Objektorientierte Programmierung

Sommersemester 2022 - Erster Antritt

Hinweis:

Roter Text kennzeichnet Ihre Eingaben in der Konsole, Ihre Eingaben werden nicht tatsächlich rot sein.
Weißer Text kennzeichnet die Ausgaben Ihres Programms.

Aufgabe

Schreiben Sie ein Programm, welches das Spiel "Türme von Hanoi" implementiert. In diesem Spiel gibt es eine beliebige Anzahl von Türmen (Towers), wobei der erste Turm mit einer beliebigen Anzahl von Scheiben (Slices) unterschiedlicher Größe (Size) startet. Diese Scheiben werden so angeordnet, dass sich die kleinste Scheibe ganz oben auf dem Turm befindet und die größte ganz unten.

Das Ziel des Spiels ist es, alle Scheiben vom 1. Turm (ganz links) auf den letzten Turm (ganz rechts) zu schieben. Das schwierige daran ist es, dass nur kleinere Scheiben auf größere gelegt werden dürfen. Sollten Sie zum Beispiel versuchen eine Scheibe mit der Größe (Size) 3 auf eine Scheibe der Größe 1 legen, wäre das ein ungültiger Spielzug!

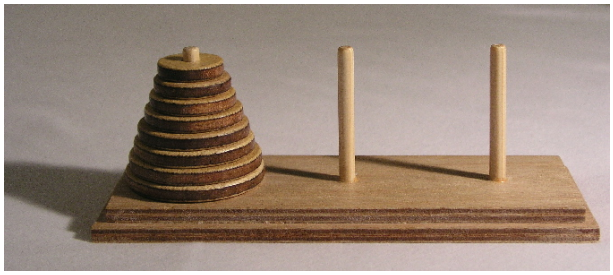

Figure 1: Beispiel der Türme von Hanoi

UML

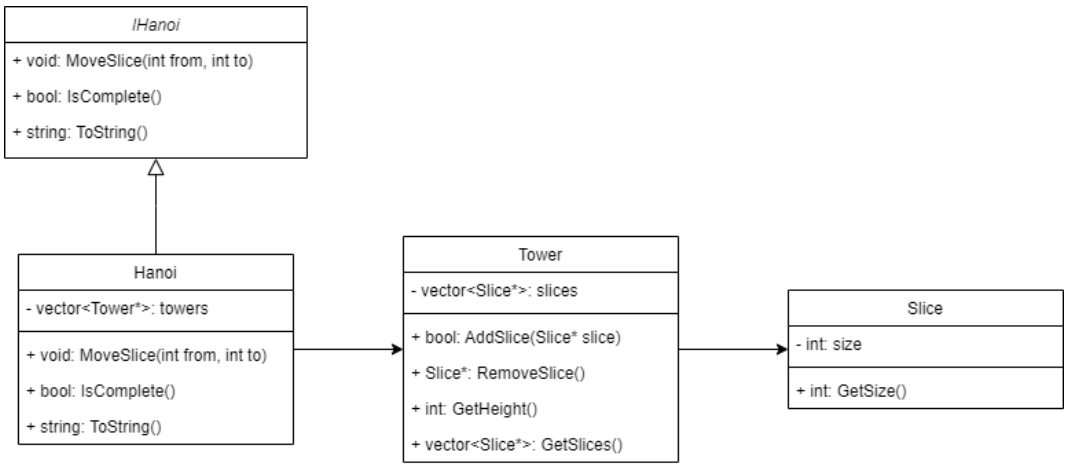

Figure 2: UML-Diagramm des Beispiels

Funktionalität

Lesen Sie die gesamte Angabe bevor Sie mit der Implementierung starten und implementieren Sie die benötigten Klassen anhand des UML-Diagramms (2).

- Erstellen Sie die Klasse *Slice*, die einen Konstruktor implementiert, welcher einen Integer als Parameter übernimmt, welcher die Größe *size* setzt.
  - *GetSize* - retourniert die Größe *size*.
- Erstellen Sie die Klasse *Tower*
  - *AddSlice* - legt eine Scheibe auf den Turm und gibt false zurück, falls es sich um einen ungültigen Spielzug handelt (wenn eine größere Scheibe auf eine kleinere Scheibe gelegt wird). Wenn der Spielzug ungültig war, wird zudem die Scheibe nicht auf den Turm gelegt.
  - *RemoveSlice* - entfernt die oberste Scheibe und retourniert diese. Sollte der Turm leer sein, wird ein *nullptr* zurückgegeben.
  - *GetHeight* - gibt die Größe des aktuellen Turms zurück.
  - *GetSlices* - gibt die Scheiben (slices) des aktuellen Turms zurück.
- Erstellen Sie die Interface-Klasse *IHanoi*, welche allen erbenden Klassen die Basis-Funktionalität des Spiels übergibt.
- Erstellen Sie die Klasse *Hanoi* die von *IHanoi* erbt. Implementieren Sie außerdem einen Konstruktor, der zwei Integer als Parameter übernimmt. Der erste Integer zeigt die

Anzahl der zu erstellenden Türme und der zweite Integer, die Anzahl der zu erstellenden Scheiben am 1. Turm. Fügen Sie deshalb am 1. Turm die entsprechende Anzahl an Scheiben in der korrekten Reihenfolge hinzu. Sie können davon ausgehen, dass sie nur valide Parameter erhalten. Achten Sie außerdem auf eine korrekte Speicherverwaltung!

