OBJEKTORIENTIERTES PROGRAMMIEREN MIT C++, SOMMERSEMESTER 2018

Ihr Betreuer: Thomas Kammerer thomas.kammerer@th-nuernberg.de

EMPLOYEE RECORD SYSTEM

VORAUSSETZUNG:

Wir haben die Klasse Employee als Übung geschrieben und mit catch2 getestet. Unser Projekt besteht also aus

- Employee.h das Headerfile deklariert die Klasse Employee
- Employee.cpp die Implementierung der Methoden der Klasse Employee
- test_employee.cpp die Catch2 Tests der Klasse Employee
- test_main.cpp die Catch2 Mainmethode zum Ausführen der Tests
- Person.h, Person.cpp und test_person.cpp stammen noch aus der letzten Übung (wir lassen Sie einfach im Projekt)

```
1  #ifndef EMPLOYEE_H
2  #define EMPLOYEE_H
3
4  #include <string>
5
6  class Employee {
7    public:
8      Employee( const std::string& first_name, const std::string& last_name );
9
10     const std::string& get_first_name() const;
11     const std::string& get_last_name() const;
12     int get_id() const;
13     int get_salary() const;
14
15     void set_first_name( const std::string& first_name );
16     void set_last_name( const std::string& last_name );
17     void set_id( int id );
18     void set_salary( int salary );
19
20     void promote( int raise_amount );
21     void demote( int demerit_amount );
22     void hire();
23     void fire();
24     bool is_hired() const;
25
26     void display() const;
27
28   private:
29     std::string first_name_;
30     std::string last_name_;
31     int id_;
32     int salary_;
33     bool is_hired_;
34 };
35
36 #endif // EMPLOYEE_H
```

Abbildung 1: Employee.h

DATABASE

Wir wollen jetzt ein Employee-Record-System erstellen. Dazu brauchen wir eine Klasse Database, die alle Employees einer Firma verwaltet. Die Klasse sollte folgende Features haben:

- Einen neuen Employee zu Datenbank hinzufügen (er wird dadurch auch angestellt und erhält eine eindeutige ID).
- Suchen nach Employee über Vorname und Nachname.
- Suchen nach Employee über die ID.
- Die Anzahl der gespeicherten Employees abfragen.
- Die (hired) Employees als Liste zurückgeben.
- Die „gefeuerten“ Employees als Liste zurückgeben.
- Alle Employees anzeigen

