Technische Universität Berlin **Neuronale Informationsverarbeitung**

C/C++ Übungsblatt 8 (Block 2)

Prof. Dr. Klaus Obermayer und Mitarbeiter

Vererbung & Polymorphie

Verfügbar ab:	**12.12.2022**
Abgabe bis:	**02.01.2023**

Aufgabe 1: Verständnis

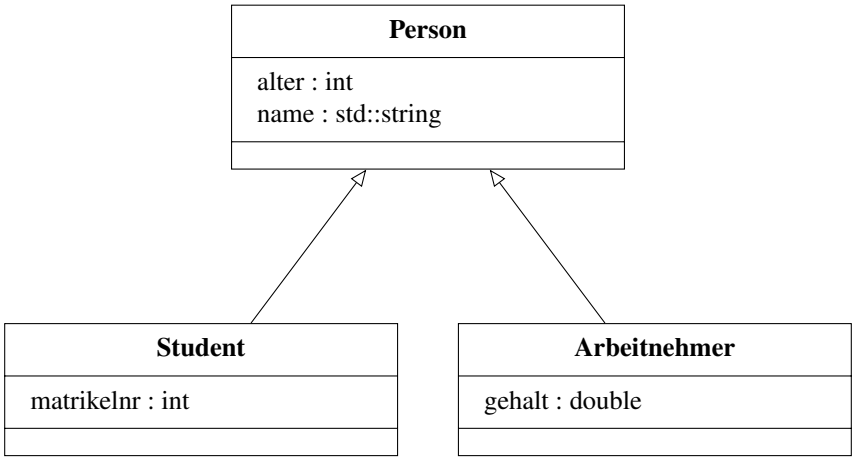
2 Punkte

Betrachten Sie die folgenden Klassen und Attribute:

Klasse
`Vehikel`
`Flugzeug`
`Automobil`
`Boot`
`Containerschiff`

Attribute
`double gewicht`
`double fluegelspannweite`
`boolean istGelaendetauglich`
`int anzahlContainer`
`double tiefgang`
`std::string name`

Stellen Sie grafisch, d.h. in einer Baumstruktur dar, in welchem Vererbungsverhältnis die gegebenen Klassen zueinander stehen. Ordnen Sie dann die gegebenen Attribute den entsprechenden Klassen sinnvoll zu.
Hinweis: Jedes Attribut darf nur einer Klasse zugeordnet werden. Für das Person-Beispiel aus dem Tutorium würde der Vererbungs-Baum ohne Berücksichtigung der Methoden und Sichtbarkeiten so aussehen:



Achtung: Der Datentyp steht hinter dem Bezeichner. Wird dies nicht eingehalten, werden Punkte abgezogen!

Aufgabe 2: Haustiere **2 Punkte**

Gegeben ist die Datei `haustierverwaltung.cpp`, die die Klasse `Haustier` mit drei öffentlichen Attributen enthält:

Listing 1: haustierverwaltung.cpp

```cpp
/*
 * Hauptprogramm, welches unsere Haustier-Klasse enthaelt
 */

#include <iostream>
#include <string>

class Haustier
{
public:
    std::string name;
    int alter;
    bool istSaeugetier;
};

int main()
{
    // TODO
}
```

Implementieren Sie in der `main`-Funktion der `haustierverwaltung.cpp` folgende Anweisungen:

- Deklarieren Sie die Variablen `weissbauchigel` und `chinchilla`, beide vom Typ `Haustier`
- Weisen Sie den Attributen der `Haustier`-Objekte `weissbauchigel` und `chinchilla` Werte gemäß unten angegebener Tabelle zu.
- Die Chinchilla wird ein Jahr älter. Erhöhen Sie das Attribut `alter` von `chinchilla` um eins.
- Geben Sie die Werte der Attribute `alter` und `name` von `chinchilla` und `weissbauchigel` auf der Konsole aus.

Haustier	Alter	Name	Säugetier
Weissbauchigel	9	Isolde	Ja
Chinchilla	4	Chili	Ja

Die Ausgabe Ihres Programmes könnte z.B. so aussehen:

```
Der Weissbauchigel ist 9 Jahre alt und heisst Isolde.
Die Chinchilla ist 4 Jahre alt und heisst Chili.
```

Beantworten Sie außerdem die folgende Frage:

- Wie lauten die Werte der Attribute `alter` und `istSaeugetier` der Objekte `weissbauchigel` bzw. `chinchilla` direkt nach der Erzeugung (d.h. *nach* Deklaration der `Haustier`-Variablen, jedoch *vor* Zuweisung der Attributswerte gemäß obiger Tabelle)?

