Die Grundidee der Objektorientierung

- die prozedurale Programmierung orientiert sich an der Arbeitsweise von Computern
  - ein Computer kann im Prinzip nichts anderes als endliche Folgen von Maschinenbefehlen (= Programme) der Reihe nach mit großer Geschwindigkeit ausführen
  - in welcher Reihenfolge welche Befehle auszuführen sind, muss ihm jeweils mitgeteilt werden:

- wir haben daher gelernt, Probleme durch Algorithmen zu lösen
  - ein Algorithmus besteht aus einer Folge von Handlungsanweisungen, nach deren Ausführung ein Problem gelöst ist
- prozedurale Programmiersprachen wie C oder Pascal sind exakt darauf abgestimmt
- sie gestatten es, einfache Algorithmen nahezu 1:1 in Programmcode zu übertragen
  - ein Algorithmus ist genau dann einfach, wenn er mit Hilfe von Struktogrammen beschrieben werden kann
- Compiler überführen den Programmcode in ausführbare Folgen von Maschinenbefehlen

- im Verlauf der Zeit stieß dieses – seit Beginn der Programmierung übliche – Vorgehen an Grenzen
  - in der Praxis wurden immer komplexere Probleme angegangen
  - dabei wurde es immer unnatürlicher und schwieriger, sie in Form von Algorithmen zu lösen
- allmählich entstand die Idee, die Lösung von Problemen eher daran auszurichten, wie Menschen vorgehen
  - und nicht an der Funktionsweise von Computern

- andererseits müssen die Lösungen immer noch so gestaltet sein, dass Compiler in der Lage sind, sie in ausführbaren Maschinencode zu überführen
  - dazu wurden immer leistungsfähigere Compiler entwickelt
- die Umsetzung der Idee hat zu dem geführt, was heute objektorientierte Programmierung (OOP) genannt wird
  - von den Anfängen der OOP (1962) bis zum Durchbruch in der Praxis (etwa ab 1990) sind allerdings nahezu 30 Jahre vergangen
  - in der Anwendungsprogrammierung ist die OOP (bei Neuentwicklungen) inzwischen Standard

- worin besteht die Grundidee der Objektorientierung?
- nehmen wir an, ein Mensch will (als komplexes Problem) ein modernes Einfamilienhaus bauen
- dann wird er die Dienste vieler Experten in Anspruch nehmen
  - das können zum Beispiel sein
    - Finanzberater
    - Notar
    - Architekt
    - Baggerfahrer

    - Betonbauer
    - Maurer
    - Dachdecker
    - Elektriker
    - Klempner
    - Innenarchitekt
    - Tischler
    - Fußbodenverleger
    - Maler
  - und viele andere mehr
- der Mensch wird die Arbeit der Experten lediglich koordinieren (als Projektleiter)

- hat die Koordination geklappt und jeder Experte seine Aufgabe korrekt erledigt, wird das Einfamilienhaus fertig sein
  - wodurch das komplexe Problem gelöst ist
- zur Lösung des Problems macht der Mensch also nicht jeden Arbeitsschritt selbst, sondern er nutzt die Fähigkeit bereits vorhandener Experten (Wiederverwendung)
  - er selbst muss deren Fähigkeiten nicht haben
  - er braucht auch nicht zu wissen, wie die Experten ihre Arbeit erledigen

- für ihn ist nur wichtig, dass die Experten die vereinbarten Leistungen erbringen

- was zeichnet Experten aus?
  - sie haben jeweils spezielle Fähigkeiten, die sie zur Nutzung zur Verfügung stellen
    - manche nur wenige
    - andere viele
  - auch sie können (wie der Projektleiter) zur Erledigung von Teilaufgaben andere Experten beauftragen und koordinieren
    - der Architekt technische Zeichner
    - der Notar Schreibkräfte

    - der Dachdecker Schreiner und Blechschlosser
    - die Maurer Bauhilfsarbeiter
    - usw.
- manche Experten nutzen Maschinen
  - der Baggerfahrer einen Bagger
  - die Maurer eine Mischmaschine
  - usw.
- lösen wir uns von der Vorstellung, dass Experten Menschen sein müssen, können wir allgemeiner auch Maschinen als Experten bezeichnen
  - denn auch sie stellen das, was sie können, zur Nutzung zur Verfügung

- Maschinen bestehen aus Teilen
  - sind diese komplex, können sie selbst als (Teil-)Maschinen, angesehen werden, also auch als Experten
- reduzieren wir die menschlichen Experten (durch Abstraktion) auf die Dinge, die zur Nutzung ihrer Fähigkeiten erforderlich sind, bestehen auch sie aus Teilen:
  - Name
  - Adresse
  - Telefon
  - usw.