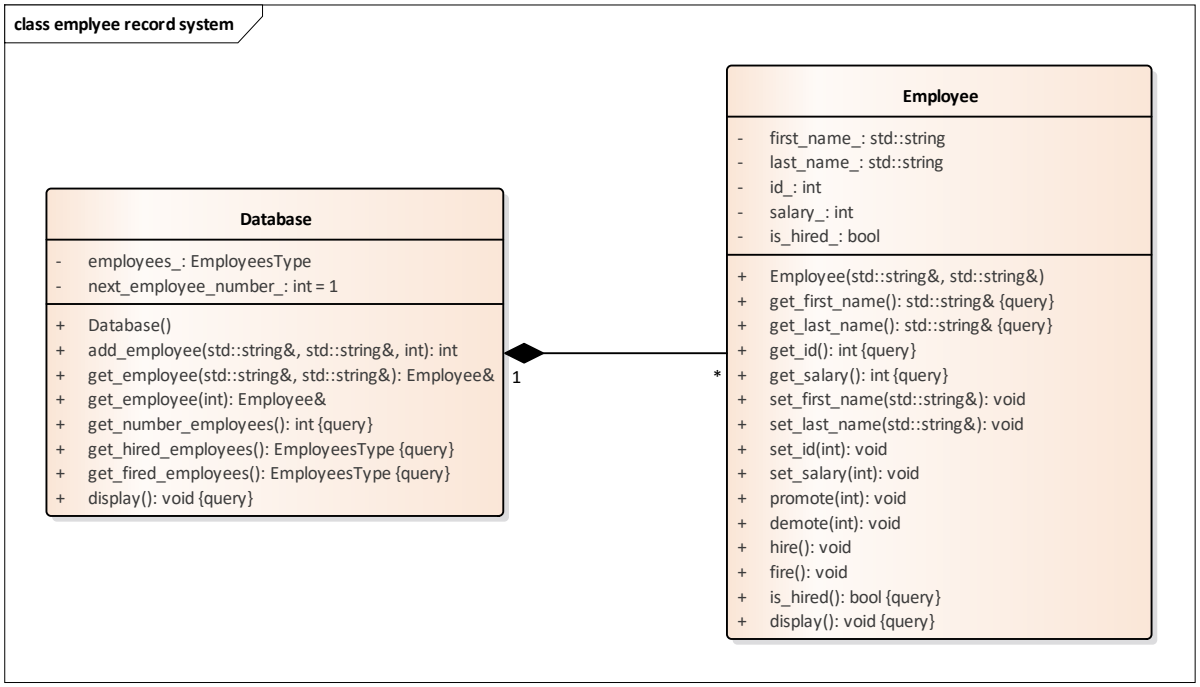

Abbildung 2: UML Diagramm: Employee Record System

Schreiben Sie die Klasse Database, verwenden Sie als Datencontainer einen std::vector<Employee>. Trennen Sie die Deklaration von der Implementierung indem sie Database.h und Database.cpp anlegen. Schreiben Sie Unittests in test_database.cpp.

Bitte testen Sie jede Methode Ihrer Klasse!

EIGENE MAIN.EXE, DIE UNSERE LIBRARY BENUTZT

Als nächstes verwenden wir die getestete Library in einer eigenen main.exe.

Legen Sie hierzu im Projekt ein neues Build-Target: Main an:

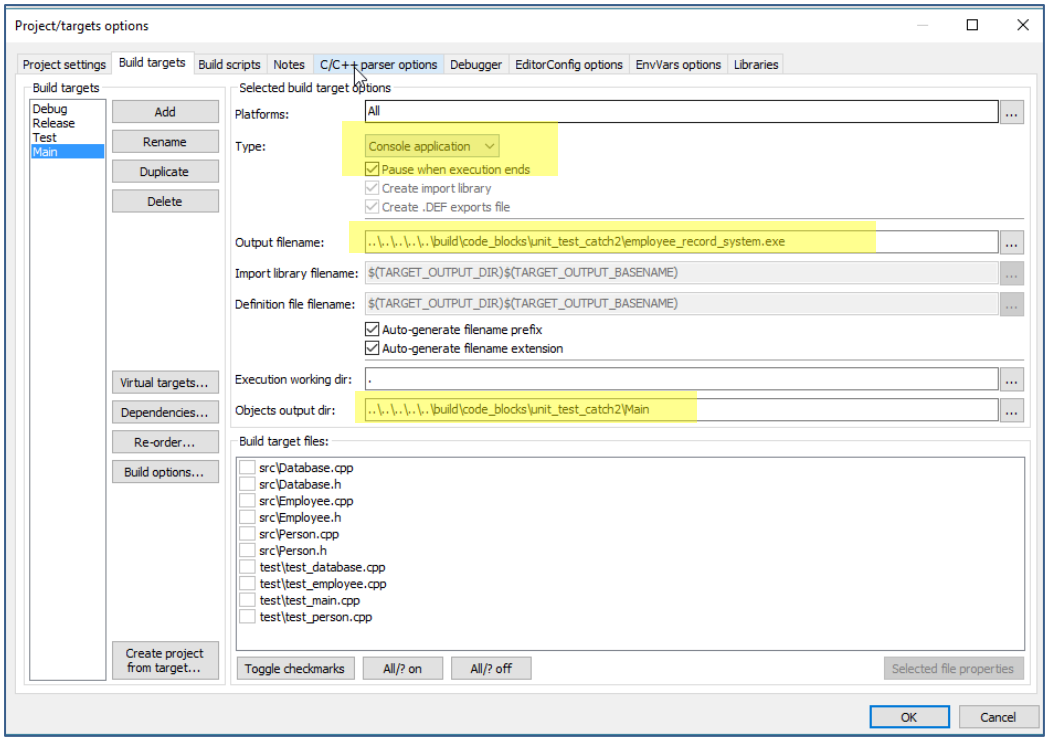

Abbildung 3: neues Build Target Main

Beachten Sie:

- Main ist eine Console Application
- Pause when execution ends Haken setzen!
- Output file und Objects wieder in unseren Buildpfad erzeugen lassen (bei mir z.B: ..\..\..\..\..\build\code_blocks\unit_test_catch2\Main)

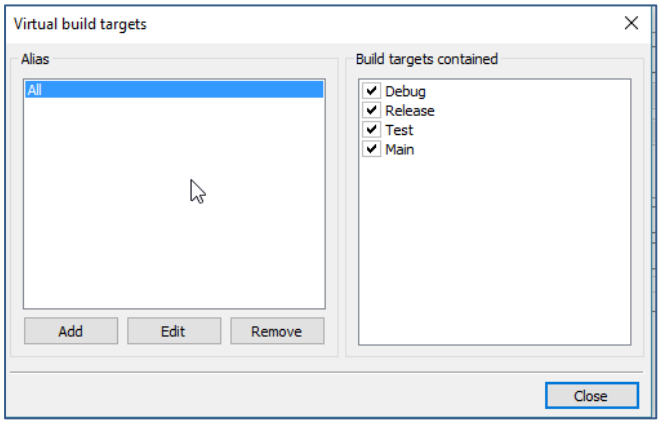

Abbildung 4: Anpassen des virtuellen Build Target All

Unser virtuelles Build-Target All anpassen, damit es auch Main erzeugt! (Haken setzen!)