Aufgabe 3: Monster **6 Punkte**

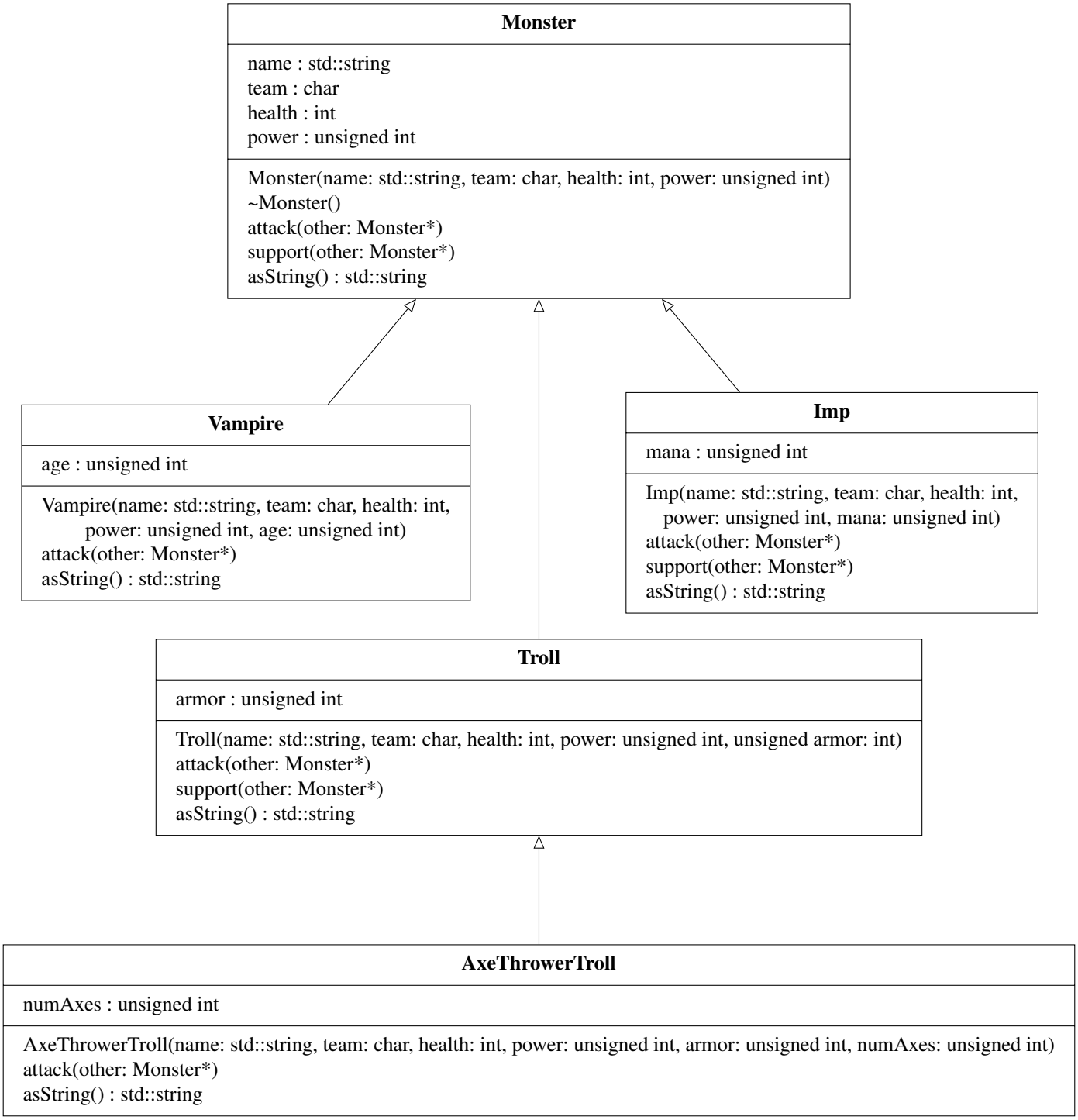
In dieser Aufgabe wird ein kleines Monsterspiel unter Nutzung von Vererbung erstellt.

Im Spiel existieren Monster, die in Teams eingeteilt werden und dann gegen Monster der anderen Teams kämpfen (Methode attack()), wobei der angegriffene Gesundheit (health) verliert. Alternativ können auch Verbündete Monster unterstützt werden (Methode support()). Das Basismonster verursacht Schaden in Höhe seiner Stärke (power) und stellt sich beim unterstützen als nutzlos heraus. Andere, speziellere Monster haben teilweise verbesserte Fähigkeiten. Die Monster sind als Klassen zu implementieren. Die *battleground.cpp* erstellt Monster und lässt diese miteinander kämpfen, bis ein Gewinner feststeht.

