

Name(1): Daniel Weyrer

Abgabetermin: 5.11.2019

Name(2): Viktoria Streibl

Punkte: 17,5

Übungsgruppe: Gruppe 1

korrigiert: REM

Geschätzter Aufwand in Ph: 10 | 10

Effektiver Aufwand in Ph: Auch hier eintragen

-0,5

Beispiel 1 (24 Punkte) Gehaltsberechnung: Entwerfen Sie aus der nachfolgenden Spezifikation ein Klassendiagramm, instanzieren Sie dieses und implementieren Sie die Funktionalität entsprechend:

Eine Firma benötigt eine Software für die Verwaltung ihrer Mitarbeiter. Es wird unterschieden zwischen verschiedenen Arten von Mitarbeitern, für die jeweils das Gehalt unterschiedlich berechnet wird.

Jeder Mitarbeiter hat: einen Vor- und einen Nachnamen, ein Namenskürzel (3 Buchstaben), eine Sozialversicherungsnummer (z.B. 1234020378 -> Geburtsdatum: 2. März 1978) und ein Einstiegsjahr (wann der Mitarbeiter zur Firma gekommen ist).

Bei der Bezahlung wird unterschieden zwischen:

- *CommissionWorker*: Grundgehalt + Fixbetrag pro verkauftem Stück
- *HourlyWorker*: Stundenlohn x gearbeitete Monatsstunden
- *PieceWorker*: Summe erzeugter Stücke x Stückwert
- *Boss*: monatliches Fixgehalt

Überlegen Sie sich, welche Members und Methoden die einzelnen Klassen benötigen, um mindestens folgende Abfragen zu ermöglichen:

- Wie viele Mitarbeiter hat die Firma?
- Wie viele *CommissionWorker* arbeiten in der Firma?
- Wie viele Stück wurden im Monat erzeugt?

- Wie viele Stück wurden im Monat verkauft?
- Wie viele Mitarbeiter sind vor 1970 geboren?
- Wie hoch ist das Monatsgehalt eines Mitarbeiters?
- Gibt es einen Mitarbeiter zu einem gegebenen Namenskürzel?
- Welche(r) Mitarbeiter ist/sind am längsten in der Firma?
- Ausgabe aller Datenblätter der Mitarbeiter

Zur Vereinfachung braucht nur ein Monat berücksichtigt werden (d.h. pro Mitarbeiter nur ein Wert für Stückzahl oder verkaufte Stück). Realisieren Sie die Ausgabe des Datenblattes als *Template Method*. Der Ausdruck hat dabei folgendes Aussehen:

```
*****
Fa. Hofer, Linz
*****
Datenblatt
-----
Name: Max Huber
Kürzel: mhu
Sozialversicherungsnummer: 1234010273
Einstiegsjahr: 2005
Mitarbeiterklasse: CommissionWorker
Grundgehalt: 2500 EUR
Provision: 350 EUR
Gesamtgehalt: 2850 EUR
-----
v1.0 Oktober 2019
-----
```

Achten Sie bei Ihrem Entwurf auf die Einhaltung der Design-Prinzipen!

Schreiben Sie einen Testtreiber, der mehrere Mitarbeiter aus den unterschiedlichen Gruppen anlegt. Die erforderlichen Abfragen werden von einer Klasse `Client` durchgeführt und die Ergebnisse ausgegeben. Achten Sie darauf, dass diese Klasse nicht von Implementierungen abhängig ist.

Treffen Sie für alle unzureichenden Angaben sinnvolle Annahmen und begründen Sie diese. Verfassen Sie weiters eine Systemdokumentation (Funktionalität, Klassendiagramm, Schnittstellen der beteiligten Klassen, etc.)!

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung**! Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den **Testoutput** mit ab!

SDP - Exercise 02

winter semester 2019/20

Viktoria Streibl - S1810306013

Daniel Weyrer - S1820306044

November 5, 2019

Contents

1	Organizational	6
1.1	Team	6
1.2	Roles and responsibilities	6
1.2.1	Jointly	6
1.2.2	Viktoria Streibl	6
1.2.3	Daniel Weyrer	6
1.3	Effort	6
1.3.1	Viktoria Streibl	6
1.3.2	Daniel Weyrer	6
2	Requirement Definition(System Specification)	7
3	System Design	9
3.1	Classdiagram	9
3.2	Design Decisions	10
3.2.1	cTor's Derived Classes (Base: Employee)	10
3.2.2	isDateValid	10
3.2.3	Search Employee	10
4	Component Design	10
4.1	Class Client	10
4.2	Class ICompany	11
4.3	Class Company	12
4.4	Class Employee	12
4.5	Class CommissionWorker	12
4.6	Class HourlyWorker	12
4.7	Class PiecesWorker	12
4.8	Class Boss	12
4.9	TestDriver	13
5	Test Protocol	14
5.1	Console Output	14
6	Source Code	17
6.1	Class Client	17
6.1.1	Client.h	17
6.1.2	Client.cpp	18
6.2	Interface ICompany	21
6.2.1	ICompany.h	21
6.3	Class Company	22
6.3.1	Company.h	22
6.3.2	Company.cpp	23
6.4	Class Employee	26
6.4.1	Employee.h	26
6.4.2	Employee.cpp	28
6.5	Class CommissionWorker	31
6.5.1	CommissionWorker.h	31

6.5.2	CommissionWorker.cpp	32
6.6	Class HourlyWorker	33
6.6.1	HourlyWorker.h	33
6.6.2	HourlyWorker.cpp	34
6.7	Class PieceWorker	35
6.7.1	PieceWorker.h	35
6.7.2	PieceWorker.cpp	36
6.8	Class Boss	37
6.8.1	Boss.h	37
6.8.2	Boss.cpp	38
6.8.3	TestDriver.cpp	38

1 Organizational

1.1 Team

- Viktoria Streibl - S1810306013
- Daniel Weyrer - S1820306044

1.2 Roles and responsibilities

1.2.1 Jointly

- planning
- Documentation
- Systemdocumentation
- Class Diagram

1.2.2 Viktoria Streibl

- Main Class Company
- Interface ICompany
- Testdriver Client
- Main Testdriver

1.2.3 Daniel Weyrer

- Base Class for Employee
- Derived Classes
 - Class Commission Worker
 - Class Hourly Worker
 - Class Pieces Worker
 - Class Boss

1.3 Effort

1.3.1 Viktoria Streibl


- estimated: 10ph
- actually: 11 ph

1.3.2 Daniel Weyrer

- estimated: 10 ph
- actually: 11 ph

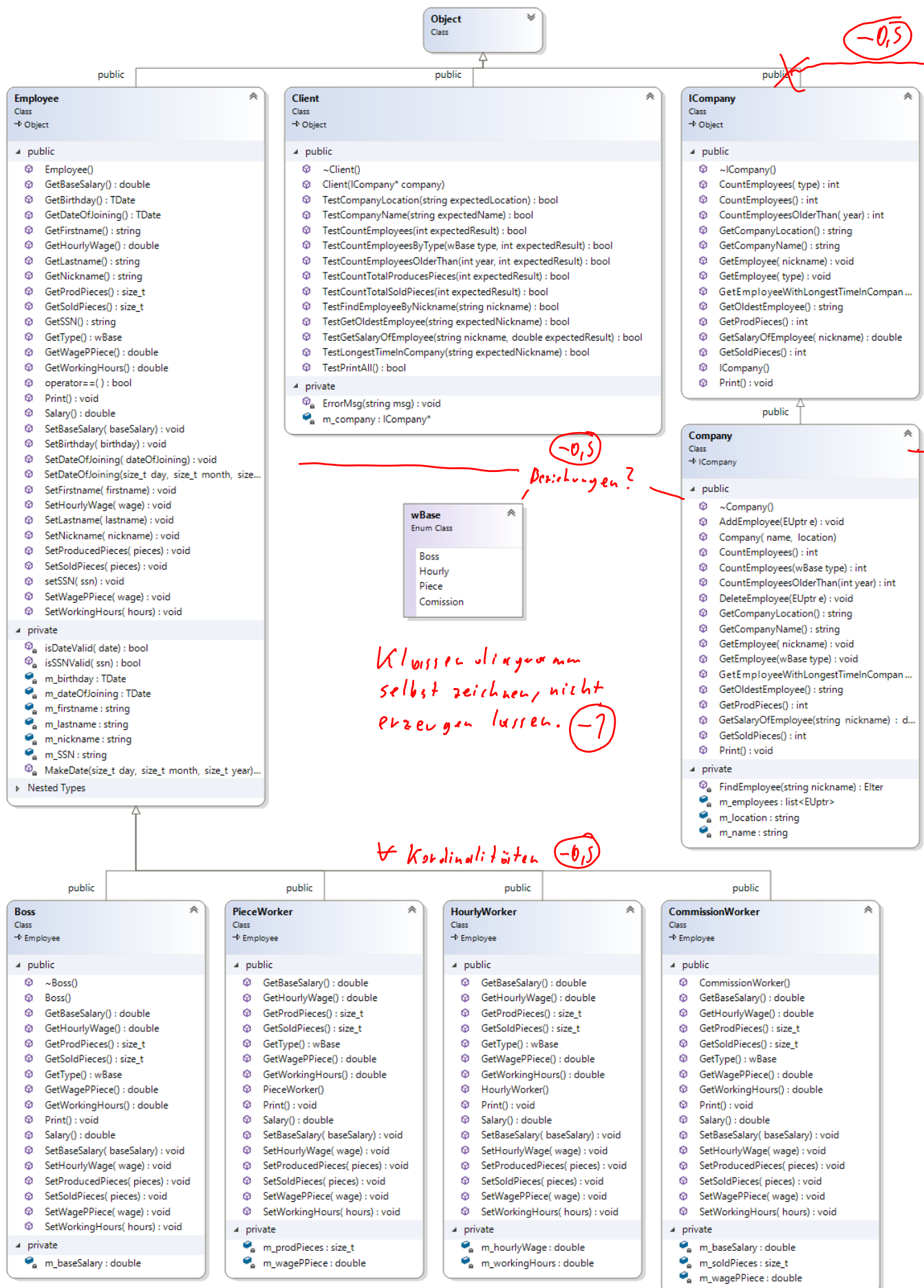
2 Requirement Definition(System Specification)

