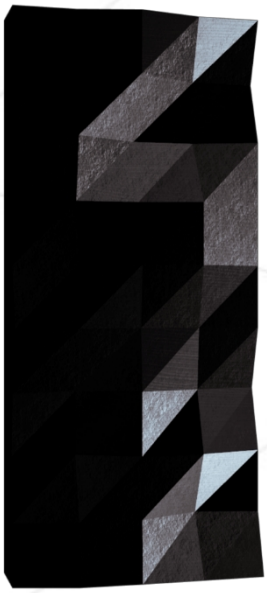

C++ - Modul 04

Subtyp-Polymorphismus, abstrakte Klassen, Schnittstellen

Zusammenfassung:
Dieses Dokument enthält die Übungen des Moduls 04 aus den C++ Modulen.

Fassung: 11

Inhalt

Kapitel I

Einleitung

C++ ist eine allgemeine Programmiersprache, die von Bjarne Stroustrup als Weiterentwicklung der Programmiersprache C oder "C with Classes" (Quelle: Wikipedia) entwickelt wurde.

Das Ziel dieser Module ist es, Sie in die **objektorientierte Programmierung** einzuführen. Dies wird der Ausgangspunkt für Ihre C++-Reise sein. Viele Sprachen werden empfohlen, um OOP zu lernen. Wir haben uns für C++ entschieden, da es von Ihrem alten Freund C abgeleitet ist. Da es sich um eine komplexe Sprache handelt, und um die Dinge einfach zu halten, wird Ihr Code dem Standard C++98 entsprechen.

Wir sind uns bewusst, dass modernes C++ in vielerlei Hinsicht anders ist. Wenn Sie also ein kompetenter C++-Entwickler werden wollen, liegt es an Ihnen, nach den 42 Common Core weiterzugehen!

Kapitel II

Allgemeine Vorschriften

Kompilieren

- Kompilieren Sie Ihren Code mit C++ und den Flags -Wall -Wextra -Werror
- Ihr Code sollte sich trotzdem kompilieren lassen, wenn Sie das Flag -std=c++98 hinzufügen

Formatierungs- und Benennungskonventionen

- Die Übungsverzeichnisse werden folgendermaßen benannt: ex00, ex01, ... , exn
- Benennen Sie Ihre Dateien, Klassen, Funktionen, Mitgliedsfunktionen und Attribute wie in den Richtlinien gefordert.
- Schreiben Sie Klassennamen im Format **UpperCamelCase**. Dateien, die Klassencode enthalten, werden immer nach dem Klassennamen benannt. Zum Beispiel: Klassenname.hpp/Klassenname.h, Klassenname.cpp, oder Klassenname.tpp. Wenn Sie also eine Header-Datei haben, die die Definition einer Klasse "BrickWall" enthält, die für eine Ziegelmauer steht, lautet ihr Name BrickWall.hpp.
- Wenn nicht anders angegeben, müssen alle Ausgabemeldungen mit einem Zeilenumbruch abgeschlossen und auf der Standardausgabe ausgegeben werden.
- *Auf Wiedersehen Norminette!* In den C++-Modulen ist kein Kodierungsstil vorgeschrieben. Sie können Ihrem Lieblingsstil folgen. Aber denken Sie daran, dass ein Code, den Ihre Mitbewerber nicht verstehen, auch nicht benotet werden kann. Geben Sie Ihr Bestes, um einen sauberen und lesbaren Code zu schreiben.

Erlaubt/Verboten

Sie programmieren nicht mehr in C. Zeit für C++! Deshalb:

- Sie dürfen fast alles aus der Standardbibliothek verwenden. Anstatt sich also an das zu halten, was Sie bereits kennen, wäre es klug, so viel wie möglich die C++-ähnlichen Versionen der C-Funktionen zu verwenden, an die Sie gewöhnt sind.
- Sie können jedoch keine andere externe Bibliothek verwenden. Das bedeutet, dass C++11 (und abgeleitete Formen) und Boost-Bibliotheken verboten sind. Die

folgenden Funktionen sind ebenfalls verboten: *printf(), *alloc() und free(). Wenn Sie sie verwenden, wird Ihre Note 0 sein und das war's.

- Beachten Sie, dass, sofern nicht ausdrücklich anders angegeben, der using-Namensraum <ns_name> und
  Freundschaftswörter sind verboten. Andernfalls wird Ihre Note -42 sein.
- **Du darfst die STL nur in den Modulen 08 und 09 verwenden.** Das bedeutet: keine **Container** (Vektor/Liste/Map/usw.) und keine **Algorithmen** (alles, was den <algorithm>-Header erfordert) bis dahin. Andernfalls wird Ihre Note -42 sein.

