FH-OÖ Hagenberg/HSD
SDP3, WS 2025
Übung 2

Name: Simon Offenberger / Simon Vogelhuber	Aufwand in h: siehe Doku
Mat.Nr: S2410306027 / S2410306014	Punkte:
Übungsgruppe: 1	korrigiert:

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 2025
-------------------------------------------
```

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

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 (entsprechend den Vorgaben aus Übung1)!

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!

FH OBERÖSTERREICH HSD

FH-HAGENBERG

Systemdokumentation Projekt Gehaltsberechnung

Version 1.0

S. Offenberger, S. Vogelhuber

Hagenberg, 30. Oktober 2025

Inhaltsverzeichnis

1 Organisatorisches

1.1 Team

- Simon Offenberger, Matr.-Nr.: S2410306027, E-Mail: Simon.Offenberger@fh-hagenberg.at
- Simon Vogelhuber, Matr.-Nr.: S2410306014, E-Mail: s2410306014@fhooe.at

1.2 Aufteilung der Verantwortlichkeitsbereiche

- Simon Offenberger
  - Design Klassendiagramm
  - Implementierung und Test der Klassen:
    - Company
    - Company Interface
    - Client
  - Implementierung des Testtreibers
  - Dokumentation
- Simon Vogelhuber
  - Design Klassendiagramm
  - Implementierung und Komponententest der Klassen:
    - Employee
    - Boss
    - ComissionWorker

    * PieceWorker
    * HourlyWorker
  - Implementierung des Testtreibers
  - Dokumentation

1.3 Aufwand

- Simon Offenberger: geschätzt 10 Ph / tatsächlich 7 Ph
- Simon Vogelhuber: geschätzt 10 Ph / tatsächlich 8,5 Ph

2 Anforderungsdefinition (Systemspezifikation)

In diesem Projekt geht es darum die Mitarbeiter eines Unternehmens zu verwalten und deren Gehälter zu berechnen. Es gibt verschiedene Arten von Mitarbeitern, welche unterschiedliche Gehaltsberechnungen haben. Der Zugriff auf die Mitarbeiter soll über eine gemeinsame Schnittstelle erfolgen.

Funktionen der Firmenschnittstelle

- Zugriff auf die wichtigsten Mitarbeiter und Firmendaten

Funktionen der Firma

- Abfage nach der Anzahl der Mitarbeiter.
- Abfage nach der Anzahl eines Mitarbeitertyps in der Firma
- Wie viele Stück wurden im Monat erzeugt?
- Wie viele Stück wurden im Monat verkauft?
- Wie viele Mitarbeiter sind vor einem bestimmten Datum 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

Funktionen der Mitarbeiter

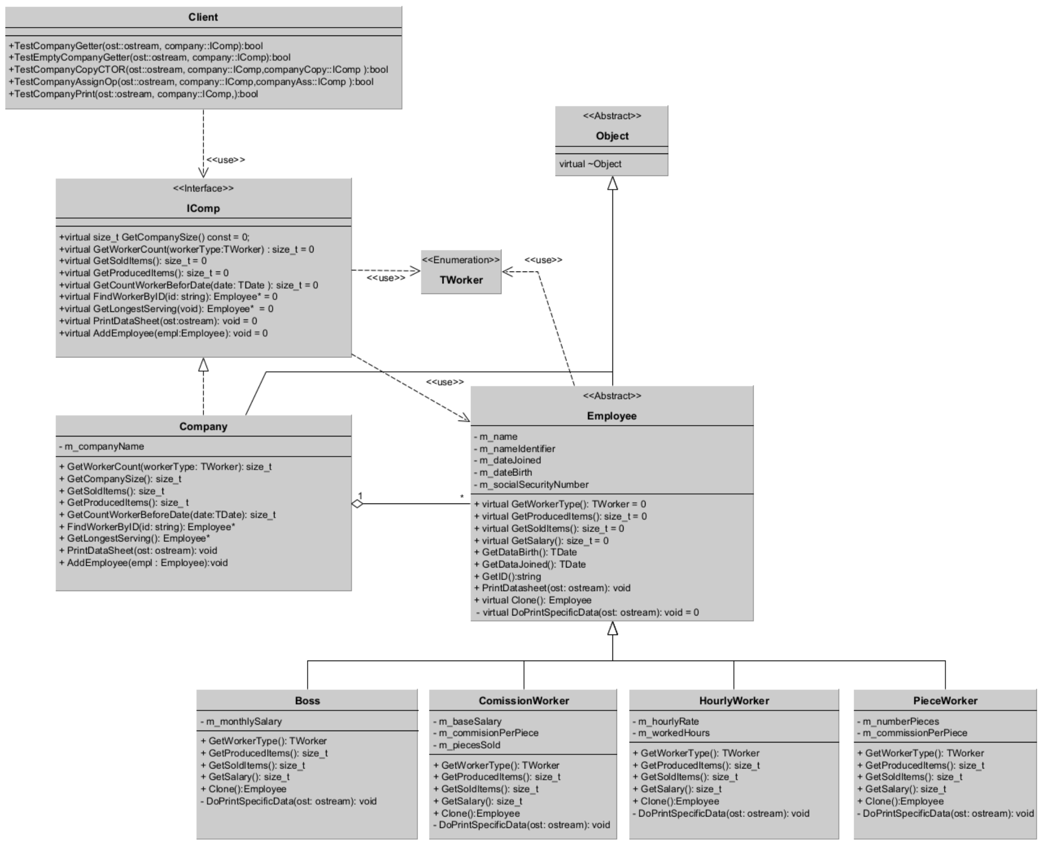
- Speichern von Mitarbeiterdaten.
  - Name
  - Namenskürzel

  - Sozialversicherungsnummer
  - Einstiegsjahr
  - Geburtsjahr

- Berechnung des Gehalts je nach Mitarbeiterklasse.
- Ausgabe von Mitarbeiterinformationen in form eines Datenblatts.

3 Systementwurf

3.1 Klassendiagramm


Client
+TestCompanyGetter(ost::ostream, company::IComp):bool
+TestEmptyCompanyGetter(ost::ostream, company::IComp):bool
+TestCompanyCopyCTOR(ost::ostream, company::IComp,companyCopy::IComp):bool
+TestCompanyAssignOp(ost::ostream, company::IComp,companyAss::IComp):bool
+TestCompanyPrint(ost::ostream, company::IComp,):bool
<<use>>
<<Abstract>>
Object
virtual ~Object
<<Interface>>
IComp
+virtual size_t GetCompanySize() const = 0;
+virtual GetWorkerCount(workerType:TWorker) : size_t = 0
+virtual GetSoldItems(): size_t = 0
+virtual GetProducedItems(): size_t = 0
+virtual GetCountWorkerBeforDate(date: TDate): size_t = 0
+virtual FindWorkerByID(id: string): Employee* = 0
+virtual GetLongestServing(void): Employee* = 0
+virtual PrintDataSheet(ost:ostream): void = 0
+virtual AddEmployee(empl:Employee): void = 0
<<use>>
<<Enumeration>>
TWorker
<<use>>
<<use>>
Company
- m_companyName
+ GetWorkerCount(workerType: TWorker): size_t
+ GetCompanySize(): size_t
+ GetSoldItems(): size_t
+ GetProducedItems(): size_ t
+ GetCountWorkerBeforeDate(date:TDate): size_t
+ FindWorkerByID(id: string): Employee*
+ GetLongestServing(): Employee*
+ PrintDataSheet(ost: ostream): void
+ AddEmployee(empl : Employee):void
1
*
<<Abstract>>
Employee
- m_name
- m_nameIdentifier
- m_dateJoined
- m_dateBirth
- m_socialSecurityNumber
+ virtual GetWorkerType(): TWorker = 0
+ virtual GetProducedItems(): size_t = 0
+ virtual GetSoldItems(): size_t = 0
+ virtual GetSalary(): size_t = 0
+ GetDataBirth(): TDate
+ GetDataJoined(): TDate
+ GetID():string
+ PrintDatasheet(ost: ostream): void
+ virtual Clone(): Employee
- virtual DoPrintSpecificData(ost: ostream): void = 0
Boss
- m_monthlySalary
+ GetWorkerType(): TWorker
+ GetProducedItems(): size_t
+ GetSoldItems(): size_t
+ GetSalary(): size_t
+ Clone():Employee
- DoPrintSpecificData(ost: ostream): void
ComissionWorker
- m_baseSalary
- m_commisionPerPiece
- m_piecesSold
+ GetWorkerType(): TWorker
+ GetProducedItems(): size_t
+ GetSoldItems(): size_t
+ GetSalary(): size_t
+ Clone():Employee
- DoPrintSpecificData(ost: ostream): void
HourlyWorker
- m_hourlyRate
- m_workedHours
+ GetWorkerType(): TWorker
+ GetProducedItems(): size_t
+ GetSoldItems(): size_t
+ GetSalary(): size_t
+ Clone():Employee
- DoPrintSpecificData(ost: ostream): void
PieceWorker
- m_numberPieces
- m_commissionPerPiece
+ GetWorkerType(): TWorker
+ GetProducedItems(): size_t
+ GetSoldItems(): size_t
+ GetSalary(): size_t
+ Clone():Employee
- DoPrintSpecificData(ost: ostream): void

3.2 Designentscheidungen

Das Interface **ICompany** wurde erstellt, um dem zugreifenden **Client** eine Schnittstelle zur Verfügung zu stellen. Dadurch kann sich der Client auf die Schnittstelle konzentrieren und muss sich nicht um die Implementierungsdetails der Firma kümmern.

Die Firma speichert einen polymorphen Container, der Objekte der abstrakten Klasse **Employee** verwaltet. Bei dem Container wurde eine Map verwendet, da die Mitarbeiter über eine eindeutige ID angesprochen werden können. Somit ist auch das Suchen nach einem Mitarbeiter sehr performant gelöst.

Die Klasse **Employee** ist abstrakt, da es keine generellen Mitarbeiter geben soll, sondern nur spezielle Arten von Mitarbeitern. Die einzelnen Mitarbeiter speichern Daten, die für die Gehaltsberechnung notwendig sind. Die Gehaltsberechnung wird über eine virtuelle Funktion realisiert, die in den abgeleiteten Klassen überschrieben wird. Weiters soll die Ausgabe eines Datenblatts zu jedem Mitarbeiter möglich sein dies wurde mittels **Template Methode Pattern** gelöst!

Das Enum mit dem Mitarbeitertypen **TWorker** wurde eingebaut, da die Company den Typen des Mitarbeiters kennen muss, um den Mitarbeiter korrekt zu verwalten. Hierbei wurde aktiv auf RTTI verzichtet, um die Kopplung zwischen Company und den konkreten Klassen die von Employee ableiten zu reduzieren. Weiters wurden die konkreten Mitarbeiterklassen so gestaltet, dass sie ohne großen Aufwand zu testen sind. Aus diesem Grund werden alle Daten im Konstruktor übergeben und es gibt keine Setter-Methoden. Würde dieses Design in der Praxis verwendet werden, müsste man noch Setter Methoden hinzufügen. Da dies hier nicht im Fokus steht, wurde dies nicht umgesetzt.

4 Dokumentation der Komponenten (Klassen)

Die HTML-Startdatei befindet sich im Verzeichnis ./../doxy/html/index.html

5 Testprotokollierung

