Klasse (objektorientierte Programmierung)

aus Wikipedia, der freien Enzyklopädie

Klasse ist in der objektorientierten Programmierung ein abstrakter Oberbegriff für die Beschreibung der gemeinsamen Struktur und/oder des gemeinsamen Verhaltens von Objekten (*Klassifizierung*). Sie dient dazu, Dinge (Objekte) der realen Welt zu abstrahieren. Im Zusammenspiel mit anderen Klassen ermöglichen sie die Modellierung eines abgegrenzten Systems der realen Welt (*siehe* Objektorientiertes Design).

Die Struktur einer Klasse bilden die Attribute (auch *Eigenschaften*), das Verhalten die Methoden (auch Operation, Funktion, Prozedur) der Klasse.

Aus Klassen erzeugte Objekte werden als **Exemplare** oder **Instanzen** der Klasse bezeichnet.

In manchen Programmiersprachen gibt es zu jeder Klasse ein bestimmtes Objekt (Klassenobjekt), das dazu da ist, die Klasse zur Laufzeit zu repräsentieren; dieses Klassenobjekt ist dann auch zuständig für die Erzeugung von Objekten der Klasse und den Aufruf der korrekten Methode.

Klassen können von anderen Klassen *abgeleitet* werden (Vererbung). Dabei erbt die Klasse die Datenstruktur (*Attribute*) und die Methoden von der *vererbenden* Klasse (Basisklasse). Die abgeleitete Klasse (Subklasse) kann Methoden der Basisklasse *überschreiben* (sofern diese es erlaubt; die Möglichkeit, dies zu verbieten, hängt von der Programmiersprache ab), d.h. die Methode neu implementieren, und eigene Methoden und Daten (*Attribute*) hinzufügen. Ein Objekt der abgeleiteten Klasse kann überall verwendet werden, wo ein Objekt der Basisklasse erwartet wird; überschriebene Methoden werden dann auf der abgeleiteten Klasse ausgeführt (Polymorphie).

Klassen werden in der Regel in Form von Klassenbibliotheken zusammengefasst, die häufig thematisch organisiert sind. So können Anwender einer objektorientierten Programmiersprache Klassenbibliotheken erwerben, die den Zugriff auf Datenbanken ermöglichen.

Beispiele

Angenommen es wird eine Klasse für eine Tabelle einer einfachen Datenbank entworfen. Ein Objekt ist dann ein einzelner Datensatz (= Zeile) in dieser Tabelle.

Ein Klasse *Bankkonto* könnte beispielsweise mit folgenden Eigenschaften definiert werden:

- hat eine Kontonummer (Attribut)
- hat einen Kontoinhaber (Referenz auf einen Kunden)
- hat eine Liste von Buchungen (Liste von Referenzen auf Buchungen)
- hat einen Saldo (Attribut)

- kann eine Einzahlung durchführen (Methode)
- kann eine Auszahlung durchführen (Methode)

Jede Kontobewegung würde dann einem Methodenaufruf entsprechen und der Liste der Buchungen ein Element hinzufügen sowie den Saldo modifizieren.

Um eine Klasse "Schueler" zu erzeugen, kann man verschiedene Eigenschaften definieren: Name, Vorname, Geburtstag, Geschlecht, Adresse.
Ein spezieller Schüler ist dann ein **Objekt** dieser Klasse, wenn für ihn eine Instanz erzeugt wurde und wenn seine Daten in die Instanzattribute eingetragen sind.

Programmbeispiel

Das folgende Beispiel ist in C++ geschrieben:

```
class Basisklasse {
  public:
    virtual void machwas() {
      std::cout << "Basisklasse::machwas\n";
    }
};
```

```
class abgeleitete_Klasse : public Basisklasse {
  public:
    virtual void machwas() {
      std::cout << "abgeleitete_Klasse::machwas\n";
    }
};
```

```
void mach_was_mit(Basisklasse& b) {
  b.machwas();
}
```

```
int main() {
  Basisklasse        objekt1;
  abgeleitete_Klasse objekt2;
  mach_was_mit(objekt1);
  mach_was_mit(objekt2);
}
```

Dieses Programm definiert eine Klasse `Basisklasse` und eine davon abgeleitete Klasse `abgeleitete_Klasse`.

Die Basisklasse hat eine Methode namens `machwas()`, die `Basisklasse::machwas` ausgibt (dafür ist die Zeile mit `std::cout` zuständig). `virtual` bedeutet in C++ "kann überschrieben werden". Das macht die abgeleitete Klasse dann auch, sie definiert die Methode um, so dass sie `abgeleitete_Klasse::machwas` ausgibt.

Anschließend folgt die Definition einer eigenständigen Funktion `mach_was_mit(...)` , die ein Objekt der Basisklasse als Argument bekommt. Auf diesem Objekt wird die Methode `machwas()` gerufen.

(`objekt1`) als auch der abgeleiteten Klasse (`objekt2`) definiert, und dann `mach_was_mit(...)` zuerst mit einem Objekt der Basisklasse, dann mit einem Objekt der abgeleiteten Klasse aufruft.

Wird dieses Programm kompiliert und ausgeführt, so gibt es

```
Basisklasse::machwas
abgeleitete_Klasse::machwas
```

aus, obwohl die Funktion `mach_was_mit(...)` **nur** für ein Basisklassenobjekt als Argument definiert ist (in der Tat hätte die abgeleitete Klasse sogar *in einer später dazu gefügten Quelldatei* geschrieben werden können, nachdem die Funktion `mach_was_mit(...)` schon lange fertig kompiliert war). Dieses Verhalten nennt man **Polymorphie**.

Einige Programmiersprachen mit Klassen

- Smalltalk
- Python
- Java
- Objective C
- Object Pascal, Delphi
- C#
- C++
- Objective CAML

Eingeordnet unter: Objektorientierte Programmierung