Einige Designanforderungen

- Speicherlecks treten auch in C++ auf. Wenn Sie Speicher zuweisen (mit der Funktion new
  Schlüsselwort), müssen Sie **Speicherlecks** vermeiden.
- Von Modul 02 bis Modul 09 muss der Unterricht in der **orthodoxen kanonischen Form** gestaltet werden**, es sei denn, es ist ausdrücklich etwas anderes angegeben**.
- Jede Funktionsimplementierung in einer Header-Datei (mit Ausnahme von Funktionsschablonen) bedeutet 0 für die Übung.
- Sie sollten in der Lage sein, jeden Ihrer Header unabhängig von anderen zu verwenden. Daher müssen sie alle Abhängigkeiten einschließen, die sie benötigen. Allerdings müssen Sie das Problem der doppelten Einbindung vermeiden, indem Sie **Include-Guards** hinzufügen. Andernfalls wird Ihre Note 0 sein.

Lies mich

- Sie können bei Bedarf zusätzliche Dateien hinzufügen (z. B. um Ihren Code aufzuteilen). Da diese Aufgaben nicht von einem Programm überprüft werden, können Sie dies gerne tun, solange Sie die vorgeschriebenen Dateien einreichen.
- Manchmal sehen die Richtlinien einer Übung kurz aus, aber die Beispiele können Anforderungen aufzeigen, die nicht ausdrücklich in den Anweisungen stehen.
- Lesen Sie jedes Modul vollständig durch, bevor Sie beginnen! Wirklich, tun Sie es.
- Bei Odin, bei Thor! Benutze deinen Verstand!!!

Sie werden eine Menge Klassen implementieren müssen. Das kann mühsam sein, es sei denn, Sie können in Ihrem Lieblings-Texteditor Skripte schreiben.

Sie haben einen gewissen Spielraum bei der Durchführung der Übungen. Halten Sie sich jedoch an die vorgeschriebenen Regeln und seien Sie nicht faul. Sie würden eine Menge nützlicher Informationen verpassen! Zögern Sie nicht, sich über theoretische Konzepte zu informieren.

Kapitel III

Übung 00: Polymorphismus

	Übung : 00
Polymorphismus	
Turn-in-Verzeichnis : *ex00/*	
Einzureichende Dateien: `Makefile, main.cpp, *.cpp, *.{h, hpp}`	
Verbotene Funktionen : `Keine`	

Für jede Übung müssen Sie **die vollständigsten Tests** vorlegen, die Sie durchführen können.
Konstruktoren und Destruktoren jeder Klasse müssen spezifische Meldungen anzeigen. Verwenden Sie nicht die gleiche Nachricht für alle Klassen.

Beginnen Sie mit der Implementierung einer einfachen Basisklasse namens **Animal**. Sie hat ein geschütztes Attribut:

- `std::string type;`

Implementieren Sie eine Hundeklasse, die von Animal erbt.
Implementieren Sie eine Katzenklasse, die von Animal erbt.

Diese beiden abgeleiteten Klassen müssen ihr Typfeld in Abhängigkeit von ihrem Namen festlegen. Dann wird der Typ des Hundes auf "Hund" und der Typ der Katze auf "Katze" initialisiert. Der Typ der Klasse Animal kann leer bleiben oder auf einen Wert Ihrer Wahl gesetzt werden.

Jedes Tier muss in der Lage sein, die Mitgliederfunktion zu nutzen:
makeSound()
Es wird ein entsprechender Ton ausgegeben (Katzen bellen nicht).

Die Ausführung dieses Codes sollte die spezifischen Geräusche der Klassen Hund und Katze ausgeben, nicht die des Tieres.

```
int main()
{
    const Animal* meta = new Animal();
    const Animal* j = new Dog();
    const Animal* i = new Cat();

    std::cout << j->getType() << " " << std::endl;
    std::cout << i->getType() << " " << std::endl; i-
    >makeSound(); //gibt das Katzengeräusch
    aus!
    j->makeSound();
    meta->makeSound();
    ...

    0 zurückgeben;
}
```

Um sicherzustellen, dass Sie verstanden haben, wie es funktioniert, implementieren Sie eine WrongCat-Klasse, die von einer WrongAnimal-Klasse erbt. Wenn Sie im obigen Code das Tier und die Katze durch die falschen ersetzen, sollte die WrongCat den WrongAnimal-Sound ausgeben.

Führen Sie weitere Tests als die oben genannten durch und reichen Sie sie ein.