```

*******************************************
             TESTCASE START
*******************************************

Test Company Get Comission Worker Cnt & Add Empl
[Test OK] Result: (Expected: 2 == Result: 2)

Test Company Get Houerly Worker Cnt & Add Empl
[Test OK] Result: (Expected: 1 == Result: 1)

Test Company Get Boss Cnt & Add Empl
[Test OK] Result: (Expected: 1 == Result: 1)

Test Company Get Piece Worker Cnt & Add Empl
[Test OK] Result: (Expected: 2 == Result: 2)

Test Company FindWorker by ID
[Test OK] Result: (Expected: Si1 == Result: Si1)

Test Company FindWorker by empty ID
[Test OK] Result: (Expected: 0000000000000000 == Result:
   ↪ 0000000000000000)

Test Company Get Size
[Test OK] Result: (Expected: 6 == Result: 6)

Test Company Get Count worker bevor 1930 date
[Test OK] Result: (Expected: 0 == Result: 0)

Test Company Get Count worker bevor 1951 date
[Test OK] Result: (Expected: 2 == Result: 2)

Test Company Get longest serving employee
[Test OK] Result: (Expected: 0 == Result: 0)

Test Company Get total pieces produced
[Test OK] Result: (Expected: 50 == Result: 50)

Test Company Get total pieces sold
[Test OK] Result: (Expected: 2700 == Result: 2700)

```

```

****************************************


****************************************
            TESTCASE START
****************************************

Test Company Copy Ctor
[Test OK] Result: (Expected: true == Result: true)


****************************************


****************************************
            TESTCASE START
****************************************

Test Company Assign Operator
[Test OK] Result: (Expected: true == Result: true)


****************************************


****************************************
            TESTCASE START
****************************************

Test Company Print Exception
[Test OK] Result: (Expected: ERROR: Provided Ostream is bad ==
  ↪  Result: ERROR: Provided Ostream is bad)


****************************************


****************************************
            TESTCASE START
****************************************

Test Empty Company Get Comission Worker Cnt & Add Empl
[Test OK] Result: (Expected: 0 == Result: 0)
```

```

Test Empty Company Get Houerly Worker Cnt & Add Empl
[Test OK] Result: (Expected: 0 == Result: 0)

Test Empty Company Get Boss Cnt & Add Empl
[Test OK] Result: (Expected: 0 == Result: 0)

Test Empty Company Get Piece Worker Cnt & Add Empl
[Test OK] Result: (Expected: 0 == Result: 0)

Test Empty Company FindWorker by ID
[Test OK] Result: (Expected: 0000000000000000 == Result:
    ↪ 0000000000000000)

Test Empty Company FindWorker by ID empty ID
[Test OK] Result: (Expected: 0000000000000000 == Result:
    ↪ 0000000000000000)

Test Empty Company Get Size
[Test OK] Result: (Expected: 0 == Result: 0)

Test Empty Company Get Count worker bevor 1930 date
[Test OK] Result: (Expected: 0 == Result: 0)

Test Empty Company Get Count worker bevor 1951 date
[Test OK] Result: (Expected: 0 == Result: 0)

Test Empty Company Get longest serving employee
[Test OK] Result: (Expected: 0000000000000000 == Result:
    ↪ 0000000000000000)

Test Empty Company Get total pieces produced
[Test OK] Result: (Expected: 0 == Result: 0)

Test Empty Company Get total pieces sold
[Test OK] Result: (Expected: 0 == Result: 0)

Test Company Add nullptr
[Test OK] Result: (Expected: ERROR: Passed in Nullptr! ==
    ↪ Result: ERROR: Passed in Nullptr!)


*******************************************

```

```

*******************************************
             TESTCASE START
*******************************************

Test - Boss.GetSalary()
[Test OK] Result: (Expected: 7800 == Result: 7800)

Test - Boss.GetSoldItems()
[Test OK] Result: (Expected: 0 == Result: 0)

Test - Boss.GetProducedItems()
[Test OK] Result: (Expected: 0 == Result: 0)

Test - Boss.GetWorkerType()
[Test OK] Result: (Expected: 0 == Result: 0)

Test - Boss.GetDateBirth()
[Test OK] Result: (Expected: 2000-11-22 == Result: 2000-11-22)

Test - Boss.GetDateJoined()
[Test OK] Result: (Expected: 2022-11-23 == Result: 2022-11-23)

Test - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test Boss.Clone()
[Test OK] Result: (Expected: true == Result: true)

Test - error buffer
[Test OK] Result: (Expected: true == Result: true)

Boss Constructor bad ID
[Test OK] Result: (Expected: ERROR: An employees ID is limited
    ↪  to 3 characters. == Result: ERROR: An employees ID is
    ↪ limited to 3 characters.)

Boss Constructor bad SV - invalid character
[Test OK] Result: (Expected: ERROR: Invalid Sozial Security
    ↪ Number == Result: ERROR: Invalid Sozial Security Number)

Boss Constructor bad SV - too many nums
[Test OK] Result: (Expected: ERROR: Invalid Sozial Security
    ↪ Number == Result: ERROR: Invalid Sozial Security Number)
```

```

Boss bad ostream
[Test OK] Result: (Expected: ERROR: Provided Ostream is bad ==
    ↪  Result: ERROR: Provided Ostream is bad)


*******************************************


*******************************************
             TESTCASE START
*******************************************

Test - HourlyWorker.GetSalary()
[Test OK] Result: (Expected: 3360 == Result: 3360)

Test - HourlyWorker.GetSoldItems()
[Test OK] Result: (Expected: 0 == Result: 0)

Test - HourlyWorker.GetProducedItems()
[Test OK] Result: (Expected: 0 == Result: 0)

Test - HourlyWorker.GetWorkerType()
[Test OK] Result: (Expected: 2 == Result: 2)

Test - HourlyWorker.GetDateBirth()
[Test OK] Result: (Expected: 2000-11-22 == Result: 2000-11-22)

Test - HourlyWorker.GetDateJoined()
[Test OK] Result: (Expected: 2022-11-23 == Result: 2022-11-23)

Test - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test testPieceWorker.Clone()
[Test OK] Result: (Expected: true == Result: true)

Test - error buffer
[Test OK] Result: (Expected: true == Result: true)

HourlyWorker Constructor bad ID
[Test OK] Result: (Expected: ERROR: An employees ID is limited
    ↪  to 3 characters. == Result: ERROR: An employees ID is
    ↪ limited to 3 characters.)
```

```

HourlyWorker Constructor bad SV - invalid character
[Test OK] Result: (Expected: ERROR: Invalid Sozial Security
   ↪ Number == Result: ERROR: Invalid Sozial Security Number)

HourlyWorker Constructor bad SV - too many nums
[Test OK] Result: (Expected: ERROR: Invalid Sozial Security
   ↪ Number == Result: ERROR: Invalid Sozial Security Number)

HourlyWorker bad ostream
[Test OK] Result: (Expected: ERROR: Provided Ostream is bad ==
   ↪  Result: ERROR: Provided Ostream is bad)


*******************************************


*******************************************
             TESTCASE START
*******************************************

Test - PieceWorker.GetSalary()
[Test OK] Result: (Expected: 1900 == Result: 1900)

Test - PieceWorker.GetSoldItems()
[Test OK] Result: (Expected: 0 == Result: 0)

Test - PieceWorker.GetProducedItems()
[Test OK] Result: (Expected: 950 == Result: 950)

Test - PieceWorker.GetWorkerType()
[Test OK] Result: (Expected: 3 == Result: 3)

Test - PieceWorker.GetDateBirth()
[Test OK] Result: (Expected: 2000-11-22 == Result: 2000-11-22)

Test - PieceWorker.GetDateJoined()
[Test OK] Result: (Expected: 2022-11-23 == Result: 2022-11-23)

Test - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test testPieceWorker.Clone()
[Test OK] Result: (Expected: true == Result: true)
```

```

Test - error buffer
[Test OK] Result: (Expected: true == Result: true)

PieceWorker Constructor bad ID
[Test OK] Result: (Expected: ERROR: An employees ID is limited
    ↪  to 3 characters. == Result: ERROR: An employees ID is
    ↪ limited to 3 characters.)

PieceWorker Constructor bad SV - invalid character
[Test OK] Result: (Expected: ERROR: Invalid Sozial Security
    ↪ Number == Result: ERROR: Invalid Sozial Security Number)

PieceWorker Constructor bad SV - too many nums
[Test OK] Result: (Expected: ERROR: Invalid Sozial Security
    ↪ Number == Result: ERROR: Invalid Sozial Security Number)

PieceWorker bad ostream
[Test OK] Result: (Expected: ERROR: Provided Ostream is bad ==
    ↪  Result: ERROR: Provided Ostream is bad)


*******************************************


*******************************************
             TESTCASE START
*******************************************

Test - ComissionWorker.GetSalary()
[Test OK] Result: (Expected: 2900 == Result: 2900)

Test - ComissionWorker.GetSoldItems()
[Test OK] Result: (Expected: 300 == Result: 300)

Test - ComissionWorker.GetProducedItems()
[Test OK] Result: (Expected: 0 == Result: 0)

Test - ComissionWorker.GetWorkerType()
[Test OK] Result: (Expected: 1 == Result: 1)

Test - ComissionWorker.GetDateBirth()
[Test OK] Result: (Expected: 2000-11-22 == Result: 2000-11-22)

```

```
Test - ComissionWorker.GetDateJoined()
[Test OK] Result: (Expected: 2022-11-23 == Result: 2022-11-23)

Test - error buffer
[Test OK] Result: (Expected: true == Result: true)

Test testPieceWorker.Clone()
[Test OK] Result: (Expected: true == Result: true)

Test - error buffer
[Test OK] Result: (Expected: true == Result: true)

ComissionWorker Constructor bad ID
[Test OK] Result: (Expected: ERROR: An employees ID is limited
    ↪  to 3 characters. == Result: ERROR: An employees ID is
    ↪ limited to 3 characters.)

ComissionWorker Constructor bad SV - invalid character
[Test OK] Result: (Expected: ERROR: Invalid Sozial Security
    ↪ Number == Result: ERROR: Invalid Sozial Security Number)

ComissionWorker Constructor bad SV - too many nums
[Test OK] Result: (Expected: ERROR: Invalid Sozial Security
    ↪ Number == Result: ERROR: Invalid Sozial Security Number)

ComissionWorker bad ostream
[Test OK] Result: (Expected: ERROR: Provided Ostream is bad ==
    ↪  Result: ERROR: Provided Ostream is bad)


*******************************************


*******************************************
             TESTCASE START
*******************************************

Test Exception in Company Add Duplicate
[Test OK] Result: (Expected: ERROR: Duplicate Employee! ==
    ↪ Result: ERROR: Duplicate Employee!)