Überlegen Sie sich zunächst Antworten auf die folgenden Fragestellungen (keine Abgabe der Antworten):

1. Was bedeutet Polymorphie?
2. Was bedeutet Überschreiben von Methoden (nicht Überladen)?
3. Was ist der Zweck des Schlüsselwortes `virtual`?

Betrachten Sie nun folgendes Klassendiagramm:



Im Folgenden sollen diese Klassen zur Verfügung gestellt werden.

1. Erstellen Sie die Klasse `Monster`
   a) Definieren Sie die Klasse in der Datei `monster.hpp` gemäß des Klassendiagramms.
      - Wählen Sie zur Vereinfachung stets das Sichtbarkeitslevel **`public`**.
      - Deklarieren Sie den Destruktor und die beiden Methoden als **`virtual`**.
   b) Implementieren Sie Konstruktor, Destruktor und die Methoden in der Datei `monster.cpp`.
      - Der Konstruktor weist über die Initialisierungsliste allen Klassenattributen die übergebenen Parameter zu.

- Der Konstruktor prüft im Konstruktorrumpf die Eingabe des Teams. Das Nullliteral (`\0`) soll abgefangen und stattdessen die Ziffer 0 als Teamname verwendet werden. Der Nutzer soll eine Warnung auf der Konsole erhalten, die erklärt, was passiert ist.
- Der Konstruktor gibt unter Verwendung des Namens eine Meldung auf der Konsole aus, dass ein Monster erschaffen wurde.
- Der Destruktor gibt unter Verwendung des Namens eine Meldung auf der Konsole aus, dass ein Monster verschwunden ist.
- Die `std::string Monster::asString()`-Methode erstellt einen Beschreibungsstring, der alle Attribute lesbar zusammenfasst. Nutzen Sie einen `std::stringstream` aus der Bibliothek `sstream` zum Zusammenfügen des Beschreibungsstrings (vgl. Tutoriumsaufgaben).
- Die **`void`** `Monster::attack(Monster* other)`-Methode wird wie folgt aufgerufen. `hugo->attack(herbert);`. Die Ausführung soll dem anderen Monster (hier herbert) `health` in Höhe der `power` des Angreifers (hier hugo) abziehen. Es soll ein Satz auf der Konsole ausgegeben werden, der Namen der Beteiligten und den verursachten Schaden beschreibt.
- Die **`void`** `Monster::support(Monster* other)`-Methode funktioniert wie der Angriff, nur auf ein Verbündetes Monster. Da das Basismonster etwas unbeholfen ist, soll es sein Ziel nur mit offenem Mund anstarren. Sorgen Sie erneut für eine entsprechende Konsolenausgabe. (Manche der spezielleren Monster verhalten sich geschickter... dazu später mehr.)

c) Kompilieren Sie nun ihre Monster-Implementierung, bis sie ohne Warnungen und Fehler kompiliert: *g++ -c -Wall -std=c++11 monster.cpp*

2. Laden Sie sich nun die fertige Klasse `Vampire` von ISIS herunter und speichern Sie sie im selben Verzeichnis.
   a) Kompilieren Sie diese ebenfalls, zu Prüfen, dass Sie kompatibel zu ihrer Monster-Implementierung ist: *g++ -c -Wall -std=c++11 vampire.cpp*
   b) Werden Fehler oder Warnungen angezeigt, müss die Monster-Implementierung korrigiert werden.
3. Erstellen Sie die Klasse `Troll`.
   a) Definieren Sie die Klasse in der Datei `troll.hpp` gemäß des Klassendiagramms.
      - Wählen Sie zur Vereinfachung stets das Sichtbarkeitslevel **`public`**.
      - Deklarieren Sie die beiden Methoden als **`virtual`**.

   b) Implementieren Sie Konstruktor und die Methoden in der Datei `troll.cpp`.
      - Der Konstruktor weist über die Initialisierungsliste allen Klassenattributen die übergebenen Parameter zu. Verwenden Sie den Elternkonstruktor der Klasse `Monster`.
      - Die `std::string Troll::asString()`-Methode soll die gleichnamige Methode der Elternklasse überschreiben, damit auch das Attribut `armor` ausgegeben wird. Wenn Sie möchten, können Sie auf der Elternmethode aufbauen.
      - Die **`void`** `Troll::attack(Monster* other)`-Methode soll die gleichnamige Methode der Elternklasse überschreiben. Der Troll fügt dem Angreifer Schaden in Höhe seiner **doppelten** `power` abzüglich seiner `armor` zu (da letztere ihn behindert). Es soll jedoch mindestens ein (1) Schaden zugefügt werden, auch wenn die Rüstung viel zu schwer ist. Sorgen Sie für eine Konsolenausgabe der Namen und des Schadens.
      - Die **`void`** `Troll::support(Monster* other)`-Methode soll die gleichnamige Methode der Elternklasse überschreiben. Der Troll verstärkt durch einen motivierenden Kampfschrei die `power` des anderen Monsters um eins (1). Sorgen Sie für eine angemessene Konsolenausgabe.