Kapitel IV

Übung 01: Ich will die Welt nicht in Brand setzen

	Übung : 01
Ich will die Welt nicht in Brand setzen	
Turn-in-Verzeichnis : *ex01/*	
Einzureichende Dateien: `Dateien aus der vorherigen Übung + *.cpp, *.{h, hpp}`	
Verbotene Funktionen : `Keine`	

Konstruktoren und Destruktoren jeder Klasse müssen bestimmte Meldungen

anzeigen. Implementieren Sie eine Klasse **Brain**. Sie enthält ein Array von 100

std::string namens ideas.
Auf diese Weise haben Hund und Katze ein privates Attribut Gehirn*.
Bei der Erstellung werden Hund und Katze ihr Gehirn mit new Brain() erstellen;
Nach der Zerstörung löschen Hund und Katze ihr Gehirn.

Erstellen und füllen Sie in Ihrer Hauptfunktion ein Array mit Tierobjekten. Die Hälfte davon wird aus **Hunden** und die andere Hälfte **aus Katzen bestehen**. Am Ende der Programmausführung durchlaufen Sie dieses Array in einer Schleife und löschen jedes Tier. Sie müssen Hunde und Katzen direkt als Animals löschen. Die entsprechenden Destruktoren müssen in der erwarteten Reihenfolge aufgerufen werden.

Vergessen Sie nicht, auf **Speicherlecks** zu achten.

Eine Kopie eines Hundes oder einer Katze darf nicht oberflächlich sein. Sie müssen also prüfen, ob Ihre Kopien tiefgründige Kopien sind!

```
int main()
{
    const Animal* j = new Dog();
    const Animal* i = new Cat();


    delete j;//sollte kein Leck verursachen
    i löschen;
    ---

    0 zurückgeben;
}
```

Führen Sie weitere Tests als die oben genannten durch und reichen Sie sie ein.

Kapitel V

Übung 02: Abstrakte Klasse

	Übung : 02
Abstrakte Klasse	
Turn-in-Verzeichnis : *ex02/*	
Einzureichende Dateien: `Dateien der vorherigen Übung + *.cpp, *.{h, hpp}`	
Verbotene Funktionen : `Keine`	

Die Erstellung von Tierobjekten macht also doch keinen Sinn. Es ist wahr, sie machen keine Geräusche! Um mögliche Fehler zu vermeiden, sollte die Standardklasse Animal nicht instanzierbar sein.

Reparieren Sie die Klasse Animal, so dass niemand sie instanziieren kann. Alles sollte wie vorher funktionieren.

Wenn Sie möchten, können Sie den Klassennamen aktualisieren, indem Sie ein A als Präfix zu Animal hinzufügen.

Kapitel VI

Übung 03: Schnittstelle & Rekapitulation

	Übung : 03
Schnittstelle & Rekapitulation	
Turn-in-Verzeichnis : *ex03/*	
Einzureichende Dateien: `Makefile, main.cpp, *.cpp, *.{h, hpp}`	
Verbotene Funktionen : `Keine`	

Schnittstellen gibt es in C++98 nicht (auch nicht in C++20). Reine abstrakte Klassen werden jedoch üblicherweise als Schnittstellen bezeichnet. In dieser letzten Übung wollen wir also versuchen, Schnittstellen zu implementieren, um sicherzustellen, dass Sie dieses Modul verstanden haben.

Vervollständigen Sie die Definition der folgenden AMateria-Klasse und implementieren Sie die erforderlichen Mitgliedsfunktionen.

```
Klasse AMateria
{
    geschützt:
           [...]

    öffentlich:
           AMateria(std::string const &
           type); [...]

           std::string const & getType() const; //Returnt den Materie-Typ

           virtual AMateria* clone() const = 0;
           virtual void use(ICharacter& target);
};
```

Implementieren Sie die konkreten Materias-Klassen **Ice** und **Cure**. Verwenden Sie ihre Namen in Kleinbuchstaben ("ice" für Ice, "cure" für Cure), um ihre Typen festzulegen. Natürlich gibt ihre Mitgliedsfunktion clone() eine neue Instanz desselben Typs zurück (d.h. wenn Sie eine Ice Materia klonen, erhalten Sie eine neue Ice Materia).

Die Mitgliedsfunktion use(ICharacter&) wird angezeigt:

- Eis: "* schießt einen Eisblitz auf <Name> *"
- Heilen: "* heilt <Name>s Wunden *"

<Name> ist der Name des als Parameter übergebenen Zeichens. Die spitzen Klammern (< und >) werden nicht gedruckt.

Beim Zuordnen einer Materia zu einer anderen macht es keinen Sinn, den Typ zu kopieren.

