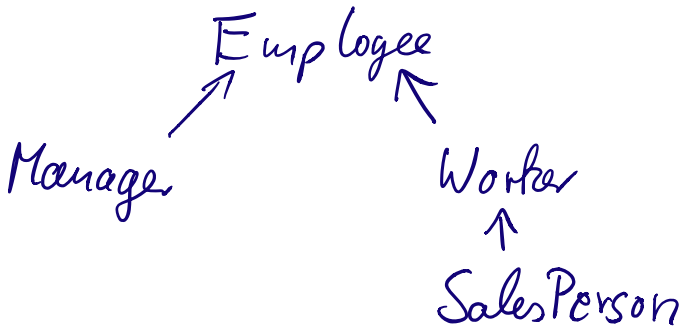
Klassenhierarchie & STL



- Container mit einer Klasse

```
{
  vector<Manager> vm{ Manager("Willi", 100), ... };
  sort(vm.begin(), vm.end());
}
```

oder

```
operator<(Manager const& a, Manager const& b)
Manager::operator<(Manager const& b) const
```

- Kann ich einen Container mit Instanzen verschieden Klassen meiner Hierarchie anlegen?
  - nicht mit STL-containern → boost-Library
  - kein Polymorphismus zwischen Containerelementen.

Ausweg:

- Container von Basisklassenpointern

a) raw pointer

```
{ vector<Employee*>
    vm{ new Manager("Willi", 100),
        new Worker("Hugo", 160, 15.5f, ....};
  ....
  for (auto &it : vm) delete it;
}
```

dynamische Speicherallokierung

Freigabe des allokierten Speichers

- Sortieren des Containers:

```
{ ...
  sort(vm.begin(), vm.end());
}
```

→ Elemente des Containers sind Pointer also
operator<(Employee *a, Employee *b)
vergleicht nur Speicheradressen

⟹ Eigene binäre (unäre) Funktionen für STL-Algorithmen stets nötig

⟹
```
sort(vm.begin(), vm.end(),
     [](Employee *a, Employee *b)
       { return *a < *b; }
);
```

operator<(Employee const& a,
Employee const& b)

alternativ ist natürlich auch möglich:

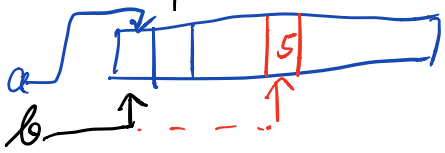
```
[] (Employee *a, Employee *b)
{ return a->payment() < b->payment(); }
```

(*a).payment()

Dereferenzieren* des Pointer und Zugriff° auf dessen Eigenschaften

- raw pointer sollten vermieden werden,
  da häufig Speicher nicht freigegeben wird,
  bereits freigegebener Speicher nochmals deallokiert wird,
  auf freigegebenen Speicher immer noch zugegriffen werden kann,
  shallow-copies eine enorme Fehlerquelle sind

```
int *a = new int [100]
int *b = a;
delete [] a;
b[5] = -1;
```



⟹ shared_ptr oder unique_ptr

b) shared_ptr

```
#include <memory>
{
vector<shared_ptr<Employee>>
  vm{make_shared<Manager>("Willi", 100),
     make_shared<Worker>("Hugo", 40, 15.0)
     ....
  };

sort(vm.begin(), sort.end(),
   [](shared_ptr<Employee> const a,
                          const b)
   { return *a < *b; });
```

Speicher allokiert

} ← Speicherfreigabe im Destructor von shared_ptr

- shared Pointer: raw pointer mit Counter
  - jeder Aufruf Copy-Konstruktor und jede Zuweisung erhöht den Counter um 1
  - jeder Aufruf des Destruktor verringert den Counter um 1
  - Speicherfreigabe erst wenn 0 == Counter