LANDESBERUFSSCHULE 4 SALZBURG

Informatik

Klassen, Objekte, Instanzen

LBS 4

Dieses Skript dient als zusätzliche Lernunterlage für Informatik

Inhalt

Coding Style

Namen so wählen, dass ein Bezug hergestellt werden kann!

Variable/Attribut:	my_number	getrennt durch _
Methoden:	getNumber()	2. Teil des Namens groß
Klasse:	class PrivateRoom{}	Anfangsbuchstabe groß
Member Variablen (private):	m_seats / seats_	prefix mit m_ / suffix _
Klasse:	zuerst Attribute und dann Methoden	
Typennamen	string, int32_t	klein
Konstanten	TAX_RATE	alles groß
Namensraum:	namespace	alles klein
globale Variablen:	:: uint32_t my_global_number	

Kommentare:

Am Anfang:	Name, Datum, Bedeutung, Version
Im Code:	// einzeilige Kommentare
	/* mehrzeilige Kommentare */

Nur die wichtigen Methoden, Algorithmen kommentieren

Klassen und Objekte

Die Beziehung zwischen Klassen und Objekte muss klar definiert sein. Eine Klasse kann man wie einen Bauplan zu einem Objekt sehen. Sie enthalten die Eigenschaften (Attribute) und das Verhalten (Methoden) eines Objektes. Diese können über definierte Schnittstellen verwendet werden.

Objekt:

In der OOP ist ein Objekt eine Abstraktion von einem Gegenstand oder einem Gedanken, wie zum Beispiel ein Tisch, ein Baum, ein Bleistift oder ein Vorhaben. Eigentlich ist jedes Objekt für sich einzigartig, andererseits gibt es für jedes Objekt gleichartige Eigenschaften. Wenn Sie in ein Schreibwarengeschäft gehen, finden Sie dort blaue, grüne, rote, dicke, dünne Stifte.

Für die Bezeichnung gleichartiger Objekte wird der Oberbegriff Stifte verwendet. Diese Bezeichnung sagt aber nichts darüber aus, ob der Stift rot, dünn oder dick ist.

In der OOP werden derartige Oberbegriffe Klassen genannt.

Klassen sind allgemeine Beschreibungen für eine Gruppe gleichartiger Objekte. Sie haben zwei wichtige Aufgaben:

- Klassen beschreiben Dinge (Eigenschaften/Verhalten) die in einem Programm verwendet werden
- sie dienen als Vorlage für Objekte mit denen wir arbeiten wollen

Klassen beschreiben ihre Objekte mit Eigenschaften (Attribute) und Verhaltensweisen (Methoden). Sehr oft besitzt ein reales Objekt viel mehr Eigenschaften und Verhaltensweisen wie ein Objekt in der OOP. Die Abstraktion muss auf das wesentliche gesetzt werden.

Instanzen sind Objekte von Klassen zur Laufzeit.

Aufbau von Klassen in C++

Funktionsweise

Eine Klasse kann Methoden und Daten kapseln. Das bedeutet, dass nur auf bestimmte Werte von außen zugegriffen werden kann. Die Definition erfolgt in einer Header-Datei. (Dateiendung hpp, h++, …)

Jede Quellcodedatei, die diese Klasse verwendet, muss die Headerdatei inkludieren.

Die Methoden werden in der Quellcodedatei ausprogrammiert (*.cpp), wobei die Datei gleich wie die Klasse benannt wird.

Bsp.: #include “name.h“ #include „name.hpp“, ……

Klassen haben drei **Zugriffsspezifizierer** mit denen die Daten und Methoden geschützt werden:

- public
- private
- protected

Prinzipiell ist alles *private* wenn nicht das Schlüsselwort *public* oder *protected* verwendet wird. Das wird später noch genauer erklärt.

