Die Programmiersprache Smalltalk (ST)
(Beispiele: IBM Smalltalk und Visual Age Version 2.0)

1 Einführung

1.1 Warum objektorientierte Programmierung (OOP) ?

Ursache: Enorme **Kostensteigerung** bei der Softwareentwicklung
1985: 140 Milliarden Dollar
1990: 250 Milliarden Dollar
2000: c.a.800 Milliarden Dollar

Entwurfsziel: Das Schreiben von Programmen zu erleichtern, insbesondere durch die Realisierung von objektorientierten Methoden. Im Mittelpunkt steht dabei die Erstellung und Wiederverwendbarkeit von Software-Bausteinen.

Es entstanden verschiedene OOP-Sprachen, auf prozedurale Sprachen aufgesetzt (C++, Pascal), die dann auch oft eine gemischte (pruzedurale und objektorientierte Programmierung) zulassen, aber auch völlige neuentwickelte Sprachen.
Smalltalk wurde eine der ersten reinen OOP-Sprachen, 1972 entwickelt, stand bis vor kurzem völlig im Schatten von C++. Erst in jüngster Zeit ist ein Boom in Richtung Smalltalk festzustellen.
Smalltalk ist in keiner Weise standardisiert. Literatur lehnt sich stets an ein spezielles Produkt an, so dass sich die dortigen Beispiele nur nach größerem Aufwand übertragen lassen.

(ST-Geschichte)

Literatur:
Andrew Valencia, A Tutorial for GNU Smalltalk, Informatik-Bibliothek,
aix550: /usr/lpp/oberon/SmallTalk/smalltalk-tutorial.ps.

Josef Mittendorfer, Objektorientierte Programmierung mit Smalltalk/V für Windows, 1992,
Addison-Wesley, Bonn.

Peter Coad u. a., Objektorientierte Programmierung, 1994, Prentice Hall Verlag GmbH, München.

Matthias C. Bücker u. a., Programmieren in Smalltalk mit VisualWorks, 1995, Springer-Verlag, Berlin.

Die grundlegenden Konzepte in OOP-Sprachen sind:
- **Objekte** als Datengrundbausteine, aus **Eigenschaften** (Daten) und **Methoden** (Anweisungen) zusammengefasst.
- **Klassen** zur Datenabstraktion und Datenkapselung, hierarchisch aufgebaut, mit **Vererbung, Funktions- und Operatorenüberlagerung** zur Wiederverwendbarkeit von Softwarebausteinen.

- **Polymorphimus,** dynamisches Binden von Objekten und Methoden während des Programmlaufes.

Objekte kommunizieren miteinander über **Nachrichten**, welche durch objekteigene Methoden verarbeitet werden.

1.2 Grundbegriffe der objektorientierten Programmierung

1.2.1 Objekte

Herkömmlicher **prozeduraler** Ansatz: Problem wurde durch einen Algorithmus beschrieben, dieser wurde dann schrittweise in genügend kleine Algorithmen zerlegt (Top-Down-Analyse) und danach in eine zur Verfügung stehende Programmiersprache codiert.