It was a company desired the various types of employees includes, such as commission worker, hourly worker, pieces worker and boss. Each employee type should also include and output some key data such as name, SSN, date of joining, salary and birthday. In addition, each company has to ouput the name and location. Any number of employees can be added and deleted in the programm, but the client is not allowed to do so. It is possible to search for employees by nickname as well as by the type. The Client can also get all produces and all sold pieces.



3 System Design

3.1 Classdiagram



3.2 Design Decisions

3.2.1 cTor's Derived Classes (Base: Employee)

The cTor's of the derived classes set every wage and number to 0, to identify uninitialized variables (as the Salary is 0 when one of the two parameters are 0!).

3.2.2 isDateValid

The Date get's it's validation in two steps:

- correct date-syntax
- is the entered date in a valid range

The first step is achieved by checking with number of days in each month while taking care of the leapyears!

The second step is done by comparing the current date with the entered date (which has to be in the past!).

3.2.3 Search Employee

Employee is searched by nickname because it has to be unique. To be sure that the nickname is unique we check it while adding new employee.

4 Component Design

4.1 Class Client

The Client simulate a person which use the interface. The following functions tests the functionality:

- Tests the company name
- Tests the company location
- Tests if it is possible to find a employee by nickname
- Tests if the employee which is search by his nickname is correct
- Counts all employees and check if it is correct
- Counts all employees of one type and check it
- Counts all produced pieces and check them
- Counts all sold pieces and check them
- Count how many employees are older than a specific year and check it
- Tests if the salary of an employee is correct
- Tests if the oldest employee is correct
- Check for the employee with is the oldest member

- Print all data of company and employees

All these methods are only for testing the functionality of Company. Furthermore, the Client gets the company with the constructor as ICompany.

4.2 Class ICompany

Is an interface which is used by the Client. It contains the following functions:

- Get the company name
- Get the company location
- Get an employee by nickname
- Get all employees of the same type
- Get all sold pieces
- Get all produced pieces
- Get salary of an employee
- Get oldest employee
- Get employee with the longest time in the company
- Count all employees in the company
- Count all employees with the same type
- Count all employees which are older than a specific year
- Output all employees in the company and some general data

The ICompany is the interface between an client and the company. The Client is not allowed to manipulate the employees. It defines the methods with can be used by the client.

"GetCompanyName", returns the name of the company. "GetCompanyLocation", returns the location of the company. "GetEmployee", can be used with the nickname or the type and returns the employees. "GetSoldPieces", counts all pieces which are sold by the company. "GetProdPieces", counts all pieces which are produces by the employees. "GetSalaryOfEmployee", returns the salary of an employee. "GetOldestEmployee", returns the oldest employee. "GetEmployeeWithLongest-TimeInCompany", returns the nickname of the employee with the oldest join in date "CountEmployees", returns the number of employees in the company, can be specified with type. "CountEmployeesOlderThan", counts how many employees are older than a specific year. "Print", outputs the name and location of the company, as well as all employees.

4.3 Class Company

Manages all employees in the company. It implements the interface ICompany. It contains the following functions:

- Add a new Employee
- Remove a Employee
- All functions from ICompany
- findEmployee

The Company class manages all Employees. It uses unique pointers stored in a vector, to avoid shallow-copies. With the method "AddEmployee" a new employee can be created. Should a employee already exist with the same nickname. So this employee is not stored and an error message output. "DeleteEmployee" deletes a employee. If none is stored with the nickname an exception get's thrown and caught in the same method. The private function "FindEmployee", loop through the list and checks if there is any employee with the specific nickname.

4.4 Class Employee

Is the base class of all employee types. It contains the following functions:

- Contains the Get and Set pure virtual functions for the derived classes
- Overloaded Output-Operator for date-struct

4.5 Class CommissionWorker

This class represents a commission worker.

- The ComissionWorker gets a base Salary and a commssion for every sold piece

4.6 Class HourlyWorker

This class represents a hourly worker.

- contains workinghours and the hourly wage the worker gets for each hour.

4.7 Class PiecesWorker

This class represents a pieces worker.

- saves number of produced pieces and the wage the worker gets for every piece he produces

4.8 Class Boss

This class represents a boss.

- BaseSalary, as it's independent from the Boss' actions!

4.9 TestDriver

The Testdriver test alle functions of the Client. It adds commissoon worker, hourly worker, pieces worker and a boss. The functions "TestLinzAG", "TestSequality" and "TestTractive" call the testing functions of client and check if everything worked. The test-functions in Client returns a bool. If the bool returns false something didn't pass and this Testcase cannot get valid again. It ouputs a error message if there was no successful run.

5 Test Protocol

It has been tested in the file "TestDriver", the following points have been tested:

- get name of the company
- get location of the company
- count all employees in the company
- find an employee by his nickname
- searching for the oldest employee
- count all employees older than 1990.
- get the employee with the oldest joining date.
- count all employees by the same type
- count all sold pieces this month
- test the salary of a employee

5.1 Console Output

```
1 #####
2 Client test for Linz AG
3 #####
4 Name: Daniel Weyrer
5 Kuerzel: DaW
6 Sozialversicherungsnummer: 7733177331
7 Einstiegsjahr: 2015
8 Mitarbeiterklasse: HourlyWorker
9 Arbeitsstunden: 80
10 Stundenlohn: 13 EUR
11 Gehalt: 1040 EUR
12 *****
13 Linz AG, Linz
14 *****
15 Datenblatt
16 -----
17
18 Name: Christian Grey
19 Kuerzel: CGB
20 Sozialversicherungsnummer: 1234512345
21 Einstiegsjahr: 0
22 Mitarbeiterklasse: Boss
23 Gehalt: 4800 EUR
24
25 Name: Viktoria Streibl
26 Kuerzel: ViS
27 Sozialversicherungsnummer: 1290012900
28 Einstiegsjahr: 2010
29 Mitarbeiterklasse: ComissionWorker
30 Grundgehalt: 2100 EUR
31 Provision: 88
32 Gehalt: 2188 EUR
33
34 Name: Daniel Weyrer
35 Kuerzel: DaW
36 Sozialversicherungsnummer: 7733177331
37 Einstiegsjahr: 2015
38 Mitarbeiterklasse: HourlyWorker
```

```
39 Arbeitsstunden: 80
40 Stundenlohn: 13 EUR
41 Gehalt: 1040 EUR
42
43 Name: John Doe
44 Kuerzel: JoD
45 Sozialversicherungsnummer: 1230502539
46 Einstiegsjahr: 2003
47 Mitarbeiterklasse: PieceWorker
48 Stueckzahl: 10
49 Stueckzahl: 5 EUR
50 Gehalt: 50 EUR
51 -----
52 v1.0 Oktober 2019
53 -----
54 Everything OK...
55
56 #####
57 Client test for Sequality GmbH
58 #####
59 *****
60 Sequality GmbH, Hagenberg
61 *****
62 Datenblatt
63 -----
64
65 Name: Christian Grey
66 Kuerzel: CGB
67 Sozialversicherungsnummer: 1234512345
68 Einstiegsjahr: 0
69 Mitarbeiterklasse: Boss
70 Gehalt: 4800 EUR
71
72 Name: Viktoria Streibl
73 Kuerzel: ViS
74 Sozialversicherungsnummer: 1290012900
75 Einstiegsjahr: 2010
76 Mitarbeiterklasse: ComissionWorker
77 Grundgehalt: 2100 EUR
78 Provision: 88
79 Gehalt: 2188 EUR
80 -----
81 v1.0 Oktober 2019
82 -----
83 Everything OK...
84
85 #####
86 Client test for Tractive
87 #####
88 *****
89 Tractive, Pasching
90 *****
91 Datenblatt
92 -----
93
94 Name: Christian Grey
95 Kuerzel: CGB
96 Sozialversicherungsnummer: 1234512345
97 Einstiegsjahr: 0
98 Mitarbeiterklasse: Boss
99 Gehalt: 4800 EUR
100
101 Name: Daniel Weyrer
102 Kuerzel: DaW
103 Sozialversicherungsnummer: 7733177331
104 Einstiegsjahr: 2015
105 Mitarbeiterklasse: HourlyWorker
106 Arbeitsstunden: 80
107 Stundenlohn: 13 EUR
108 Gehalt: 1040 EUR
109
110 Name: John Doe
111 Kuerzel: JoD
```

```
112 Sozialversicherungsnummer: 1230502539
113 Einstiegsjahr: 2003
114 Mitarbeiterklasse: PieceWorker
115 Stueckzahl: 10
116 Stueckzahl: 5 EUR
117 Gehalt: 50 EUR
118 -----
119 v1.0 Oktober 2019
120 -----
121 Everything OK...
```


