

Grundlagen der Informatik II – 9. Übung

Schwerpunkt: Einführung in die Objektorientierte Programmierung, Konstruktor und Destruktor, Beispielklasse zur Verwaltung von Studenten

Grundidee der Objektorientierten Programmierung

- Vereinigung verschiedenster Eigenschaften und Funktionen eines Objektes in einem kompakten abstrakten Datentypen – der „Klasse“ (Konstruktionsplan für ein Objekt)
- Beispiel: Klasse zur Verwaltung von Studenten

Student	
<i>Eigenschaften, Attribute</i>	<ul style="list-style-type: none">• Name• Studienfach• Kartenguthaben• Matrikeljahr ...
<i>Funktionen, Methoden</i>	<ul style="list-style-type: none">• Immatrikulieren• Studiendauer ermitteln• Details ausgeben• Studienfach ändern...

- Implementation der Klassen-Deklaration (Bekanntmachung) in C++:

```
class Student
{
    private:
        char name[40], fach[40];
        int matrikeljahr;
        float guthaben;
        char kontonummer[12];
    public:
        void immatrikuliere(char n[], char f[], int jahr);
        int studiidauer(int aktuellesjahr);
        void ausgabe();
        void set_fach(char f[]);
};
```

- Konvention:
 - Attribute werden „private“ gemacht, d.h. Zugriff auf diese nur innerhalb der Klassenmethoden möglich
 - Methoden, die relevant für die Benutzung sind, werden „public“ gemacht – sie stellen damit die „Schnittstelle“ nach aussen dar
- zur Änderung von Attributen von aussen verwendet man vorzugsweise „set_...“- und „get_...“-Funktionen (z.B. set_fach, set_name, get_matrikeljahr, ...)

Trennung zwischen Deklaration und Implementierung

- analog zu Funktionsprototypen wird die Klasse erst deklariert und dann implementiert
- Implementation einer Methode muss zur Klasse zugehörig gemacht werden über den „::“-Operator:

```
[Rückgabety] Klassenname :: Methode ( ... ) { }
```

- Beispiel für Methode „studiendauer“ der Klasse „Student“:

```
int Student::studiendauer(int aktuellesjahr)
{
    return aktuellesjahr - matrikeljahr;
}
```

Objekte einer Klasse erzeugen und mit ihnen arbeiten

- Erzeugung von Objekten dort, wo sie gebraucht werden: globale Objekte, Objekte in Funktionen, als Rückgabe von Funktionen etc.
- hier: Beschränkung auf Objekte, die in der main()-Funktion erzeugt werden
- vereinfachte Syntax für die Variablendeklaration:

```
Klassenname Objektvariable;
```

- Syntax für den Zugriff auf Methoden eines Objektes einer Klasse:

```
Objektvariable . Methodename ( Argumente ) ;
```

- Beispiel für Klasse „Student“:

```
int main()
{
    Student xy;
    xy.immatrikuliere("Fischer", "Biologie", 2002);
    xy.ausgabe();
    cout << "Studiendauer: " << xy.studiendauer(2005);
    return 0;
}
```

Initialisierung und Deinitialisierung von Objekten

- Objekte werden bei der Erzeugung „initialisiert“ und es wird immer folgende Funktion aufgerufen:

```
Klassenname :: Klassenname ( )
```

- diese Funktion nennt sich „Konstruktor“ einer Klasse und kann bei Bedarf neu definiert werden, um eigene Initialisierungen vorzunehmen (ohne Funktionstyp!)
- Beispiel-Konstruktor für die Klasse „Student“:

```
Student::Student() {  
    guthaben = 30.00;  
    strcpy(fach, "");  
    strcpy(name, "Nobody");  
    matrikeljahr = 0;  
}
```

- analog zur Erzeugung wird bei der Freigabe eines Objektes (beim Programmende oder beim Aufruf von „delete“ für ein dynamisch erzeugtes Objekt) stets folgende Funktion aufgerufen:

```
Klassenname :: ~ Klassenname ( )
```

- diese Funktion nennt sich „Destruktor“ einer Klasse und kann ebenfalls neu definiert werden, um eigene Aufräumarbeiten vorzunehmen (wieder ohne Funktionstyp!)
- Beispiel-Destruktor für die Klasse „Student“:

```
Student::~~Student() {  
    cout << name << " wurde exmatrikuliert!";  
    ueberweisung_restguthaben(guthaben);  
    cout << "Das Restguthaben wurde gutgeschrieben!";  
}
```

- sowohl Konstruktor als auch Destruktor müssen in der Klassendeklaration vorkommen, wenn sie dann wie eben gezeigt neu implementiert werden sollen:

```
class Student  
{  
    private:  
        // ...  
    public:  
        Student();  
        ~Student();  
        // ...  
};
```