FH-OÖ Hagenberg/HSD
SDP3, WS 2019
Übung 2

Name(1): Daniel Weyrer — Abgabetermin: 5.11.2019

Name(2): Viktoria Streibl — Punkte:

Übungsgruppe: Gruppe 1 — korrigiert:

Geschätzter Aufwand in Ph: 10 | 10 — Effektiver Aufwand in Ph:

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

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: - ph

1.3.2 Daniel Weyrer

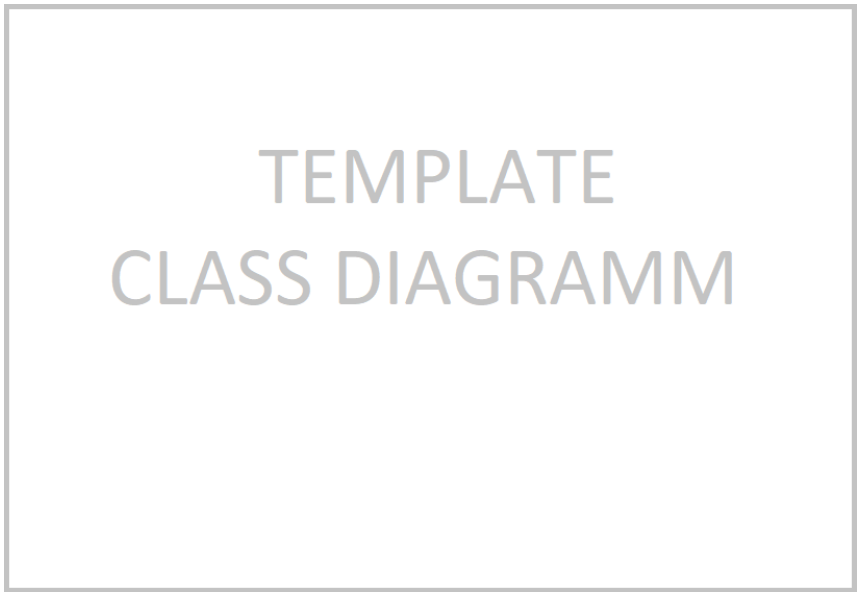
- estimated: 10 ph
- actually: - ph

2 Requirenment 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

3.2.2

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:

- Test the company name
- Test the company location

- Test if it is possible ot find a employee by nickname
- Test if it is possible ot find a employee by birthday

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 employees by birthday
- Get all sold pieces
- Get all produced pieces
- Count all employees in the company
- Count all employees with the same age
- Ouput 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 methodes 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 with the birthday and returns the employees. "GetSoldPieces", counts all pieces which are sold by the company. "GetProdPieces", counts all pieces which are produces by the employees. "CountEmployees", returns the number of employees in the company. "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

The Company class manages all Employees. It uses unique pointers stored in a vector, to avoid shallow-copies. With the method "AddEmploye" 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.

4.4 Class Employee

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

-

4.5 Class CommissionWorker

This class represents a comission worker.

-

4.6 Class HourlyWorker

This class represents a hourly worker.

-

4.7 Class PiecesWorker

This class represents a pieces worker.

-

4.8 Class Boss

This class represents a boss.

-

4.9 TestDriver

The Testdriver test alle functions of the Client. It adds commisson worker, hourly worker, pieces worker and a boss and deletes them. It searches employees by nickname and birthday and print all of them.

5 Test Protocol

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

-

5.1 Console Output

6 Source Code

6.1 Class Client

6.1.1 Client.h

```
/* ------------------------------------------------------------------
| Workfile : Client .h
| Description : [ HEADER ] Class for the Client to act with an Company
| Name : Viktoria Streibl     PKZ : S1810306013
| Date : 04.11.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------ */
#ifndef CLIENT_H
#define CLIENT_H

#include "ICompany.h"

class Client : public Object {

public:
  Client(ICompany* const company);
  ~Client() = default;

  //tests if the company name is correct
  bool TestCompanyName(std::string expectedName) const;
  //tests if the company location is correct
  bool TestCompanyLocation(std::string expectedLocation) const;
  //tests if the employee which is search by his nickname is correct
  bool TestFindEmployeeByNickname(std::string nickname) const;
  //counts all employees and check if it is correct
  bool TestCountEmployees(int expectedResult) const;
  //counts all employees of one type and check it
  bool TestCountEmployeesByType(wBase type, int expectedResult) const;
  //count all procuded pieces and check them
  bool TestCountTotalProducesPieces(int expectedResult) const;
  //count all sold pieces and check them
  bool TestCountTotalSoldPieces(int expectedResult) const;
  //count how many employees are older than a specific year and check it
  bool TestCountEmployeesOlderThan(int year, int expectedResult) const;
  //tests if the salary of an employee is correct
  bool TestGetSalaryOfEmployee(std::string nickname, double expectedResult) const;
  //tests if the oldest employee is correct
  bool TestGetOldestEmployee(std::string expectedNickname) const;
  //check for the employee with is the oldest member
  bool TestLongestTimeInCompany(std::string expectedNickname) const;
  //let print all data of company and employees
  bool TestPrintAll() const;

private:
  ICompany* m_company;

  void ErrorMsg(std::string msg) const;
};
#endif //CLIENT_H
```