6 Source Code

6.1 Class Client

6.1.1 Client.h

```
1  /* -----
2  | Workfile : Client .h
3  | Description : [ HEADER ] Class for the Client to act with an Company
4  | Name : Viktoria Streibl      PKZ : S1810306013
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9  #ifndef CLIENT_H
10 #define CLIENT_H
11
12 #include "ICompany.h"
13
14 class Client : public Object {
15
16 public:
17     Client(ICompany* const company);
18     ~Client() = default;
19
20     //tests if the company name is correct
21     bool TestCompanyName(std::string& expectedName) const;
22     //tests if the company location is correct
23     bool TestCompanyLocation(std::string& expectedLocation) const;
24     //tests if the employee which is search by his nickname is correct
25     bool TestFindEmployeeByNickname(std::string& nickname) const;
26     //counts all employees and check if it is correct
27     bool TestCountEmployees(int expectedResult) const;
28     //counts all employees of one type and check it
29     bool TestCountEmployeesByType(wBase type, int expectedResult) const;
30     //count all procuded pieces and check them
31     bool TestCountTotalProducesPieces(int expectedResult) const;
32     //count all sold pieces and check them
33     bool TestCountTotalSoldPieces(int expectedResult) const;
34     //count how many employees are older than a specific year and check it
35     bool TestCountEmployeesOlderThan(int year, int expectedResult) const;
36     //tests if the salary of an employee is correct
37     bool TestGetSalaryOfEmployee(std::string& nickname, double expectedResult) const;
38     //tests if the oldest employee is correct
39     bool TestGetOldestEmployee(std::string& expectedNickname) const;
40     //check for the employee with is the oldest member
41     bool TestLongestTimeInCompany(std::string& expectedNickname) const;
42     //let print all data of company and employees
43     bool TestPrintAll() const;
44
45 private:
46     ICompany* m_company;
47
48     void ErrorMsg(std::string msg) const;
49 };
50 #endif //CLIENT_H
```

Strings nicht kopieren + const

(-7)

6.1.2 Client.cpp

```
1  /* -----
2  | Workfile : Client.cpp
3  | Description : [ SOURCE ] Class for the Client to act with an Company
4  | Name : Viktoria Streibl      PKZ : S1810306013
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9  #include "Client.h"
10
11 Client::Client(ICompany* const company) {
12     m_company = company;
13 }
14
15 bool Client::TestCompanyName(std::string expectedName) const {
16     //get company name and compare with expected name
17     std::string name = m_company->GetCompanyName();
18     if (name == expectedName) {
19         return true;
20     }
21     else {
22         ErrorMsg("Company Name was wrong");
23         return false;
24     }
25 }
26
27 bool Client::TestCompanyLocation(std::string expectedLocation) const {
28     //get company location and compare with expected location
29     std::string name = m_company->GetCompanyLocation();
30     if (name == expectedLocation) {
31         return true;
32     }
33     else {
34         ErrorMsg("Company Location was wrong");
35         return false;
36     }
37 }
38
39 bool Client::TestFindEmployeeByNickname(std::string nickname) const {
40     //search for employee by nickname and print it
41     m_company->GetEmployee(nickname);
42     return true;
43 }
44
45 bool Client::TestCountEmployees(int expectedResult) const {
46     //get number of employees and compare with expected result
47     int currEmployees = m_company->CountEmployees();
48     if (expectedResult == currEmployees) {
49         return true;
50     }
51     else {
52         ErrorMsg("Number of employees was wrong");
53         return false;
54     }
55 }
56
57 bool Client::TestCountEmployeesByType(wBase type, int expectedResult) const {
58     //get number of employees of specific type and compare with expected result
59     int currEmployees = m_company->CountEmployees(type);
60     if (expectedResult == currEmployees) {
61         return true;
62     }
63     else {
64         ErrorMsg("Numbers of employees by the same type was wrong");
65         return false;
66     }
67 }
68
69 bool Client::TestCountTotalProducesPieces(int expectedResult) const {
70     //get total produced pieces and compare with expected result
71     int totalProdPieces = m_company->GetProdPieces();
72     if (expectedResult == totalProdPieces) {
```

```
71     return true;
72 }
73 else {
74     ErrorMsg("Numbers of produces pieces was wrong");
75     return false;
76 }
77 }
78
79 bool Client::TestCountTotalSoldPieces(int expectedResult) const {
80     //get total sold pieces and compare with expected result
81     int totalSoldPieces = m_company->GetSoldPieces();
82     if (expectedResult == totalSoldPieces) {
83         return true;
84     }
85     else {
86         ErrorMsg("Numbers of sold pieces was wrong");
87         return false;
88     }
89 }
90
91 bool Client::TestCountEmployeesOlderThan(int year, int expectedResult) const {
92     //get number of employees older than year and compare with expected result
93     int employeesOlderThan = m_company->CountEmployeesOlderThan(year);
94     if (expectedResult == employeesOlderThan) {
95         return true;
96     }
97     else {
98         ErrorMsg("Numbers of older-than employees was wrong");
99         return false;
100    }
101 }
102
103 bool Client::TestGetSalaryOfEmployee(std::string nickname, double expectedResult) const {
104     //get salary of employee and compare with expected result
105     double salaryOfEmployee = m_company->GetSalaryOfEmployee(nickname);
106     if (expectedResult == salaryOfEmployee) {
107         return true;
108     }
109     else {
110         ErrorMsg("Salaray of employee was wrong");
111         return false;
112     }
113 }
114
115 bool Client::TestGetOldestEmployee(std::string expectedNickname) const {
116     //get nickname of oldest employee compare with expected nickname
117     std::string nickname = m_company->GetOldestEmployee();
118     if (expectedNickname == nickname) {
119         return true;
120     }
121     else {
122         ErrorMsg("Finding oldest employee was wrong");
123         return false;
124     }
125 }
126
127 bool Client::TestLongestTimeInCompany(std::string expectedNickname) const {
128     //get nickname of oldest employee compare with expected nickname
129     std::string nickname = m_company->GetEmployeeWithLongestTimeInCompany();
130     if (expectedNickname == nickname) {
131         return true;
132     }
133     else {
134         ErrorMsg("Finding employee which is in the company for the longest time was wrong");
135         return false;
136     }
137 }
138
139 bool Client::TestPrintAll() const {
140     //print everything
141     m_company->Print();
142     return true;
143 }
```

```
144
145 void Client::ErrorMsg(std::string msg) const{
146     //outputs the error message
147     std::cout << "!!!!!! Error: " << msg << std::endl;
148 }
```

