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

Hardwarenahe Programmierung II
SS 2020
LV 2512

Übungsblatt T3

Aufgabe T3.1 (Test 3):

Die Ergebnisse zu den folgenden Testaufgaben müssen für die Berücksichtigung der Leistung bis zum Tagesende des 04.07.2020 als PDF-Dokument (mit darin eingebettetem Quellcode) mit dem Dateinamen `<Nachname>_<Vorname>_hwp2s20_t03.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.

Sollte von Ihnen zur Beantwortung angegebener Quellcode in aufeinanderfolgenden Aufgabenteilen weiter verändert werden (z.B. c)⇒d)), geben Sie eine separate Quellcode-Version pro Aufgabenteil an.

a) Betrachten Sie den folgenden Code und geben sie für 1) bis 5) jeweils an, ob es sich um einen polymorphen (p) oder nicht-polymorphen Aufruf (n) handelt. Falls (n), müssen Sie den Grund angeben, warum der Aufruf nicht polymorph sein kann.(**2 P**)

   Hinweis: Beim Compilieren des Quellcodes tritt sogar ein Fehler auf; dies sollte bereits ein Indiz für eine der Antworten sein.

```
class Animal {
public:
  virtual void move() {}
  void sleep() {}
};

class Bird : private Animal {
public:
  virtual void move() {}
  void sleep() {}
};
```

```
class Bee : public Animal {
public:
  virtual void move() {}
  void sleep() {}
};

class Cat : public Animal {
public:
  void move() {}
  void sleep() {}
};

int main(void) {
  Bird bird; Bee bee; Cat cat;

  Animal *pAnimalCat = &cat;
  Animal *pAnimalBird = &bird;
  Animal &rAnimalBee = bee;

  bee.move();          // 1)
  pAnimalCat->move();  // 2)
  pAnimalCat->sleep(); // 3)
  rAnimalBee.move()(); // 4)
  pAnimalBird->move(); // 5)

  return 0;
}
```

Betrachten Sie für die Aufgabenteile b), c) und d) die folgenden Klassen:

```
class Food {
protected:
  int weight;
};

class Vegetable {
};

class Soup {
};
```

b) Geben Sie eine Klasse `VegetableSoup` an, die sowohl von `Vegetable` als auch von `Soup` `public` erbt.

c) Fügen Sie `VegetableSoup` eine Methode `bool checkForUniqueWeight()` hinzu, die die *Adresse* des über `Vegetable` geerbten Data Members `weight` mit der Adresse des über `Soup` geerbten Data Members `weight` vergleicht und bei Gleichheit `true` zurückgibt. Was wird zurückgegeben, wenn Sie die Methode testweise für ein Objekt `VegetableSoup vs` aufrufen?

d) Ändern Sie die Klassen (dabei aber nicht `checkForUniqueWeight()`) so, dass sich beim Testen von `checkForUniqueWeight()` die umgekehrte Antwort ergibt.

e) Ändern Sie die Klasse `Animal` aus Aufgabe a) so, dass sie zu einer abstrakten Klasse wird.

Beantworten Sie dann: Warum lässt sich der Rest des Quelltextes aus a) trotzdem noch unverändert compilieren?

Hinweis: Geben Sie als Gegenbeispiel an, wie die Klasse `Bee` geändert werden müsste, damit die Fehlermeldung `"cannot declare variable 'bee' to be of abstract type 'Bee'"` vom Compiler ausgegeben wird.

(**2 P**)

f) Machen Sie den folgenden Quellcode duch Einfügen von `friend`-Deklarationen (ansonsten keine Änderungen) compilierbar.

```
class Car {
   int mileage;
};

class CarManufacturer {
  void resetMileage(Car &c) {
    c.mileage = 0;
  }
};

void resetCarMileage(Car *c) {c->mileage = 0;}
```

g) Betrachten Sie die Klasse

```
class Car {
};
```

und verändern Sie sie so, dass das Definieren eines `Car`-Objekts als Variable, z.B. durch `Car c1;` nicht mehr erlaubt ist.

Schreiben Sie dann eine (globale) *Generatorfunktion* `createCar()` (also *keine* Member Function von `Car`), die mit `new` ein `Car`-Objekt erzeugt und einen Pointer darauf zurückgibt.

Verändern Sie falls nötig die Klasse `Car` so, dass `createCar()` compilierbar wird, das Erzeugen von `Car`-Objekten von anderer Stelle aus aber nach wie vor nicht möglich ist.

(**2 P**)