Ich habe eine main.cpp bereits vorbereitet, sie finden das File unter:

cpp_summer_course_2018\lessons\001_first_classes\001_unit_tests_catch2\main

Bitte fügen Sie main.cpp dem Projekt hinzu.

Wählen Sie main.cpp nur für das Main-Target aus.

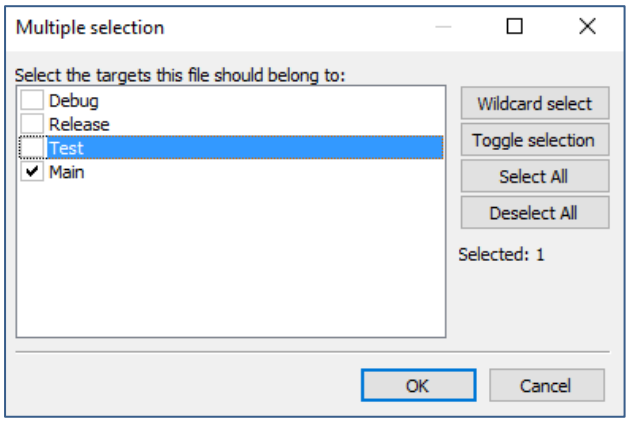

Abbildung 5: main.cpp soll nur im Main-Target gebaut werden!

Auch das Main-Target muss noch konfiguriert werden:

- C++11 oder höher Haken setzen in Build Options/Compiler Settings
- Die Library lib001_unit_tests_catch2.a dazu linken unter Build Options/Linker Settings
- Suchverzeichnis src setzen unter Build Options/Search Directories

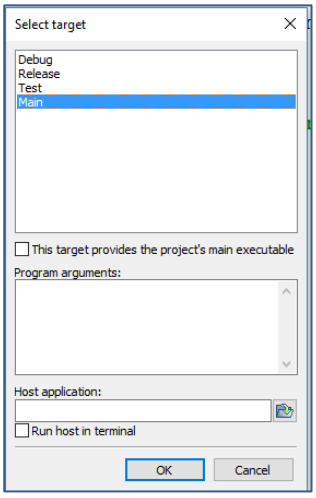

Abbildung 6: Main auswählen!

Übersetzen und ausführen (Main als Target beim Starten auswählen)!

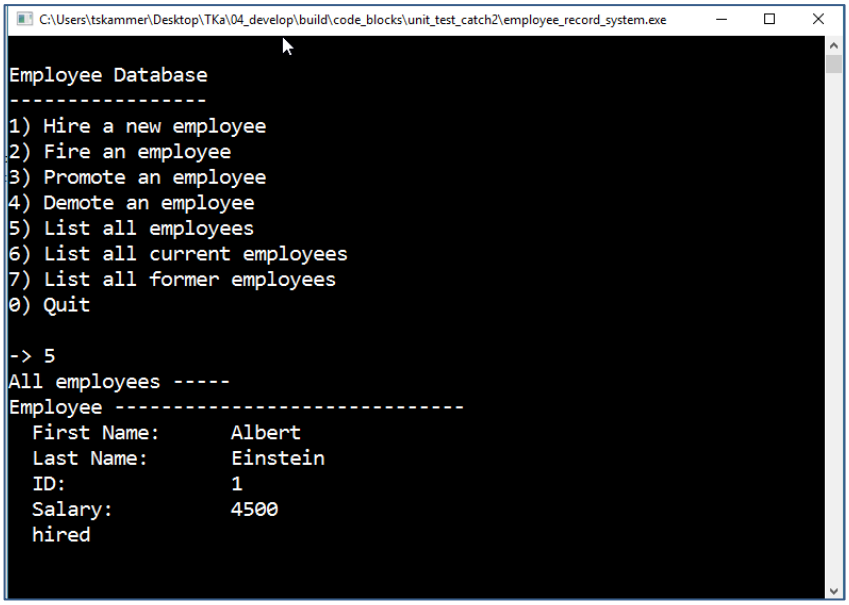

Abbildung 7: Main in Aktion!

Wenn Sie das alles geschafft haben: GRATULATION !!!
Wenn Sie das noch nicht geschafft haben: KEINE PANIK! und DRAN BLEIBEN!

Schicken Sie Ihr Projekt gezipp an:
thomas.kammerer@th-nuernberg.de