Die Funktion heit schon error

-0,5

6.2 Interface ICompany

6.2.1 ICompany.h

```
1  /* -----
2  | Workfile : ICompany .h
3  | Description : [ Interface ] Interface between Client and Company
4  | Name : Viktoria Streibl      PKZ : S1810306013
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9  #ifndef ICOMPANY_H
10 #define ICOMPANY_H
11
12 #include <stdio.h>
13 #include <string>
14 #include <list>
15
16 #include "Object.h"
17 #include "Employee.h"
18
19 class ICompany : public Object {
20 public:
21     ICompany() = default;
22     ~ICompany() = default;
23
24     //returns the name of the company
25     virtual std::string GetCompanyName() = 0;
26     //returns the location of the company
27     virtual std::string GetCompanyLocation() = 0;
28     //print a employee found by the nickname
29     virtual void GetEmployee(std::string const nickname) = 0;
30     //print all employees of the type
31     virtual void GetEmployee(wBase const type) = 0;
32     //return total sold pieces last month
33     virtual int GetSoldPieces() = 0;
34     //return total produced pieces last month
35     virtual int GetProdPieces() = 0;
36     //return the salary of the employee
37     virtual double GetSalaryOfEmployee(std::string const nickname) = 0;
38     //returns the nickname of the oldest employee
39     virtual std::string GetOldestEmployee() = 0;
40     //check for the employee with is the oldest member
41     virtual std::string GetEmployeeWithLongestTimeInCompany() = 0;
42     //returns the number of employees in the company
43     virtual int CountEmployees() = 0;
44     //returns the number of employees of a specific type in the company
45     virtual int CountEmployees(wBase const type) = 0;
46     //returns the number of employees older than a specific year
47     virtual int CountEmployeesOlderThan(int const year) = 0;
48     //print all data of the company and employees
49     virtual void Print() = 0;
50 };
51 #endif //ICOMPANY_H
```

6.3 Class Company

6.3.1 Company.h

```

1  /* -----
2  | Workfile : Company .h
3  | Description : [ HEADER ] Class Company to store all data
4  | Name : Viktoria Streibl      PKZ : S1810306013
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9  #ifndef COMPANY_H
10 #define COMPANY_H
11
12 #include <string>
13 #include <list>
14
15 #include "ICompany.h"
16 #include "Employee.h"
17
18 typedef std::unique_ptr<Employee> EUptr;
19 typedef std::list<EUptr>::const_iterator EIter;
20
21 class Company : public ICompany
22 {
23
24 public:
25     //create company with name and location
26     Company(std::string const name, std::string const location);
27     ~Company() = default;
28
29     //returns the name of the company
30     std::string GetCompanyName() override;
31     //returns the location of the company
32     std::string GetCompanyLocation() override;
33     //print a employee found by the nickname
34     void GetEmployee(std::string const nickname) override;
35     //print all employees of the type
36     void GetEmployee(wBase type) override;
37     //return total sold pieces last month
38     int GetSoldPieces() override;
39     //return total produced pieces last month
40     int GetProdPieces() override;
41     //return the salary of the employee
42     double GetSalaryOfEmployee(std::string nickname) override;
43     //returns the nickname of the oldest employee
44     std::string GetOldestEmployee() override;
45     //check for the employee with is the oldest member
46     std::string GetEmployeeWithLongestTimeInCompany() override;
47     //returns the number of employees in the company
48     int CountEmployees() override;
49     //returns the number of employees of a specific type in the company
50     int CountEmployees(wBase type) override;
51     //returns the number of employees older than a specific year
52     int CountEmployeesOlderThan(int year) override;
53     //print all data of the company and employees
54     void Print() override;
55
56     //add an employee
57     void AddEmployee(EUptr e);
58     //delete an employee
59     void DeleteEmployee(EUptr e);
60
61 private:
62     std::string m_name;
63     std::string m_location;
64     std::list<EUptr> m_employees;
65
66     //find an employee by nickname
67     EIter FindEmployee(std::string nickname);
68 };

```

```
69 #endif //COMPANY_H
```

6.3.2 Company.cpp

```
1  /* -----
2  | Workfile : Company.cpp
3  | Description : [ SOURCE ] Class Company to store all data
4  | Name : Viktoria Streibl      PKZ : S1810306013
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9  #include "Company.h"
10
11 using namespace std;
12
13 Company::Company(std::string const name, std::string const location) {
14     m_name = name;
15     m_location = location;
16 }
17
18 string Company::GetCompanyName() {
19     return m_name;
20 }
21
22 string Company::GetCompanyLocation() {
23     return m_location;
24 }
25
26 void Company::GetEmployee(std::string const nickname) {
27     EIter itList;
28     //loop through list and search for nickname
29     for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
30         if (nickname == (**itList).GetNickname()) {
31             (**itList).Print();
32         }
33     }
34 }
35
36 void Company::GetEmployee(wBase type){
37     EIter itList;
38     //loop through list and count every employee with the same type
39     for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
40         if (type == (**itList).GetType()) {
41             (**itList).Print();
42         }
43     }
44 }
45
46 int Company::GetSoldPieces() {
47     int sumSoldPieces = 0;
48
49     list<EUptr>::const_iterator itList;
50     //loop through list and sum all sold pieces
51     for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
52         sumSoldPieces += (**itList).GetSoldPieces();
53     }
54     return sumSoldPieces;
55 }
56
57 int Company::GetProdPieces() {
58     int sumProdPieces = 0;
59     list<EUptr>::const_iterator itList;
60     //loop through list and sum all produced pieces
61     for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
62         sumProdPieces += (**itList).GetProdPieces();
63     }
64     return sumProdPieces;
65 }
66
67 double Company::GetSalaryOfEmployee(std::string nickname) {
68     //get nickname of expected employee
69     EIter iter = FindEmployee(nickname);
```

```
69  if (iter == m_employees.cend()) {
70      cout << "Warning: No employee was found." << endl;
71      return 0;
72  }
73  //return salary of employee
74  return (**iter).Salary();
75 }
76
77 string Company::GetOldestEmployee() {
78     list<EUptr>::const_iterator itList = m_employees.cbegin();
79     //get nickname and birthday of first employee
80     string nickname = (**itList).GetNickname();
81     Employee::TDate birthday = (**itList).GetBirthday();
82
83     //loop through and check if the current employee's birthday is older than the
84     //last saved one.
85     for (itList = ++m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
86         if ((**itList).GetBirthday().year == birthday.year) {
87             if ((**itList).GetBirthday().month == birthday.month) {
88                 if ((**itList).GetBirthday().day < birthday.day) {
89                     nickname = (**itList).GetNickname();
90                     birthday = (**itList).GetBirthday();
91                 }
92             }
93             else if ((**itList).GetBirthday().year < birthday.year) {
94                 nickname = (**itList).GetNickname();
95                 birthday = (**itList).GetBirthday();
96             }
97         }
98         else if ((**itList).GetBirthday().year < birthday.year) {
99             nickname = (**itList).GetNickname();
100            birthday = (**itList).GetBirthday();
101        }
102    }
103    //return nickname of oldest employee
104    return nickname;
105 }
106
107 string Company::GetEmployeeWithLongestTimeInCompany() {
108     list<EUptr>::const_iterator itList = m_employees.cbegin();
109     //get nickname and joinDate of first employee
110     string nickname = (**itList).GetNickname();
111     Employee::TDate joinDate = (**itList).GetDateOfJoining();
112
113     //loop through and check if the current employee's joinDate is older than the
114     //last saved one.
115     for (itList = ++m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
116         if ((**itList).GetDateOfJoining().year == joinDate.year) {
117             if ((**itList).GetDateOfJoining().month == joinDate.month) {
118                 if ((**itList).GetDateOfJoining().day < joinDate.day) {
119                     nickname = (**itList).GetNickname();
120                     joinDate = (**itList).GetDateOfJoining();
121                 }
122             }
123             else if ((**itList).GetDateOfJoining().year < joinDate.year) {
124                 nickname = (**itList).GetNickname();
125                 joinDate = (**itList).GetDateOfJoining();
126             }
127         }
128         else if ((**itList).GetDateOfJoining().year < joinDate.year) {
129             nickname = (**itList).GetNickname();
130             joinDate = (**itList).GetDateOfJoining();
131         }
132     }
133     //return nickname of oldest employee
134     return nickname;
135 }
136
137 int Company::CountEmployees() {
138     //return how many employees are in the company
139     return m_employees.size();
140 }
141 }
```



```

142 int Company::CountEmployees(wBase type) {
143     //compare types
144     auto PredType = [type](EUptr const& e) {
145         return (type == (*e).GetType());
146     };
147     //count if types are equal
148     return count_if(m_employees.begin(), m_employees.end(), PredType);
149 }
150
151 int Company::CountEmployeesOlderThan(int year){
152     //compare birthday year
153     auto PredBirthday = [year](EUptr const& e) {
154         return (year > (*e).GetBirthday().year);
155     };
156     //count if types are older than year
157     return count_if(m_employees.begin(), m_employees.end(), PredBirthday);
158 }
159
160 void Company::Print() {
161     list<EUptr>::const_iterator itList;
162     cout << "*****" << endl;
163     cout << m_name << ", " << m_location << endl;
164     cout << "*****" << endl;
165     cout << "Datenblatt" << endl;
166     cout << "-----" << endl;
167     for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
168         cout << endl;
169         (**itList).Print();
170     }
171     cout << "-----" << endl;
172     cout << "v1.0 Oktober 2019" << endl;
173     cout << "-----" << endl;
174 }
175
176 void Company::AddEmployee(EUptr e) {
177     m_employees.emplace_back(move(e));
178 }
179
180 void Company::DeleteEmployee(EUptr e) {
181     try {
182         std::string nickname = (*e).GetNickname();
183         EIter iter = FindEmployee(nickname);
184
185         if (iter == m_employees.cend()) {
186             throw exception(" Delete failed : The employee is not registered in this company!");
187         }
188         else {
189             m_employees.erase(iter);
190         }
191     }
192     catch (exception const& ex) {
193         cerr << ex.what() << endl;
194     }
195 }
196
197 EIter Company::FindEmployee(string nickname) {
198     //compare nicknames
199     auto PredBirthday = [nickname](unique_ptr<Employee> const& e) {
200         return (nickname == (*e).GetNickname());
201     };
202     //find the correct employee by nickname
203     return find_if(m_employees.begin(), m_employees.end(), PredBirthday);
204 }

