I. Objekt-orientierte Programmierung

Bei der objektorientierten Programmierung werden Daten und Anweisungen zusammengefasst zu Objekten. Ein Objekt besitzt also Eigenschaften (die sogenannten Attribute) und Verhaltensweisen (die sogenannten Methoden oder Elementfunktionen). Die wichtigsten Konzepte bei der OOP sind:

- Kapselung: auf die Attribute kann nicht direkt zugegriffen werden, sondern nur über Methoden des Objekts
- Vererbung: ein abgeleitetes Objekt erbt alle Eigenschaften und Verhaltensweisen. Diese können erweitert oder angepasst werden.
- Polymorphie: eine Variable kann nicht auch Instanzen einer abgeleiteten Klasse speichern und dennoch zur Laufzeit auf die „richtigen" Methoden zugreifen.

Die wichtigsten Begriffe:

- Klasse: Gesamtheit aller Objekte der gleichen Art (Bsp: BA-Student).
- Instanz: Ein Objekt einer bestimmten Klasse (jeder einzelne Student). Das Erzeugen einer Instanz heisst Instanziieren (das Einschreiben an der BA).

In C++ sind Klassen eine Erweiterung der Strukturen um Methoden. Dabei kann man verschiedene Arten von Methoden unterscheiden:

- Verwaltungsmethoden — technische Aufgaben wie z.B. Initialisierung (Konstruktor, Destruktor)
- Implementierungsmethoden — Funktionalität
- Hilfsmethoden — unterstützende Aufgaben; typischerweise private
- Zugriffsmethoden — Methoden, um auf die privaten Attribute lesend/schreibend zuzugreifen

Es gibt spezielle Methoden, die beim Instanziieren bzw. beim Entfernen der Instanz automatisch aufgerufen werden. Dieses sind die Konstruktoren bzw. Destruktoren. **Achtung!** Weder Konstruktor noch Destruktor besitzen einen Rückgabetyp (auch nicht void!). Destruktoren besitzen ausserdem keine Parameterliste (auch nicht void!).

Auch bei den Konstruktoren unterscheidet man verschiedene Arten:

- Standardkonstruktor — wenn kein anderer Konstruktor in Frage kommt (keine Argumente)
- allg. Konstruktor — beliebige Argumente; kann überladen werden
- Kopierkonstruktor — konstante Referenz als Argument; Kopie einer Instanz erstellen
- Typumwandlungskonstruktor — ein Argument; dient zur Typumwandlung (wenn keine Umwandlung erlaubt werden soll, muss in der Deklaration `explicit` vor dem Konstruktor stehen)

A. Kapselung

Um den Zugriff auf einzelne Bestandteile (Attribute/Methoden) einer Klasse zu verhindern, können diese Teile als privat deklariert werden. Soll dagegen der Zugriff von „aussen" zulässig sein, so sind diese Teile öffentlich.

In manchen Fällen möchte man einer einzelnen anderen Klasse Zugriff auf die privaten Bestandteile der eigenen Klasse gewähren. Dazu definiert man diese Klasse als befreundet (`friend class andere_klasse`). Friend-Deklarationen werden üblicherweise am Anfang der Deklarationen aufgeführt.

B. Vererbung

Die Spezialisierung eines Objekts heisst Vererbung. Vererbt werden Attribute und Methoden. Die Klasse, die spezialisiert wird (von der also eine weitere Klasse abgeleitet wird) heisst Basisklasse, die spezialisierte Klasse heisst abgeleitete Klasse.

Bei der Vererbung in C++ ist zu beachten, welche Zugriffsmöglichkeiten auf die einzelnen geerbten Attribute und Methoden bestehen. Die abgeleitete Klasse hat zum Beispiel keinen Zugriff auf die privaten Attribute der Basisklasse. Soll der Zugriff hierauf möglich sein, ohne dass diese Atrribute allgemein zugänglich sind, so müssen sie als geschützt definiert werden.

Zugriffsmöglichkeiten:

Attribut/Methode	Klasse + Freunde	Erben	Fremde
private	√	X	X
protected	√	√	X
public	√	√	√

Auch bei der Vererbung muss zwischen drei verschiedenen Arten unterschieden werden. Diese unterscheiden sich in der Art, in welche Zugriffszone die geerbten Attribute und Methoden eingeordnet werden:

Zugriffsrechte nach dem Ableiten (Zeile: Zugriffsart vor dem Ableiten; Spalte: Art der Vererbung):

	private	protected	public
private	private	private	private
protected	private	protected	protected
public	private	proteced	public

C. Polymorphie

Mit Hilfe der Polymorphie ist es möglich, eine Liste zu definieren, die Zeiger auf Instanzen einer gewünschten Klasse aufnimmt. In dieser Liste können dann nicht nur Zeiger auf Instanzen dieser Klasse, sondern auch Zeiger auf Instanzen einer beliebigen davon abgeleiteten Klasse gespeichert werden.

Wird eine Methode in der abgeleiteten Klasse neu implementiert, so kann durch die Definition als virtuelle Methode sichergestellt werden, dass zur Laufzeit ermittelt wird, von welcher Klasse die gerade betrachtete Instanz der Liste tatsächlich ist, und die entsprechende Ausprägung dieser Methode aufgerufen wird. Die Definition einer virtuellen Methode erfolgt durch das Schlüsselwort „virtual“, das der Methodendefintion in der Klasse vorangestellt wird.

Beispiel:

```
class c1
{
  ...
  virtual void display (void);
  ...
};

class c2 : public c1
{
  ...
  virtual void display (void);
  ...
};
...
c1 *liste [2];
liste [0] = new (c1);
liste [1] = new (c2);
liste [0]->display ();  // Methode c1::display
liste [1]->display ();  // Methode c2::display
```

Wird eine Methode in der Basisklasse als virtuell definiert, dann ist sie automatisch in jeder abgeleiteten Klasse virtuell. Ihre Schnittstelle (=Signatur) darf sich dann beim Ableiten nicht ändern (da beim Aufruf der Methode die Anzahl und die Typen der Parameter immer die gleichen sind).

Um die Polymorphie benutzen zu können, müssen die beteiligten Klassen also eine gemeinsame Basisklasse besitzen, in der die benötigten Methoden bereits definiert sind. In manchen Fällen muss diese Basisklasse erst „künstlich“ erzeugt werden, ohne dass eine Implementation dieser Methoden in dieser Basisklasse Sinn machen würde und ohne dass von dieser Basisklasse eine Instanz erzeugt werden soll. In diesem Fall kann die virtuelle Methode als rein-virtuell definiert werden. Dann bezeichnet man die zugehörige Klasse als „abstrakt“ und es ist nicht möglich, eine Instanz dieser Klasse zu anzulegen. Eine virtuelle Methode wird durch die Zuweisung „= 0“ zu einer rein-virtuellen Methode:

```
class c1
{
  ...
  virtual void display (void) = 0;   // rein-virtuelle Methode => abstrakte Klasse
  ...
};
```

Jede abgeleitete Klasse, von der Instanzen angelegt werden sollen, muss dann zumindestens diese rein-virtuellen Methoden neu implementieren, da sie andernfalls auch zu einer abstrakten Klasse würde.