Aufgabe	1	2	3	4	5	Summe	Note
Maximum	10	20	25	25	20	100	
Erreicht							

Organisatorische Hinweise:

- Dauer der Klausur: 90 Minuten
- Es darf ein DinA4-Blatt mit handschriftlichen Notizen verwendet werden (Vorder- und Rückseite beschrieben).
- Es dürfen keine elektronischen Geräte in irgendeiner Form benutzt werden. Also kein: Taschenrechner, Notebook, Handy usw.

Verhalten während der Klausur:

- Beschriften Sie sofort das Deckblatt mit Ihrem Namen und Ihrer Matrikel-Nr.
- Verwenden Sie die vorgegebenen Klausuraufgabenzettel für Ihre Lösungen (ggf. auch die Rückseite).
- Benutzen Sie keinen Bleistift und keine rote Tinte. Entsprechende Aufzeichnungen werden nicht gewertet.
- Nutzen Sie die anfängliche Fragestunde, um alle Unklarheiten zu klären. Während der Bearbeitungszeit werden keine Fragen mehr beantwortet!!
- Legen Sie Ihren Studierendenausweis gut sichtbar auf den Tisch.
- Sofern Sie ein (max. DinA4-)Hilfsblatt vorgefertigt haben, legen Sie es zum Ausweis.
- Melden Sie sich bitte mit Handzeichen, wenn Sie zur Toilette müssen (zeitgleich darf nur ein Klausurteilnehmer den Raum verlassen)
- Vermeiden Sie möglichst Störungen!

Genereller Hinweis zur Klausur:

- Halten Sie sich an die vorgegebenen Namenskonventionen.
- Achten Sie bitte auf einen strukturierten und gut lesbaren Code (d.h. Coding-Style).
- Achten Sie auf Effizienz beim Programmieren.
- Evtl. notwendige **`#include`**-Anweisungen können Sie voraussetzen.

Viel Erfolg!

Aufgabe 1 [10 Punkte]

a) Erklären Sie den Unterschied zwischen einer normalen Klasse und einer Interface Klasse in C++. [3 Punkte]

b) Was ist der Unterschied zwischen Interface Polymorphismus und Klassenvererbung in Bezug auf ihre Attribute und Methoden? [3 Punkte]

c) Nennen Sie je **zwei** wichtige Eigenschaften von Quick Sort und Bubble Sort. Welcher Algorithmus ist der Schnellere? [4 Punkte]

Raumüberwachungssystem Beschreibung

Ein Raumüberwachungssystem soll die Raumfeuchtigkeit eines Raumes überwachen, die sich innerhalb der Grenzen *MIN* (inklusive) und *MAX* (inklusive) bewegen soll. Für die Implementierung dieses Überwachungssystems werden zwei Klassen benötigt:

1) Die Klasse ***Humidifier*** sorgt dafür, dass der Motor des Luftbefeuchters an- und ausgeschaltet wird.
    - Beim *Befeuchten* des Raums wird der Motor des Luftbefeuchters eingeschaltet und das Zimmerfenster geschlossen.
    - Beim *Entfeuchten* des Raums wird der Motor des Luftbefeuchters ausgeschaltet und das Zimmerfenster geöffnet.
    - Im *Normalbetrieb* ist der Motor ausgeschaltet und das Zimmerfenster geschlossen.