```

Das kann nicht ever Ernst sein

*(*var). func() $\hat{=}$ var -> func()*

-0,5

6.4 Class Employee

6.4.1 Employee.h

```

1  /* -----
2  | Workfile : Employee.h
3  | Description : [ HEADER ] Baseclass for derived classes (diff worktype)
4  | Name : Daniel Weyrer          PKZ : S1820306044
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9
10
11 #ifndef EMPLOYEE_H
12 #define EMPLOYEE_H
13 #include "Object.h"
14 #include <string>
15 #include <time.h>
16 #include <iostream>
17 #include <algorithm>
18
19 enum class wBase { Boss, Hourly, Piece, Comission };
20 std::ostream& operator<<(std::ostream& ost, wBase const& base);
21
22 class Employee : public Object {
23 public:
24     //struct to save dates
25     typedef struct {
26         size_t day = 0;
27         size_t month = 0;
28         size_t year = 0;
29     } TDate;
30     //Overloaded output-operator
31     friend std::ostream& operator<<(std::ostream& ost, TDate const& date);
32     Employee() : m_firstname{ "" }, m_lastname{ "" }, m_nickname{ 0 }, m_SSN{ 0 } {}
33
34     //returns type of derived class
35     virtual wBase GetType() const = 0;
36
37     //returns specific salary (depends on type)
38     virtual double Salary() const = 0;
39
40     //pure virtual Getter/Setter Methods; Getter return 0 if requested value is not contained
41     //in the derived class!
42     virtual void SetProducedPieces(size_t const pieces) = 0;
43     virtual std::size_t GetProdPieces() const = 0;
44
45     //Set/Get sold pieces
46     virtual void SetSoldPieces(size_t const pieces) = 0;
47     virtual std::size_t GetSoldPieces() const = 0;
48
49     //Set/Get base salary
50     virtual void SetBaseSalary(double const baseSalary) = 0;
51     virtual double GetBaseSalary() const = 0;
52
53     //Set/Get Workinghours
54     virtual void SetWorkingHours(double const hours) = 0;
55     virtual double GetWorkingHours() const = 0;
56
57     //Set/Get wage per hour
58     virtual void SetHourlyWage(double const wage) = 0;
59     virtual double GetHourlyWage() const = 0;
60
61     //Set/Get wage per Piece
62     virtual void SetWagePPiece(double const wage) = 0;
63     virtual double GetWagePPiece() const = 0;
64
65     //Prints Base and Derived Class
66     virtual void Print();
67
68     //Getter/Setter for Baseclass

```

```
69 void SetFirstname(std::string const& firstname);
70 std::string GetFirstname();
71
72 void SetLastname(std::string const& lastname);
73 std::string GetLastname() const;
74
75 void SetNickname(std::string const& nickname);
76 std::string GetNickname() const;
77
78 //Sets SSN after its validation
79 void setSSN(std::string const& ssn);
80 std::string GetSSN() const;
81
82 //Sets Birthday after a Date and Age-Verification
83 void SetBirthday(TDate const& birthday);
84 TDate GetBirthday() const;
85
86 //Sets Date of Joining after a Date-Verification
87 TDate GetDateOfJoining() const;
88 void SetDateOfJoining(TDate const& dateOfJoining);
89 void SetDateOfJoining(std::size_t day, std::size_t month, std::size_t year);
90
91 //overloaded ==-Operator (nickname is unique)
92 bool operator ==(Employee const&);
93
94 private:
95     std::string m_firstname;
96     std::string m_lastname;
97     std::string m_nickname;
98     std::string m_SSN;
99     TDate m_birthday;
100    TDate m_dateOfJoining;
101
102    //returns true if Date-format is valid and not in the future
103    bool isDateValid(TDate const& date);
104
105    //returns true if the string contains only numbers and is 10 digits long
106    bool isSSNValid(std::string const& ssn);
107
108    //returns the created struct based on the given values
109    TDate MakeDate(std::size_t day, std::size_t month, std::size_t year);
110
111 };
112
113 #endif //EMPLOYEE_H
```

6.4.2 Employee.cpp

```
1  /* -----
2  | Workfile : Employee.cpp
3  | Description : [ SOURCE ] Implementation of Baseclass
4  | Name : Daniel Weyrer          PKZ : S1820306044
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9
10 #include "Employee.h"
11
12 //minimum age in Austria is 15 years!
13 size_t static const minimumAge = 15;
14
15
16 void Employee::SetNickname(std::string const& nickname) {
17     m_nickname = nickname;
18 }
19
20 std::string Employee::GetNickname() const {
21     return m_nickname;
22 }
23
24
25 void Employee::setSSN(std::string const& ssn) {
26     try {
27         if (isSSNValid(ssn)) {
28             m_SSN = ssn;
29         }
30         else {
31             throw("SSN invalid!");
32         }
33     }
34     catch (std::exception const& ex) {
35         std::cerr << "SSN-Exception: " << ex.what() << std::endl;
36     }
37 }
38
39 std::string Employee::GetSSN() const {
40     return m_SSN;
41 }
42
43 void Employee::SetBirthday(Employee::TDate const& birthday) {
44     try {
45         if (isDateValid(birthday)) {
46             //get current date (time-library)
47             time_t now = time(0);
48             tm ltm;
49             localtime_s(&ltm, &now);
50
51             //Worker needs to be older than the minimum Age; ltm.tm_years = years since 1900!
52             if (((ltm.tm_year + 1900) - minimumAge) >= birthday.year) {
53                 m_birthday = birthday;
54             }
55             else {
56                 throw std::exception ("Employee is not allowed to work yet!");
57             }
58         }
59         else {
60             throw std::exception ("Entered date is invalid");
61         }
62     }
63     catch (std::exception const& ex) {
64         std::cerr << "Date-Exception: " << ex.what() << std::endl;
65     }
66 }
67
68 Employee::TDate Employee::GetBirthday() const {
69     return m_birthday;
70 }
```

```
71
72 void Employee::SetDateOfJoining(Employee::TDate const& dateOfJoining) {
73     try {
74         if (isDateValid(dateOfJoining)) {
75             m_dateOfJoining = dateOfJoining;
76         }
77         else {
78             throw std::exception ("Entered Date is invalid");
79         }
80     }
81     catch (std::exception const& ex) {
82         std::cerr << "Date-Exception: " << ex.what() << std::endl;
83     }
84 }
85
86 void Employee::SetDateOfJoining(std::size_t day, std::size_t month, std::size_t year) {
87     SetDateOfJoining(MakeDate(day, month, year));
88 }
89
90 bool Employee::operator==(Employee const& e) {
91     return (this->GetNickname() == e.GetNickname());
92 }
93
94 Employee::TDate Employee::MakeDate(std::size_t day, std::size_t month, std::size_t year) {
95     TDate tmp;
96     tmp.day = day; tmp.month = month; tmp.year = year;
97     return tmp;
98 }
99
100 bool Employee::isDateValid(Employee::TDate const& date) {
101     //get current date
102     time_t now = time(0);
103     tm ltm;
104     localtime_s(&ltm, &now);
105
106     //gregorian dates started in 1582
107     if (!(1582 <= date.year)) {
108         return false;
109     }
110     if (!(1 <= date.month && date.month <= 12)) {
111         return false;
112     }
113     if (!(1 <= date.day && date.day <= 31)) {
114         return false;
115     }
116     //Months with 30 days
117     if ((date.day == 31) && (date.month == 4 || date.month == 6 || date.month == 9 || date.month == 11)) {
118         return false;
119     }
120     //february has a max of 29 days in a leap year
121     if ((date.day == 30) && (date.month == 2)) {
122         return false;
123     }
124     //Leap-day at 29th of february every 4 years, but not every hundredth year (to keep it in sync
125     //with earth-rotation)
126     if ((date.month == 2) && (date.day == 29) && (date.year % 4 != 0)) {
127         return false;
128     }
129     if ((date.month == 2) && (date.day == 29) && (date.year % 400 == 0)) {
130         return true;
131     }
132     if ((date.month == 2) && (date.day == 29) && (date.year % 100 == 0)) {
133         return false;
134     }
135     //Check if current date is in the past
136     if (date.year > ltm.tm_year && date.month > ltm.tm_mon && date.day > ltm.tm_mday) {
137         return false;
138     }
139
140     return true;
141 }
```

```
142
143 bool Employee::isSSNValid(std::string const& ssn) {
144     if (ssn.length() != 10) {
145         return false;
146     }
147     auto PredSSN = [](char const c) {return (isdigit(c)); };
148     return std::all_of(ssn.cbegin(), ssn.cend(), PredSSN);
149 }
150
151 Employee::TDate Employee::GetDateOfJoining() const {
152     return m_dateOfJoining;
153 }
154 }
155
156 void Employee::Print() {
157     std::cout << "Name: " << this->GetFirstname() << " " << this->GetLastname() << std::endl;
158     std::cout << "Kürzel: " << this->GetNickname() << std::endl;
159     std::cout << "Sozialversicherungsnummer: " << this->GetSSN() << std::endl;
160     std::cout << "Einstiegsjahr: " << m_dateOfJoining.year << std::endl;
161 }
162
163 void Employee::SetFirstname(std::string const& firstname) {
164     m_firstname = firstname;
165 }
166
167 std::string Employee::GetFirstname() {
168     return m_firstname;
169 }
170
171 void Employee::SetLastname(std::string const& lastname) {
172     m_lastname = lastname;
173 }
174
175 std::string Employee::GetLastname() const {
176     return m_lastname;
177 }
178
179 //Overloaded Output-Operators for date struct and enum class
180
181 std::ostream& operator<<(std::ostream& ost, Employee::TDate const& date) {
182     if (ost.good()) {
183         ost << date.day << "." << date.month << "." << date.year;
184     }
185     return ost;
186 }
187
188 std::ostream& operator<<(std::ostream& ost, wBase const& base) {
189     if (ost.good()) {
190         switch (base) {
191             case wBase::Boss: ost << "Boss"; break;
192             case wBase::Hourly: ost << "HourlyWorker"; break;
193             case wBase::Piece: ost << "PieceWorker"; break;
194             case wBase::Comission: ost << "ComissionWorker"; break;
195         }
196     }
197     return ost;
198 }
```

