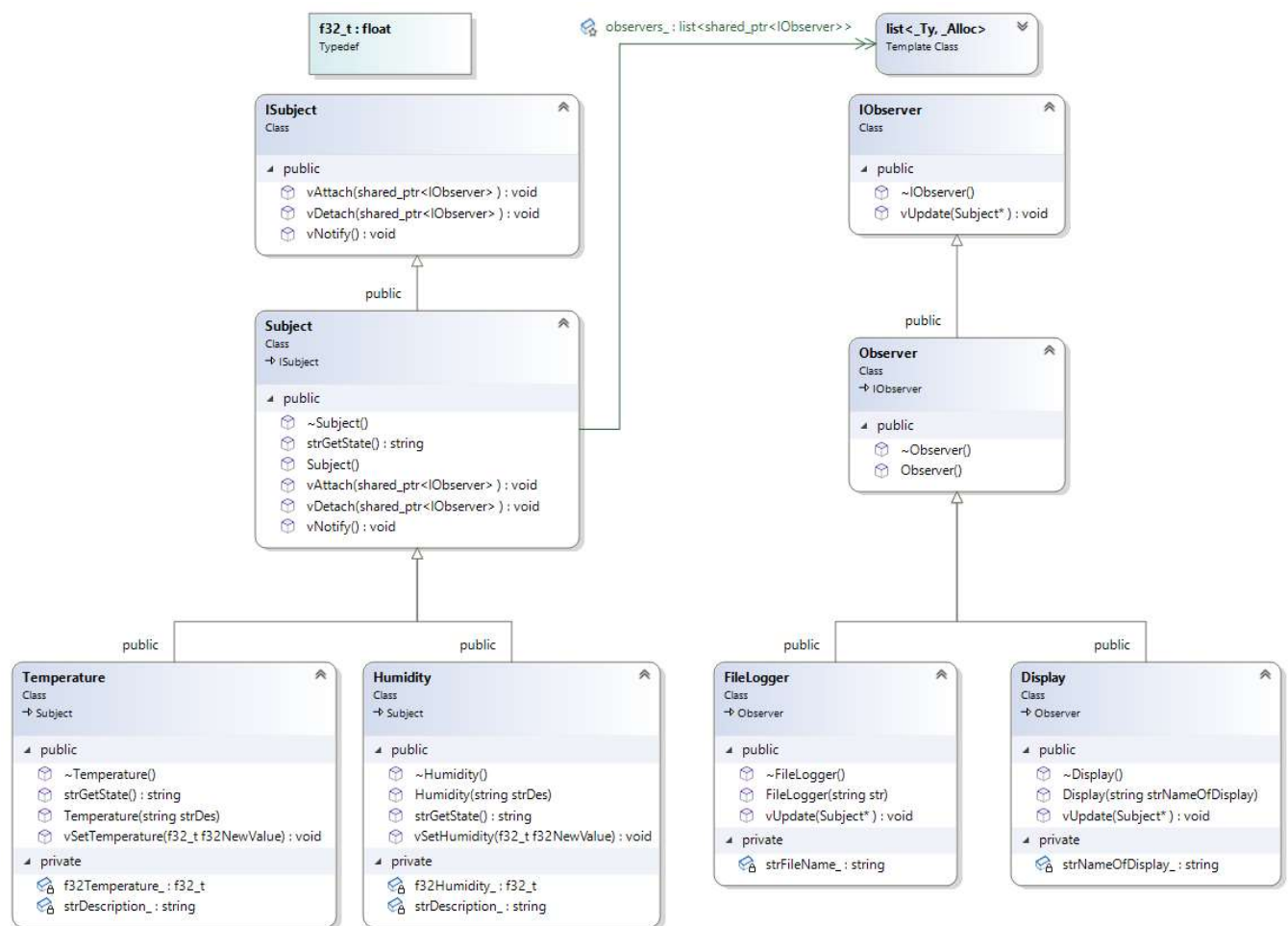
Folgende Einstellungen sind für Debug und Release (All Configurations) vorzunehmen:

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

Aufgabe 1: Observer Pattern

Arbeiten Sie sich in die Thematik des Observer Pattern ein.