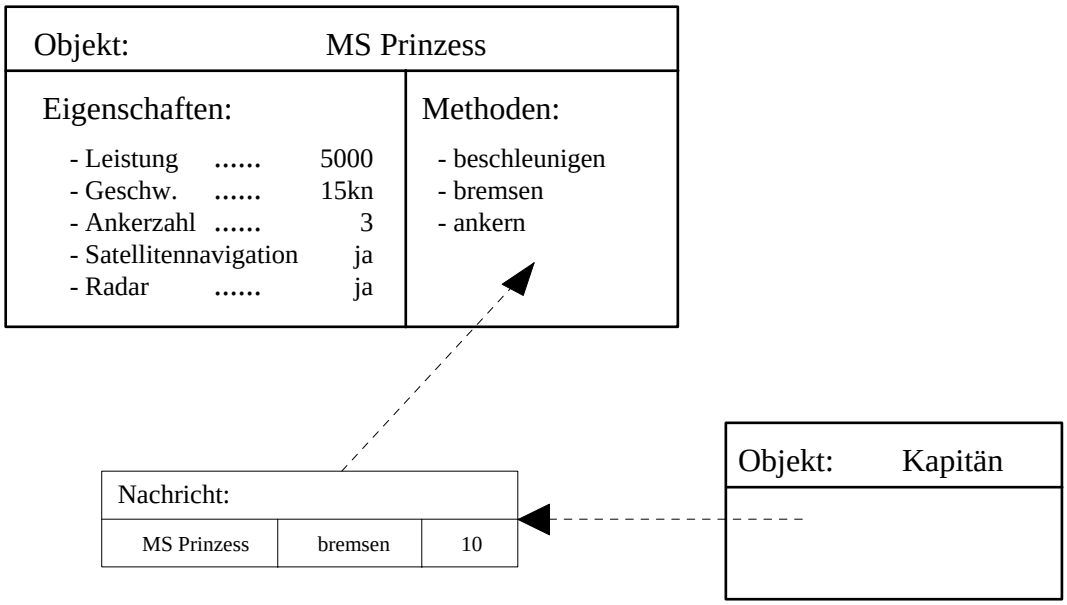
Neuer **objektorientierter** Ansatz: Das menschliche Denkverhalten des Klassifizierens und Abstrahierens wird ausgenutzt. Das zu modellierende Gesamtsystem wird in gegebene Teilsysteme zerlegt, bestehend aus miteinander kommunizierenden Einheiten. Jeder solchen Einheit wird ein Objekt zugeordnet. (Beispiel: Fahrstuhlsteuerung in einem Hochhaus: Objekte sind Personen, Etagen, Fahrstühle, aber auch Etagenschalter, Fahrstuhlknöpfe)

Ein **Objekt** im Sinne von OOP besteht nun aus
- **Eigenschaften**: Daten des Objekts (Strukturkomponente des Objekts).
- **Methoden**: Operationen zur Manipulation seiner eigenen Daten (Funktionskomponente des Objekts, in Smalltalk sind dies keine Funktionen => Begriff Methode).

Objekt:	MS Prinzess
Eigenschaften:	Methoden:
- Leistung ..... 5000 - Geschw. ..... 15kn - Ankerzahl ..... 3 - Satellitennavigation ja - Radar ..... ja	- beschleunigen - bremsen - ankern

Dieses Zusammenfügen von Eigenschaften (Daten) und Methoden (Funktionen) wird als **Kapselung (information hiding)** bezeichnet, eine wichtige Eigenschaft der OOP: Daten und Funktionen werden als ein Modul behandelt. Die Daten eines Objekts sollen **nur** durch eigene Funktionen manipuliert werden. Eine Veränderung der Daten von außen wiederspricht der Programmierphilosophie.

Zur Kommunikation mit anderen Objekten werden **Nachrichten (Botschaften)** verschickt, die bei den Objekten eine Ausführung einer oder mehrerer Methoden auslösen, welche eine Änderung seiner Daten nach sich ziehen kann.



Eine Nachricht besteht immer aus einem Zielobjekt, dem Empfänger (*receiver)* der Nachricht, einer Methode, die das Zielobjektes „ verstehen sollte “ und evtl. notwendige Parameter, die die Methode der Zielobjektes zur Bearbeitung der Nachricht benötigt.
Die Auswahl der Methode erfolgt evtl. erst zur Laufzeit (bei Smalltalk generell, bei Pascal und C++ wahlweise). Das wiederum ermöglicht, die Auswahl der entsprechenden Methode vom aufgerufenen Objekt abhängig zu machen.

1.2.2 Klassen

Die **Datenabstraktion** besteht nun darin, dass gleichartige Objekte in **Klassen** zusammengefasst werden. Die Klassen sind damit Datentypdefinitionen im üblichem Sinne und legen die Eigenschaften und Methoden der Objekte gleichen Typs fest.

Eine **Klasse** definiert:

(1) die Namen der objektspezifischen Eigenschaften,
(2) die Methoden gleichartiger Objekte,
(3) klassenspezifische Eigenschaften,
(4) klassenspezifische Methoden.