6.5 Class CommissionWorker

6.5.1 CommissionWorker.h

```
1  /* -----
2  | Workfile : CommssionWorker.h
3  | Description : [ HEADER ] Derived Class CommissionWorker
4  | Name : Daniel Weyrer          PKZ : S1820306044
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9
10 #ifndef COMMISSIONWORKER_H
11 #define COMMISSIONWORKER_H
12
13 #include <string>
14
15 #include "Employee.h"
16
17
18 class CommissionWorker : public Employee {
19 public:
20     CommissionWorker() : m_soldPieces{ 0 }, m_wagePPiece{ 0 }, m_baseSalary{ 0 }{}
21     virtual wBase GetType() const override;
22     virtual double Salary() const override;
23
24     virtual void SetSoldPieces(size_t const pieces) override;
25     virtual std::size_t GetSoldPieces() const override;
26
27     virtual void SetBaseSalary(double const baseSalary) override;
28     virtual double GetBaseSalary() const override;
29
30     virtual void SetProducedPieces(size_t const pieces) override;
31     virtual std::size_t GetProdPieces() const override;
32
33     virtual void SetWorkingHours(double const hours) override;
34     virtual double GetWorkingHours() const override;
35
36     virtual void SetHourlyWage(double const wage);
37     virtual double GetHourlyWage() const;
38
39     virtual void SetWagePPiece(double const wage);
40     virtual double GetWagePPiece() const;
41
42     virtual void Print() override;
43
44 private:
45     size_t m_soldPieces;
46     double m_wagePPiece;
47
48     double m_baseSalary;
49 };
50 #endif //COMMISSIONWORKER_H
```

6.5.2 CommissionWorker.cpp

```

1  /* -----
2  | Workfile : CommissionWorker.cpp
3  | Description : [ SOURCE ] Derived Class CommissionWorker
4  | Name : Daniel Weyrer          PKZ : S1820306044
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9
10
11 #include "CommissionWorker.h"
12
13 wBase CommissionWorker::GetType() const {
14     return wBase::Comission;
15 }
16
17 double CommissionWorker::Salary() const {
18     return (m_baseSalary + (m_wagePPiece * m_soldPieces));
19 }
20
21 void CommissionWorker::SetSoldPieces(size_t const pieces) {
22     m_soldPieces = pieces;
23 }
24
25 std::size_t CommissionWorker::GetSoldPieces() const {
26     return m_soldPieces;
27 }
28
29 void CommissionWorker::SetBaseSalary(double const baseSalary) {
30     m_baseSalary = baseSalary;
31 }
32
33 double CommissionWorker::GetBaseSalary() const {
34     return m_baseSalary;
35 }
36
37 void CommissionWorker::SetProducedPieces(size_t const pieces) {
38 }
39
40 std::size_t CommissionWorker::GetProdPieces() const {
41     return 0;
42 }
43
44 void CommissionWorker::SetWorkingHours(double const hours) {
45 }
46
47 double CommissionWorker::GetWorkingHours() const {
48     return 0.0;
49 }
50
51 void CommissionWorker::SetHourlyWage(double const wage) {
52 }
53
54 double CommissionWorker::GetHourlyWage() const {
55     return 0.0;
56 }
57
58 void CommissionWorker::SetWagePPiece(double const wage) {
59     m_wagePPiece = wage;
60 }
61
62 double CommissionWorker::GetWagePPiece() const {
63     return m_wagePPiece;
64 }
65
66 void CommissionWorker::Print() {
67     Employee::Print();
68     std::cout << "Mitarbeiterklasse: " << this->GetType() << std::endl;
69     std::cout << "Grundgehalt: " << this->GetBaseSalary() << " EUR" << std::endl;
70     std::cout << "Provision: " << (this->GetSoldPieces() * this->GetWagePPiece()) << std::endl;

```

entweder auf <0 prüfen, oder size_t

-0,5

Unnötige Leerimplementierungen
gilt für alle Workerklassen

-7


```
71     std::cout << "Gehalt: " << this->Salary() << " EUR" << std::endl;  
72 }
```

6.6 Class HourlyWorker

6.6.1 HourlyWorker.h

```
1  /* -----  
2  | Workfile : HourlyWorker.h  
3  | Description : [ Header ] Derived Class HourlyWorker  
4  | Name : Daniel Weyrer          PKZ : S1820306044  
5  | Date : 04.11.2019  
6  | Remarks : -  
7  | Revision : 0  
8  | ----- */  
9  
10  
11 #ifndef HOURLYWORKER_H  
12 #define HOURLYWORKER_H  
13  
14 #include "Employee.h"  
15 #include <string>  
16  
17 class HourlyWorker : public Employee {  
18 public:  
19     HourlyWorker() : m_workingHours{ 0 }, m_hourlyWage{ 0 }{}  
20  
21     virtual wBase GetType() const override;  
22     virtual double Salary() const override;  
23  
24     virtual void SetProducedPieces(size_t const pieces) override;  
25     virtual std::size_t GetProdPieces() const override;  
26  
27     virtual void SetSoldPieces(size_t const pieces) override;  
28     virtual std::size_t GetSoldPieces() const override;  
29  
30     virtual void SetBaseSalary(double const baseSalary) override;  
31     virtual double GetBaseSalary() const override;  
32  
33     virtual void SetWorkingHours(double const hours) override;  
34     virtual double GetWorkingHours() const override;  
35  
36     virtual void SetHourlyWage(double const wage);  
37     virtual double GetHourlyWage() const;  
38  
39     virtual void SetWagePPiece(double const wage);  
40     virtual double GetWagePPiece() const;  
41  
42     virtual void Print() override;  
43  
44 private:  
45     double m_workingHours;  
46     double m_hourlyWage;  
47 };  
48  
49 #endif //HOURLYWORKER_H
```