6.1.2 Client.cpp

```cpp
/* ---------------------------------------------------------------------
| Workfile : Client.cpp
| Description : [ SOURCE ] Class for the Client to act with an Company
| Name : Viktoria Streibl     PKZ : S1810306013
| Date : 04.11.2019
| Remarks : -
| Revision : 0
| --------------------------------------------------------------------- */
#include "Client.h"

Client::Client(ICompany* const company) {
  m_company = company;
}

bool Client::TestCompanyName(std::string expectedName) const{
  //get company name and compare with expected name
  std::string name = m_company->GetCompanyName();
  if (name == expectedName) {
    return true;
  }
  else {
    ErrorMsg("Company Name was wrong");
    return false;
  }
}
bool Client::TestCompanyLocation(std::string expectedLocation) const {
  //get company location and compare with expected location
  std::string name = m_company->GetCompanyLocation();
  if (name == expectedLocation) {
    return true;
  }
  else {
    ErrorMsg("Company Location was wrong");
    return false;
  }
}
bool Client::TestFindEmployeeByNickname(std::string nickname) const {
  //search for employee by nickname and print it
  m_company->GetEmployee(nickname);
  return true;
}

bool Client::TestCountEmployees(int expectedResult) const {
  //get number of employees and compare with expected result
  int currEmployees = m_company->CountEmployees();
  if (expectedResult == currEmployees) {
    return true;
  }
  else {
    ErrorMsg("Number of employees was wrong");
    return false;
  }
}

bool Client::TestCountEmployeesByType(wBase type, int expectedResult) const {
  //get number of employees of specific type and compare with expected result
  int currEmployees = m_company->CountEmployees(type);
  if (expectedResult == currEmployees) {
    return true;
  }
  else {
    ErrorMsg("Numbers of employees by the same type was wrong");
    return false;
  }
}

bool Client::TestCountTotalProducesPieces(int expectedResult) const {
  //get total produced pieces and compare with expected result
  int totalProdPieces = m_company->GetProdPieces();
  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 }
```

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
```

```
| Date : 04.11.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------------ */
#ifndef COMPANY_H
#define COMPANY_H

#include <string>
#include <list>

#include "ICompany.h"
#include "Employee.h"

typedef std::unique_ptr<Employee> EUptr;
typedef std::list<EUptr>::const_iterator EIter;

class Company : public ICompany
{

public:
  //create company with name and location
  Company(std::string const name, std::string const location);
  ~Company() = default;

  //returns the name of the company
  std::string GetCompanyName() override;
  //returns the location of the company
  std::string GetCompanyLocation() override;
  //print a employee found by the nickname
  void GetEmployee(std::string const nickname) override;
  //print all employees of the type
  void GetEmployee(wBase type) override;
  //return total sold pieces last month
  int GetSoldPieces() override;
  //return total produced pieces last month
  int GetProdPieces() override;
  //return the salary of the employee
  double GetSalaryOfEmployee(std::string nickname) override;
  //returns the nickname of the oldest employee
  std::string GetOldestEmployee() override;
  //check for the employee with is the oldest member
  std::string GetEmployeeWithLongestTimeInCompany() override;
  //returns the number of employees in the company
  int CountEmployees() override;
  //returns the number of employees of a specific type in the company
  int CountEmployees(wBase type) override;
  //returns the number of employees older than a specific year
  int CountEmployeesOlderThan(int year) override;
  //print all data of the company and employees
  void Print() override;

  //add an employee
  void AddEmployee(EUptr e);
  //delete an employee
  void DeleteEmployee(EUptr e);

private:
  std::string m_name;
  std::string m_location;
  std::list<EUptr> m_employees;

  //find an employee by nickname
  EIter FindEmployee(std::string nickname);
};
#endif //COMPANY_H
```

6.3.2 Company.cpp

```
/* ------------------------------------------------------------------------
| Workfile : Company.cpp
| Description : [ SOURCE ] Class Company to store all data
| Name : Viktoria Streibl     PKZ : S1810306013
```