Variablen einer Klasse haben Objekte als Wert. Man bezeichnet sie als **Instanzen** und entsprechend wird unterschieden:

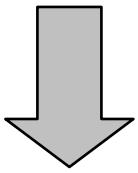
(1) **Instanzvariablen** Haben bei verschiedenen Objekten verschiedene Werte.
(2) **Instanzmethoden** Werden vom Objekt zur Datenmanipulation ausgeführt.
(3) **Klassenvariablen** Haben immer den gleichen Wert.
(4) **Klassenmethoden** Wird von der Klasse(z.B. Erzeugen eines Objekts) ausgeführt.

Klassen sind in einigen OOP-Sprachen wie auch Smalltalk selbst wieder Objekte, dann können diese auch Nachrichten verstehen und eigene Methoden ausführen. In C++ und Pascal sind Klassen keine Objekte, haben folglich keine Klassenmethoden und keine oder nicht manipulierbare Klassenvariablen, d.h. (1) und (2) sind vorhanden, (3) und (4) nicht.

Objekt:	MS Prinzess
Eigenschaften:	Methoden:
- Leistung ..... 5000 - Geschw. ..... 15kn - Ankerzahl ..... 3 - Satellitennavigation ja - Radar ..... ja	- beschleunigen - bremsen - ankern

Objekt:	MS Karibik
Eigenschaften:	Methoden:
- Leistung ..... 3000 - Geschw. ..... 10kn - Ankerzahl ..... 2 - Satellitennavigation ja - Radar ..... ja	- beschleunigen - bremsen - ankern

Objekt:	MS Wangenrooge
Eigenschaften:	Methoden:
- Leistung ..... 2000 - Geschw. ..... 10kn - Ankerzahl ..... 2 - Satellitennavigation ja - Radar ..... ja	- beschleunigen - bremsen - ankern



Klasse:	Motorboot
Eigenschaften:	Methoden:
- Leistung - Geschwindigkeit - Ankerzahl	- beschleunigen - bremsen - ankern
- Satellitennavigation - Radar	- Objektbildung

Instanzvariablen (1)

Instanzmethoden (2)

(3) **Klassenvariablen**

(4) **Klassenmethoden**

Zusammenfassend kann man feststellen, dass durch die Verwendung von Klassen zur Beschreibung abstrakter Datentypen einerseits die **Datensicherheit** erhöht wird, indem der Zugriff von außen unterbunden wird, und andererseits die Modifikationsfreundlichkeit erheblich verbessert wird, da ausführbare Operationen und Methoden bereitgestellt werden.
Durch sauber definierte **Schnittstellen** wird die Interaktion mit anderen Programmteilen geregelt. Dabei kann sich, ohne Einfluss auf die Programmierumgebung, die Implementation einer Methoden verändern, solange die Schnittstelle selbst gleich bleibt. Eine Klasse kann unabhängig von der Programmierumgebung ausgetestet werden.
Diese Art der Kapselung unterstützt die Entwicklung von Software, insbesondere bei mehreren Programmierern, wobei die Vorgehensweise nach dem Baukastensystem erfolgt. Den Mittelpunkt der Programmierung bilden nunmehr die Objekte der Klassen und nicht mehr die Prozeduren. Die prozedurale Programmierung wird damit von der objektorientierten Programmierung abgelöst.

Beschreibung der Klasse **Motorboot** in C++:

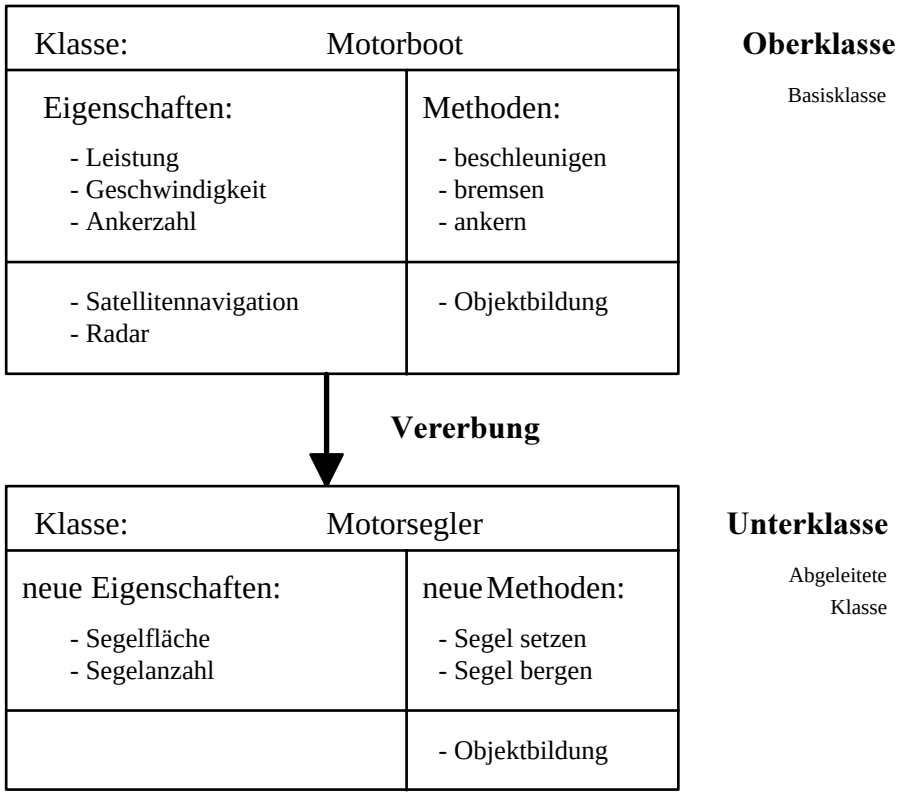
```
class Motorboot
{   private:
      int Leistung;                    // in PS
      int Geschwindigkeit;             // in Knoten
      int Ankerzahl;
      const boolean Satellitennavigation = true;
      const boolean Radar = true;
    public:
      void beschleunige( int delta);
      void bremsen( int delta);
      void ankern();
      void new();
};
```

Beschreibung der Klasse **Motorboot** in Smalltalk:

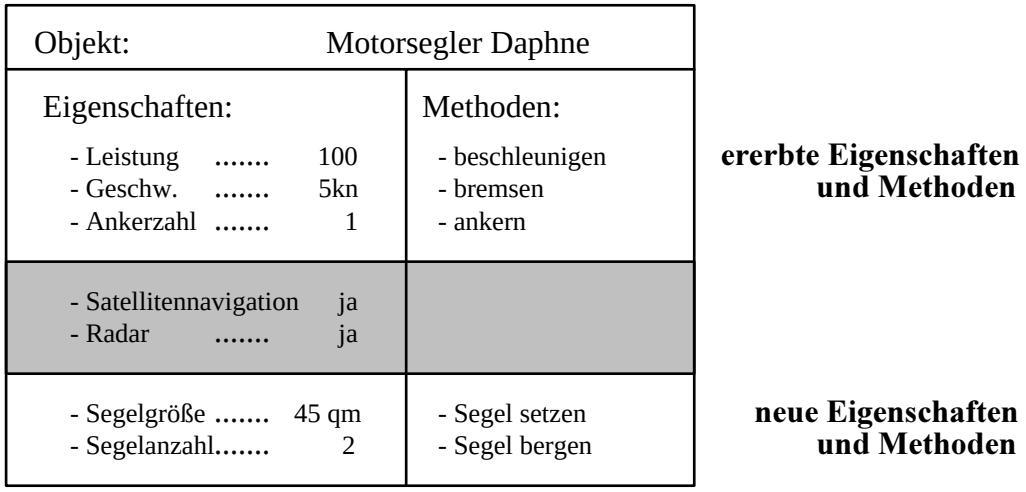
```
Object subclass: #Motorboot
    instanceVariableNames:
      'leistung geschwindigkeit ankerzahl'
    classVariableNames:
      'satellitennavigation radar'
    poolDicionary: ' '

    beschleunige: delta
      methodenDefinition

    bremsen: delta
      methodenDefinition

    ankern
      methodenDefinition

    new
      methodenDefinition
```

1.2.3 Vererbung

Während man gleichartige Objekte zu einer Klasse zusammenfasst, können ähnliche Objekte Klassen zugeordnet werden, welche voneinander abgeleitet werden.
Dabei besitzt eine von einer Klasse (**Oberklasse**) abgeleitete Klasse (**Unterklasse**) neben den Methoden und Eigenschaften der Oberklasse noch einige, zusätzliche Methoden und/oder Eigenschaften.



