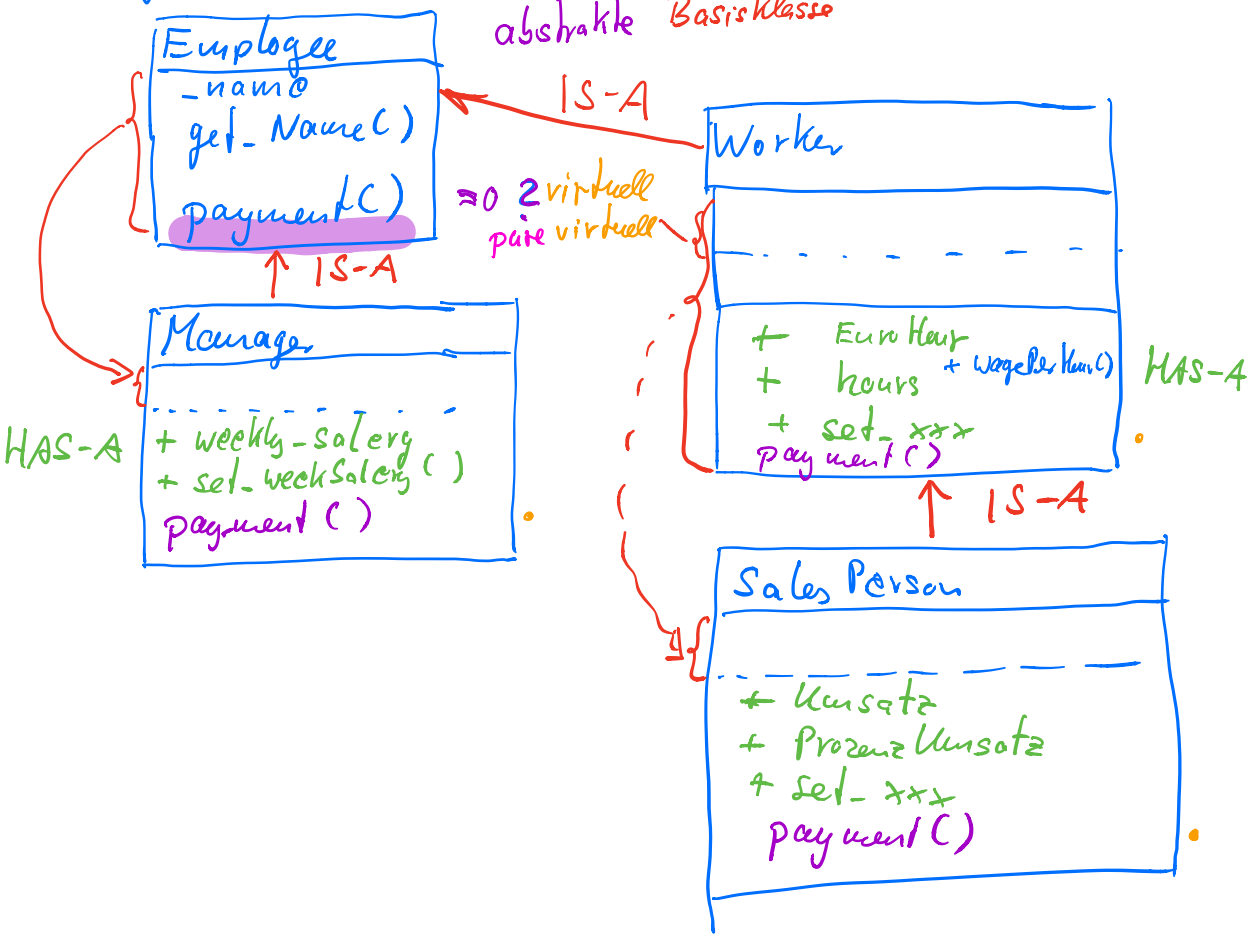
Klassenhierarchien: A Basis Klasse

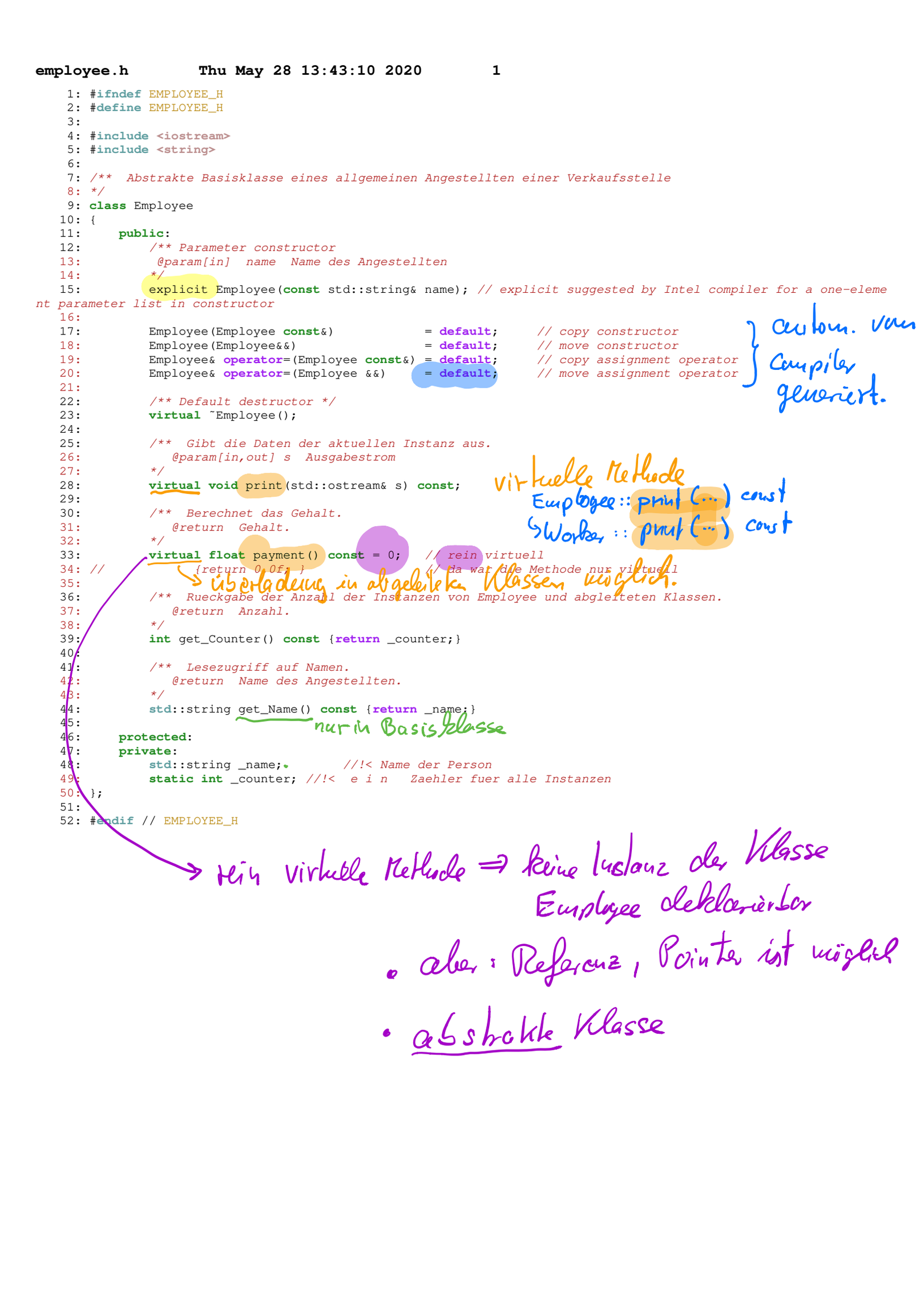
↑ IS-A

B HAS-A additional property

Beispiel Unternehmen:



```
 1: #ifndef EMPLOYEE_H
 2: #define EMPLOYEE_H
 3:
 4: #include <iostream>
 5: #include <string>
 6:
 7: /**  Abstrakte Basisklasse eines allgemeinen Angestellten einer Verkaufsstelle
 8: */
 9: class Employee
10: {
11:     public:
12:         /** Parameter constructor
13:          @param[in]  name  Name des Angestellten
14:         */
15:         explicit Employee(const std::string& name); // explicit suggested by Intel compiler for a one-eleme
nt parameter list in constructor
16:
17:         Employee(Employee const&)             = default;     // copy constructor
18:         Employee(Employee&&)                  = default;     // move constructor
19:         Employee& operator=(Employee const&)  = default;     // copy assignment operator
20:         Employee& operator=(Employee &&)      = default;     // move assignment operator
21:
22:         /** Default destructor */
23:         virtual ~Employee();
24:
25:         /**  Gibt die Daten der aktuellen Instanz aus.
26:            @param[in,out] s  Ausgabestrom
27:         */
28:         virtual void print(std::ostream& s) const;
29:
30:         /**  Berechnet das Gehalt.
31:            @return  Gehalt.
32:         */
33:         virtual float payment() const = 0;   // rein virtuell
34: //          {return 0.0f; }                  // da war die Methode nur virtuell
35:
36:         /**  Rueckgabe der Anzahl der Instanzen von Employee und abgeleiteten Klassen.
37:            @return  Anzahl.