Aufbau einer Klasse:

```
class MyCar{
  private: // kein Zugriff von außen
        int8_t number; //privates Attribut

  public: //Zugriff von außen
        MyCar(); //Konstruktor wird beim Erstellen der Klasse aufgerufen
        ~MyCar(); //Destruktor wird beim Zerstören der Klasse aufgerufen
        void setNumber(int8_t num); //Settermethode
        int getNumber(); //Gettermethode
};
```

Bestandteile der C++ Klassen:

Klassenname:	Schlüsselwort class und Name MyCar der Klasse
Konstruktor:	wird beim Initialisieren automatisch aufgerufen; gleicher Name wie die Klasse MyCar();
Destruktor:	Anweisungen um Objekte zu vernichten; wird beim Zerstören der Klasse aufgerufen ~MyCar();
Methoden:	Setter, Getter und andere Methoden um gekapselte Werte zu verwenden setNumber(int8_t num);
Daten:	Variable oder Strukturen, int8_t number;

Die Definition der Klasse sowie die Deklaration der Attribute und Methoden erfolgt in eigenen Datei. (Header -Datei). Kurze Methoden können darin ausprogrammiert werden. (z.b.: setter und getter).

Getter und Setter sind Methoden die eine Schnittstelle zur Klasse definieren.

Für die Implementierung erzeugen Sie eine gleichnamige Datei *.ccp.
Diese enthält den Code für die Methoden. Jede Klasse soll in einer eigenen Quellcode-Datei erstellt werden.

Beispiel:

test.h	//Definition der Klasse, Attribute, Methoden
test.cpp	//ausprogrammierte Methoden

Daten schützen: Kapselung von Attributen

Die Kapselung ist ein wichtiges Grundkonzept der OOP. Kapselung bedeutet, dass Objekte den Zugriff auf ihre Daten selbst kontrollieren. Es wird sichergestellt, dass die Attribute (Daten) kontrollierte Werte besitzen.

Kapselung bedeutet, dass Attribute (Daten) des Objekts nur mit definierten Schnittstellen verwendet werden können. Getter- und Setter Methoden stellen die Möglichkeit zum Schreiben und Lesen der Information zur Verfügung. Diese Methoden können dann den Zugriff auf die Daten sehr gut kontrollieren.

Beispiel: Klasse

```
//Rectangle.h
class Rectangle
{
private:                                  //nur von den Klassenmethoden bearbeitbar
   uint32_t m_length, m_width;            // die Länge
public:                                   //von außen aufrufbar
   void setLength(int length);            // Methode zum Setzen der Länge
   int getLength();                       // Methode zum Lesen der Länge
   double getCircum();                    // Methode zur Berechnung des Umfangs
};
```

```
 //Rectangle.cpp

void Rectangle::setLength(int length)     // Methode zum Setzen der Länge
  {
    if (length > 0)
      this->m_length = length;
    else
      this->m_length = 0;
  }
  int32_t Rectangle::getLength()          // Methode zum Lesen der Länge
  {
    return this->m_length;
  }
  double Rectangle::getCircum()           // Methode zur Berechnung des Umfangs
  {
    return 2*(this->m_length + this->m_width);
  }
```

Vorgehensweise beim Definieren einer Klasse:

1. Klassengerüst erstellen
2. Eigenschaften definieren
3. Methoden definieren
4. Zugriffsspezifizierer richtig setzen
5. Dokumentation der Methoden ev. Eigenschaften
6. Methoden ausprogrammieren

Konstruktor:

Bei der Erstellung eines Objektes wird automatisch der Konstruktor der Klasse aufgerufen. Das bietet die Möglichkeit die Attribute auf einen bestimmten Anfangswert zu setzen.

Der Konstruktor hat denselben Namen wie die Klasse und hat keinen Rückgabewert.

Beispiel: Konstruktor

```
class Test{
private:
int32_t m_number;
public:
        Test();

void setNumber(int32_t number);
}
```

Wenn kein Konstruktor in der Klasse definiert ist, dann wird ein virtueller Konstruktor erstellt.