Ein Objekt der Klasse Motorsegler besitzt auch **alle Eigenschaften** der Klasse Motorboot und kann **alle Methoden** der beiden Klassen ausführen.



Beschreibung der Klasse **Motorsegler** in C++:

```
class Motorsegler: public Motorboot   //     abgeleitete
Klasse
{   private:
      int Segelfläche;                // in Quadratmeter
      int Segelanzahl;
    public:
      void Segel_setzen( int qm);
      void Segel_bergen( int qm);
};
```

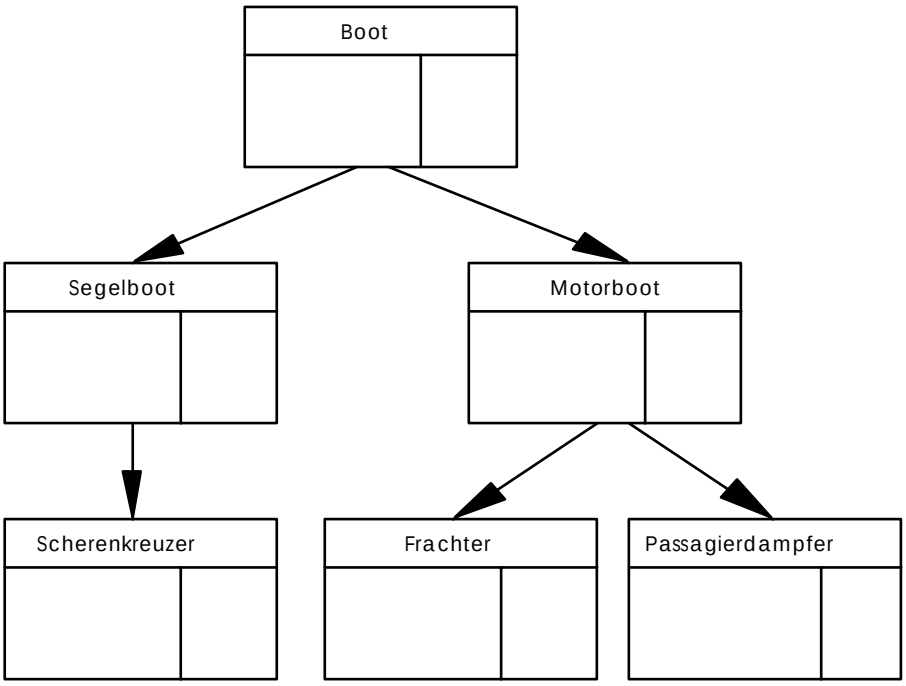
Beschreibung der Klasse **Motorsegler** in Smalltalk:

```
Motorboot subclass: #Motorsegler
    instanceVariableNames:
      'segelfläche segelanzahl'
    classVariableNames: ' '
    poolDicionary: ' '

    segel_setzen: qm
      methodenDefinition

    segel_bergen: qm
      methodenDefinition
```