   c) Kompilieren Sie nun ihre Troll-Implementierung, bis sie ohne Warnungen und Fehler kompiliert: *g++ -c -Wall -std=c++11 troll.cpp*
4. Erstellen Sie die Klasse `AxeThrowerTroll`.

a) Definieren Sie die Klasse in der Datei `axeThrowerTroll.hpp` gemäß des Klassendiagramms. Achten Sie darauf, von `Troll` statt von `Monster` zu erben!
   - Wählen Sie zur Vereinfachung stets das Sichtbarkeitslevel **`public`**.

b) Implementieren Sie Konstruktor und die Methoden in der Datei `axeThrowerTroll.cpp`.
   - Der Konstruktor weist über die Initialisierungsliste allen Klassenattributen die übergebenen Parameter zu. Verwenden Sie den Elternkonstruktor der Klasse `Troll`.
   - Die `std::string AxeThrowerTroll::asString()`-Methode soll die gleichnamige Methode der Elternklasse überschreiben, damit auch das Attribut `numAxes` ausgegeben wird. Wenn Sie möchten, können Sie auf der Elternmethode aufbauen.
   - Die **`void`** `AxeThrowerTroll::attack(Monster* other)`-Methode soll die gleichnamige Methode der Elternklasse überschreiben. Der Troll schaut nach, ob er noch über Äxte verfügt (numAxes > 0). Wenn ja, wirft er eine Axt (diese geht verloren) und fügt dabei Schaden in Höhe des **dreifachen** seiner `power` zu. Hat er keine Äxte mehr, verhält er sich wie ein normaler Troll (rufen Sie dann die Elternmethode auf).

c) Kompilieren Sie nun ihre AxeThrowerTroll-Implementierung, bis Sie ohne Warnungen und Fehler kompiliert: *g++ -c -Wall -std=c++11 axeThrowerTroll.cpp*

5. Erstellen Sie die Klasse `Imp`.

   a) Definieren Sie die Klasse in der Datei `imp.hpp` gemäß des Klassendiagramms.
      - Wählen Sie zur Vereinfachung stets das Sichtbarkeitslevel **`public`**.

   b) Implementieren Sie Konstruktor und die Methoden in der Datei `imp.cpp`.
      - Der Konstruktor weist über die Initialisierungsliste allen Klassenattributen die übergebenen Parameter zu. Verwenden Sie den Elternkonstruktor der Klasse `Monster`.
      - Die `std::string Imp::asString()`-Methode soll die gleichnamige Methode der Elternklasse überschreiben, damit auch das Attribut `mana` ausgegeben wird. Wenn Sie möchten, können Sie auf der Elternmethode aufbauen.
      - Die **`void`** `Imp::attack(Monster* other)`-Methode soll die gleichnamige Methode der Elternklasse überschreiben. Falls der Imp mindestens zwei (2) mana hat, wirft er einen Feuerball auf den Gegner, der Schaden in Höhe des `dreifachen` seiner Power verursacht (und dabei zwei (2) Mana verbraucht). Hat er kein Mana, regeneriert einer ein (1) Mana. Sorgen Sie erneut für eine entsprechende Konsolenausgabe.
      - Die **`void`** `Imp::support(Monster* other)`-Methode soll die gleichnamige Methode der Elternklasse überschreiben. Falls der Imp mana hat, heilt er das andere Monster um das **doppelte** seiner `power` und verbraucht dabei ein (1) mana. Falls es sich beim anderen Monster ebenfalls um einen Imp handelt (Stichwort: **`dynamic_cast`**`<>()`), wird bei diesem ein (1) mana generiert. Hat der aktive Imp kein Mana, regeneriert er ein (1) Mana.

   c) Kompilieren Sie nun ihre Imp-Implementierung, bis Sie ohne Warnungen und Fehler kompiliert: *g++ -c -Wall -std=c++11 imp.cpp*

6. Laden Sie sich das fertige Programm `battleground.cpp` von ISIS herunter und speichern Sie es im selben Verzeichnis.

   a) Kompilieren Sie es und linken Sie es gegen ihre Implementierungen: *g++ -Wall -std=c++11 battleground.cpp monster.o vampire.o troll.o axeThrowerTroll.o imp.o -o battleground* Oder inclusive Neukompilierung der Klassen: *g++ -Wall -std=c++11 battleground.cpp monster.cpp vampire.cpp troll.cpp axeThrowerTroll.cpp imp.cpp -o battleground*

   b) Korrigieren Sie eventuell vorhandene syntaktische und semantische Fehler.