*******************************************

```

```
TEST OK!!
```

6 Quellcode

6.1 Object.hpp

```
/*****************************************************************//**
 * \file   Object.hpp
 * \brief  Root of all Objects
 *
 * \author Simon
 * \date   October 2025
 *********************************************************************/
#ifndef OBJECT_HPP
#define OBJECT_HPP

class Object {
public:

        /**
         * \brief Constant for Exception Bad Ostream.
         */
        inline static const std::string ERROR_BAD_OSTREAM = "ERROR:␣Provided␣Ostream␣is␣bad";

        /**
         * \brief Constant for Exception Fail Write.
         */
        inline static const std::string ERROR_FAIL_WRITE = "ERROR:␣Fail␣to␣write␣on␣provided␣Ostream";

        /**
         * \brief Constant for Exception Nullprt.
         */
        inline static const std::string ERROR_NULLPTR = "ERROR:␣Passed␣in␣Nullptr!";

protected:

        /**
         * \brief protected CTOR -> abstract Object.
         *
         */
        Object() = default;

        /**
         * \brief virtual DTOR -> once Virtual always virtual.
         *
         */
        virtual ~Object() = default;

};

#endif // !OBJECT_HPP
```

6.2 Client.hpp

```
/*********************************************************************//**
 * \file   Client.hpp
 * \brief  Client Class that uses the Class Company via the Interface IComp
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#ifndef CLIENT_HPP
#define CLIENT_HPP

#include <iostream>
#include "IComp.hpp"

class Client {
public:
        /**
         * Constant for Exception Bad Ostream.
         */
        inline static const std::string ERROR_BAD_OSTREAM = "ERROR:␣Provided␣Ostream␣is␣bad";

        /**
         * Constant for Exception Write Fail.
         */
        inline static const std::string ERROR_FAIL_WRITE = "ERROR:␣Fail␣to␣write␣on␣provided␣Ostream";

        /**
         * \brief Test Methode for the Getter Methodes of the Company via the Interface.
         *
         * \param ost Refernce to an ostream where the Test results should be printed at
         * \param company Reference to a company interface
         * \return true -> Test OK
         * \return false -> Test NOK
         */
        bool TestCompanyGetter(std::ostream & ost,const IComp& company) const;

        /**
         * \brief Test Methode for the Getter Methodes of an Empty Company via the Interface.
         *
         * \param ost Refernce to an ostream where the Test results should be printed at
         * \param company Reference to a company interface
         * \return true -> Test OK
         * \return false -> Test NOK
         */
        bool TestEmptyCompanyGetter(std::ostream & ost, IComp& company) const;

        /**
         * \brief Test Methode for testing the Copy Ctor of the Company
         *
         * \param ost Refernce to an ostream where the Test results should be printed at
         * \param company Reference to a company interface
         * \param companyCopy Reference to the copy of company
         * \return true -> Test OK
         * \return false -> Test NOK
         */
        bool TestCompanyCopyCTOR(std::ostream & ost,const IComp& company,const IComp& companyCopy) const;

        /**
         * \brief Test Methode for the Assign Operator of Company
         *
         * \param ost Refernce to an ostream where the Test results should be printed at
         * \param company Reference to a company interface
         * \param companyAss Reference to the assigned Company should be Equal to company
         * \return true -> Test OK
         * \return false -> Test NOK
         */
        bool TestCompanyAssignOp(std::ostream & ost,const IComp& company,const IComp& companyAss) const;

        /**
         * \brief Test Methode for the Print Methode of Company
         *
         * \param ost Refernce to an ostream where the Test results should be printed at
         * \param company Reference to a company interface
         * \return true -> Test OK
         * \return false -> Test NOK
         */
        bool TestCompanyPrint(std::ostream & ost,const IComp& company) const;

};

#endif // !CLIENT_HPP
```

6.3 Client.cpp

```cpp
/*******************************************************************//**
 * \file   Client.hpp
 * \brief  Client Class that uses the Class Company via the Interface IComp
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/

#include "Client.hpp"
#include "Test.hpp"
#include <sstream>
#include <fstream>

using namespace std;
using namespace std::chrono;

bool Client::TestCompanyGetter(std::ostream& ost,const IComp & company) const
{
        if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";


        try {

                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣Comission␣Worker␣Cnt␣&␣Add␣Empl",  static_cast<size_t>(2), company.GetWorkerCount(TWorker::E_CommisionWorker));
                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣Houerly␣Worker␣Cnt␣&␣Add␣Empl",    static_cast<size_t>(1), company.GetWorkerCount(TWorker::E_HourlyWorker));
                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣Boss␣Cnt␣&␣Add␣Empl",                              static_cast<size_t>(1), company.GetWorkerCount(TWorker::E_Boss));
                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣Piece␣Worker␣Cnt␣&␣Add␣Empl",              static_cast<size_t>(2), company.GetWorkerCount(TWorker::E_PieceWorker));


                TestOK = TestOK && check_dump(ost, "Test␣Company␣FindWorker␣by␣ID",                     static_cast<std::string>("Si1"), company.FindWorkerByID("Si1")->GetID());
                TestOK = TestOK && check_dump(ost, "Test␣Company␣FindWorker␣by␣empty␣ID",   static_cast<const Employee *>(nullptr), company.FindWorkerByID(""));

                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣Size",                                     static_cast<size_t>(6), company.GetCompanySize());

                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣Count␣worker␣bevor␣1930␣date", static_cast<size_t>(0), company.GetCountWorkerBeforDate({ 1930y,November,23d }));
                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣Count␣worker␣bevor␣1951␣date", static_cast<size_t>(2), company.GetCountWorkerBeforDate({ 1951y,November,23d }));

                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣longest␣serving␣employee", TWorker::E_Boss, company.GetLongestServing()->GetWorkerType());

                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣total␣pieces␣produced", static_cast<size_t>(50), company.GetProducedItems());

                TestOK = TestOK && check_dump(ost, "Test␣Company␣Get␣total␣pieces␣sold", static_cast<size_t>(2700), company.GetSoldItems());

        }
        catch (const string& err) {
                error_msg = err;
                TestOK = false;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
                TestOK = false;
        }
        catch (const exception& err) {
                error_msg = err.what();
                TestOK = false;
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
                TestOK = false;
        }

        TestEnd(ost);

        if (ost.fail()) throw Client::ERROR_FAIL_WRITE;

        return TestOK;
}

```

```
bool Client::TestEmptyCompanyGetter(std::ostream& ost,IComp& company) const
{
        if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";


        try {

                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣Comission␣Worker␣Cnt␣&␣Add␣Empl", static_cast<size_t>(0), company.GetWorkerCount(TWorker::E_CommisionWorker));
                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣Houerly␣Worker␣Cnt␣&␣Add␣Empl", static_cast<size_t>(0), company.GetWorkerCount(TWorker::E_HourlyWorker));
                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣Boss␣Cnt␣&␣Add␣Empl", static_cast<size_t>(0), company.GetWorkerCount(TWorker::E_Boss));
                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣Piece␣Worker␣Cnt␣&␣Add␣Empl", static_cast<size_t>(0), company.GetWorkerCount(TWorker::E_PieceWorker));


                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣FindWorker␣by␣ID", static_cast<const Employee *>(nullptr), company.FindWorkerByID("Si1"));
                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣FindWorker␣by␣ID␣empty␣ID", static_cast<const Employee *>(nullptr), company.FindWorkerByID(""));


                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣Size", static_cast<size_t>(0), company.GetCompanySize());

                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣Count␣worker␣bevor␣1930␣date", static_cast<size_t>(0), company.GetCountWorkerBeforDate({ 1930y,November,23d }));
                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣Count␣worker␣bevor␣1951␣date", static_cast<size_t>(0), company.GetCountWorkerBeforDate({ 1951y,November,23d }));

                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣longest␣serving␣employee", static_cast<const Employee*>(nullptr), company.GetLongestServing());


                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣total␣pieces␣produced", static_cast<size_t>(0), company.GetProducedItems());

                TestOK = TestOK && check_dump(ost, "Test␣Empty␣Company␣Get␣total␣pieces␣sold", static_cast<size_t>(0), company.GetSoldItems());

        }
        catch (const string& err) {
                error_msg = err;
                TestOK = false;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
                TestOK = false;
        }
        catch (const exception& err) {
                error_msg = err.what();
                TestOK = false;
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
                TestOK = false;
        }

        try {

                company.AddEmployee(nullptr);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }


        TestOK = TestOK && check_dump(ost, "Test␣Company␣Add␣nullptr", Object::ERROR_NULLPTR, error_msg);

        TestEnd(ost);

        if (ost.fail()) throw Client::ERROR_FAIL_WRITE;

        return TestOK;
}

bool Client::TestCompanyCopyCTOR(std::ostream& ost,const IComp& company,const IComp& companyCopy) const
{

        if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

                stringstream result;
                stringstream expected;

                company.PrintDataSheet(expected);
                companyCopy.PrintDataSheet(result);

                TestOK = TestOK && check_dump(ost, "Test␣Company␣Copy␣Ctor", true ,expected.str() == result.str());

        }
        catch (const string& err) {
                error_msg = err;
                TestOK = false;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
                TestOK = false;
        }
        catch (const exception& err) {
                error_msg = err.what();
                TestOK = false;
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
                TestOK = false;
        }

        TestEnd(ost);

        if (ost.fail()) throw Client::ERROR_FAIL_WRITE;

        return TestOK;

        return false;
}

bool Client::TestCompanyAssignOp(std::ostream& ost,const IComp& company,const IComp& companyAss) const
{
        if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

                stringstream result;
                stringstream expected;

                company.PrintDataSheet(expected);
                companyAss.PrintDataSheet(result);

                TestOK = TestOK && check_dump(ost, "Test␣Company␣Assign␣Operator", true, expected.str() == result.str());

        }
        catch (const string& err) {
                error_msg = err;
                TestOK = false;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
                TestOK = false;
        }
        catch (const exception& err) {
                error_msg = err.what();
                TestOK = false;
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
                TestOK = false;
        }

        TestEnd(ost);

        if (ost.fail()) throw Client::ERROR_FAIL_WRITE;

        return TestOK;

        return false;
}

bool Client::TestCompanyPrint(std::ostream& ost, const IComp& company) const
{
        if (!ost.good()) throw Client::ERROR_BAD_OSTREAM;

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        fstream badstream;
        badstream.setstate(ios::badbit);

        try {

                company.PrintDataSheet(badstream);

        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣Company␣Print␣Exception", Client::ERROR_BAD_OSTREAM, error_msg);

        badstream.close();

        TestEnd(ost);

        if (ost.fail()) throw Client::ERROR_FAIL_WRITE;

        return TestOK;

        return false;
}
```

6.4 IComp.hpp

```cpp
/*********************************************************************//**
 * \file   IComp.hpp
 * \brief  Interface which is implemented by the company and used by the client
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#ifndef ICOMP_HPP
#define ICOMP_HPP

#include <string>
#include "TWorker.hpp"
#include "Employee.hpp"

class IComp{
public:

        /**
         * \brief Gets the current size of the company.
         *
         * \return Size of the company
         */
        virtual size_t GetCompanySize() const = 0;

        /**
         * \brief Get the Count of a specific Worker Type.
         *
         * \param workerType Worker Type from which the count should be determined
         * \return Count of the Worker Type in the Company
         */
        virtual size_t GetWorkerCount(const TWorker & workerType) const = 0;

        /**
         * \brief Get the amount of Sold Items in the whole company.
         *
         * \return Amout of Sold Items
         */
        virtual size_t GetSoldItems() const = 0;

        /**
         * \brief Get the amount of produced items.
         *
         * \return Amout of produced Items
         */
        virtual size_t GetProducedItems() const = 0;

        /**
         * \brief Get the of worker with birth date bevor date.
         *
         * \param date to get the employees which are older
         * \return Amout of employees which are older than the passed in birthdate
         */
        virtual size_t GetCountWorkerBeforDate(const TDate & date) const = 0;

        /**
         * \brief Find a worker with a specific ID.
         *
         * \param id ID for which should be searched for
         * \return nullptr if no Empl is found
         * \return Pointer to Employee
         */
        virtual Employee const * FindWorkerByID(const std::string & id) const = 0;

        /**
         * \brief Get the Employee which has been the longest serving.
         *
         * \return nullptr if company is empty
         * \return Pointer to Employee
         */
        virtual Employee const * GetLongestServing(void) const = 0;

        /**
         * \brief Prints a Datasheet for each employee.
         *
         * \param ost ostream where the Datasheet should be printed at
         * \return referenced ostream
         */
        virtual std::ostream& PrintDataSheet(std::ostream& ost) const = 0;

        /**
         * \brief Adds am Employee to the Company
         * \brief The company now owns the Employee and is responsible for destructing of Employee.
         *
         * \param empl Employee that should be added to the Company
         * \throw ERROR_DUPLICATE_EMPL if ID of Employee is already in the collection
         * \throw ERROR_NULLPTR if an Nullptr is passed in
         */
        virtual void AddEmployee(Employee const* empl) = 0;

        /**
         * \brief Virtual Dtor of Icomp.
         *
         */
        virtual ~IComp() = default;
};

#endif // !ICOMP_HPP
```

6.5 Company.hpp

```
/*********************************************************************//**
 * \file   Company.hpp
 * \brief  Company that holds Employees and provides information about the
 * \brief  Employees of the company.
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#ifndef COMPANY_HPP
#define COMPANY_HPP

#include <map>
#include <string>
#include "Object.hpp"
#include "IComp.hpp"

/**
 * Declaration of an alias for the used Container.
 */
using  TContEmployee = std::map<const std::string,Employee const *>;

class Company : public Object, public IComp{
public:
        /**
         * Constant for the Excetion of an Duplicate Employee.
         */
        inline static const std::string ERROR_DUPLICATE_EMPL = "ERROR:␣Duplicate␣Employee!";

        /**
         * \brief CTOR for a Company.
         *
         * \param name Name of the Company
         */
        Company(const std::string & name) : m_companyName{ name } {}

        /**
         * \brief Copy Ctor of the Company.
         *
         * \param comp Refernce to the company that should be copied
         */
        Company(const Company & comp);

        /**
         * \brief Assignoperator for a company uses Copy and Swap.
         *
         * \param comp Copy of the company
         */
        void operator=(Company comp);

        /**
         * \brief Adds am Employee to the Company
         * \brief The company now owns the Employee and is responsible for destructing of Employee.
         *
         * \param empl Employee that should be added to the Company
         * \throw ERROR_DUPLICATE_EMPL if ID of Employee is already in the collection
         * \throw ERROR_NULLPTR if an Nullptr is passed in
         */
        virtual void AddEmployee(Employee const* empl) override;

        /**
         * \brief Gets the current size of the company.
         *
         * \return Size of the company
         */
        virtual size_t GetCompanySize() const override;

        /**
         * \brief Get the Count of a specific Worker Type.
         *
         * \param workerType Worker Type from which the count should be determined
         * \return Count of the Worker Type in the Company
         */
        virtual size_t GetWorkerCount(const TWorker& workerType) const override;
```

```cpp

        /**
         * \brief Get the amount of Sold Items in the whole company.
         *
         * \return Amout of Sold Items
         */
        virtual size_t GetSoldItems() const override;

        /**
         * \brief Get the amount of produced items.
         *
         * \return Amout of produced Items
         */
        virtual size_t GetProducedItems() const override;

        /**
         * \brief Get the of worker with birth date bevor date.
         *
         * \param date to get the employees which are older
         * \return Amout of employees which are older than the passed in birthdate
         */
        virtual size_t GetCountWorkerBeforDate(const TDate& date) const override;

        /**
         * \brief Find a worker with a specific ID.
         *
         * \param id ID for which should be searched for
         * \return nullptr if no Empl is found
         * \return Pointer to Employee
         */
        virtual Employee const * FindWorkerByID(const std::string& id) const override;

        /**
         * \brief Get the Employee which has been the longest serving.
         *
         * \return nullptr if company is empty
         * \return Pointer to Employee
         */
        virtual Employee const * GetLongestServing(void) const override;

        /**
         * \brief Prints a Datasheet for each employee.
         *
         * \param ost ostream where the Datasheet should be printed at
         * \return referenced ostream
         */
        virtual std::ostream& PrintDataSheet(std::ostream& ost) const override;

        /**
         * \brief DTOR of the Company.
         *
         */
        ~Company();

private:

        std::string m_companyName;
        TContEmployee m_Employees;
};

#endif // !COMPANY_HPP
```

6.6 Company.cpp

```
/*********************************************************************//**
 * \file   Company.cpp
 * \brief  Company that holds Employees and provides information about the
 * \brief  Employees of the company.
 *
 * \author Simon Offenberger
 * \date   October 2025
 *********************************************************************/
#include <algorithm>
#include <numeric>
#include <iostream>
#include "Company.hpp"
#include "Employee.hpp"
using namespace std;

/**
 * \brief Ostream manipulater for creating a horizontal line.
 *
 * \return string
 */
static ostream & hline(ostream & ost) {

        ost << string(60, '-') << endl;
        return ost;
}

/**
 * \brief Ostream manipulater for creating a horizontal line.
 *
 * \return string
 */
static ostream & hstar(ostream & ost) {

        ost << string(60, '*') << endl;
        return ost;
}

void Company::AddEmployee(Employee const* empl)
{
        if (empl == nullptr) throw Object::ERROR_NULLPTR;
        // insert returns a pair. First = Iterator, Second bool -> bool indicates if the insertion was succsessful.
        if (!m_Employees.insert({ empl->GetID(),empl }).second) throw Company::ERROR_DUPLICATE_EMPL;
}

Company::Company(const Company& comp)
{
        // copy Company name
        m_companyName = comp.m_companyName;

        // clone all employees from one company to the other
        for_each(
                comp.m_Employees.cbegin(), comp.m_Employees.cend(),
                [&](auto& e) {AddEmployee(e.second->Clone());
                });
}

void Company::operator=(Company comp)
{
        // copy and swap
        std::swap(m_Employees, comp.m_Employees);
        std::swap(m_companyName, comp.m_companyName);
}

size_t Company::GetCompanySize() const
{
        return m_Employees.size();
}

size_t Company::GetWorkerCount(const TWorker& workerType) const
{
        // Count all Employees where workerType is equal
        return count_if(m_Employees.cbegin(), m_Employees.cend(),
                                        [&](auto& e) {return e.second->GetWorkerType() == workerType;});
```

```cpp
}

size_t Company::GetSoldItems() const
{
        return accumulate(m_Employees.cbegin(), m_Employees.cend(),static_cast<size_t>(0),
                [](size_t val, const auto& e) { return val + e.second->GetSoldItems();});
}

size_t Company::GetProducedItems() const
{
        return accumulate(m_Employees.cbegin(), m_Employees.cend(), static_cast<size_t>(0),
                [](size_t val, const auto& e) { return val + e.second->GetProducedItems();});
}

size_t Company::GetCountWorkerBeforDate(const TDate& date) const
{
        return count_if(m_Employees.cbegin(), m_Employees.cend(),
                [&](const auto& e) {return e.second->GetDateBirth() < date;});
}

Employee const * Company::FindWorkerByID(const std::string& id) const
{
        auto empl = m_Employees.find(id);

        if (empl == m_Employees.cend()) return nullptr;
        else return empl->second;
}

Employee const * Company::GetLongestServing(void) const
{
        auto minElem = min_element(m_Employees.cbegin(), m_Employees.cend(),
                [](const auto& lhs, const auto& rhs) { return lhs.second->GetDateJoined() < rhs.second->GetDateJoined();});

        if (minElem == m_Employees.end()) return nullptr;
        else return minElem->second;

}

std::ostream& Company::PrintDataSheet(std::ostream& ost) const
{

        // convert system clock.now to days -> this can be used in CTOR for year month day
        std::chrono::year_month_day date{ floor<std::chrono::days>(std::chrono::system_clock::now()) };

        if (!ost.good()) throw Object::ERROR_BAD_OSTREAM;

        ost << hstar;
        ost << m_companyName << endl;
        ost << hstar;

        for_each(m_Employees.cbegin(), m_Employees.cend(), [&](const auto& e) { e.second->PrintDatasheet(ost);});

        ost << hline;
        ost << date.month() << "␣" << date.year() << endl;
        ost << hline;

        if (ost.fail()) throw Object::ERROR_FAIL_WRITE;

        return ost;
}

Company::~Company()
{
        for (auto & elem : m_Employees)
        {
                delete elem.second;
        }

        m_Employees.clear();
}
```

6.7 TWorker.hpp

```cpp
/*****************************************************************//**
 * \file   TWorker.hpp
 * \brief  Enum for indicating the worker Type
 *
 * \author Simon
 * \date   October 2025
 *********************************************************************/
#ifndef TWORKER_HPP
#define TWORKER_HPP

// changed naming convention because of
// name clashes with the actual classes
// that had the same name.
enum TWorker
{
    E_Boss,
    E_CommisionWorker,
    E_HourlyWorker,
    E_PieceWorker
};

#endif // !TWORKER_HPP
```

6.8 Employee.hpp

```
/*****************************************************************//**
 * \file   Employee.hpp
 * \brief  Abstract Class for constructing Employees of all types
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string>
#include <chrono>
#include "Object.hpp"
#include "TWorker.hpp"

using TDate = std::chrono::year_month_day;

class Employee : public Object
{
public:

    inline static const std::string ERROR_BAD_ID = "ERROR: An employees ID is limited to 3 characters.";
    inline static const std::string ERROR_BAD_SOZIAL_SEC_NUM = "ERROR: Invalid Sozial Security Number";

    /**
     * \brief Returns the ID of an Employee.
     *
     * \return String indication the ID
     */
    std::string GetID() const;

    /**
     * \brief Constructor needs every
     * member set to be called.
     * \return TWorker enum
     */
    Employee(
        const std::string &    name,
        const std::string &    nameID,
        const TDate       &    dateJoined,
        const TDate       &    TDateBirthdaydateBirth,
        const std::string &    socialSecurityNumber
    );

