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

Name(1):	Abgabetermin:
Name(2):	Punkte:
Übungsgruppe:	korrigiert:
Geschätzter Aufwand in Ph:	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 4, 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

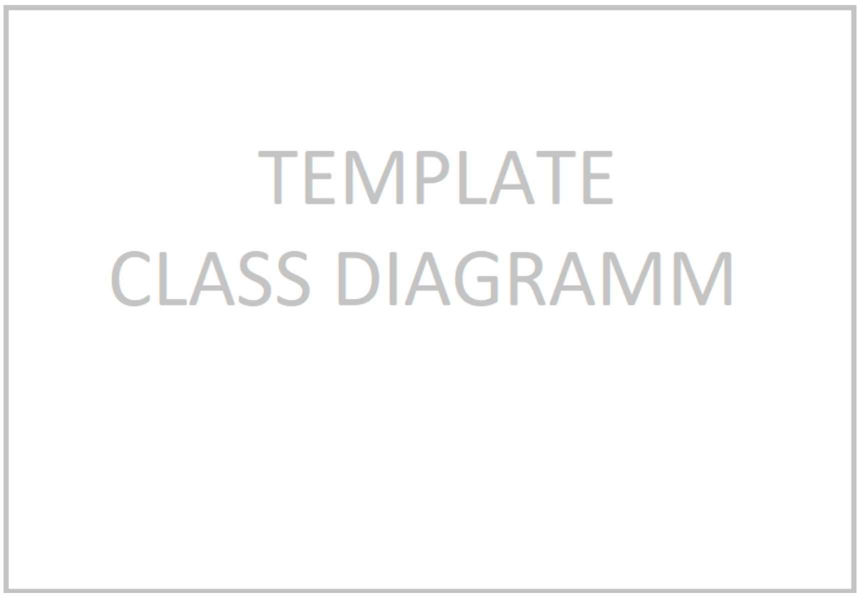
- estimated: 10 ph
- actually: - ph

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

```cpp
#pragma once
#include "ICompany.h"

class Client {


private:
  ICompany& company;

  bool TestCompanyName();
  bool TestCompanyLocation();
  bool TestFindEmployeeByNickname();
  bool TestFindEmployeeByBirthday();
};
```

6.1.2 Client.cpp

```cpp
#include "ICompany.h"
```

6.2 Interface ICompany

6.2.1 ICompany.h

```cpp
#pragma once
#include <stdio.h>
#include <string>

#include "Employee.h"

class ICompany {
public:
  virtual std::string GetCompanyName() = 0;
  virtual std::string GetCompanyLocation() = 0;
  virtual Employee* GetEmployee(std::string nickname) = 0;
  //virtual Employee* GetEmployee(Type type) = 0;
  virtual int GetSoldPieces() = 0;
  virtual int GetProdPieces() = 0;
  virtual int CountEmployees() = 0;
  //virtual int CountEmployess(Date birthday) = 0;
  virtual void Print() = 0;
};
```

6.3 Class Company

6.3.1 Company.h

```cpp
#pragma once

#include <string>
#include <list>

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

class Company : public ICompany
{

public:
  Company(std::string name, std::string location);

  std::string GetCompanyName() override;
  std::string GetCompanyLocation() override;
  Employee* GetEmployee(std::string nickname) override;
  //Employee* GetEmployee(Type type) override;
  int GetSoldPieces() override;
  int GetProdPieces() override;
  int CountEmployees() override;
  //int CountEmployess(Date birthday) override;
  void Print() override;

  void AddEmployee(Employee e);
  void DeleteEmployee(Employee e);

private:
  std::string m_name;
  std::string m_location;
  std::list<Employee> m_employees;
};
```

6.3.2 Company.cpp

```cpp
#include "Company.h"

using namespace std;
```

```cpp

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

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

Employee* Company::GetEmployee(std::string nickname) {
  for (Employee emp : m_employees) {
    if (nickname == emp.GetNickname()) {
      return emp;
    }
  };
  return new Employee();
}

//Employee* Company::GetEmployee(Type type){}

int Company::GetSoldPieces() {
  int sumSoldPieces = 0;
  for (Employee emp : m_employees) {
    sumSoldPieces += emp.GetSoldPieces();
  }
  return sumSoldPieces;
}

int Company::GetProdPieces() {
  int sumProdPieces = 0;
  for (Employee emp : m_employees) {
    sumProdPieces += emp.GetProdPieces();
  }
  return sumProdPieces;
}

int Company::CountEmployees() {
  return m_employees.size();
}

//int Company::CountEmployess(Date birthday) {}

void Company::Print() {
  for (Employee emp : m_employees) {
    //cout << emp;
  }
}

void Company::AddEmployee(Employee e) {
  m_employees.push_back(e);
}

void Company::DeleteEmployee(Employee e) {
  for (Employee emp : m_employees) {
    /*if (emp == e) {
      //delete
    }*/
  }
}
```

6.4 Class Employee

6.4.1 Employee.h

```cpp
#pragma once
class Employee
{
};
```

6.4.2 Employee.cpp

```
#include "Employee.h"
```

6.5 Class CommissionWorker

6.5.1 CommissionWorker.h

```
#pragma once
class CommissionWorker
{
};
```

6.5.2 CommissionWorker.cpp

```cpp
#include "CommissionWorker.h"
```

6.6 Class HourlyWorker

6.6.1 HourlyWorker.h

```cpp
#pragma once
class HourlyWorker
{
};
```

6.6.2 HourlyWorker.cpp

```
#include "HourlyWorker.h"
```

6.7 Class PieceWorker

6.7.1 PieceWorker.h

```
#pragma once
class PieceWorker
{
};
```

6.7.2 PieceWorker.cpp

```cpp
#include "PieceWorker.h"
```

6.8 Class Boss

6.8.1 Boss.h

```cpp
#pragma once
class Boss
{
};
```

6.8.2 Boss.cpp

```
#include "Boss.h"
```

6.8.3 TestDriver.cpp

```
#include "Client.h"
#include "Company.h"

int main() {

  Company linzag("Linz AG", "Linz");
  Company sequality("Sequality GmbH", "Hagenberg");
  Company tractive("Tractive", "Pasching");

  //cast company auf icompany?

  return 0;
}
```