Hinweis: Sollten Sie bei irgendeinem Problem nicht weiterkommen, schauen Sie nach einer vergleichbaren Problemstellung in den Lösungen der letzten Tutorienblätter. Alles, was Sie benötigen, wurde dort bereits demonstriert.

Die Ausgabe Ihres Testprogramms könnte z.B. so aussehen:

```
Vorbereitung...
Monstroesitaet erschaffen.
Vampy erschaffen.
Monsterchen erschaffen.
Dracularius erschaffen.
Trollopa erschaffen.
Trolline erschaffen.
Wichtelantius erschaffen.
Berserkerus erschaffen.
Wichtelontias erschaffen.
Kukundi erschaffen.

Moegen die Spiele beginnen...
Monstroesitaet greift Vampy an und verursacht 1 Schaden.
Vampy starrt Dracularius nutzlos mit offenem Mund an.
Monsterchen greift Vampy an und verursacht 3 Schaden.
Vampy ist am Ende seiner Kraefte. Vampy verschwindet.
Dracularius starrt Trollopa nutzlos mit offenem Mund an.
Trollopa greift Monstroesitaet an und verursacht 11 Schaden.
Trolline greift Dracularius an und verursacht 3 Schaden.
Wichtelantius nutzt 1 Mana und heilt Berserkerus um 8 Gesundheit.
Berserkerus wirft eine Axt nach Monstroesitaet und verursacht 3 Schaden.
Monstroesitaet starrt Monsterchen nutzlos mit offenem Mund an.
Monsterchen starrt Monstroesitaet nutzlos mit offenem Mund an.
Dracularius beisst Monstroesitaet, verursacht 2 Schaden, heilt sich um 1 Gesundheit und altert um 1.
Trollopa greift Monstroesitaet an und verursacht 11 Schaden.
Trolline laesst einen Kampfschrei erklingen, der Monstroesitaet anspornt und ihm 1 Power gewaehrt.
Wichtelantius regeneriert 1 Mana.
Berserkerus laesst einen Kampfschrei erklingen, der Wichtelantius anspornt und ihm 1 Power gewaehrt.
Monstroesitaet greift Dracularius an und verursacht 2 Schaden.
Monsterchen starrt Monstroesitaet nutzlos mit offenem Mund an.
Dracularius starrt Trollopa nutzlos mit offenem Mund an.
Trollopa greift Monstroesitaet an und verursacht 11 Schaden.
Monstroesitaet ist am Ende seiner Kraefte. Monstroesitaet verschwindet.
Trolline greift Dracularius an und verursacht 3 Schaden.
Dracularius ist am Ende seiner Kraefte. Dracularius verschwindet.
Wichtelantius regeneriert 1 Mana.
Berserkerus wirft eine Axt nach Monsterchen und verursacht 3 Schaden.
Monsterchen ist am Ende seiner Kraefte. Monsterchen verschwindet.
Trollopa greift Trolline an und verursacht 11 Schaden.
Trolline ist am Ende seiner Kraefte. Trolline verschwindet.
Wichtelantius nutzt 1 Mana, wirft einen Feuerball auf Trollopa und verursacht 15 Schaden.
Trollopa ist am Ende seiner Kraefte. Trollopa verschwindet.
Berserkerus moechte angreifen, findet aber keinen Gegner mehr.

Es gewinnt das Team C.
Sieger:
Monster: name="Wichtelantius",team="C",health="2",power="5",mana="0"
Monster: name="Berserkerus",team="C",health="11",power="1",armor="1",numAxes="1"

Rest aufraeumen...
Wichtelantius verschwindet.
Berserkerus verschwindet.
```

Optional: Wenn Sie möchten - und alle anderen Aufgaben erfüllt haben - können Sie nun gerne weitere Monsterklassen implementieren und/oder ein weiteres Testprogramm `battleground_custom.cpp` erstellen (dies kann auf `battleground.cpp` aufbauen). Die Funktionalität der `battleground.cpp` darf dadurch nicht beeinträchtigt werden. Zusatzpunkte werden keine vergeben.