Zu den grundlegenden Programmiermethoden gehörten bisher die Deklaration von Datentypen und die Bearbeitung dieser Daten durch die Benutzung von vordefinierten oder benutzerdefinierten Funktionen. Bei den Funktionen wird lediglich überprüft, ob die Datentypen bei der Verwendung korrekt sind. Es wird keine Rücksicht darauf genommen, ob die Funktion für die Verarbeitung der Daten tatsächlich sinnvoll ist oder nicht. Beispielsweise ist das Guthaben auf einem Bankkonto vom Typ `double`. Zulässig wäre eine Funktion, welche das Guthaben quadriert. Das könnte zwar sehr schön sein ist aber sicherlich nicht zulässig. Sinnvolle Funktionen für das Konto wären beispielsweise Ein- oder Auszahlung. Durch die Einführung der Objektorientierten Programmierung (OOP) wird dieses Problem behoben.

Ein Objekt kann alles sein, was sich durch bestimmte **Eigenschaften** beschreiben lässt. Durch fest zugeordnete Funktionen können diese Eigenschaften verändert oder die Objekte in anderer Form beeinflusst werden. In diesem Zusammenhang nennt man die zugeordneten Funktionen auch **Methoden**.

Die wesentliche Erweiterung in C++ gegenüber C ist die Einführung von Klassen. Eine Klasse beschreibt die Eigenschaften und die Methoden eines Objektes. Die Deklaration erfolgt mit dem Schlüsselwort **`class`** gefolgt von einem Bezeichner. Der Bezeichner der Klasse bildet einen neuen Datentyp. Von diesem Datentyp können Objekte angelegt werden (so genannte Instanzen des Objektes). Im allgemeinen kann auf diese Objekte nur mit den zugeordneten Methoden zugegriffen werden. Für die Methoden gelten die bekannten Konventionen der benutzerdefinierten Funktionen.

Der Entwurf derartiger Software erfolgt beispielsweise durch Klassendiagramme mit UML[1].

Klasse = Eigenschaften + Methoden

Vorteile der Objektorientierten Programmierung:

1. Ein wesentlicher Vorteil ist die **Wiederverwendung** des Programmcodes, wenn die Objekte sinnvoll und allgemein gültig angelegt werden. So lassen sich beispielsweise Klassenbibliotheken erstellen, welche mittels einfacher #include-Anweisungen in neue Projekte eingebunden werden können.

2. Durch die verwendeten Methoden kann sichergestellt werden, dass die Eigenschaften der Objekte keine unzulässigen Werte erhalten können. Dadurch wird die Sicherheit der Daten gewährleistet (**Kapselung** der Daten).

3. Durch die Technik der **Vererbung** können von vorhandenen Klassen weitere Klassen abgeleitet werden. Für die abgeleiteten Klassen können zusätzliche Eigenschaften und Methoden vereinbart werden.

4. Objekte können bei Bedarf erst zur Laufzeit des Programms neu angelegt werden (**späte Bindung**). Dadurch wird erreicht, dass Speicherplatz nur für die vom Benutzer benötigten Objekte belegt wird. Die Erzeugung von Objekten zur Laufzeit erfolgt mit dem Schlüsselwort **`new`**.

5. Beim Anlegen der Objekte können die Eigenschaften durch so genannte Konstruktoren einen vordefinierten Wert erhalten (Standardeigenschaft).

[1] Unified modeling language

Ein einfaches Beispiel:

Ein Rechteck auf dem Bildschirm hat die wesentlichen Eigenschaften Länge und Breite. Die einfachsten Methoden sind z.B. die Veränderung dieser Eigenschaften oder die Berechnung der Fläche.

Beispiel für ein Klassendiagramm in UML:

CRechteck
private: double laenge double breite
public: void setlaenge() void setbreite() double getflaeche()

Für die Ein-/Ausgabe stehen die Methoden cin und cout zur Verfügung (in der Header-Datei `iostream` mit dem Namensraum `std`). Die vordefinierten Funktionen könnten in eigenen Namensräumen neu definiert werden.

C++-Programm :

```
#include<conio.h>
#include <iostream>     //Achtung: Ohne Erweiterung .h
using namespace std;
//Innerhalb eines Namensraumes haben Objekte einen eindeutigen Namen
class CRechteck
{
private:
      double laenge;
      double breite;
public:
      void setlaenge();
      void setbreite();
      double getflaeche();
};
/***********Initialisierung der Methoden***************/
void CRechteck::setlaenge()
{
      cout << "\nLaenge: ";
      cin >> laenge;
}
void CRechteck::setbreite()
{
      cout << "\nBreite: ";
      cin >> breite;
}
double CRechteck::getflaeche()
{
      return (laenge*breite);
}
```

```
int main()
{
      CRechteck wiese;        //Erzeugung des Objektes
      wiese.setlaenge();
      wiese.setbreite();
      cout << "\nFlaeche: " << wiese.getflaeche() << endl;
      _getch();
      return(0);
}
```

Elemente mit dem Attribut **private** sind nur für Methoden der eigenen Klasse verfügbar. Ein Zugriff auf die Eigenschaften `laenge` und `breite` innerhalb der main()-Funktion ist nicht möglich.

Funktionen mit dem Attribut **public** sind in allen Funktionen des Projektes verfügbar. Der Aufruf der Methoden erfolgt durch Angabe des Objektnamens gefolgt von dem .(Punkt)-Operator und dem Namen der Methode.

Die Initialisierung der Methoden erfolgt in diesem Beispiel außerhalb der Klassenbeschreibung. Zur Deklaration wird zunächst der Rückgabewert angegeben. Danach folgt der Objektname, der ::-Operator (Sichtbarkeitsoperator) und der Methodenname.

Übung:

Gemäß folgender UML soll eine Klasse CZylinder zur Berechnung von Oberfläche und Volumen eines Zylinders erstellt werden.

CZylinder
private: double durchmesser double hoehe
public: void setdurchmesser() void sethoehe() double getoberflaeche() double getvolumen()

Erstellen Sie bitte das Programm für die Klasse und ein Hauptprogramm zum Testen!

Zeiger auf Objekte :

Mit dem Schlüsselwort new können Objekte zur Laufzeit angelegt werden. Die Daten werden in einem dynamischen Speicherbereich, dem so genannten ‚Heap' (engl. Haufen) abgelegt. Der Vorteil liegt darin, dass die Größe und die Anzahl der Objekte nicht durch den Programmcode begrenzt werden.

Mit dem Schlüsselwort new wird der Speicher für ein Objekt im Heap bereit gestellt und ein Zeiger auf dieses Objekt eingerichtet. Der Zugriff auf Elemente und Methoden des Objektes erfolgt über die Zeigervariable gefolgt von dem -> (Pfeil-) Operator.

```
#include<conio.h>
#include<iostream>
using namespace std;
class CRechteck
{
private:
      double laenge;
      double breite;
public:
      void setlaenge();
      void setbreite();
      double getflaeche();
};

void CRechteck::setlaenge()
{
      cout << "\nLaenge: ";
      cin >> laenge;
}
void CRechteck::setbreite()
{
      cout << "\nBreite: ";
      cin >> breite;
}
double CRechteck::getflaeche()
{
      return (laenge*breite);
}
int main()
{
      CRechteck* wiese = new CRechteck; //wiese ist ein Zeiger auf das Objekt
      wiese->setlaenge();
      wiese->setbreite();
      cout << "\nFlaeche: " << wiese->getflaeche() << endl;
      _getch();
      return(0);
}
```