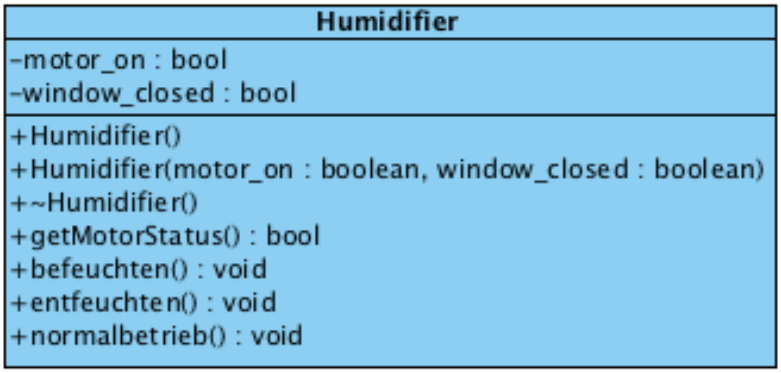
2) Die Klasse ***HumidFSM*** definiert die Steuerungslogik von *Humidifier* anhand einer Zustandsmaschine. Die Steuerungslogik der Zustandsmaschine soll wie folgt aussehen:
    - Nach der Inbetriebnahme des Raumüberwachungssystem wird die aktuelle Raumfeuchtigkeit überwacht.
    - Unterschreitet die Feuchtigkeit den unteren Grenzwert *MIN*, so wird der Raum befeuchtet.
    - Übersteigt die Feuchtigkeit den oberen Grenzwert MAX, so wird der Raum entfeuchtet.
    - Liegt die Feuchtigkeit zwischen *MIN* und *MAX*, so ist das System im Normalbetrieb.
    - Nach jedem Zustandswechsel wird der aktuelle Zustand auf der Konsole ausgegeben.
    - Es gibt keinen direkten Übergang zwischen Befeuchten und Entfeuchten, d.h. nach Be- bzw. Entfeuchten erreicht die FSM immer zuerst den Normalzustand.

Aufgabe 2: *Humidifier* [20 Punkte]

Die Klasse *Humidifier* sorgt dafür, dass der Motor des Luftbefeuchters an- und ausgeschaltet wird.

- Beim **Befeuchten** des Raums wird das Fenster geschlossen und der Motor des Luftbefeuchters eingeschaltet.
- Beim **Entfeuchten** des Raums wird das Fenster geöffnet und der Motor des Luftbefeuchters ausgeschaltet.
- Im **Normalbetrieb** ist das Fenster geschlossen und der Motor ausgeschaltet.

a) Deklarieren Sie die Klasse in *Humidifier.h* anhand des gegebenen UML Klassendiagramms. [5 Punkte]



Antwort:

b) Beim Kreiieren eines neuen Objektes der Klasse *Humidifier* soll der Motor standardgemäß ausgeschaltet und das Fenster geschlossen sein. Implementieren Sie das in *Humidifier.cpp*. [4 Punkte]

Antwort:

c) Der folgende Overloaded Constructor wurde mangelhaft implementiert, dabei sollen die Übergabeparameterwerte an die Attribute des Objektes übergeben werden. Finden Sie den Fehler und korrigieren Sie diese. [3 Punkte]

```
Humidifier::Humidifier(bool motor_on, bool window_closed) {
                        motor_on = motor_on;
                        window_closed = window_closed;
}
```

d) Beim Befeuchten des Raumes soll das Fenster geschlossen und der Motor angeschaltet werden. Zusätzlich soll auf der Konsole folgende Nachricht angezeigt werden: „`Raum wird befeuchtet`“. Implementieren Sie die Methode **`void befeuchten()`** in *Humidifier.cpp*. [4 Punkte]

Antwort:

e) Gegeben ist das folgende Hauptprogramm:

```
1       int main(int argc, char** argv) {
2           Humidifier myHum;
3           for (int i=0; i<3; i++) {
5               myHum.getMotorStatus();
6           }
7           return 0;
8       }
```

In welcher Zeile wird der Konstruktor des Objekts aufgerufen, und in welcher Zeile wird sein Destruktor aufgerufen? [4 Punkte]

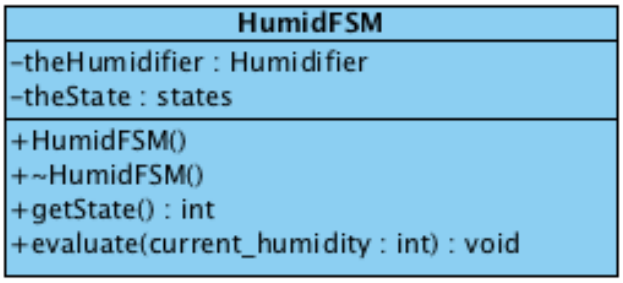
Antwort:

Aufgabe 3: *HumidFSM* [25 Punkte]

Die Klasse ***HumidFSM*** definiert die Steuerungslogik von *Humidifier* anhand einer Zustandsmaschine. Dabei wird der FSM der aktuelle Feuchtigkeitswert *current_humidity* in *main.cpp* übergeben. Die Steuerungslogik der Zustandsmaschine soll wie folgt aussehen:

- Nach der Inbetriebnahme des Raumüberwachungssystem wird die Raumfeuchtigkeit überwacht.
- Unterschreitet die Feuchtigkeit den unteren Grenzwert *MIN*, so wird der Raum befeuchtet.
- Übersteigt die Feuchtigkeit den oberen Grenzwert MAX, so wird der Raum entfeuchtet.
- Erreicht die Feuchtigkeit wieder Werte zwischen *MIN* und *MAX*, so geht das System wieder in den Normalbetrieb.
- Nach jedem Zustandswechsel wird der aktuelle Zustand auf der Konsole ausgegeben.
- Es gibt keinen direkten Übergang zwischen Befeuchten und Entfeuchten, d.h. nach Be- bzw Entfeuchten erreicht die FSM immer zuerst den Normalzustand.

Die Klasse *HumidFSM* hat folgende Deklaration:



private Attribute:

- Zustände (*theStates*) der Zustandsmaschine;
- Luftentfeuchter (*theHumidifier*) als Gegenstand der Steuerungslogik;

public Methoden:

- Default Constructor *HumidFSM()*;
- Destructor *~HumidFSM();*
- Methode *void evaluate(int)* mit Parameter *current_humidity*, ohne Rückgabewert;

a) Designen Sie die Zustandsmaschine für das Raumüberwachungssystem anhand von UML State Machine. Achten Sie darauf, dass die Namen Ihrer Zustände und Transitionen selbsterklärend sind. [10 Punkte]

Antwort:

b) Definieren Sie den Typ *states* für Ihre FSM in *HumidFSM.h*. [2 Punkte]

Antwort:

c) Implementieren Sie die Methode *void evaluate(int)* in *HumidFSM.cpp,* die die Steuerungslogik Ihrer FSM aus 3a) auswertet. [13 Punkte]

Antwort:

```
void HumidFSM::evaluate(int){

}
```

Aufgabe 4: Testen und Filehandling [25 Punkte]

a) Analysieren Sie Ihre FSM aus Aufgabe 3a) und ermitteln Sie anhand von folgenden Testgenerierungs-Methoden eine minimale Anzahl von Testdaten und -sequenzen für den Aufruf von *evaluate(current_Humidity)*. [6 Punkte]

i. 100% Zutandsüberdeckung
Antwort:

current_Humidity =

ii. 100% Übergangsüberdeckung
Antwort:

current_Humidity =

iii. Grenzwertanalyse
Antwort:

current_Humidity =

b) Implementieren Sie eine Test-Methode *void evalute_Test()* für die 100% Zustandsüberdeckung aus a)i. [4 Punkte]

Antwort:

```
#include <assert.h>

void evaluate_Test() {




}
```

c) Das Raumüberwachungssystem soll nun automatisiert getestest werden. Dafür wird die Klasse *FileHandler* benötigt, der Daten aus einer Datei ausliest.