    /**
     * \brief Gives Information about what kind
     * of Worker it is.
     * \return TWorker enum
     */
    virtual TWorker GetWorkerType() const = 0;

    /** Pure Virtual Function
     * \brief return produced items.
     * \return size_t
     */
    virtual size_t GetProducedItems() const = 0;

    /** Pure Virtual Function
     * \brief returns sold items
     * \return size_t
     */
    virtual size_t GetSoldItems() const = 0;

    /** Pure Virtual Function
     * \brief returns total pay a worker
     * recieves.
     * \return size_t
     */
    virtual size_t GetSalary() const = 0;

    /**
     * \brief returns date of birth of a given worker.
     * \return TDate
     */
    TDate GetDateBirth() const;

    /**
     * \brief returns the date a worker.
     * has started working at the company.
     * \return TDate
     */
    TDate GetDateJoined() const;

    /**
     * \brief Prints information about a worker.
     * \return std::ostream&
     */
    std::ostream& PrintDatasheet(std::ostream& ost) const;



    /** Pure virtual function
    * \brief creates a copy of the worker and puts it on the heap.
    * \return Employee*
    */
    virtual Employee* Clone() const = 0;

private:

    /** Pure virtual function
    * \brief Prints specific information for a type of worker.
    * \return std::ostream&
    */
    virtual std::ostream& DoPrintSpecificData(std::ostream& ost) const = 0;


    std::string m_name;
    std::string m_nameIdentifier;
    TDate m_dateJoined;
    TDate m_dateBirth;
    std::string m_socialSecurityNumber;

    const size_t SozialSecNumLen = 4;
};

#endif // EMPLOYEE_H
```

6.9 Employee.cpp

```cpp
/**********************************************************************//**
 * \file   Employee.cpp
 * \brief  Abstract Class for constructing Employees of all types
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#include "Employee.hpp"
#include <cctype>
#include <algorithm>

Employee::Employee(
    const std::string &     name,
    const std::string &     nameID,
    const TDate &       dateJoined,
    const TDate &  dateBirth,
    const std::string  &   socialSecurityNumber
) : m_name{ name },
m_nameIdentifier{ nameID },
m_dateJoined{ dateJoined },
m_dateBirth{ dateBirth }
{
    if (nameID.length() != 3) throw ERROR_BAD_ID;

    if (! std::all_of(socialSecurityNumber.begin(), socialSecurityNumber.end(), ::isdigit))  throw ERROR_BAD_SOZIAL_SEC_NUM;

    if (! (socialSecurityNumber.size() == SozialSecNumLen) )  throw ERROR_BAD_SOZIAL_SEC_NUM;

    m_socialSecurityNumber = socialSecurityNumber;

}

std::string Employee::GetID() const
{
    return m_nameIdentifier;
}

TDate Employee::GetDateBirth() const
{
    return m_dateBirth;
}

TDate Employee::GetDateJoined() const
{
    return m_dateJoined;
}

std::ostream& Employee::PrintDatasheet(std::ostream& ost) const
{
    if (ost.bad())
    {
        throw Object::ERROR_BAD_OSTREAM;
    }

    ost << "Datenblatt\n---------------\n";
    ost << "Name:␣" << m_name << std::endl;
    ost << "Kuerzel:␣" << m_nameIdentifier << std::endl;
    ost << "Sozialversicherungsnummer:␣" << m_socialSecurityNumber;
    ost << m_dateBirth.day() << static_cast<unsigned>(m_dateBirth.month()) << static_cast<int>(m_dateBirth.year())%100 << std::endl;
    ost << "Geburtstag:␣" << m_dateBirth << std::endl;
    ost << "Einstiegsjahr:␣" << m_dateJoined.year() << std::endl;

    DoPrintSpecificData(ost);

    ost << std::endl;

    return ost;
}
```

6.10 Boss.hpp

```
/*******************************************************************//**
 * \file   Boss.hpp
 * \brief  Boss Class - inherits from Employee
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#ifndef BOSS_H
#define BOSS_H

#include "Employee.hpp"

class Boss : public Employee
{
public:

    Boss(
        const std::string & name,
        const std::string & nameID,
        const TDate & dateJoined,
        const TDate & dateBirth,
        const std::string & socialSecurityNumber,
        const size_t & salary
    );


    /**
     * \brief Just here because of whacky class structure.
     * Worker does not strictly produce items!
     */
    size_t GetProducedItems() const override { return 0; };

    /**
     * \brief Just here because of whacky class structure.
     * Worker Does not sell items!
     */
    size_t GetSoldItems() const override { return 0; };

    /**
    * \brief Returns the total earnings for an
    * worker in this month.
    * \return size_t
    */
    size_t GetSalary() const override;

    /**
     * \brief Returns the type of worker.
     * \return TWorker
     */
    TWorker GetWorkerType() const override;

    /**
     * \brief Creates a clone on the Heap
     * and returns a pointer.
     * \return Employee*
     */
    Employee* Clone() const override;

private:
    /**
     * \brief Prints worker specific information
     * \param std::ostream& ost
     * \return std::ostream&
     */
    std::ostream& DoPrintSpecificData(std::ostream& ost) const override;

    size_t m_salary;
};

#endif // BOSS_H
```

6.11 Boss.cpp

```cpp
/*******************************************************************//**
 * \file   Boss.cpp
 * \brief  Boss Class - inherits from Employee
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#include "Boss.hpp"

Boss::Boss(
    const std::string & name,
    const std::string & nameID,
    const TDate & dateJoined,
    const TDate & dateBirth,
    const std::string & socialSecurityNumber,
    const size_t & salary
) :
    Employee(name, nameID, dateJoined, dateBirth, socialSecurityNumber),
    m_salary{ salary } {}

std::ostream& Boss::DoPrintSpecificData(std::ostream& ost) const
{
    if (ost.bad())
    {
        throw Object::ERROR_BAD_OSTREAM;
        return ost;
    }
    ost << "Role:␣Boss" << std::endl;
    ost << "Salary:␣" << m_salary << "␣EUR" << std::endl;

    return ost;
}

size_t Boss::GetSalary() const
{
    return m_salary;
}

TWorker Boss::GetWorkerType() const
{
    return E_Boss;
}

Employee* Boss::Clone() const
{
    return new Boss{ *this };
}
```

6.12 HourlyWorker.hpp

```
/*******************************************************************//**
 * \file   HourlyWorker.hpp
 * \brief  HourlyWorker Class - Inherits from Employee
 * \author Simon
 * \date   October 2025
 *********************************************************************/
#ifndef HOURLY_WORKER_HPP
#define HOURLY_WORKER_HPP

#include "Employee.hpp"

class HourlyWorker : public Employee
{
public:


    HourlyWorker(
        const std::string & name,
        const std::string & nameID,
        const TDate & dateJoined,
        const TDate & dateBirth,
        const std::string & socialSecurityNumber,
        const size_t & hourlyRate,
        const size_t & workedHours
    );


    /**
     * \brief Just here because of whacky class structure.
     * Worker does not strictly produce items!
     */
    size_t GetProducedItems() const override { return 0; };

    /**
     * \brief Just here because of whacky class structure.
     * Worker Does not sell items!
     */
    size_t GetSoldItems() const override { return 0; };

    /**
    * \brief Returns the total earnings for an
    * worker in this month.
    * \return size_t
    */
    size_t GetSalary() const override;

    /**
     * \brief Returns the type of worker.
     * \return TWorker
     */
    TWorker GetWorkerType() const override;

    /**
     * \brief Creates a clone on the Heap
     * and returns a pointer.
     * \return Employee*
     */
    Employee* Clone() const override;

private:
    /**
     * \brief Prints worker specific information
     * \param std::ostream& ost
     * \return std::ostream&
     */
    std::ostream& DoPrintSpecificData(std::ostream& ost) const override;

    size_t m_hourlyRate;
    size_t m_workedHours;
};

#endif // !HOURLY_WORKER_HPP
```

6.13 HourlyWorker.cpp

```cpp
/*********************************************************************//**
 * \file   HourlyWorker.cpp
 * \brief  HourlyWorker Class - Inherits from Employee
 * \author Simon
 * \date   October 2025
 *********************************************************************/

#include "HourlyWorker.hpp"

HourlyWorker::HourlyWorker(
    const std::string & name,
    const std::string & nameID,
    const TDate & dateJoined,
    const TDate & dateBirth,
    const std::string & socialSecurityNumber,
    const size_t & hourlyRate,
    const size_t & workedHours
) :
    Employee(name, nameID, dateJoined, dateBirth, socialSecurityNumber),
    m_hourlyRate{ hourlyRate },
    m_workedHours{ workedHours }
{}

std::ostream& HourlyWorker::DoPrintSpecificData(std::ostream& ost) const
{
    if (ost.bad())
    {
        throw Object::ERROR_BAD_OSTREAM;
        return ost;
    }
    ost << "Role: HourlyWWorker" << std::endl;
    ost << "Hourly rate: " << m_hourlyRate << " EUR" << std::endl;
    ost << "Hours worked: " << m_workedHours << " EUR" << std::endl;

    return ost;
}

size_t HourlyWorker::GetSalary() const
{
    return m_hourlyRate * m_workedHours;
}

TWorker HourlyWorker::GetWorkerType() const
{
    return E_HourlyWorker;
}

Employee* HourlyWorker::Clone() const
{
    return new HourlyWorker{*this};
}
```

6.14 PieceWorker.hpp

```
/*********************************************************************//**
 * \file   PieceWorker.hpp
 * \brief  PieceWorker Class - inherits from Employee
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#ifndef PIECE_WORKER_H
#define PIECE_WORKER_H

#include "Employee.hpp"

class PieceWorker : public Employee
{
public:

    PieceWorker(
        const std::string & name,
        const std::string & nameID,
        const TDate & dateJoined,
        const TDate & dateBirth,
        const std::string & socialSecurityNumber,
        const size_t & m_numberPieces,
        const size_t & m_commisionPerPiece
    );

    /**
     * \brief Returns the number of pieces the
     * worker has produced
     */
    size_t GetProducedItems() const override;

    /**
     * \brief Just here because of whacky class structure.
     * Worker does not strictly sell items!
     */
    size_t GetSoldItems() const override { return 0; };

    /**
    * \brief Returns the total earnings for an
    * worker in this month.
    * \return size_t
    */
    size_t GetSalary() const override;

    /**
     * \brief Returns the type of worker.
     * \return TWorker
     */
    TWorker GetWorkerType() const override;

    /**
     * \brief Creates a clone on the Heap
     * and returns a pointer.
     * \return Employee*
     */
    Employee* Clone() const override;

private:
    /**
     * \brief Prints worker specific information
     * \param std::ostream& ost
     * \return std::ostream&
     */
    std::ostream& DoPrintSpecificData(std::ostream& ost) const override;

    size_t m_numberPieces;
    size_t m_commisionPerPiece;
};