38:         */
39:         int get_Counter() const {return _counter;}
40:
41:         /**  Lesezugriff auf Namen.
42:            @return  Name des Angestellten.
43:         */
44:         std::string get_Name() const {return _name;}
45:
46:     protected:
47:     private:
48:         std::string _name;        //!< Name der Person
49:         static int _counter; //!<  e i n   Zaehler fuer alle Instanzen
50: };
51:
52: #endif // EMPLOYEE_H
```



```
 1: #include "employee.h"
 2: #include <iostream>
 3: #include <string>
 4: using namespace std;
 5:
 6: int Employee::_counter = 0; //!<  Der  e i n e   Zaehler fuer alle Instanzen wird definiert.
 7:
 8: Employee::Employee(const std::string& name)
 9: : _name(name)
10: {
11:     //ctor
12:     ++_counter;
13: }
14:
15: Employee::~Employee()
16: {
17:     //dtor
18:     --_counter;
19: }
20:
21:
22: void Employee::print(std::ostream& s) const
23: {
24:     s << "Name: " << _name << endl;
25:     s << "Bezahlung: " << payment() << endl;
26: //                        Bei virtuellen Methoden wird hier zur Laufzeit
27: //                        via der VMT die konkrete Methode  Klasse::payment()
28: //                        aufgerufen.
29: }
30:
```

```
 1: #ifndef MANAGER_H
 2: #define MANAGER_H
 3:
 4: #include "employee.h"
 5: #include <iostream>
 6: #include <string>
 7:
 8: /**      Manager einer Verkaufsstelle
 9: */
10: class Manager : public Employee
11: {
12:     public:
13:         /** Parameter constructor
14:          @param[in]  name  Name des Angestellten
15:          @param[in]  wageWeek Wochengehalt
16:         */
17:         Manager(const std::string& name, float wageWeek);
18:
19:         Manager(Manager const&)             = default;      // copy constructor
20:         Manager(Manager&&)                  = default;      // move constructor
21:         Manager& operator=(Manager const&) = default;      // copy assignment operator
22:         Manager& operator=(Manager &&)      = default;      // move assignment operator
23:
24:         /** Default destructor */
25:         virtual ~Manager() override;
26:
27:         /**  Gibt die Daten der aktuellen Instanz aus.
28:            @param[in,out] s  Ausgabestrom
29:         */
30:         void print(std::ostream& s) const override;
31:
32:         /**  Berechnet das Gehalt.
33:            @return  Gehalt.
34:         */
35:         float payment() const override {return _wageWeek;}
36:     protected:
37:     private:
38:         float _wageWeek; //!< Wochengehalt
39: };
40:
41: #endif // MANAGER_H
```

```
 1: #include "manager.h"
 2: #include <iostream>
 3: #include <string>
 4: using namespace std;
 5:
 6: Manager::Manager(const string& name, float wageWeek)
 7: : Employee(name), _wageWeek(wageWeek)
 8: {
 9:     //ctor
10: }
11:
12: Manager::~Manager()
13: {
14:     //dtor
15: }
16:
17:
18: void Manager::print(ostream& s) const
19: {
20:     Employee::print(s);
21: //   s << "Wochenlohn: " << _wageWeek << endl;   // dank virtueller Methode payment() nicht mehr noetig
22: }
```

```
 1: #ifndef WORKER_H
 2: #define WORKER_H
 3:
 4: #include "employee.h"
 5: #include <iostream>
 6: #include <string>
 7:
 8: /**      Normaler Arbeiter (Packer) in einer Verkaufsstelle.
 9: */
10: class Worker : public Employee
11: {
12:     public:
13:         /** Parameter constructor
14:          @param[in]  name  Name des Angestellten
15:          @param[in]  hours Arbeitsstunden
16:          @param[in]  wageHours  Stundenlohn
17:         */
18:         Worker(const std::string& name, float hours, float wageHours);
19:
20:         Worker(Worker const&)             = default;      // copy constructor
21:         Worker(Worker&&)                  = default;      // move constructor
22:         Worker& operator=(Worker const&) = default;      // copy assignment operator
23:         Worker& operator=(Worker &&)      = default;      // move assignment operator
24:
25:         /** Default destructor */
26:         ~Worker() override;
27:
28:         /**  Gibt die Daten der aktuellen Instanz aus.
29:            @param[in,out] s  Ausgabestrom
30:         */
31:         void print(std::ostream& s) const override;
32:
33:         /**  Berechnet das Gehalt.
34:            @return  Gehalt.
35:         */
36:         float payment() const override
37:             {return _hours*_wageHours ; }
38:
39:         /**  Abfrage Studenlohn.
40:            @return  Studenlohn in EUR.