Jede Zeile der Datei Namens „TestDaten.txt“ beinhaltet einen Feuchtigkeitswert. Die Methode *int readFile(FILE* testFile)* liest die *testFile* aus und gibt einen Feuchtigkeitswert als Rückgabewert zurück. Implementieren Sie die Methode in *FileHandler.cpp*. [8 Punkte]

Antwort:

```
int FileHandler::readFile(FILE* testFile) const{

}
```

d) Im Hauptprogramm wird der ausgelesene Feuchtigkeitswert von der Steuerungslogik ausgewertet. Die Daten wird solange ausgewertet, bis die Datei leer ist. Implementieren Sie das im Hauptprogramm *main.cpp.* [7 Punkte]

Antwort:

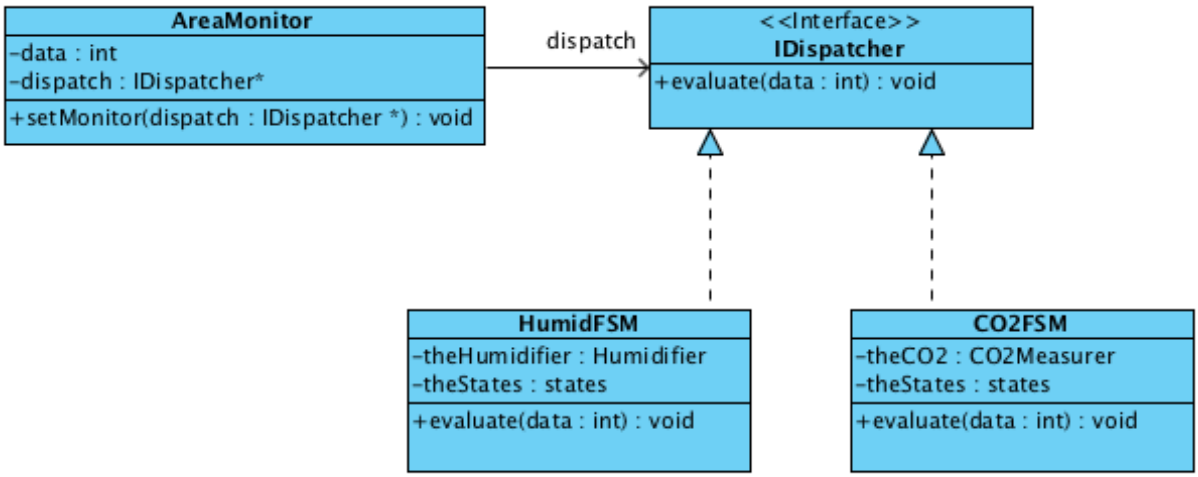
```
int main(){




}
```

Aufgabe 5: Interfaces [20 Punkte]

Das Raumüberwachungssystem bietet eine Schnittstelle *IDispatcher* an, die Daten an unterschiedliche Überwachungseinheiten zur Verarbeitung weiterleitet, z.B. Temperatur, CO2-Gehalt, Raumfeuchtigkeit etc.



Hinweis: Konstruktoren und Destruktoren sind zu vernachlässigen.

a) Deklarieren Sie die Interface Klasse *IDispatcher.h*. [4 Punkte]

Antwort:

b) Deklarieren Sie erneut die realisierende Klasse *HumidFSM.h.* [4 Punkte]

Antwort:

c) Deklarieren Sie die Consumer Klasse *AreaMonitor.h.* [3 Punkte]

Antwort:

d) Die Methode *void setMonitor(IDispatcher*)* der Klasse *AreaMonitor* ruft über die Schnittstelle *IDispatcher* die Methode *evaluate()* der jeweiligen FSMs auf. Implementieren Sie die Methode *void setMonitor(IDispatcher*)* in *AreaMonitor.cpp.* [6 Punkte]

Antwort:

```
void AreaMonitor::setMonitor(IDispatcher* dispatcher){

}
```

e) Füllen Sie die Lücken im Hauptprogramm. [3 Punkte]

Antwort:

```
int main() {
        AreaMonitor myMonitor;
        CO2FSM myCO2;
        HumidFSM myHumid;
        // Aufruf von CO2FSM:



        // Aufruf von HumidFSM:



        return 0;
}
```