6.6.2 HourlyWorker.cpp

```
1  /* -----  
2  | Workfile : HourlyWorker.cpp  
3  | Description : [ SOURCE ] Derived Class HourlyWorker  
4  | Name : Daniel Weyrer          PKZ : S1820306044  
5  | Date : 04.11.2019  
6  | Remarks : -  
7  | Revision : 0  
8  | ----- */  
9  
10  
11 #include "HourlyWorker.h"  
12  
13 wBase HourlyWorker::GetType() const {  
14     return wBase::Hourly;  
15 }  
16  
17 double HourlyWorker::Salary() const {  
18     double tmpWage;  
19     tmpWage = m_workingHours * m_hourlyWage;  
20     return tmpWage;  
21 }  
22  
23 void HourlyWorker::SetProducedPieces(size_t const pieces) {  
24 }  
25  
26 void HourlyWorker::SetSoldPieces(size_t const pieces) {  
27 }  
28  
29 void HourlyWorker::SetWorkingHours(double const hours) {  
30     m_workingHours = hours;  
31 }  
32  
33 std::size_t HourlyWorker::GetProdPieces() const {  
34     return 0;  
35 }  
36  
37 std::size_t HourlyWorker::GetSoldPieces() const {  
38     return 0;  
39 }  
40  
41 void HourlyWorker::SetBaseSalary(double const baseSalary) {  
42 }  
43  
44 double HourlyWorker::GetBaseSalary() const {  
45     return 0.0;  
46 }  
47  
48 double HourlyWorker::GetWorkingHours() const {  
49     return m_workingHours;  
50 }  
51  
52 void HourlyWorker::SetHourlyWage(double const wage) {  
53     m_hourlyWage = wage;  
54 }  
55  
56 double HourlyWorker::GetHourlyWage() const {  
57     return m_hourlyWage;  
58 }  
59  
60 void HourlyWorker::SetWagePPiece(double const wage) {  
61 }  
62  
63 double HourlyWorker::GetWagePPiece() const {  
64     return 0;  
65 }  
66  
67 void HourlyWorker::Print() {  
68     Employee::Print();  
69     std::cout << "Mitarbeiterklasse: " << this->GetType() << std::endl;  
70     std::cout << "Arbeitsstunden: " << this->GetWorkingHours() << std::endl;
```

```

71  std::cout << "Stundenlohn: " << this->GetHourlyWage() << " EUR" << std::endl;
72  std::cout << "Gehalt: " << this->Salary() << " EUR" << std::endl;
73  }

```

6.7 Class PieceWorker

6.7.1 PieceWorker.h

```

1  /* -----
2  | Workfile : PieceWorker.h
3  | Description : [ HEADER ] Derived Class PieceWorker
4  | Name : Daniel Weyrer          PKZ : S1820306044
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9
10 #ifndef PIECEWORKER_H
11 #define PIECEWORKER_H
12 #include <string>
13
14 #include "Employee.h"
15
16 class PieceWorker : public Employee {
17 public:
18     PieceWorker() : m_prodPieces{ 0 }, m_wagePPiece{ 0 } {}
19
20     virtual wBase GetType() const override;
21     virtual double Salary() const override;
22
23     virtual void SetProducedPieces(size_t const pieces) override;
24     virtual std::size_t GetProdPieces() const override;
25
26     virtual void SetSoldPieces(size_t const pieces) override;
27     virtual std::size_t GetSoldPieces() const override;
28
29     virtual void SetBaseSalary(double const baseSalary) override;
30     virtual double GetBaseSalary() const override;
31
32     virtual void SetWorkingHours(double const hours) override;
33     virtual double GetWorkingHours() const override;
34
35     virtual void SetHourlyWage(double const wage) override;
36     virtual double GetHourlyWage() const override;
37
38     virtual void SetWagePPiece(double const wage) override;
39     virtual double GetWagePPiece() const override;
40
41     virtual void Print() override;
42
43 private:
44     std::size_t m_prodPieces;
45     double m_wagePPiece;
46 };
47
48 #endif //PIECEWORKER_H

```

6.7.2 PieceWorker.cpp

```
1  /* -----  
2  | Workfile : PieceWorker.cpp  
3  | Description : [ SOURCE ] Derived Class PieceWorker  
4  | Name : Daniel Weyrer          PKZ : S1820306044  
5  | Date : 04.11.2019  
6  | Remarks : -  
7  | Revision : 0  
8  | ----- */  
9  
10  
11 #include "PieceWorker.h"  
12  
13 wBase PieceWorker::GetType() const {  
14     return wBase::Piece;  
15 }  
16  
17 double PieceWorker::Salary() const {  
18     return (m_wagePPiece * m_prodPieces);  
19 }  
20  
21 void PieceWorker::SetProducedPieces(size_t const pieces) {  
22     m_prodPieces = pieces;  
23 }  
24  
25 std::size_t PieceWorker::GetProdPieces() const {  
26     return m_prodPieces;  
27 }  
28  
29 void PieceWorker::SetSoldPieces(size_t const pieces) {  
30 }  
31  
32 std::size_t PieceWorker::GetSoldPieces() const {  
33     return 0;  
34 }  
35  
36 void PieceWorker::SetBaseSalary(double const baseSalary) {  
37 }  
38  
39 double PieceWorker::GetBaseSalary() const {  
40     return 0.0;  
41 }  
42  
43 void PieceWorker::SetWorkingHours(double const hours) {  
44 }  
45  
46 double PieceWorker::GetWorkingHours() const {  
47     return 0.0;  
48 }  
49  
50 void PieceWorker::SetHourlyWage(double const wage) {  
51 }  
52  
53 double PieceWorker::GetHourlyWage() const {  
54     return 0.0;  
55 }  
56  
57 void PieceWorker::SetWagePPiece(double const wage) {  
58     m_wagePPiece = wage;  
59 }  
60  
61 double PieceWorker::GetWagePPiece() const {  
62     return m_wagePPiece;  
63 }  
64  
65 void PieceWorker::Print() {  
66     Employee::Print();  
67     std::cout << "Mitarbeiterklasse: " << this->GetType() << std::endl;  
68     std::cout << "Stückzahl: " << this->GetProdPieces() << std::endl;  
69     std::cout << "Stückwert: " << this->GetWagePPiece() << " EUR" << std::endl;  
70     std::cout << "Gehalt: " << this->Salary() << " EUR" << std::endl;
```

71 }

6.8 Class Boss

6.8.1 Boss.h

```
1  /* -----
2  | Workfile : Boss.h
3  | Description : [ HEADER ] Derived Class Boss
4  | Name : Daniel Weyrer          PKZ : S1820306044
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9
10 #ifndef BOSS_H
11 #define BOSS_H
12
13 #include <string>
14 #include "Employee.h"
15
16 class Boss : public Employee {
17 public:
18     Boss() : m_baseSalary{ 0 } {}
19     virtual ~Boss() override = default;
20
21     virtual wBase GetType() const override;
22     virtual double Salary() const override;
23
24     virtual void SetBaseSalary(double const baseSalary) override;
25     virtual double GetBaseSalary() const override;
26
27     virtual void SetProducedPieces(size_t const pieces) override;
28     virtual std::size_t GetProdPieces() const override;
29
30     virtual void SetSoldPieces(size_t const pieces) override;
31     virtual std::size_t GetSoldPieces() const override;
32
33
34     virtual void SetWorkingHours(double const hours) override;
35     virtual double GetWorkingHours() const override;
36
37     virtual void SetHourlyWage(double const wage) override;
38     virtual double GetHourlyWage() const override;
39
40     virtual void SetWagePPiece(double const wage) override;
41     virtual double GetWagePPiece() const override;
42
43     virtual void Print() override;
44
45 private:
46     double m_baseSalary;
47 };
48
49 #endif //BOSS_H
```

6.8.2 Boss.cpp

```
1  /* -----
2  | Workfile : Boss.cpp
3  | Description : [ SOURCE ] Derived Class Boss
4  | Name : Daniel Weyrer          PKZ : S1820306044
5  | Date : 04.11.2019
6  | Remarks : -
7  | Revision : 0
8  | ----- */
9
10 #include "Boss.h"
11
12 wBase Boss::GetType() const {
13     return wBase::Boss;
14 }
15
16 double Boss::Salary() const {
17     return m_baseSalary;
18 }
19
20 void Boss::SetBaseSalary(double const baseSalary) {
21     m_baseSalary = baseSalary;
22 }
23
24 double Boss::GetBaseSalary() const {
25     return m_baseSalary;
26 }
27
28 void Boss::SetProducedPieces(size_t const pieces) {
29 }
30
31 std::size_t Boss::GetProdPieces() const {
32     return 0;
33 }
34
35 void Boss::SetSoldPieces(size_t const pieces) {
36 }
37
38 std::size_t Boss::GetSoldPieces() const {
39     return 0;
40 }
41
42 void Boss::SetWorkingHours(double const hours) {
43 }
44
45 double Boss::GetWorkingHours() const {
46     return 0.0;
47 }
48
49 void Boss::SetHourlyWage(double const wage) {
50 }
51
52 double Boss::GetHourlyWage() const {
53     return 0.0;
54 }
55
56 void Boss::SetWagePPiece(double const wage) {
57 }
58
59 double Boss::GetWagePPiece() const {
60     return 0.0;
61 }
62
63 void Boss::Print() {
64     Employee::Print();
65     std::cout << "Mitarbeiterklasse: " << this->GetType() << std::endl;
66     std::cout << "Gehalt: " << this->Salary() << " EUR" << std::endl;
67 }
```