#endif // !PIECE_WORKER_H
```

6.15 PieceWorker.cpp

```
/*******************************************************************//**
 * \file   PieceWorker.cpp
 * \brief  PieceWorker Class - inherits from Employee
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#include "PieceWorker.hpp"

PieceWorker::PieceWorker(
    const std::string & name,
    const std::string & nameID,
    const TDate & dateJoined,
    const TDate & dateBirth,
    const std::string & socialSecurityNumber,
    const size_t & m_numberPieces,
    const size_t & m_commisionPerPiece
) :
    Employee(name, nameID, dateJoined, dateBirth, socialSecurityNumber),
    m_numberPieces{ m_numberPieces },
    m_commisionPerPiece{ m_commisionPerPiece }{}

std::ostream& PieceWorker::DoPrintSpecificData(std::ostream& ost) const
{
    if (ost.bad())
    {
        throw Object::ERROR_BAD_OSTREAM;
        return ost;
    }
    ost << "Role:␣PieceWorker" << std::endl;
    ost << "Pieces␣produced:␣" << m_numberPieces << std::endl;
    ost << "Pay␣per␣piece:␣" << m_commisionPerPiece << "␣EUR" << std::endl;

    return ost;
}

size_t PieceWorker::GetProducedItems() const
{
    return m_numberPieces;
}

size_t PieceWorker::GetSalary() const
{
    return m_numberPieces * m_commisionPerPiece;
}

TWorker PieceWorker::GetWorkerType() const
{
    return E_PieceWorker;
}

Employee* PieceWorker::Clone() const
{
    return new PieceWorker{ *this };
}
```

6.16 ComissionWorker.hpp

```
/*******************************************************************//**
 * \file   ComissionWorker.hpp
 * \brief  ComissionWorker Class - inherits from Employee
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#ifndef COMISSION_WORKER_H
#define COMISSION_WORKER_H

#include "Employee.hpp"

class ComissionWorker : public Employee
{
public:

    ComissionWorker(
        const std::string & name,
        const std::string & nameID,
        const TDate & dateJoined,
        const TDate & dateBirth,
        const std::string & socialSecurityNumber,
        const size_t & baseSalary,
        const size_t & commisionPerPiece,
        const size_t & piecesSold
    );

    /**
     * \brief Just here because of whacky class structure.
     * Worker does not strictly produce items!
     */
    size_t GetProducedItems() const override { return 0; };

    /**
     * \brief returns how many items the commision worker has sold
     * \return size_t sold items
     */
    size_t GetSoldItems() const override;

    /**
    * \brief Returns the total earnings for an
    * worker in this month.
    * \return size_t
    */
    size_t GetSalary() const override;

    /**
     * \brief Returns the type of worker.
     * \return TWorker
     */
    TWorker GetWorkerType() const override;

    /**
     * \brief Creates a clone on the Heap
     * and returns a pointer.
     * \return Employee*
     */
    Employee* Clone() const override;

private:
    /**
     * \brief Prints worker specific information
     * \param std::ostream& ost
     * \return std::ostream&
     */
    std::ostream& DoPrintSpecificData(std::ostream& ost) const override;

    size_t m_baseSalary;
    size_t m_commisionPerPiece;
    size_t m_piecesSold;
};

#endif // !COMISSION_WORKER_H
```

6.17 ComissionWorker.cpp

```
/*********************************************************************//**
 * \file   ComissionWorker.cpp
 * \brief  ComissionWorker Class - inherits from Employee
 * \author Simon Vogelhuber
 * \date   October 2025
 *********************************************************************/
#include "ComissionWorker.hpp"

ComissionWorker::ComissionWorker(
    const std::string & name,
    const std::string & nameID,
    const TDate & dateJoined,
    const TDate & dateBirth,
    const std::string & socialSecurityNumber,
    const size_t & baseSalary,
    const size_t & commisionPerPiece,
    const size_t & piecesSold
) :
    Employee(name, nameID, dateJoined, dateBirth, socialSecurityNumber),
    m_baseSalary{ baseSalary },
    m_commisionPerPiece{ commisionPerPiece },
    m_piecesSold { piecesSold }
{}

std::ostream& ComissionWorker::DoPrintSpecificData(std::ostream & ost) const
{
    if (ost.bad())
    {
        throw Object::ERROR_BAD_OSTREAM;
        return ost;
    }
    ost << "Role: ComissionWorker" << std::endl;
    ost << "Base salary: " << m_baseSalary << " EUR" << std::endl;
    ost << "Comission per piece: " << m_commisionPerPiece << " EUR" << std::endl;
    ost << "Pieces sold: " << m_piecesSold << std::endl;

    return ost;
}

size_t ComissionWorker::GetSoldItems() const
{
    return m_piecesSold;
}

size_t ComissionWorker::GetSalary() const
{
    return m_baseSalary + m_piecesSold * m_commisionPerPiece;
}

TWorker ComissionWorker::GetWorkerType() const
{
    return E_CommisionWorker;
}

Employee* ComissionWorker::Clone() const
{
    return new ComissionWorker{ *this };
}
```

6.18 main.cpp

```cpp
/*****************************************************************//**
 * \file   main.cpp
 * \brief  Testdriver for the Company
 *
 * \author Simon
 * \date   October 2025
 *********************************************************************/
#include "Company.hpp"
#include "Employee.hpp"
#include "HourlyWorker.hpp"
#include "vld.h"
#include "Client.hpp"
#include "Test.hpp"
#include "ComissionWorker.hpp"
#include "Boss.hpp"
#include "PieceWorker.hpp"
#include <iostream>
#include <fstream>
#include <cassert>

using namespace std;
using namespace std::chrono;

static bool TestEmployeeBoss(std::ostream& ost);
static bool TestEmployeeHourlyWorker(std::ostream& ost);
static bool TestEmployeePieceWorker(std::ostream& ost);
static bool TestEmployeeComissionWorker(std::ostream& ost);
static bool TestCompanyAdd(std::ostream& ost);

#define WRITE_OUTPUT true

int main(void){
        bool TestOK = true;
        ofstream testoutput;
        try {

                if (WRITE_OUTPUT == true) {
                        testoutput.open("TestOutput.txt");
                }

                Company comp{ "Offenberger␣Devices" };
                Client TestClient;
                ComissionWorker* cWork = new ComissionWorker{ "Simon␣1", "Si1", { 2022y,November,23d }, { 2000y,November,22d }, "4711", 2500, 25, 2500 };
                ComissionWorker* cWork2 = new ComissionWorker{ "Simon␣6", "Si6", { 2022y,November,23d }, { 2000y,November,22d }, "4711", 2500, 25, 200 };
                HourlyWorker* hWork = new HourlyWorker{ "Simon␣2", "Si2", { 2022y,November,23d }, { 1934y,November,23d },"4712",20,25 };
                Boss* boss = new Boss{ "Simon␣3", "Si3", { 2000y,November,23d }, { 1950y,November,23d },"4712",35000 };
                PieceWorker* pWork = new PieceWorker{ "Simon␣4", "Si4", { 2022y,November,23d }, { 2010y,November,23d },"4712",25,25 };
                PieceWorker* pWork2 = new PieceWorker{ "Simon␣5", "Si5", { 2022y,November,23d }, { 2011y,November,23d },"4712",25,25 };

                comp.AddEmployee(cWork);
                comp.AddEmployee(cWork2);
                comp.AddEmployee(hWork);
                comp.AddEmployee(boss);
                comp.AddEmployee(pWork);
                comp.AddEmployee(pWork2);

                TestOK = TestOK && TestClient.TestCompanyGetter(cout, comp);
                if (WRITE_OUTPUT) TestOK = TestOK && TestClient.TestCompanyGetter(testoutput, comp);

                // Copy Ctor Call !
                Company compCopy = comp;

                TestOK = TestOK && TestClient.TestCompanyCopyCTOR(cout, comp, compCopy);
                if (WRITE_OUTPUT) TestOK = TestOK && TestClient.TestCompanyCopyCTOR(testoutput, comp, compCopy);

                // Test Assign Operator
                Company compAss{ "Assign␣Company" };
                compAss = comp;

                TestOK = TestOK && TestClient.TestCompanyAssignOp(cout, comp, compAss);
                if (WRITE_OUTPUT) TestOK = TestOK && TestClient.TestCompanyAssignOp(testoutput, comp, compAss);


```

```
		TestOK = TestOK && TestClient.TestCompanyPrint(cout, comp);
		if (WRITE_OUTPUT) TestOK = TestOK && TestClient.TestCompanyPrint(testoutput, comp);

		Company emptyComp{ "empty" };

		TestOK = TestOK && TestClient.TestEmptyCompanyGetter(cout, emptyComp);
		if (WRITE_OUTPUT) TestOK = TestOK && TestClient.TestEmptyCompanyGetter(testoutput, emptyComp);

		// Test Boss
		TestOK = TestOK && TestEmployeeBoss(cout);
		if (WRITE_OUTPUT) TestOK = TestOK && TestEmployeeBoss(testoutput);

		// Test Hourly Worker
		TestOK = TestOK && TestEmployeeHourlyWorker(cout);
		if (WRITE_OUTPUT) TestOK = TestOK && TestEmployeeHourlyWorker(testoutput);

		// Test Piece Worker
		TestOK = TestOK && TestEmployeePieceWorker(cout);
		if (WRITE_OUTPUT) TestOK = TestOK && TestEmployeePieceWorker(testoutput);

		// Test Comission Worker
		TestOK = TestOK && TestEmployeeComissionWorker(cout);
		if (WRITE_OUTPUT) TestOK = TestOK && TestEmployeeComissionWorker(testoutput);

		// Test Company Add
		TestOK = TestOK && TestCompanyAdd(cout);
		if (WRITE_OUTPUT) TestOK = TestOK && TestCompanyAdd(testoutput);

		if (WRITE_OUTPUT) {
			if (TestOK) TestCaseOK(testoutput);
			else TestCaseFail(testoutput);

			testoutput.close();
		}

		if (TestOK) TestCaseOK(cout);
		else TestCaseFail(cout);
	}
	catch (const string& err) {
		cerr << err;
	}
	catch (bad_alloc const& error) {
		cerr << error.what();
	}
	catch (const exception& err) {
		cerr << err.what();
	}
	catch (...) {
		cerr << "Unhandelt␣Exception";
	}

	if (testoutput.is_open()) testoutput.close();

}



static bool TestEmployeeBoss(std::ostream& ost)
{

	assert(ost.good());

	TestStart(ost);

	bool TestOK = true;
	string error_msg = "";

	try {
		size_t testSalary = 7800;
		string svr = "4711";
		TDate dateBorn = { 2000y,November,22d };
		TDate dateJoined = { 2022y,November,23d };
		string name = "Max␣Musterman";
		string id = "MAX";

