Die "Rule of Five"

Die "Rule of Five" ist ein Konzept in C++, das sich auf das Verwalten von Ressourcen in Klassen bezieht. Es erweitert die "Rule of Three" aus C++98, indem es zwei zusätzliche Methoden hinzufügt, die durch C++11 eingeführt wurden: den Move-Konstruktor und den Move-Zuweisungsoperator. Hier ist eine detaillierte Erklärung der "Rule of Five" mit dem Move-Zuweisungsoperator ausgeschaltet:

Die "Rule of Five" besagt, dass wenn eine Klasse eine der folgenden fünf Methoden benötigt, sie wahrscheinlich alle fünf benötigt:

1. Destruktor
2. Kopierkonstruktor
3. Kopierzuweisungsoperator
4. Move-Konstruktor
5. Move-Zuweisungsoperator (ausgeschaltet in diesem Beispiel)

Beispielklasse

Hier ist ein Beispiel, das die "Rule of Five" implementiert, wobei der Move-Zuweisungsoperator ausgeschaltet ist:

```cpp
#include <iostream>
#include <utility> // Für std::move

class MyClass {
public:
    // Konstruktor
    MyClass(int size) : data(new int[size]), size(size) {
        std::cout << "Constructor called" << std::endl;
    }

    // Destruktor
    ~MyClass() {
        delete[] data;
        std::cout << "Destructor called" << std::endl;
    }

    // Kopierkonstruktor
    MyClass(const MyClass& other) : data(new int[other.size]),
size(other.size) {
        std::copy(other.data, other.data + other.size, data);
        std::cout << "Copy Constructor called" << std::endl;
    }

    // Kopierzuweisungsoperator
    MyClass& operator=(const MyClass& other) {
        if (this == &other) {
            return *this;
        }
        delete[] data;
        size = other.size;
        data = new int[size];
```

```
        std::copy(other.data, other.data + other.size, data);
        std::cout << "Copy Assignment Operator called" << std::endl;
        return *this;
    }

    // Move-Konstruktor
    MyClass(MyClass&& other) noexcept : data(other.data),
size(other.size) {
        other.data = nullptr;
        other.size = 0;
        std::cout << "Move Constructor called" << std::endl;
    }

    // Move-Zuweisungsoperator (ausgeschaltet)
    MyClass& operator=(MyClass&& other) noexcept = delete;

private:
    int* data;
    int size;
};

int main() {
    MyClass obj1(10); // Konstruktor
    MyClass obj2 = obj1; // Kopierkonstruktor
    MyClass obj3(std::move(obj1)); // Move-Konstruktor
    // MyClass obj4;
    // obj4 = std::move(obj2); // Move-Zuweisungsoperator
(ausgeschaltet)

    return 0;
}
```

Kommentare

1. Konstruktor:
   Allokiert Ressourcen (in diesem Fall ein dynamisches Array).
2. Destruktor:
   Gibt die allokierten Ressourcen frei.
3. Kopierkonstruktor:
   Erstellt eine tiefe Kopie des Objekts. Dies ist notwendig, um sicherzustellen, dass jedes Objekt seine eigenen Ressourcen verwaltet.
4. Kopierzuweisungsoperator:
   Kopiert die Daten von einem anderen Objekt und gibt zuerst die aktuellen Ressourcen frei, um Speicherlecks zu vermeiden.
5. Move-Konstruktor:
   Übernimmt die Ressourcen eines temporären Objekts, anstatt sie zu kopieren. Dies ist effizienter, da keine neuen Ressourcen allokiert werden müssen.
6. Move-Zuweisungsoperator:
   Wurde in diesem Beispiel ausgeschaltet (`= delete`), was bedeutet, dass er nicht verwendet werden kann. Dies könnte sinnvoll sein, wenn Move-Semantik für die Klasse nicht erwünscht oder nicht sicher ist.

Warum den Move-Zuweisungsoperator ausschalten?

Es gibt Situationen, in denen das Ausschalten des Move-Zuweisungsoperators sinnvoll sein kann:

- **Sicherheitsgründe**: Wenn die Klasse Ressourcen verwaltet, die nicht sicher verschoben werden können (z.B. Dateihandles, Netzwerkverbindungen).
- **Design-Entscheidungen**: Wenn die Klasse nicht dafür vorgesehen ist, nach ihrer Erstellung verschoben zu werden.
- **Komplexität**: Wenn die Implementierung des Move-Zuweisungsoperators zu komplex oder fehleranfällig ist.

Durch das Ausschalten des Move-Zuweisungsoperators wird sichergestellt, dass Objekte dieser Klasse nicht durch Move-Zuweisung verschoben werden können, was potenzielle Fehler oder undefiniertes Verhalten verhindert.