Hochschule RheinMain
Fachbereich Design Informatik Medien
Marcus Thoss, M.Sc.

Hardwarenahe Programmierung II
SS 2020
LV 2512

Übungsblatt 7

Aufgabe 7.1 (`friend`, Referenzen und `const`):

Erweitern Sie Ihr Klassensystem aus **Blatt 5** wie folgt:

a) Die Klasse `Vet` soll nun eine `const`-Referenz auf ein `DomesticAnimal` enthalten und soll in `inspect()` nach der Untersuchung der Elemente des Tiers zusätzlich noch einmal `print()` an dem Tier aufrufen.

   Welches Problem ergibt sich nun (das sich auch mit `friend` nicht lösen lässt)? Finden Sie eine Lösung.

b) Ergänzen Sie `Being` um ein Data Member `int printCount`, das mit jedem `print()`-Aufruf um 1 erhöht wird. Wieder wird der Compiler ein Problem bei der Verwendung in `Vet::inspect()` melden.

   Lösen Sie es, indem Sie die Deklaration von `printCount` so modifizieren, dass es auch an `const`-referenzierten Objekten verändert werden darf, da es ja in seiner Bedeutung kein echtes, funktionales Attribut eines `Being`s darstellt.

Aufgabe 7.2 (Test 2):

Die Ergebnisse zu den folgenden Testaufgaben müssen für die Berücksichtigung der Leistung bis zum Tagesende des 19.06.2020 als PDF-Dokument (mit darin eingebettetem Quellcode) mit dem Dateinamen `<Nachname>_<Vorname>_hwp2s20_t02.pdf` per Email an `marcus.thoss@hs-rm.de` gesendet werden.

Sofern nicht anders angegeben, wird für eine vollständig gelöste Teilaufgabe ein Punkt vergeben. Jeglicher C++-Code muss den Styleguide (`StyleGuide.pdf`) einhalten, der auf dem Laborserver zur Verfügung gestellt ist.

a) Warum wird das Argument eines Copy Constructors als `const &` deklariert? Berücksichtigen Sie in Ihrer Antwort folgende Aspekte:

- Warum als Referenz und nicht als Objekt?
- Warum `const &` und nicht nur `&`?

b) Korrigieren Sie den folgenden Code sinnvoll (ohne Verwendung von `struct`) unter Berücksichtigung der Prinzips der Datenkapselung.

```
class Car {
  Car() : mileage(0) {}
  int getMileage() {return mileage;}
  int mileage;
};

int main(void) {
  Car car1;
  int m = car1.getMileage();
  return m;
}
```

c) Korrigieren Sie den Code aus Aufgabe b) durch Verwendung von `struct`. Achten Sie trotzdem auf sinnvolle Datenkapselung.

Betrachten Sie nun die folgende Klasse:

```
class Bird {
public:
  Bird(float initWeight) : weight(initWeight) {} // ctor A
  Bird(float initWeight, int dummyInt=0) : weight(initWeight) {} // ctor B
private:
  float weight;
};
```

d) Welcher Constructor wird jeweils aufgerufen (A oder B oder gibt es einen Fehler)?

```
...
Bird b1;          //(1)
Bird b2(1.5);     //(2)
Bird b3(1.5, 2);  //(3)
```

e) Geben Sie für alle fehlerhaften Varianten der vorigen Frage eine Korrektur der Klasse an, die den Aufruf ermöglicht.

Betrachten Sie nun die folgende Klasse:

```
class Runway {
public:
  Runway(int dir) : direction(dir) {}
private:
  int direction;
};
```

f) Ergänzen Sie die Klasse `Runway` um ein Data Member `airTemperature` des Typs `float`, das für alle `Runway`-Objekte gelten soll (die Anwendung verwaltet nur einen einzigen Flughafen). Geben sie für `airTemperature` die vollständige Deklaration und Definition an!

Hinweis: Testen Sie ihren Code zur Sicherheit mit einer `main()`-Funktion!

g) Ergänzen Sie die Klasse um eine Methode `getAirTemperature()`, die den Wert von `airTemperature` zurückliefert.

h) Geben Sie eine `main()`-Funktion mit jeweils einem Aufruf von `getAirTemperature()`

1) an einem `Runway`-Objekt
2) ohne ein `Runway`-Objekt

an.

i) Ergänzen Sie die `Runway`-Klasse zu der vollständigen *Canonical Class Form*. Geben Sie auch Implementierungen für die hinzugefügten Methoden an (**2 P**).