```
| Date : 04.11.2019
| Remarks : -
| Revision : 0
| ------------------------------------------------------------------------ */
#include "Company.h"

using namespace std;

Company::Company(std::string const name, std::string const location) {
  m_name = name;
  m_location = location;
}

string Company::GetCompanyName() {
  return m_name;
}
string Company::GetCompanyLocation() {
  return m_location;
}

void Company::GetEmployee(std::string const nickname) {
  EIter itList;
  //loop through list and search for nickname
  for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
    if (nickname == (**itList).GetNickname()) {
      (**itList).Print();
    }
  }
}

void Company::GetEmployee(wBase type){
  EIter itList;
  //loop through list and count every employee with the same type
  for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
    if (type == (**itList).GetType()) {
      (**itList).Print();
    }
  }
}

int Company::GetSoldPieces() {
  int sumSoldPieces = 0;

  list<EUptr>::const_iterator itList;
  //loop through list and sum all sold pieces
  for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
    sumSoldPieces += (**itList).GetSoldPieces();
  }
  return sumSoldPieces;
}

int Company::GetProdPieces() {
  int sumProdPieces = 0;
  list<EUptr>::const_iterator itList;
  //loop through list and sum all produced pieces
  for (itList = m_employees.cbegin(); itList != m_employees.cend(); ++itList) {
    sumProdPieces += (**itList).GetProdPieces();
  }
  return sumProdPieces;
}

double Company::GetSalaryOfEmployee(std::string nickname) {
  //get nickname of expected employee
  EIter iter = FindEmployee(nickname);
  if (iter == m_employees.cend()) {
    cout << "Warning: No employee was found." << endl;
    return 0;
  }
  //return salary of employee
  return (**iter).Salary();
}

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 }
```

6.4 Class Employee

6.4.1 Employee.h

```
1  #ifndef EMPLOYEE_H
2  #define EMPLOYEE_H
3  #include "Object.h"
4  #include <string>
5  #include <time.h>
6  #include <iostream>
7  #include <algorithm>
8
9  enum class wBase { Boss, Hourly, Piece, Comission };
10 std::ostream& operator<<(std::ostream& ost, wBase const& base);
11
12 class Employee : public Object {
13 public:
```

```
14   //struct to save dates
15   typedef struct  {
16     size_t day;
17     size_t month;
18     size_t year;
19   } TDate;
20   //Overloaded output-operator
21   friend std::ostream& operator<<(std::ostream& ost, TDate const& date);
22   Employee() = default;
23
24   //returns type of derived class
25   virtual wBase GetType() const = 0;
26
27   //returns specific salary (depends on type)
28   virtual double Salary() const = 0;
29
30   //pure virtual Getter/Setter Methods; Getter return 0 if requested value is not contained
31   //in the derived class!
32   virtual void SetProducedPieces(size_t const pieces) = 0;
33   virtual std::size_t GetProdPieces() const = 0;
34
35   virtual void SetSoldPieces(size_t const pieces) = 0;
36   virtual std::size_t GetSoldPieces() const = 0;
37
38   virtual void SetBaseSalary(double const baseSalary) = 0;
39   virtual double GetBaseSalary() const = 0;
40
41   virtual void SetWorkingHours(double const hours) = 0;
42   virtual double GetWorkingHours() const = 0;
43
44   virtual void SetHourlyWage(double const wage) = 0;
45   virtual double GetHourlyWage() const = 0;
46
47   virtual void SetWagePPiece(double const wage) = 0;
48   virtual double GetWagePPiece() const = 0;
49
50   //Prints Base and Derived Class
51   virtual void Print();
52
53   //Getter/Setter for Baseclass
54   void SetFirstname(std::string const& firstname);
55   std::string GetFirstname();
56
57   void SetLastname(std::string const& lastname);
58   std::string GetLastname() const;
59
60   void SetNickname(std::string const& nickname);
61   std::string GetNickname() const;
62
63   void setSSN(std::string const& ssn);
64   std::string GetSSN() const;
65
66   void SetBirthday(TDate const& birthday);
67   TDate GetBirthday() const;
68
69   TDate GetDateOfJoining() const;
70   void SetDateOfJoining(TDate const& dateOfJoining);
71   void SetDateOfJoining(std::size_t day, std::size_t month, std::size_t year);
72
73   //overloaded ==-Operator (nickname is unique)
74   bool operator ==(Employee const&);
75
76 private:
77   std::string m_firstname;
78   std::string m_lastname;
79   std::string m_nickname;
80   std::string m_SSN;
81   TDate m_birthday;
82   TDate m_dateOfJoining;
83
84   //returns true if Date-format is valid and not in the future
85   bool isDateValid(TDate const& date);
86
```

