Hochschule Furtwangen
Fakultät Informatik
Prof. Dr. Lothar Piepmeyer

Praktikum ‚Objektorientierte Programmierung'

Aufgabenblatt 11

Aufgabe 1:
a. Stellen Sie sicher, dass der Code, den Sie im vorhergehenden Aufgabenblatt entwickelt haben, den folgenden Test besteht:

```
void test(){
    auto employee= new  Employee(4, "Duck", "Tick");
    assert(employee->salary()==0);
    auto worker= new  Worker(4, "Duck", "Tick", 9, 40);
    employee=worker;
    assert(employee->salary()==0);
    auto seller= new Seller(5, "Duck", "Trick", 1000, 2000);
    employee=seller;
    assert(employee->salary()==0);
}
```

Aufgabe 2:
Überführen Sie die Klasse `Employee` in eine rein virtuelle Klasse und sorgen Sie dafür, dass die Methode `salary` polymorph genutzt wird. Arbeiten Sie - wie in der Vorlesung erläutert - in den Unterklassen mit dem Schlüsselwort `override`. Passen Sie den Test aus Aufgabe 1 so an, dass die polymorphe Nutzung geprüft wird.

Aufgabe 3:
In Ihren Tests nutzen Sie Zeiger. Wir hatten uns aber vorgenommen, nicht mehr mit Zeigern, sondern mit Smart-Pointern zu arbeiten. Prüfen Sie, ob Shared-Pointer vom Typ `std::unique_ptr` polymorphe Methoden auch polymorph nutzen. Modifizieren Sie dazu Ihren Test aus Aufgabe 2 indem Sie die Zeiger geeignet durch Smart-Pointer ersetzen.

Hinweis: Zu C++ haben Bjarne Stroustrup und Herb Sutter die 'C++ Core Guidelines' verfasst, einen Satz von Regeln, der Ihnen - wenn es Sie interessiert - weitere Einsichten in C++ verschafft. Für diese Aufgabe ist die Regel C.35 wichtig, auf die ich in meiner Vorlesung nicht aufmerksam gemacht habe:
C.35: A base class destructor should be either public and virtual, or protected and nonvirtual
Prüfen Sie auch was passiert, wenn Sie gegen diese Regel verstoßen.