Das zu erstellende C++ Programm enthält nur das Observer Pattern (Pull-Variante) – weitere Klassen sind hier nicht notwendig. ISubject und IObserver sind Interfaces. Subject und Observer sind dabei abstrakte Klassen. Die Kindklassen von Subject sollen dabei die Klassen Temperature und Humidity (Luftfeuchtigkeit) sein. Kindklassen von Observer sind Display und FileLogger.



Übungen OOSWE/Progr. 2 (C++) – KE 12

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 "Temperature.h"
#include "Humidity.h"
#include "Display.h"
#include "FileLogger.h"

static void vClientCode(void);

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

    return 0;
}

static void vClientCode(void)
{
    // Creation of Temperature, Humidity, FileLogger and Display-Objects
    // shared Pointers!!!
}
```

Übungen OOSWE/Progr. 2 (C++) – KE 12

1.2 Implementieren Sie das Pattern beginnend mit den Interfaces ISubject und IObservable sowie den abstrakten Klassen Subject und Observer. ISubject und Subject sowie IObservable und Observer können in einer *.h-/*.cpp-Datei deklariert/implementiert werden. Namespace ist MyDesignPattern.

Subject.h

```
#pragma once
#include <memory>
#include <string>
#include <list>
#include "Observer.h"
typedef float f32_t;

namespace MyDesignPattern
{
    class ISubject
    {
    public:
        virtual void vAttach(std::shared_ptr <MyDesignPattern::IObservable>) = 0;
        virtual void vDetach(std::shared_ptr <MyDesignPattern::IObservable>) = 0;
        virtual void vNotify() = 0;
    };
    class Subject : public ISubject
    {
    public:
        Subject() = default;
        virtual ~Subject() = default;
        virtual std::string strGetState() = 0;
        void vAttach(std::shared_ptr <MyDesignPattern::IObservable>);
        void vDetach(std::shared_ptr <MyDesignPattern::IObservable>);
        void vNotify();
    protected:
        std::list<std::shared_ptr <MyDesignPattern::IObservable>> observers_;
    };
}
```

Auch hier kommt eine Liste mit Shared-Pointern zur Anwendung.

Observer.h:

```
#pragma once
#include <iostream>

namespace MyDesignPattern
{
    class Subject;
    class IObservable
    {
    public:
        virtual ~IObservable() = default;
        virtual void vUpdate(Subject*) = 0;
    };
    class Observer : public IObservable
    {
    public:
        Observer() = default;
        virtual ~Observer() = default;
    };
}
```

Übungen OOSWE/Progr. 2 (C++) – KE 12

1.3 Realisieren Sie die konkreten Klassen Humidity sowie Temperature. Beide Klassen erhalten über den Konstruktor die strDescription_ sowie einen initialen Wert (fHumidity_ oder fTemperature_).

Implementieren Sie danach die Klassen Display und FileLogger. FileLogger erhält im Konstruktor den Namen der Logdatei übergeben.

FileLogger und Display erhalten über die Update-Methode einen Shared-Pointer auf ein Subject. Dessen Getstate-Methode wird aufgerufen (Pull-Variante). Der erhaltene String soll bei Display nur auf der Konsole ausgegeben werden. Bei FileLogger soll dieser in einer Logdatei abgespeichert werden. Konstruktoren als implizite Inline-Funktionen ☺ (Wie macht man dies?)

```
#include <iostream>
#include <ios>
#include <fstream>
#include "FileLogger.h"
#include "Subject.h"

void MyDesignPattern::FileLogger::vUpdate(Subject* pSubject)
{
    std::ofstream filelogger (strFileName_, std::ios_base::out | std::ios_base::app);
    filelogger << pSubject->strGetState() << "\n";
}
```

1.4 Instanziiieren Sie sich zwei Temperature und ein Humidity Objekt in vTestClient (Smart Pointers). Instanziiieren Sie zwei Display- und zwei FileLogger-Objekte (Smart Pointers).

„Attachen“ sie jetzt Observer zu den Subjects. Gesamter Code unten!

Sobald Sie in den Subjects vSetTemperature oder vSetHumidity aufrufen, werden automatisch alle Observer über die Methode Notify (ruft alle Update-Methoden der „attachen“ Observer auf) aufgerufen. Diese erhalten einen Smartpointer auf das Subject und holen sich dann die Daten (hier nur ein String) vom Subject (Pull-Variante).