```
87   //returns true if the string contains only numbers and is 10 digits long
88   bool isSSNValid(std::string const& ssn);
89
90   //returns the created struct based on the given values
91   TDate MakeDate(std::size_t day, std::size_t month, std::size_t year);
92
93 };
94
95 #endif //EMPLOYEE_H
```

6.4.2 Employee.cpp

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

```
60 }
61
62 void Employee::SetDateOfJoining(Employee::TDate const& dateOfJoining) {
63   try {
64     if (isDateValid(dateOfJoining)) {
65       m_dateOfJoining = dateOfJoining;
66     }
67     else {
68       throw std::exception ("Entered Date is invalid");
69     }
70   }
71   catch (std::exception const& ex) {
72     std::cerr << "Date-Exception: " << ex.what() << std::endl;
73   }
74 }
75
76 void Employee::SetDateOfJoining(std::size_t day, std::size_t month, std::size_t year) {
77   SetDateOfJoining(MakeDate(day, month, year));
78 }
79
80 bool Employee::operator==(Employee const& e) {
81   return (this->GetNickname() == e.GetNickname());
82 }
83
84 Employee::TDate Employee::MakeDate(std::size_t day, std::size_t month, std::size_t year) {
85   TDate tmp;
86   tmp.day = day; tmp.month = month; tmp.year = year;
87   return tmp;
88 }
89
90 bool Employee::isDateValid(Employee::TDate const& date) {
91   //get current date
92   time_t now = time(0);
93   tm ltm;
94   localtime_s(&ltm, &now);
95
96   //gregorian dates started in 1582
97   if (!(1582 <= date.year)) {
98     return false;
99   }
100  if (!(1 <= date.month && date.month <= 12)) {
101    return false;
102  }
103  if (!(1 <= date.day && date.day <= 31)) {
104    return false;
105  }
106  //Months with 30 days
107  if ((date.day == 31) && (date.month == 4 || date.month == 6 || date.month == 9 || date.month ==
       11)) {
108    return false;
109  }
110  //february has a max of 29 days in a leap year
111  if ((date.day == 30) && (date.month == 2)) {
112    return false;
113  }
114  //Leap-day at 29th of february every 4 years, but not every hundredth year (to keep it in sync
       with earth-rotation)
115  if ((date.month == 2) && (date.day == 29) && (date.year % 4 != 0)) {
116    return false;
117  }
118  if ((date.month == 2) && (date.day == 29) && (date.year % 400 == 0)) {
119    return true;
120  }
121  if ((date.month == 2) && (date.day == 29) && (date.year % 100 == 0)) {
122    return false;
123  }
124
125  //Check if current date is in the past
126  if (date.year > ltm.tm_year && date.month > ltm.tm_mon && date.day > ltm.tm_mday) {
127    return false;
128  }
129
130  return true;
```

```
131 }
132
133 bool Employee::isSSNValid(std::string const& ssn) {
134   if (ssn.length() != 10) {
135     return false;
136   }
137   auto PredSSN = [](char const c) {return (isdigit(c)); };
138   return std::all_of(ssn.cbegin(), ssn.cend(), PredSSN);
139 }
140
141 Employee::TDate Employee::GetDateOfJoining() const {
142   return m_dateOfJoining;
143
144 }
145
146 void Employee::Print() {
147   std::cout << "Name: " << this->GetFirstname() << " " << this->GetLastname() << std::endl;
148   std::cout << "Kürzel: " << this->GetNickname() << std::endl;
149   std::cout << "Sozialversicherungsnummer: " << this->GetSSN() << std::endl;
150   std::cout << "Einstiegsjahr: " << this->GetDateOfJoining() << std::endl;
151 }
152
153 void Employee::SetFirstname(std::string const& firstname) {
154   m_firstname = firstname;
155 }
156
157 std::string Employee::GetFirstname() {
158   return m_firstname;
159 }
160
161 void Employee::SetLastname(std::string const& lastname) {
162   m_lastname = lastname;
163 }
164
165 std::string Employee::GetLastname() const {
166   return m_lastname;
167 }
168
169 //Overloaded Output-Operators for date struct and enum class
170
171 std::ostream& operator<<(std::ostream& ost, Employee::TDate const& date) {
172   if (ost.good()) {
173     ost << date.day << "." << date.month << "." << date.year;
174   }
175   return ost;
176 }
177
178 std::ostream& operator<<(std::ostream& ost, wBase const& base) {
179   if (ost.good()) {
180     switch (base) {
181     case wBase::Boss: ost << "Boss";
182     case wBase::Hourly: ost << "HourlyWorker";
183     case wBase::Piece: ost << "PieceWorker";
184     case wBase::Comission: ost << "ComissionWorker";
185     }
186   }
187   return ost;
188 }
```

6.5 Class CommissionWorker

6.5.1 CommissionWorker.h