- *MoveSlice* - bewegt eine Scheibe von einem Turm zu einem anderen, wobei die Parameter *from* und *to* dem entsprechenden Index entsprechen. Werfen Sie eine Exception mit der entsprechenden Fehlermeldung für folgende Fälle und reagieren Sie entsprechend darauf mit Try-Catch-Blöcken. Geben Sie außerdem bei allen Try-Catch-Blöcken "An unknown error occurred" aus, falls eine Fehlermeldung unbestimmten Typs geworfen wird.
  - Einer der Parameter ist Out-Of-Bounds ("One of your parameters was out of bounds!")
  - Der ausgewählte Turm, von dem eine Scheibe bewegt werden soll ist leer ("The selected tower does not have any slices to move!")
  - Ungültiger Spielzug - z.B. "tower->AddSlice(slice) == false" - wenn eine größere Scheibe auf eine kleinere Scheibe gelegt wird ("Invalid game move!")
- *IsComplete* - gibt zurück, ob sich alle Scheiben auf dem letzten Turm befinden.
- Überladen Sie den <<-Operator und ersetzen Sie alle existierenden Aufrufe von *ToString* mit dem überladenen Operator. Der überladene Operator soll die entsprechende ToString-Methode verwenden.

Hinweise

- Sie dürfen Ihre Klassen um private Hilfsmehoden erweitern, allerdings dürfen keine neuen public Methoden implementiert werden (ausgenommen überladene Operatoren für Teil 2).
- Parameter und Rückgabewerte von vorgegebenen Methoden dürfen nicht geändert werden.
- Groß-/Kleinschreibung von vorgegebenen Methoden dürfen geändert werden. Passen Sie dementsprechend vorgegebenen Code an.

Bewertungsschlüssel

- 30% Erstellen der Klassenstruktur anhand des vorgegebenen UML-Diagramms
- 15% Korrekte Implementierung der Methoden der Klasse *Hanoi*
- 15% Korrekte Implementierung der Methoden der Klasse *Tower*
- 5% Korrekte Implementierung der Methoden der Klasse *Slice*
- 15% Speicherverwaltung
- 10% Operator-Overloading
- 10% Exceptions

Achten Sie darauf, Programmierkonventionen einzuhalten und sprechende Variablennamen zu verwenden. Im Code Review kann das Ihre Bewertung beeinflussen.

Beispiel Aus- und Eingaben:

```
1
2
3
4
5
-

-

-


Move from :
>> 1
Move to :
>> 2

The selected tower does not have any slices to move!
1
2
3
4
5
-

-

-


Move from :
>> 0
Move to :
>> 1

2
3
4
5
-
```

```
1
-

-


Move from :
>> 0
Move to :
>> 1

Invalid game move!
2
3
4
5
-

1
-

-


Move from :
>> 0
Move to :
>> 5

One of your parameters was out of bounds!
2
3
4
5
-

1
-

-


Move from :
```

```
>> 0
Move to :
>> 2

3
4
5
-

1
-

2
-


Move from :
>> 1
Move to :
>> 2

3
4
5
-

-

1
2
-


Move from :
...
```

Codevorgaben

```cpp
int main()
{
    Hanoi* game = new Hanoi(3, 5);
    std::cout << game->ToString() << std::endl;

    while (!game->IsComplete()) {
        int from, to;

        std::cout << "Move from: " << std::endl;
        std::cout << " >> ";
        std::cin >> from;

        std::cout << "Move to: " << std::endl;
        std::cout << " >> ";
        std::cin >> to;

        system("cls");
        game->MoveSlice(from, to);
        std::cout << game->ToString() << std::endl;
    }

    delete game;
    return 0;
}
```

Figure 3: Vorlage der main.cpp

```cpp
#include <sstream> // <-- AM ANFANG DER DATEI INKLUDIEREN!

std::string Hanoi::ToString() const
{
        std::stringstream ss;

        for (auto tower : this->towers)
        {
                auto slices = tower->GetSlices();
                for (auto slice = slices.rbegin();
                     slice != slices.rend();
                     slice++)
                {
                        ss << (*slice)->GetSize() << std::endl;
                }
                ss << "-" << std::endl << std::endl;
        }

        return ss.str();
}
```

Figure 4: ToString-Methode der Game-Klasse