```
static void vClientCode(void)
{
    // Three Subjects
    std::shared_ptr<MyDesignPattern::Temperature> pTemp1 =
        std::make_shared<MyDesignPattern::Temperature>("Temp1");

    std::shared_ptr<MyDesignPattern::Temperature> pTemp2 =
        std::make_shared<MyDesignPattern::Temperature>("Temp2");

    std::shared_ptr<MyDesignPattern::Humidity> pHumidity =
        std::make_shared<MyDesignPattern::Humidity>("Humidity bath");

    // Four Observers
    std::shared_ptr<MyDesignPattern::Display> pDisplay1 =
        std::make_shared<MyDesignPattern::Display>("Display1");

    std::shared_ptr<MyDesignPattern::Display> pDisplay2 =
        std::make_shared<MyDesignPattern::Display>("Display2");

    std::shared_ptr<MyDesignPattern::FileLogger> pLoggerHum =
        std::make_shared<MyDesignPattern::FileLogger>("Humidity.txt");
    std::shared_ptr<MyDesignPattern::FileLogger> pLoggerTemp =
```

Übungen OOSWE/Progr. 2 (C++) – KE 12

```
std::make_shared<MyDesignPattern::FileLogger>("Temp1.txt");

pTemp1->vAttach(pLoggerTemp);
pTemp1->vAttach(pDisplay1);
pTemp1->vAttach(pDisplay2);

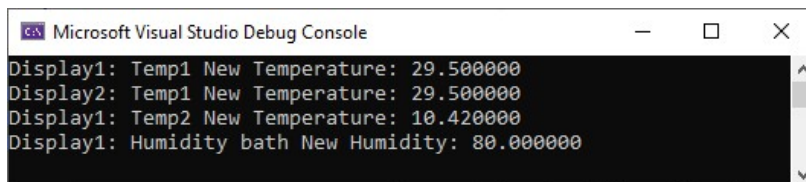
pTemp2->vAttach(pDisplay1);

pHumidity->vAttach(pDisplay1);
pHumidity->vAttach(pLoggerHum);

pTemp1->vSetTemperature(29.5F);

pTemp2->vSetTemperature(10.42F);

pHumidity->vSetHumidity(80.0);
}
```



```
Microsoft Visual Studio Debug Console
Display1: Temp1 New Temperature: 29.500000
Display2: Temp1 New Temperature: 29.500000
Display1: Temp2 New Temperature: 10.420000
Display1: Humidity bath New Humidity: 80.000000
```



```
*Humidity - Editor
Datei Bearbeiten Format Ansicht Hilfe
Humidity bath New Humidity: 80.000000
Zeile 1, Spalte 1 100% Windows (CRLF) UTF-8
```



```
*Temp1 - Editor
Datei Bearbeiten Format Ansicht Hilfe
Temp1 New Temperature: 29.500000
Zeile 2, Spalte 1 100% Windows (CRLF) UTF-8
```

Hinweis: Die Push-Variante des Observer Pattern würde die Daten direkt an den Observer liefern.

Selbstverständlich könnte man in die FileLogger noch Datum und Uhrzeit verwenden.

Es müssten keine Memory-Leaks nach der Ausführung erscheinen.

pTemp1, pTemp2 und pHumidity können beliebig andere Werte annehmen. Auf dem Bildschirm sowie in den Dateien stehen dann immer die Werte. Probieren Sie dies aus!

Übungen OOSWE/Progr. 2 (C++) – KE 12

Aufgabe 2: GoogleTest

Verwenden Sie nun den Ringbuffer aus KE2. Wie müssen Sie die Anwendung bezüglich Architektur anpassen, damit der Ringbuffer später mit GoogleTest getestet werden kann?

2.1 Führen Sie diese Änderungen durch!

2.2 Erstellen Sie ein GoogleTest Projekt mit 5 Testfällen, welche Ihren Ringbuffer testen.