```
1  #ifndef COMISSIONWORKER_H
2  #define COMISSIONWORKER_H
3
4  #include <string>
5
6  #include "Employee.h"
7
8
9  class CommissionWorker : public Employee {
10 public:
11   CommissionWorker() = default;
12   virtual wBase GetType() const override;
13   virtual double Salary() const override;
14
15   virtual void SetSoldPieces(size_t const pieces) override;
16   virtual std::size_t GetSoldPieces() const override;
17
18   virtual void SetBaseSalary(double const baseSalary) override;
19   virtual double GetBaseSalary() const override;
20
21   virtual void SetProducedPieces(size_t const pieces) override;
22   virtual std::size_t GetProdPieces() const override;
23
24   virtual void SetWorkingHours(double const hours) override;
25   virtual double GetWorkingHours() const override;
26
27   virtual void SetHourlyWage(double const wage);
28   virtual double GetHourlyWage() const;
29
30   virtual void SetWagePPiece(double const wage);
31   virtual double GetWagePPiece() const;
32
33   virtual void Print() override;
34
35 private:
36   size_t m_soldPieces;
37   double m_wagePPiece;
38
39   double m_baseSalary;
40 };
41 #endif //COMISSIONWORKER_H
```

6.5.2 CommissionWorker.cpp

```cpp
#include "CommissionWorker.h"

wBase CommissionWorker::GetType() const {
  return wBase::Comission;
}

double CommissionWorker::Salary() const {
  return m_baseSalary + m_wagePPiece * m_soldPieces;
}

void CommissionWorker::SetSoldPieces(size_t const pieces) {
  m_soldPieces = pieces;
}

std::size_t CommissionWorker::GetSoldPieces() const {
  return m_soldPieces;
}

void CommissionWorker::SetBaseSalary(double const baseSalary) {
  m_baseSalary = baseSalary;
}

double CommissionWorker::GetBaseSalary() const {
  return m_baseSalary;
}

void CommissionWorker::SetProducedPieces(size_t const pieces) {
}

std::size_t CommissionWorker::GetProdPieces() const {
  return 0;
}

void CommissionWorker::SetWorkingHours(double const hours) {
}

double CommissionWorker::GetWorkingHours() const {
  return 0.0;
}

void CommissionWorker::SetHourlyWage(double const wage) {
}

double CommissionWorker::GetHourlyWage() const {
  return 0.0;
}

void CommissionWorker::SetWagePPiece(double const wage) {
}

double CommissionWorker::GetWagePPiece() const {
  return 0.0;
}

void CommissionWorker::Print() {
  std::cout << "Mitarbeiterklasse: " << this->GetType() << std::endl;
  std::cout << "Grundgehalt: " << this->GetBaseSalary() << std::endl;
  std::cout << "Provision" << (this->GetSoldPieces()) * (this->GetWagePPiece()) << std::endl;
  std::cout << "Gehalt: " << this->Salary() << " EUR" << std::endl;
}
```

6.6 Class HourlyWorker

6.6.1 HourlyWorker.h

```cpp
#ifndef HOURLYWORKER_H
#define HOURLYWORKER_H

#include "Employee.h"
```

```
5  #include <string>
6
7  class HourlyWorker : public Employee {
8  public:
9    HourlyWorker() = default;
10
11   virtual wBase GetType() const override;
12   virtual double Salary() const override;
13
14   virtual void SetProducedPieces(size_t const pieces) override;
15   virtual std::size_t GetProdPieces() const override;
16
17   virtual void SetSoldPieces(size_t const pieces) override;
18   virtual std::size_t GetSoldPieces() const override;
19
20   virtual void SetBaseSalary(double const baseSalary) override;
21   virtual double GetBaseSalary() const override;
22
23   virtual void SetWorkingHours(double const hours) override;
24   virtual double GetWorkingHours() const override;
25
26   virtual void SetHourlyWage(double const wage);
27   virtual double GetHourlyWage() const;
28
29   virtual void SetWagePPiece(double const wage);
30   virtual double GetWagePPiece() const;
31
32   virtual void Print() override;
33
34
35 private:
36   double m_workingHours;
37   double m_hourlyWage;
38 };
39 #endif //HOURLYWORKER_H
```

6.6.2 HourlyWorker.cpp