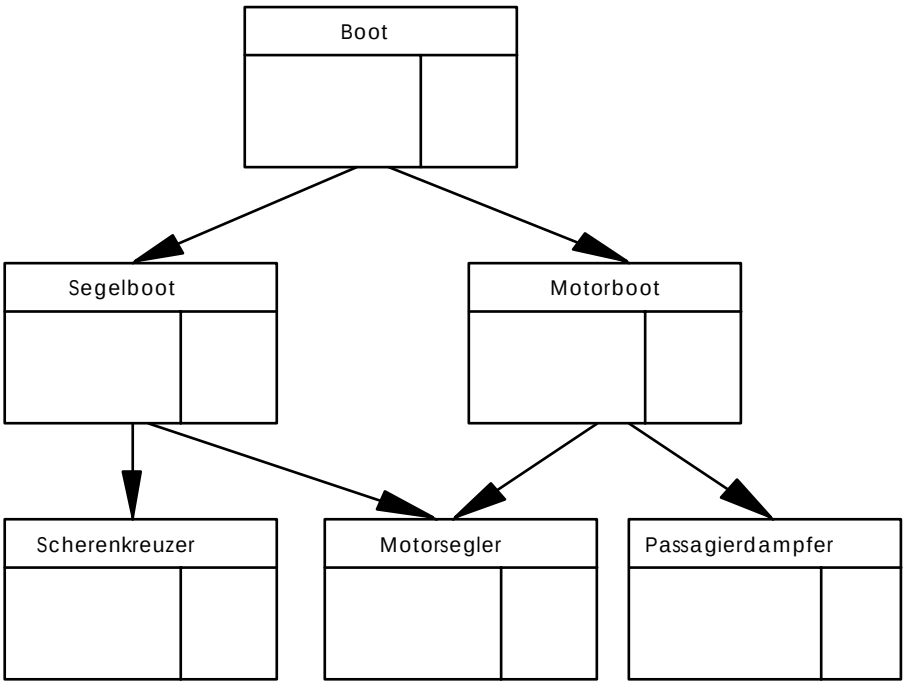
Vererbung ist **das** Konzept der objektorientierten Programmierung und ist deshalb in allen objektorientierten Sprachen enthalten. Sein Hauptzweck liegt darin, die Ähnlichkeit von neu zu schaffenden Klassen mit bereits vorhandenen Klassen auszunutzen, und zwar entweder im Sinne von Spezialisierungen oder von Abstraktion. Man unterscheidet zwei Varianten:

- **Einfachvererbung**
- **Mehrfachvererbung**

Die **einfache Vererbung** liegt dann vor, wenn jede Klasse einer Vererbungshierarchie, die nicht oberste Klasse ist, genau eine direkte Oberklasse besitzt (Baumstruktur).



Bei **mehrfacher Vererbung** hat mindestens eine Klasse mehr als eine direkte Oberklasse. Durch diese Eigenschaft ist es möglich, voneinander unabhängige Klassenhierarchien zusammenzuführen.



Bei der Mehrfachvererbung können **Konflikte** auftreten, in der Art, dass Methoden mit dem gleichen Namen in beiden Oberklassen definiert wurden (z.B. bremsen). Abhilfe schafft da nur die eigene Realisierung dieser Methode innerhalb der Klasse Motorsegler, welche dann die in beiden Oberklassen definierte Methode gleichen Namens überlagert.
Auch Eigenschaften können mehrfach definiert sein. Solche Konflikte müssen i.R. vom Programmierer ausgeschlossen werden.

In C++ ist Mehrfachvererbung erlaubt in Pascal und Smalltalk nicht.

Die Ableitung einer Klasse mir Mehrfachvererbung wird in C++ zum Unterschied zur Einfachvererbung wie folgt beschrieben:

- `class unterklasse: oberklasse {...};`
  `// einfache Vererbung`
- `class unterklasse: oberklasse1, oberklasse2,... {...};`
  `// mehrfache Vererbung`

1.2.4 Polymorphismus

Unter Polymorphismus bzw. dynamisches Binden versteht man die Fähigkeit einer Instanz, auf Objekte unterschiedlicher Klassen einer Klassenhierarchie zurückzugreifen.
Wenn eine Instanz(Variable) auf eine Oberklasse zeigt, wird erst zur Laufzeit entschieden, auf welches Objekt seiner Unterklassen die Instanz tatsächlich zurückgreift (**late binding** - späte Bindung). Damit kann in der Programmierung ein höheres Abstraktionsniveau erreicht werden, denn bei einem Methodenaufruf wird für das der Instanz zugeordnete Objekt erst zur Laufzeit der richtige Methodenrumpf (also die dem konkreten Objekt zugehörige Methode) hinzugebunden.
Zu diesem Zweck werden oft **abstrakte Klassen** eingeführt, von denen jedoch keine Instanzen gebildet werden. Diese Klassen der obersten Ebene(Wurzel des Hierarchiebaumes) definieren Eigenschaften oder/und Methoden, die von allen Unterklassen ererbt werden.