Schreiben Sie die konkrete Klasse **Character**, die die folgende Schnittstelle implementieren wird:

```
class ICharacter
{
   öffentlich:
          virtual ~ICharacter() {}
          virtual std::string const & getName() const = 0;
          virtual void equip(AMateria* m) = 0;
          virtual void unequip(int idx) = 0;
          virtual void use(int idx, ICharacter& target) = 0;
};
```

Der **Charakter** verfügt über ein Inventar von 4 Plätzen, d.h. maximal 4 Materias. Beim Bau ist das Inventar leer. Er rüstet die Materias in den ersten leeren Slot aus, den er findet. Das heißt, in dieser Reihenfolge: von Slot 0 bis Slot 3. Falls sie versuchen, eine Materia zu einem vollen Inventar hinzuzufügen oder eine nicht vorhandene Materia zu benutzen/unequipen, tun sie nichts (aber trotzdem sind Bugs verboten). Die Funktion unequip() darf die Materia NICHT löschen!

Behandeln Sie die Materias, die Ihr Charakter auf dem Boden hinterlassen hat, wie Sie wollen. Speichern Sie die Adressen, bevor Sie unequip() oder etwas anderes aufrufen, aber vergessen Sie nicht, dass Sie Speicherlecks vermeiden müssen.

Die Funktion use(int, ICharacter&) muss die Materia im Slot[idx] verwenden und den Zielparameter an die Funktion AMateria::use übergeben.

Das Inventar deines Charakters kann jede Art von AMateria aufnehmen.

Ihr **Zeichen** muss einen Konstruktor haben, der seinen Namen als Parameter annimmt. Jede Kopie (mit dem Kopierkonstruktor oder dem Kopierzuweisungsoperator) eines Charakters muss **tief** sein. Während des Kopierens müssen die Materias eines Charakters gelöscht werden, bevor die neuen zu seinem Inventar hinzugefügt werden. Natürlich müssen die Materias auch gelöscht werden, wenn ein Charakter zerstört wird.

Schreiben Sie die konkrete Klasse **MateriaSource**, die die folgende Schnittstelle implementieren wird:

```
Klasse IMateriaSource
{
    öffentlich:
            virtual ~IMateriaSource() {}
            virtual void learnMateria(AMateria*) = 0;
            virtual AMateria* createMateria(std::string const & type) = 0;
};
```

- learnMateria(AMateria*)
  Kopiert die als Parameter übergebene Materia und speichert sie im Speicher, damit sie später geklont werden kann. Wie der Charakter kann auch die **MateriaSource** maximal 4 Materias kennen. Sie sind nicht unbedingt eindeutig.
- createMateria(std::string const &)
  Gibt eine neue Materia zurück. Letztere ist eine Kopie der zuvor von der **MateriaSource** gelernten Materia, deren Typ dem als Parameter übergebenen Typ entspricht. Gibt 0 zurück, wenn der Typ unbekannt ist.

Kurz gesagt, Ihre **MateriaSource** muss in der Lage sein, "Vorlagen" von Materias zu lernen, um sie bei Bedarf zu erstellen. Dann können Sie eine neue Materie nur mit einer Zeichenkette erzeugen, die ihren Typ identifiziert.

Ausführen dieses Codes:

```
int main()
{
    IMateriaSource* src = new MateriaSource();
    src->learnMateria(new Ice());
    src->learnMateria(new Cure());

    ICharacter* me = new Character("me");

    AMateria* tmp;
    tmp = src->createMateria("ice");
    me->equip(tmp);
    tmp = src->createMateria("cure");
    me->equip(tmp);

    ICharacter* bob = new Character("bob");

    me->use(0, *bob);
    me->use(1, *bob);

    bob
    löschen;
    mich
    löschen;
    src
    löschen;

    0 zurückgeben;
}
```

Sollte ausgegeben werden:

```
$> clang++ -W -Wall -Werror *.cpp
$> ./a.out | cat -e
* schießt einen Eisblitz auf Bob *$
* heilt Bobs Wunden *$
```

Führen Sie, wie üblich, weitere Tests durch und reichen Sie diese ein.

Sie können dieses Modul bestehen, ohne die Übung 03 zu bearbeiten.

Kapitel VII

Einreichung und Peer-Evaluierung

Reichen Sie Ihre Arbeit wie gewohnt in Ihrem Git-Repository ein. Nur die Arbeit in Ihrem Repository wird während der Verteidigung bewertet. Zögern Sie nicht, die Namen Ihrer Ordner und Dateien zu überprüfen, um sicherzustellen, dass sie korrekt sind.

```
??????????? XXXXXXXXXX = $3$$6b616b91536363971573e58914295d42
```