```cpp
#include "HourlyWorker.h"

wBase HourlyWorker::GetType() const {
  return wBase::Hourly;
}

double HourlyWorker::Salary() const {
  double tmpWage;
  tmpWage = m_workingHours * m_hourlyWage;
  return tmpWage;
}

void HourlyWorker::SetProducedPieces(size_t const pieces) {
}

void HourlyWorker::SetSoldPieces(size_t const pieces) {
}

void HourlyWorker::SetWorkingHours(double const hours) {
  m_workingHours = hours;
}

std::size_t HourlyWorker::GetProdPieces() const {
  return 0;
}

std::size_t HourlyWorker::GetSoldPieces() const {
  return 0;
}

void HourlyWorker::SetBaseSalary(double const baseSalary) {
}

double HourlyWorker::GetBaseSalary() const {
  return 0.0;
}

double HourlyWorker::GetWorkingHours() const {
  return m_workingHours;
}

void HourlyWorker::SetHourlyWage(double const wage) {
  m_hourlyWage = wage;
}

double HourlyWorker::GetHourlyWage() const {
  return m_hourlyWage;
}

void HourlyWorker::SetWagePPiece(double const wage) {
}

double HourlyWorker::GetWagePPiece() const {
  return 0;
}

void HourlyWorker::Print() {
  std::cout << "Mitarbeiterklasse: " << this->GetType() << std::endl;
  std::cout << "Arbeitsstunden: " << this->GetWorkingHours() << std::endl;
  std::cout << "Stundenlohn: " << this->GetHourlyWage() << " EUR" << std::endl;
  std::cout << "Gehalt: " << this->Salary() << " EUR" << std::endl;
}
```

6.7 Class PieceWorker

6.7.1 PieceWorker.h

```cpp
#ifndef PIECEWORKER_H
#define PIECEWORKER_H
```

```
3  #include <string>
4
5  #include "Employee.h"
6
7  class PieceWorker : public Employee {
8  public:
9    PieceWorker() = default;
10
11   virtual wBase GetType() const override;
12   virtual double Salary() const override;
13
14   virtual void SetProducedPieces(size_t const pieces) override;
15   virtual std::size_t GetProdPieces() const override;
16
17   virtual void SetSoldPieces(size_t const pieces) override;
18   virtual std::size_t GetSoldPieces() const override;
19
20   virtual void SetBaseSalary(double const baseSalary) override;
21   virtual double GetBaseSalary() const override;
22
23   virtual void SetWorkingHours(double const hours) override;
24   virtual double GetWorkingHours() const override;
25
26   virtual void SetHourlyWage(double const wage) override;
27   virtual double GetHourlyWage() const override;
28
29   virtual void SetWagePPiece(double const wage) override;
30   virtual double GetWagePPiece() const override;
31
32   virtual void Print() override;
33
34
35 private:
36   std::size_t m_prodPieces;
37   double m_wagePPiece;
38 };
39
40 #endif //PIECEWORKER_H
```

6.7.2 PieceWorker.cpp

```cpp
#include "PieceWorker.h"

wBase PieceWorker::GetType() const {
  return wBase::Piece;
}

double PieceWorker::Salary() const {
  return (m_wagePPiece * m_prodPieces);
}

void PieceWorker::SetProducedPieces(size_t const pieces) {
  m_prodPieces = pieces;
}

std::size_t PieceWorker::GetProdPieces() const {
  return m_prodPieces;
}

void PieceWorker::SetSoldPieces(size_t const pieces) {
}

std::size_t PieceWorker::GetSoldPieces() const {
  return 0;
}

void PieceWorker::SetBaseSalary(double const baseSalary) {
}

double PieceWorker::GetBaseSalary() const {
  return 0.0;
}

void PieceWorker::SetWorkingHours(double const hours) {
}

double PieceWorker::GetWorkingHours() const {
  return 0.0;
}

void PieceWorker::SetHourlyWage(double const wage) {
}

double PieceWorker::GetHourlyWage() const {
  return 0.0;
}

void PieceWorker::SetWagePPiece(double const wage) {
}

double PieceWorker::GetWagePPiece() const {
  return 0.0;
}

void PieceWorker::Print() {
  std::cout << "Mitarbeiterklasse: " << this->GetType() << std::endl;
  std::cout << "Stückzahl: " << this->GetProdPieces() << std::endl;
  std::cout << "Stückwert: " << this->GetWagePPiece() << " EUR" << std::endl;
  std::cout << "Gehalt: " << this->Salary() << " EUR" << std::endl;
}
```

6.8 Class Boss

6.8.1 Boss.h