6.8.3 TestDriver.cpp

```
1  /* -----  
2  | Workfile : Testdriver.cpp  
3  | Description : [ SOURCE ] Main File for testing the program  
4  | Name : Viktoria Streibl      PKZ : S1810306013  
5  | Date : 04.11.2019  
6  | Remarks : -  
7  | Revision : 0  
8  | ----- */  
9  #include "Client.h"  
10 #include "Company.h"  
11  
12 #include "Boss.h"  
13 #include "CommissionWorker.h"  
14 #include "HourlyWorker.h"  
15 #include "PieceWorker.h"  
16  
17 void PrintTestTitle(std::string const subtitle);  
18 void TestLinzAG(Company* const linzag);  
19 void TestSequality(Company* const sequality);  
20 void TestTractive(Company* const tractive);  
21  
22 int main() {  
23     //create some companies  
24     Company linzag("Linz AG", "Linz");  
25     Company sequality("Sequality GmbH", "Hagenberg");  
26     Company tractive("Tractive", "Pasching");  
27  
28     Employee::TDate birthday;  
29     Employee::TDate joinDate;  
30  
31     //create some employees  
32     Boss b;  
33     b.SetFirstname("Christian");  
34     b.SetLastname("Grey");  
35     b.SetBaseSalary(4800);  
36     birthday.day = 12;  
37     birthday.month = 1;  
38     birthday.year = 1972;  
39     b.SetBirthDay(birthday);  
40     b.SetNickname("CGB");  
41     b.setSSN("1234512345");  
42     joinDate.day = 1;  
43     joinDate.month = 1;  
44     joinDate.year = 2001;  
45     linzag.AddEmployee(std::make_unique<Boss>(b));  
46     sequality.AddEmployee(std::make_unique<Boss>(b));  
47     tractive.AddEmployee(std::make_unique<Boss>(b));  
48  
49     CommissionWorker w;  
50     w.SetFirstname("Viktoria");  
51     w.SetLastname("Streibl");  
52     w.SetBaseSalary(2100);  
53     w.SetSoldPieces(11);  
54     w.SetWagePPiece(8);  
55     birthday.day = 29;  
56     birthday.month = 10;  
57     birthday.year = 1998;  
58     w.SetBirthDay(birthday);  
59     w.SetNickname("ViS");  
60     w.setSSN("1290012900");  
61     joinDate.day = 1;  
62     joinDate.month = 1;  
63     joinDate.year = 2010;  
64     w.SetDateOfJoining(joinDate);  
65     linzag.AddEmployee(std::make_unique<CommissionWorker>(w));  
66     sequality.AddEmployee(std::make_unique<CommissionWorker>(w));  
67  
68     HourlyWorker hw;  
69     hw.SetFirstname("Daniel");  
70     hw.SetLastname("Weyrer");  
71     hw.SetWorkingHours(80);  
72     hw.SetHourlyWage(13);  
73     birthday.day = 17;
```

*Denkt mal darüber nach einen
Konstruktor zu verwenden*

```

74 birthday.month = 1;
75 birthday.year = 1998;
76 hw.SetBirthday(birthday);
77 hw.SetNickname("DaW");
78 hw.setSSN("7733177331");
79 joinDate.day = 1;
80 joinDate.month = 1;
81 joinDate.year = 2015;
82 hw.SetDateOfJoining(joinDate);
83 linzag.AddEmployee(std::make_unique<HourlyWorker>(hw));
84 tractive.AddEmployee(std::make_unique<HourlyWorker>(hw));
85
86 PieceWorker pw;
87 pw.SetFirstname("John");
88 pw.SetLastname("Doe");
89 pw.SetProducedPieces(10);
90 pw.SetWagePPiece(5.0);
91 birthday.day = 28;
92 birthday.month = 4;
93 birthday.year = 1983;
94 pw.SetBirthday(birthday);
95 pw.SetNickname("JoD");
96 pw.setSSN("1230502539");
97 joinDate.day = 15;
98 joinDate.month = 7;
99 joinDate.year = 2003;
100 pw.SetDateOfJoining(joinDate);
101 linzag.AddEmployee(std::make_unique<PieceWorker>(pw));
102 tractive.AddEmployee(std::make_unique<PieceWorker>(pw));
103
104 //test companies
105 TestLinzAG(&linzag);
106 TestSequality(&sequality);
107 TestTractive(&tractive);
108
109 return 0;
110 }
111
112 //print subtitle of testcase
113 void PrintTestTitle(std::string const subtitle) {
114     std::cout << std::endl;
115     std::cout << "#####" << std::endl;
116     std::cout << subtitle << std::endl;
117     std::cout << "#####" << std::endl;
118 }
119
120 void TestLinzAG(Company* const linzag) {
121     PrintTestTitle("Client test for Linz AG");
122
123     ICompany* comp = dynamic_cast<ICompany*>(&(*linzag));
124     Client client_linzag(comp);
125     bool isLinzAGValid = true;
126
127     //test some functions withs client of linzag
128     isLinzAGValid = isLinzAGValid ? client_linzag.TestCompanyName("Linz AG") : false;
129     isLinzAGValid = isLinzAGValid ? client_linzag.TestCompanyLocation("Linz") : false;
130     isLinzAGValid = isLinzAGValid ? client_linzag.TestCountEmployees(4) : false;
131     isLinzAGValid = isLinzAGValid ? client_linzag.TestFindEmployeeByNickname("DaW") : false;
132     isLinzAGValid = isLinzAGValid ? client_linzag.TestCountEmployeesOlderThan(1990, 2) : false;
133     isLinzAGValid = isLinzAGValid ? client_linzag.TestGetOldestEmployee("CGB") : false;
134     isLinzAGValid = isLinzAGValid ? client_linzag.TestLongestTimeInCompany("CGB") : false;
135     client_linzag.TestPrintAll();
136
137     //check if everything works
138     if (isLinzAGValid) {
139         std::cout << "Everything OK..." << std::endl;
140     }
141     else {
142         std::cout << "Something failed..." << std::endl;
143     }
144 }
145
146 void TestSequality(Company* const sequality) {

```



```
147
148 PrintTestTitle("Client test for Sequality GmbH");
149
150 ICompany* comp = dynamic_cast<ICompany*>(&(*sequality));
151 Client client_sequality(comp);
152 bool isSequalityValid = true;
153
154 //test some functions withs client of sequality
155 isSequalityValid = isSequalityValid ? client_sequality.TestCompanyName("Sequality GmbH") : false;
156 isSequalityValid = isSequalityValid ? client_sequality.TestCompanyLocation("Hagenberg") : false;
157 isSequalityValid = isSequalityValid ? client_sequality.TestCountEmployeesByType(wBase::Comission,
158                                     1) : false;
159 isSequalityValid = isSequalityValid ? client_sequality.TestCountTotalSoldPieces(11) : false;
160 isSequalityValid = isSequalityValid ? client_sequality.TestGetSalaryOfEmployee("ViS", 2188) :
161                                     false;
162 client_sequality.TestPrintAll();
163
164 //check if everything works
165 if (isSequalityValid) {
166     std::cout << "Everything OK..." << std::endl;
167 }
168 else {
169     std::cout << "Something failed..." << std::endl;
170 }
171
172 void TestTractive(Company* const tractive) {
173     PrintTestTitle("Client test for Tractive");
174
175     ICompany* comp = dynamic_cast<ICompany*>(&(*tractive));
176     Client client_tractive(comp);
177     bool isTractiveValid = true;
178
179     //test some functions withs client of tractive
180     isTractiveValid = isTractiveValid ? client_tractive.TestCompanyName("Tractive") : false;
181     isTractiveValid = isTractiveValid ? client_tractive.TestCompanyLocation("Pasching") : false;
182     isTractiveValid = isTractiveValid ? client_tractive.TestCountTotalProducesPieces(10) : false;
183     isTractiveValid = isTractiveValid ? client_tractive.TestGetSalaryOfEmployee("DaW", 1040) : false;
184     client_tractive.TestPrintAll();
185
186     //check if everything works
187     if (isTractiveValid) {
188         std::cout << "Everything OK..." << std::endl;
189     }
190     else {
191         std::cout << "Something failed..." << std::endl;
192     }
193 }
```