Destruktor:

Der Destruktor wird für die Zerstörung der dynamisch erstellten Elemente benötigt. Jedes Objekt, dass mit *new* instanziiert wurde, muss mit *delete* zerstört werden. Der Destruktor wird am Ende der Laufzeit des Objektes automatisch aufgerufen. Das Freigeben des Speichers müssen Sie selber vornehmen.

Ausnahme SmartPointer: wird noch durchgenommen!

Beispiel:

```
//hpp
class Test{
private:
int64_t *my_array; //Zeiger vom Typ int64_t (long)
int64_t m_number;

public:
        Test();
        void setNumber(int number);
        ~Test();
```

```
        //cpp
#include ”Test.hpp”
Test::Test(){
        m_number = 0;
        my_array = new int64_t[3]; //erstellen von einem Array
        }

void Test::setNumber(int64_t number){
        m_number = number;
}
        Test::~Test(){
                delete [] my_array; //vernichten des Arrays
                my_array = NULL;
        }
}
```

This-Zeiger

Damit Methoden die Attribute der Objekte bearbeiten können, übergibt der Compiler ein zusätzliches Argument an die Methode. Der this-Zeiger zeigt auf die Instanz, von der aus der Aufruf kam.

Achtung! ***this*** ist ein Zeiger. Bei der Verwendung muss mit dem -> Operator „dereferenziert“ werden.

Beispiel: this-Zeiger

```
double getCircum (){
         return 2 * (this->m_length + this->m_width);
}
```

Copy-Konstruktor

Jede C++-Klasse besitzt neben dem normalen Konstruktor auch einen Copy-Konstruktor. Dieser besitzt ein Referenzargument vom Typ der Klasse und kopiert die Eigenschaften des übergebenen Objekts in die Eigenschaften der Instanz, für die der Copy-Konstruktor ausgerufen wird. Wenn Sie keinen eigenen Copy-Konstruktor in die Klasse integrieren, fügt der Compiler einen automatisch ein.

Der Copy-Konstruktor wird implizit verwendet, wenn Sie eine Instanz einer Klasse auf eine Variable zuweisen:

```
Rectangle r2 = r1;                    //Copy-Konstruktor wird aufgerufen
```

Es wird eine Kopie von r1 im Objekt r2 gespeichert. Beide besitzen die gleichen Werte und sind dennoch verschiedene Objekte.

Wenn Sie in C++ Objekte dynamisch erstellen und den Zeiger einem anderen Objekt zuweisen wird der Copy-Konstruktor nicht aufgerufen.

```
Rectangle *r3 = new Rectangle        //Objekt am Heap erstellen
Rectangle *r4 = r3;                  //Copy-Konstruktor wird nicht aufgerufen
```

Beide – r3 und r4 – zeigen auf dasselbe Objekt. In diesem Fall muss der Copy-Konstruktor direkt aufgerufen werden.

```
Rectangle *r5 = new Rectangle(r1);   //Copy-Konstruktor wird direkt aufgerufen
```

Der Copy-Konstruktor wird immer aufgerufen, wenn keine Zeiger und Referenzen übergeben werden. (flache Kopie)

Aufgabe:

Definieren Sie eine Klasse Auto. Überlegen Sie welche Attribute und Methoden notwendig sind und welche von außen nicht erreichbar sein sollen. Skizzieren Sie sich die Klasse zuerst auf einem Blatt Papier. (mind. 5 Attribute)

Definieren Sie die Klasse in einer eigenen Headerdatei und programmieren Sie die wichtigsten Methoden in einer eigenen Quellcodedatei aus.

Testen Sie die Klasse, instanziieren Sie mehrere Objekte (statisch und dynamisch), verwenden Sie den Copy-Konstruktor. Zerstören Sie die dynamisch erstellten Objekte (Ausgabe im Destruktor)