```cpp
#ifndef BOSS_H
#define BOSS_H

#include <string>
#include "Employee.h"
```

```
6
7  class Boss : public Employee {
8  public:
9    Boss() = default;
10   Boss(double const baseSalary);
11   virtual ~Boss() override = default;
12
13   virtual wBase GetType() const override;
14   virtual double Salary() const override;
15
16   virtual void SetBaseSalary(double const baseSalary) override;
17   virtual double GetBaseSalary() const override;
18
19   virtual void SetProducedPieces(size_t const pieces) override;
20   virtual std::size_t GetProdPieces() const override;
21
22   virtual void SetSoldPieces(size_t const pieces) override;
23   virtual std::size_t GetSoldPieces() const override;
24
25
26   virtual void SetWorkingHours(double const hours) override;
27   virtual double GetWorkingHours() const override;
28
29   virtual void SetHourlyWage(double const wage) override;
30   virtual double GetHourlyWage() const override;
31
32   virtual void SetWagePPiece(double const wage) override;
33   virtual double GetWagePPiece() const override;
34
35   virtual void Print() override;
36
37
38 private:
39   double m_baseSalary;
40 };
41
42 #endif //BOSS_H
```

6.8.2 Boss.cpp

```cpp
#include "Boss.h"

Boss::Boss(double const baseSalary) {
  m_baseSalary = baseSalary;
}

wBase Boss::GetType() const {
  return wBase::Boss;
}

double Boss::Salary() const {
  return m_baseSalary;
}

void Boss::SetBaseSalary(double const baseSalary) {
  m_baseSalary = baseSalary;
}

double Boss::GetBaseSalary() const {
  return m_baseSalary;
}

void Boss::SetProducedPieces(size_t const pieces) {
}

std::size_t Boss::GetProdPieces() const {
  return 0;
}

void Boss::SetSoldPieces(size_t const pieces) {
}

std::size_t Boss::GetSoldPieces() const {
  return 0;
}

void Boss::SetWorkingHours(double const hours) {
}

double Boss::GetWorkingHours() const {
  return 0.0;
}

void Boss::SetHourlyWage(double const wage) {
}

double Boss::GetHourlyWage() const {
  return 0.0;
}

void Boss::SetWagePPiece(double const wage) {
}

double Boss::GetWagePPiece() const {
  return 0.0;
}

void Boss::Print() {
  std::cout << "Mitarbeiterklasse: " << this->GetType() << std::endl;
  std::cout << "Gehalt: " << this->Salary() <<" EUR" << std::endl;
}
```

6.8.3 TestDriver.cpp

```cpp
/* ------------------------------------------------------------------
| Workfile : Testdriver.cpp
| Description : [ SOURCE ] Main File for testing the program
| Name : Viktoria Streibl     PKZ : S1810306013
| Date : 04.11.2019
| 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   Company linzag("Linz AG", "Linz");
24   Company sequality("Sequality GmbH", "Hagenberg");
25   Company tractive("Tractive", "Pasching");
26
27   Employee::TDate birthday;
28   Employee::TDate joinDate;
29
30   Boss b;
31   b.SetFirstname("Christian");
32   b.SetLastname("Grey");
33   b.SetBaseSalary(4800);
34   birthday.day = 12;
35   birthday.month = 1;
36   birthday.year = 1972;
37   b.SetBirthday(birthday);
38   b.SetNickname("CG-Boss");
39   b.setSSN("1234512345");
40   joinDate.day = 1;
41   joinDate.month = 1;
42   joinDate.year = 2001;
43   linzag.AddEmployee(std::make_unique<Boss>(b));
44   sequality.AddEmployee(std::make_unique<Boss>(b));
45   tractive.AddEmployee(std::make_unique<Boss>(b));
46
47   CommissionWorker w;
48   w.SetFirstname("Viktoria");
49   w.SetLastname("Streibl");
50   w.SetBaseSalary(2100);
51   w.SetSoldPieces(11);
52   birthday.day = 29;
53   birthday.month = 10;
54   birthday.year = 1998;
55   w.SetBirthday(birthday);
56   w.SetNickname("ViS");
57   w.setSSN("1290012900");
58   joinDate.day = 1;
59   joinDate.month = 1;
60   joinDate.year = 2010;
61   w.SetDateOfJoining(joinDate);
62   linzag.AddEmployee(std::make_unique<CommissionWorker>(w));
63   sequality.AddEmployee(std::make_unique<CommissionWorker>(w));
64
65   HourlyWorker hw;
66   hw.SetFirstname("Daniel");
67   hw.SetLastname("Weyrer");
68   hw.SetWorkingHours(80);
69   hw.SetHourlyWage(13);
70   birthday.day = 17;
71   birthday.month = 1;
72   birthday.year = 1998;
73   hw.SetBirthday(birthday);
74   hw.SetNickname("DaW");
75   hw.setSSN("7733177331");
76   joinDate.day = 1;
77   joinDate.month = 1;
78   joinDate.year = 2015;
79   hw.SetDateOfJoining(joinDate);
```