41:         */
42:         float wagePerHour() const
43:             {return _wageHours ; }
44:
45:     protected:
46:     private:
47:         float _hours; //!< Arbeitsstunden
48:         float _wageHours; //!< Member Stundenlohn
49: };
50:
51: #endif // WORKER_H
```

+ HAS-A

+ HAS-A

```
 1: #include "worker.h"
 2: #include <iostream>
 3: #include <string>
 4: using namespace std;
 5:
 6: Worker::Worker(const string& name, float hours, float wageHours)
 7: : Employee(name), _hours(hours),_wageHours(wageHours)
 8: {
 9:     //ctor
10: }
11:
12: Worker::~Worker()
13: {
14:     //dtor
15: }
16:
17: void Worker::print(ostream& s) const
18: {
19:     Employee::print(s);
20:  //   s << "Lohn: " << payment() << endl;     // dank virtueller Methode payment() nicht mehr noetig
21: }
```

```
 1: #ifndef SALESPERSON_H
 2: #define SALESPERSON_H
 3:
 4: #include "worker.h"
 5: #include <iostream>
 6: #include <string>
 7:
 8: /**      Verkaeufer in einer Verkaufsstelle
 9: */
10: class SalesPerson : public Worker
11: {
12:     public:
13:         /** Parameter constructor
14:          @param[in]  name  Name des Angestellten
15:          @param[in]  hours Arbeitsstunden
16:          @param[in]  wageHour  Stundenlohn
17:          @param[in]  commission Umsatz
18:          @param[in]  percent  Umsatzbeteiligung in Prozent
19:         */
20:         SalesPerson(const std::string& name, int hours, float wageHour,
21:                     float commission, float percent);
22:
23:         SalesPerson(SalesPerson const&)             = default;      // copy constructor
24:         SalesPerson(SalesPerson&&)                  = default;      // move constructor
25:         SalesPerson& operator=(SalesPerson const&) = default;      // copy assignment operator
26:         SalesPerson& operator=(SalesPerson &&)      = default;      // move assignment operator
27:
28:         /** Default destructor */
29:         virtual ~SalesPerson() override;
30:
31:         /**  Gibt die Daten der aktuellen Instanz aus.
32:            @param[in,out] s  Ausgabestrom
33:         */
34:         void print(std::ostream& s) const override;
35:
36:         /**  Berechnet das Gehalt.
37:            @return  Gehalt.
38:         */
39:         float payment() const override
40:          {return Worker::payment() + _commission*_percent;}
41:                  Basisklasse    +  spezifisches dieser Klasse
42:     protected:
43:     private:
44:         float _commission; //!< Umsatz
45:         float _percent; //!< Umsatzbeteiligung in Prozent
46: };
47:
48: #endif // SALESPERSON_H
```

```cpp
#include "salesperson.h"
#include <iostream>
#include <string>
using namespace std;

SalesPerson:: SalesPerson(const string& name, int hours, float wageHour,
                          float commission, float percent)
: Worker(name,hours,wageHour), _commission(commission),
  _percent(percent)
{
    //ctor
}

SalesPerson::~SalesPerson()
{
    //dtor
}

void SalesPerson::print(ostream& s) const
{
    Worker::print(s);
 //   s << "Sales-Gehalt:" << payment() << endl;   // dank virtueller Methode payment() nicht mehr noetig
    s << "Prozent: " << _percent << endl;
}
```

```cpp
//     Vorlesung: 29.05.2020
//     Klassenhierarchie:
//                     Employee
//             Manager         Worker
//                             SalesPerson

#include "employee.h"
#include "manager.h"
#include "salesperson.h"
#include "worker.h"

#include <iostream>
using namespace std;


ostream& operator<<(ostream &s, const Employee& org);      // Deklaration

ostream& operator<<(ostream &s, const Employee& org)       // Definition
{
    org.print(s);
    return s;
}

int main()
{
    cout << "Hello world!" << endl;

    //Employee aa("Angestellter");        // rein virtuelle Methode payment()
                                          // ==> abstrakte Basisklasse Employee
                                          // ==> keine Instanz von Employee mehr moeglich
    Worker bb("Hugo", 160, 15.50);
    SalesPerson cc("Wanda", 80, 10.2f, 10000, 0.05f);
    Manager dd("Brenda", 2000);
    //cout << aa;                         // nicht mehr moeglich, da Employee abstrakte Basisklasse ist
    cout << bb;
    cout << "##########  " << dd.get_Counter() <<" Objects ##########\n";
    cout << cc;
    cout << "----------------\n" << dd << endl;
    cout << "SalesPerson: " << cc.get_Name()  << "  has hourly wages of " << cc.wagePerHour() << " EUR\n";


    return 0;
}
```