		Boss testBoss{ name, id, dateJoined, dateBorn, svr, testSalary };
```

```

                TestOK = TestOK && check_dump(ost, "Test␣-␣Boss.GetSalary()", testSalary, testBoss.GetSalary());
                TestOK = TestOK && check_dump(ost, "Test␣-␣Boss.GetSoldItems()", static_cast<size_t>(0), testBoss.GetSoldItems());
                TestOK = TestOK && check_dump(ost, "Test␣-␣Boss.GetProducedItems()", static_cast<size_t>(0), testBoss.GetProducedItems());
                TestOK = TestOK && check_dump(ost, "Test␣-␣Boss.GetWorkerType()", E_Boss, testBoss.GetWorkerType());
                TestOK = TestOK && check_dump(ost, "Test␣-␣Boss.GetDateBirth()", dateBorn, testBoss.GetDateBirth());
                TestOK = TestOK && check_dump(ost, "Test␣-␣Boss.GetDateJoined()", dateJoined, testBoss.GetDateJoined());
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣-␣error␣buffer", error_msg.empty(), true);
        error_msg.clear();

        //clone test
        try {
                size_t testSalary = 7800;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                Boss testBoss{ name, id, dateJoined, dateBorn, svr, testSalary };
                Employee* pEmp = testBoss.Clone();
                TestOK = TestOK && check_dump(ost, "Test␣Boss.Clone()", pEmp != nullptr && pEmp != &testBoss, true);
                delete pEmp;
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣-␣error␣buffer", error_msg.empty(), true);
        error_msg.clear();

        // Unavialable ID
        try {
                size_t testSalary = 7800;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAXL";

                Boss testBoss{ name, id, dateJoined, dateBorn, svr, testSalary };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Boss␣Constructor␣bad␣ID", error_msg, Employee::ERROR_BAD_ID);
        error_msg.clear();

        // Constructor bad SV
        try {
                size_t testSalary = 7800;
                string svr = "ARGH";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                Boss testBoss{ name, id, dateJoined, dateBorn, svr, testSalary };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Boss␣Constructor␣bad␣SV␣-␣invalid␣character", Employee::ERROR_BAD_SOZIAL_SEC_NUM, error_msg);

        error_msg.clear();


        // Constructor bad SV - too many nums
        try {
                size_t testSalary = 7800;
                string svr = "ARGH";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                Boss testBoss{ name, id, dateJoined, dateBorn, svr, testSalary };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Boss␣Constructor␣bad␣SV␣-␣too␣many␣nums", Employee::ERROR_BAD_SOZIAL_SEC_NUM, error_msg);
        error_msg.clear();

        // Bad ostream
        try {
                size_t testSalary = 7800;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                Boss testBoss{ name, id, dateJoined, dateBorn, svr, testSalary };
                std::stringstream out_stream;
                out_stream.setstate(ios::badbit);
                testBoss.PrintDatasheet(out_stream);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Boss␣bad␣ostream", error_msg, Object::ERROR_BAD_OSTREAM);
        error_msg.clear();

        TestEnd(ost);
        return TestOK;
}

static bool TestEmployeeHourlyWorker(std::ostream& ost)
{
        assert(ost.good());

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {
                size_t hourlyRate = 21;
                size_t workedHours = 160;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                HourlyWorker testHourlyWorker{ name, id, dateJoined, dateBorn, svr, hourlyRate, workedHours };

                TestOK = TestOK && check_dump(ost, "Test␣-␣HourlyWorker.GetSalary()", hourlyRate * workedHours, testHourlyWorker.GetSalary());
                TestOK = TestOK && check_dump(ost, "Test␣-␣HourlyWorker.GetSoldItems()", static_cast<size_t>(0), testHourlyWorker.GetSoldItems());
                TestOK = TestOK && check_dump(ost, "Test␣-␣HourlyWorker.GetProducedItems()", static_cast<size_t>(0), testHourlyWorker.GetProducedItems());
                TestOK = TestOK && check_dump(ost, "Test␣-␣HourlyWorker.GetWorkerType()", E_HourlyWorker, testHourlyWorker.GetWorkerType());
                TestOK = TestOK && check_dump(ost, "Test␣-␣HourlyWorker.GetDateBirth()", dateBorn, testHourlyWorker.GetDateBirth());
                TestOK = TestOK && check_dump(ost, "Test␣-␣HourlyWorker.GetDateJoined()", dateJoined, testHourlyWorker.GetDateJoined());
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣-␣error␣buffer", error_msg.empty(), true);
        error_msg.clear();

        //clone test
        try {
                size_t hourlyRate = 21;
                size_t workedHours = 160;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                HourlyWorker testHourlyWorker{ name, id, dateJoined, dateBorn, svr, hourlyRate, workedHours };

                Employee* pEmp = testHourlyWorker.Clone();
                TestOK = TestOK && check_dump(ost, "Test␣testPieceWorker.Clone()", pEmp != nullptr && pEmp != &testHourlyWorker, true);
                delete pEmp;
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣-␣error␣buffer", error_msg.empty(), true);
        error_msg.clear();

        // Unavialable ID
        try {
                size_t hourlyRate = 21;
                size_t workedHours = 160;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAXL";

                HourlyWorker testHourlyWorker{ name, id, dateJoined, dateBorn, svr, hourlyRate, workedHours };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "HourlyWorker␣Constructor␣bad␣ID", error_msg, Employee::ERROR_BAD_ID);
        error_msg.clear();

        // Constructor bad SV
        try {
                size_t hourlyRate = 21;
                size_t workedHours = 160;
                string svr = "ARGH";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                HourlyWorker testHourlyWorker{ name, id, dateJoined, dateBorn, svr, hourlyRate, workedHours };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "HourlyWorker␣Constructor␣bad␣SV␣-␣invalid␣character", Employee::ERROR_BAD_SOZIAL_SEC_NUM, error_msg);

        error_msg.clear();

        // Constructor bad SV - too many nums
```

```cpp
        try {
                size_t hourlyRate = 21;
                size_t workedHours = 160;
                string svr = "ARGH";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                HourlyWorker testHourlyWorker{ name, id, dateJoined, dateBorn, svr, hourlyRate, workedHours };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "HourlyWorker␣Constructor␣bad␣SV␣-␣too␣many␣nums", Employee::ERROR_BAD_SOZIAL_SEC_NUM, error_msg);
        error_msg.clear();

        // Bad ostream
        try {
                size_t hourlyRate = 21;
                size_t workedHours = 160;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                HourlyWorker testHourlyWorker{ name, id, dateJoined, dateBorn, svr, hourlyRate, workedHours };
                std::stringstream out_stream;
                out_stream.setstate(ios::badbit);
                testHourlyWorker.PrintDatasheet(out_stream);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "HourlyWorker␣bad␣ostream", error_msg, Object::ERROR_BAD_OSTREAM);
        error_msg.clear();

        TestEnd(ost);
        return TestOK;
}

static bool TestEmployeePieceWorker(std::ostream& ost)
{
        assert(ost.good());

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {
                size_t piecesProduced = 950;
                size_t comissionPerPiece = 2;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                PieceWorker testHourlyWorker{ name, id, dateJoined, dateBorn, svr, piecesProduced, comissionPerPiece };

                TestOK = TestOK && check_dump(ost, "Test␣-␣PieceWorker.GetSalary()", piecesProduced * comissionPerPiece, testHourlyWorker.GetSalary());
                TestOK = TestOK && check_dump(ost, "Test␣-␣PieceWorker.GetSoldItems()", static_cast<size_t>(0), testHourlyWorker.GetSoldItems());
                TestOK = TestOK && check_dump(ost, "Test␣-␣PieceWorker.GetProducedItems()", piecesProduced, testHourlyWorker.GetProducedItems());
                TestOK = TestOK && check_dump(ost, "Test␣-␣PieceWorker.GetWorkerType()", E_PieceWorker, testHourlyWorker.GetWorkerType());
                TestOK = TestOK && check_dump(ost, "Test␣-␣PieceWorker.GetDateBirth()", dateBorn, testHourlyWorker.GetDateBirth());
                TestOK = TestOK && check_dump(ost, "Test␣-␣PieceWorker.GetDateJoined()", dateJoined, testHourlyWorker.GetDateJoined());
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣-␣error␣buffer", error_msg.empty(), true);
        error_msg.clear();

        //clone test
        try {
                size_t piecesProduced = 950;
                size_t comissionPerPiece = 2;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                PieceWorker testPieceWorker{ name, id, dateJoined, dateBorn, svr, piecesProduced, comissionPerPiece };
                Employee* pEmp = testPieceWorker.Clone();
                TestOK = TestOK && check_dump(ost, "Test␣testPieceWorker.Clone()", pEmp != nullptr && pEmp != &testPieceWorker, true);
                delete pEmp;
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣-␣error␣buffer", error_msg.empty(), true);
        error_msg.clear();

        // Unavialable ID
        try {
                size_t piecesProduced = 950;
                size_t comissionPerPiece = 2;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAXL";

                PieceWorker testPieceWorker{ name, id, dateJoined, dateBorn, svr, piecesProduced, comissionPerPiece };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
```

```
			error_msg = error.what();
		}
		catch (const exception& err) {
			error_msg = err.what();
		}
		catch (...) {
			error_msg = "Unhandelt␣Exception";
		}

		TestOK = TestOK && check_dump(ost, "PieceWorker␣Constructor␣bad␣ID", error_msg, Employee::ERROR_BAD_ID);
		error_msg.clear();

		// Constructor bad SV
		try {
			size_t piecesProduced = 950;
			size_t comissionPerPiece = 2;
			string svr = "ARGH";
			TDate dateBorn = { 2000y,November,22d };
			TDate dateJoined = { 2022y,November,23d };
			string name = "Max␣Musterman";
			string id = "MAX";

			PieceWorker testPieceWorker{ name, id, dateJoined, dateBorn, svr, piecesProduced, comissionPerPiece };
		}
		catch (const string& err) {
			error_msg = err;
		}
		catch (bad_alloc const& error) {
			error_msg = error.what();
		}
		catch (const exception& err) {
			error_msg = err.what();
		}
		catch (...) {
			error_msg = "Unhandelt␣Exception";
		}

		TestOK = TestOK && check_dump(ost, "PieceWorker␣Constructor␣bad␣SV␣-␣invalid␣character", Employee::ERROR_BAD_SOZIAL_SEC_NUM, error_msg);

		error_msg.clear();

		// Constructor bad SV - too many nums
		try {
			size_t piecesProduced = 950;
			size_t comissionPerPiece = 2;
			string svr = "ARGH";
			TDate dateBorn = { 2000y,November,22d };
			TDate dateJoined = { 2022y,November,23d };
			string name = "Max␣Musterman";
			string id = "MAX";

			PieceWorker testPieceWorker{ name, id, dateJoined, dateBorn, svr, piecesProduced, comissionPerPiece };
		}
		catch (const string& err) {
			error_msg = err;
		}
		catch (bad_alloc const& error) {
			error_msg = error.what();
		}
		catch (const exception& err) {
			error_msg = err.what();
		}
		catch (...) {
			error_msg = "Unhandelt␣Exception";
		}

		TestOK = TestOK && check_dump(ost, "PieceWorker␣Constructor␣bad␣SV␣-␣too␣many␣nums", Employee::ERROR_BAD_SOZIAL_SEC_NUM, error_msg);
		error_msg.clear();