```
80    linzag.AddEmployee(std::make_unique<HourlyWorker>(hw));
81    tractive.AddEmployee(std::make_unique<HourlyWorker>(hw));
82
83    PieceWorker pw;
84    pw.SetFirstname("John");
85    pw.SetLastname("Doe");
86    pw.SetProducedPieces(10);
87    pw.SetWagePPiece(5.0);
88    birthday.day = 28;
89    birthday.month = 4;
90    birthday.year = 1983;
91    pw.SetBirthday(birthday);
92    pw.SetNickname("JoD");
93    pw.setSSN("1230502539");
94    joinDate.day = 15;
95    joinDate.month = 7;
96    joinDate.year = 2003;
97    pw.SetDateOfJoining(joinDate);
98    linzag.AddEmployee(std::make_unique<PieceWorker>(pw));
99    tractive.AddEmployee(std::make_unique<PieceWorker>(pw));
100
101   TestLinzAG(&linzag);
102   TestSequality(&sequality);
103   TestTractive(&tractive);
104
105   return 0;
106 }
107
108 void PrintTestTitle(std::string const subtitle) {
109   std::cout << std::endl;
110   std::cout << "################################" << std::endl;
111   std::cout << subtitle << std::endl;
112   std::cout << "################################" << std::endl;
113 }
114
115 void TestLinzAG(Company* const linzag) {
116   PrintTestTitle("Client test for Linz AG");
117
118   ICompany* comp = dynamic_cast<ICompany*>(&(*linzag));
119   Client client_linzAG(comp);
120   bool isLinzAGValid = true;
121
122   isLinzAGValid = isLinzAGValid ? client_linzAG.TestCompanyName("Linz AG") : false;
123   isLinzAGValid = isLinzAGValid ? client_linzAG.TestCompanyLocation("Linz") : false;
124   isLinzAGValid = isLinzAGValid ? client_linzAG.TestCountEmployees(4) : false;
125   isLinzAGValid = isLinzAGValid ? client_linzAG.TestFindEmployeeByNickname("DaW") : false;
126   isLinzAGValid = isLinzAGValid ? client_linzAG.TestCountEmployeesOlderThan(1990, 2) : false;
127   isLinzAGValid = isLinzAGValid ? client_linzAG.TestGetOldestEmployee("CG-Boss") : false;
128   isLinzAGValid = isLinzAGValid ? client_linzAG.TestLongestTimeInCompany("CG-Boss") : false;
129   client_linzAG.TestPrintAll();
130
131   if (isLinzAGValid) {
132     std::cout << "Everything OK..." << std::endl;
133   }
134   else {
135     std::cout << "Something failed..." << std::endl;
136   }
137
138 }
139 void TestSequality(Company* const sequality) {
140
141   PrintTestTitle("Client test for Sequality GmbH");
142
143   ICompany* comp = dynamic_cast<ICompany*>(&(*sequality));
144   Client client_sequality(comp);
145   bool isSequalityValid = true;
146
147   isSequalityValid = isSequalityValid ? client_sequality.TestCompanyName("Sequality GmbH") : false;
148   isSequalityValid = isSequalityValid ? client_sequality.TestCompanyLocation("Hagenberg") : false;
149   isSequalityValid = isSequalityValid ? client_sequality.TestCountEmployeesByType(wBase::Comission,
          1) : false;
150   isSequalityValid = isSequalityValid ? client_sequality.TestCountTotalSoldPieces(11) : false;
151   client_sequality.TestPrintAll();
```

```
152
153   if (isSequalityValid) {
154     std::cout << "Everything OK..." << std::endl;
155   }
156   else {
157     std::cout << "Something failed..." << std::endl;
158   }
159 }
160 void TestTractive(Company* const tractive) {
161   PrintTestTitle("Client test for Tractive");
162
163   ICompany* comp = dynamic_cast<ICompany*>(&(*tractive));
164   Client client_tractive(comp);
165   bool isTractiveValid = true;
166
167   isTractiveValid = isTractiveValid ? client_tractive.TestCompanyName("Tractive") : false;
168   isTractiveValid = isTractiveValid ? client_tractive.TestCompanyLocation("Pasching") : false;
169   isTractiveValid = isTractiveValid ? client_tractive.TestCountTotalProducesPieces(10) : false;
170   isTractiveValid = isTractiveValid ? client_tractive.TestGetSalaryOfEmployee("DaW", 1040) : false;
171   client_tractive.TestPrintAll();
172
173   if (isTractiveValid) {
174     std::cout << "Everything OK..." << std::endl;
175   }
176   else {
177     std::cout << "Something failed..." << std::endl;
178   }
179 }
```