		// Bad ostream
		try {
			size_t piecesProduced = 950;
			size_t comissionPerPiece = 2;
			string svr = "4711";
			TDate dateBorn = { 2000y,November,22d };
			TDate dateJoined = { 2022y,November,23d };
```

```cpp
                string name = "Max␣Musterman";
                string id = "MAX";

                PieceWorker testPieceWorker{ name, id, dateJoined, dateBorn, svr, piecesProduced, comissionPerPiece };
                std::stringstream out_stream;
                out_stream.setstate(ios::badbit);
                testPieceWorker.PrintDatasheet(out_stream);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "PieceWorker␣bad␣ostream", error_msg, Object::ERROR_BAD_OSTREAM);
        error_msg.clear();

        TestEnd(ost);
        return TestOK;
}

static bool TestEmployeeComissionWorker(std::ostream& ost)
{
        assert(ost.good());

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {
                size_t baseSalary = 2300;
                size_t piecesSold = 300;
                size_t comissionPerPiece = 2;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                ComissionWorker testHourlyWorker{ name, id, dateJoined, dateBorn, svr, baseSalary, comissionPerPiece, piecesSold };

                TestOK = TestOK && check_dump(ost, "Test␣-␣ComissionWorker.GetSalary()", baseSalary + piecesSold * comissionPerPiece, testHourlyWorker.GetSalary());
                TestOK = TestOK && check_dump(ost, "Test␣-␣ComissionWorker.GetSoldItems()", piecesSold, testHourlyWorker.GetSoldItems());
                TestOK = TestOK && check_dump(ost, "Test␣-␣ComissionWorker.GetProducedItems()", static_cast<size_t>(0), testHourlyWorker.GetProducedItems());
                TestOK = TestOK && check_dump(ost, "Test␣-␣ComissionWorker.GetWorkerType()", E_CommisionWorker, testHourlyWorker.GetWorkerType());
                TestOK = TestOK && check_dump(ost, "Test␣-␣ComissionWorker.GetDateBirth()", dateBorn, testHourlyWorker.GetDateBirth());
                TestOK = TestOK && check_dump(ost, "Test␣-␣ComissionWorker.GetDateJoined()", dateJoined, testHourlyWorker.GetDateJoined());
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣-␣error␣buffer", error_msg.empty(), true);
        error_msg.clear();

        //clone test
        try {
                size_t baseSalary = 2300;
                size_t piecesSold = 300;
                size_t comissionPerPiece = 2;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                ComissionWorker testComissionWorker{ name, id, dateJoined, dateBorn, svr, baseSalary, comissionPerPiece, piecesSold };
                Employee* pEmp = testComissionWorker.Clone();
                TestOK = TestOK && check_dump(ost, "Test␣testPieceWorker.Clone()", pEmp != nullptr && pEmp != &testComissionWorker, true);
                delete pEmp;
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣-␣error␣buffer", error_msg.empty(), true);
        error_msg.clear();

        // Unavialable ID
        try {
                size_t baseSalary = 2300;
                size_t piecesSold = 300;
                size_t comissionPerPiece = 2;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAXL";

                ComissionWorker testComissionWorker{ name, id, dateJoined, dateBorn, svr, baseSalary, comissionPerPiece, piecesSold };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "ComissionWorker␣Constructor␣bad␣ID␣", error_msg, Employee::ERROR_BAD_ID);
        error_msg.clear();

        // Constructor bad SV - no numbers
        try {
                size_t baseSalary = 2300;
                size_t piecesSold = 300;
                size_t comissionPerPiece = 2;
                string svr = "ARGH";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                ComissionWorker testComissionWorker{ name, id, dateJoined, dateBorn, svr, baseSalary, comissionPerPiece, piecesSold };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "ComissionWorker␣Constructor␣bad␣SV␣-␣invalid␣character", Employee::ERROR_BAD_SOZIAL_SEC_NUM, error_msg);

        error_msg.clear();

        // Constructor bad SV - too many nums
        try {
                size_t baseSalary = 2300;
                size_t piecesSold = 300;
                size_t comissionPerPiece = 2;
                string svr = "47488888239874";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                ComissionWorker testComissionWorker{ name, id, dateJoined, dateBorn, svr, baseSalary, comissionPerPiece, piecesSold };
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "ComissionWorker␣Constructor␣bad␣SV␣-␣too␣many␣nums", Employee::ERROR_BAD_SOZIAL_SEC_NUM, error_msg);
        error_msg.clear();

        // Bad ostream
        try {
                size_t baseSalary = 2300;
                size_t piecesSold = 300;
                size_t comissionPerPiece = 2;
                string svr = "4711";
                TDate dateBorn = { 2000y,November,22d };
                TDate dateJoined = { 2022y,November,23d };
                string name = "Max␣Musterman";
                string id = "MAX";

                ComissionWorker testComissionWorker{ name, id, dateJoined, dateBorn, svr, baseSalary, comissionPerPiece, piecesSold };
                std::stringstream out_stream;
                out_stream.setstate(ios::badbit);
                testComissionWorker.PrintDatasheet(out_stream);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "ComissionWorker␣bad␣ostream", error_msg, Object::ERROR_BAD_OSTREAM);
        error_msg.clear();

        TestEnd(ost);
        return TestOK;
}

```

```

static bool TestCompanyAdd(std::ostream& ost)
{
        assert(ost.good());

        TestStart(ost);

        bool TestOK = true;
        string error_msg = "";

        try {

                ComissionWorker* cWork = new ComissionWorker{ "Simon␣1", "Si1", { 2022y,November,23d }, { 2000y,November,22d }, "4711", 2500, 25, 2500 };

                Company comp{"Dup"};
                comp.AddEmployee(cWork);
                comp.AddEmployee(cWork);
        }
        catch (const string& err) {
                error_msg = err;
        }
        catch (bad_alloc const& error) {
                error_msg = error.what();
        }
        catch (const exception& err) {
                error_msg = err.what();
        }
        catch (...) {
                error_msg = "Unhandelt␣Exception";
        }

        TestOK = TestOK && check_dump(ost, "Test␣Exception␣in␣Company␣Add␣Duplicate", Company::ERROR_DUPLICATE_EMPL, error_msg);
        error_msg.clear();

        TestEnd(ost);
        return TestOK;
}
```

6.19 Test.hpp

```cpp
/******************************************************************//**
 * \file   Test.hpp
 * \brief  File that provides a Test Function with a formated output
 *
 * \author Simon
 * \date   April 2025
 **********************************************************************/
#ifndef TEST_HPP
#define TEST_HPP

#include <string>
#include <iostream>
#include <vector>
#include <list>
#include <queue>
#include <forward_list>

#define ON 1
#define OFF 0
#define COLOR_OUTPUT OFF

// Definitions of colors in order to change the color of the output stream.
const std::string colorRed = "\x1B[31m";
const std::string colorGreen = "\x1B[32m";
const std::string colorWhite = "\x1B[37m";

inline std::ostream& RED(std::ostream& ost) {
        if (ost.good()) {
                ost << colorRed;
        }
        return ost;
}
inline std::ostream& GREEN(std::ostream& ost) {
        if (ost.good()) {
                ost << colorGreen;
        }
        return ost;
}
inline std::ostream& WHITE(std::ostream& ost) {
        if (ost.good()) {
                ost << colorWhite;
        }
        return ost;
}

inline std::ostream& TestStart(std::ostream& ost) {
        if (ost.good()) {
                ost << std::endl;
                ost << "*******************************************" << std::endl;
                ost << "␣␣␣␣␣␣␣␣␣␣␣␣␣TESTCASE␣START␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣" << std::endl;
                ost << "*******************************************" << std::endl;
                ost << std::endl;
        }
        return ost;
}

inline std::ostream& TestEnd(std::ostream& ost) {
        if (ost.good()) {
                ost << std::endl;
                ost << "*******************************************" << std::endl;
                ost << std::endl;
        }
        return ost;
}

inline std::ostream& TestCaseOK(std::ostream& ost) {

#if COLOR_OUTPUT
        if (ost.good()) {
                ost << colorGreen << "TEST␣OK!!" << colorWhite << std::endl;
        }
#else
        if (ost.good()) {
                ost <<  "TEST␣OK!!" <<  std::endl;
        }
#endif // COLOR_OUTPUT

        return ost;
}

inline std::ostream& TestCaseFail(std::ostream& ost) {

#if COLOR_OUTPUT
        if (ost.good()) {
                ost << colorRed << "TEST␣FAILED␣!!" << colorWhite << std::endl;

        }
#else
        if (ost.good()) {
                ost << "TEST␣FAILED␣!!" << std::endl;

        }
#endif // COLOR_OUTPUT

        return ost;
}

/**
        * \brief function that reports if the testcase was successful.
        *
        * \param testcase       String that indicates the testcase
        * \param succsessful true -> reports to cout test OK
        * \param succsessful false -> reports test failed
        */
template <typename T>
bool check_dump(std::ostream& ostr, const std::string& testcase, const T& expected, const T& result) {
        if (ostr.good()) {
#if COLOR_OUTPUT
                if (expected == result) {
                        ostr << testcase << std::endl <<  colorGreen << "[Test␣OK]␣" << colorWhite <<"Result:␣(Expected:␣" << std::boolalpha << expected << "␣==" << "␣Result:␣" << result << ")"  << std::noboolalpha << std::endl << std::endl;
                }
                else {
                        ostr << testcase << std::endl << colorRed << "[Test␣FAILED]␣" << colorWhite << "Result:␣(Expected:␣" << std::boolalpha << expected << "␣!=" << "␣Result:␣" << result << ")"  << std::noboolalpha << std::endl << std::endl;
                }
#else
                if (expected == result) {
                        ostr << testcase << std::endl << "[Test␣OK]␣"  << "Result:␣(Expected:␣" << std::boolalpha << expected << "␣==" << "␣Result:␣" << result << ")" << std::noboolalpha << std::endl << std::endl;
                }
                else {
                        ostr << testcase << std::endl  << "[Test␣FAILED]␣"  << "Result:␣(Expected:␣" << std::boolalpha << expected << "␣!=" << "␣Result:␣" << result << ")" << std::noboolalpha << std::endl << std::endl;
                }
#endif
                if (ostr.fail()) {
                        std::cerr << "Error:␣Write␣Ostream" << std::endl;
                }
        }
        else {
                std::cerr << "Error:␣Bad␣Ostream" << std::endl;
        }
        return expected == result;
}

template <typename T1, typename T2>
std::ostream& operator<< (std::ostream& ost,const std::pair<T1,T2> & p) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        ost << "(" << p.first << "," << p.second << ")";
        return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::vector<T> & cont) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
        return ost;
}
```

```

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::list<T> & cont) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
        return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::deque<T> & cont) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
        return ost;
}

template <typename T>
std::ostream& operator<< (std::ostream& ost,const std::forward_list<T> & cont) {
        if (!ost.good()) throw std::exception{ "Error␣bad␣Ostream!" };
        std::copy(cont.cbegin(), cont.cend(), std::ostream_iterator<T>{ost, "␣"});
        return ost;
}


#endif // !TEST_HPP
```