- (abstrakte) Experten haben also folgende charakteristische Eigenschaften:
  - sie setzen sich aus (endlich vielen) Teilen zusammen
  - sie besitzen Fähigkeiten, die sie zur Nutzung zur Verfügung stellen
  - man kann sie unterscheiden
    - wenn sie sich aus unterschiedlichen Teilen zusammensetzen oder unterschiedliche Fähigkeiten haben, sowieso
    - aber auch dann, wenn sie sich aus gleichen Teilen zusammensetzen und gleiche Fähigkeiten haben
      - bei Menschen zum Beispiel anhand des Namens
      - bei Maschinen zum Beispiel anhand einer Fertigungsnummer

- fassen wir zusammen:
  - der Mensch löst das komplexe Problem (Bau eines modernen Einfamilienhauses), indem er die Fähigkeiten bereits vorhandener Experten (Menschen und Maschinen) koordiniert und nutzt
    - wobei manche der Experten die Fähigkeiten anderer Experten (Menschen und Maschinen) koordinieren und nutzen, usw.
  - die eingesetzten Experten (Menschen und Maschinen) haben jeweils eine Identität, setzen sich aus Teilen zusammen und stellen ihre Fähigkeiten zur Nutzung zur Verfügung
    - dabei ist nicht wichtig, wie sie etwas machen, sondern nur, was sie können

- letztendlich wird das komplexe Problem durch das (komplexe) Zusammenwirken (vieler) derartiger Experten gelöst
  - und nicht (algorithmisch) durch umfassende Planung jedes einzelnen Arbeitsschrittes
- die Grundidee der Objektorientierung besteht darin, auch in der Programmierung so vorzugehen
  - dabei entsprechen den Experten Objekte

- **Erkenntnis:** in der objektorientierten Programmierung werden Probleme durch das Zusammenwirken von Objekten gelöst, wobei die Objekte
  - eine Gestalt haben, d.h. sich aus Teilen zusammensetzen
    - die Teile eines Objektes, aus denen es sich zusammensetzt, werden Attribute (des Objektes) genannt
  - ein Verhalten haben, d.h. Fähigkeiten, die sie zur Nutzung anbieten
    - die Fähigkeiten eines Objektes werden durch Methoden (des Objektes) beschrieben
  - eine Identität haben

- wie kann die Idee mit den Mitteln einer formalen Programmiersprache umgesetzt werden?
  - woher kommen die Objekte?
  - wie können sie zusammenwirken?
- im Detail fallen die Antworten auf die Fragen für jede (objektorientierte) Programmiersprache anders aus
  - was sich oberflächlich an der unterschiedlichen Syntax der Sprachen bemerkbar macht

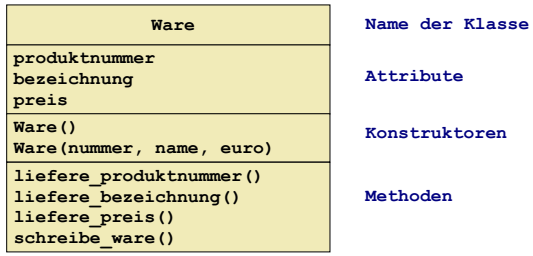
- allgemein lässt sich jedoch sagen, dass objektorientierte Programmiersprachen
  - einen Mechanismus zum Erzeugen von Objekten haben
    - dabei werden so genannte Konstruktoren eingesetzt, um sie in einen sinnvollen Initialzustand zu bringen
      - vergleichbar mit der Werkseinstellung von Apparaten
  - es ermöglichen, die zur Verfügung gestellten Fähigkeiten existierender Objekte jederzeit zu nutzen
    - indem Objekte beauftragt werden, ihre Methoden auszuführen („durch Senden einer Nachricht“)
- wie in der realen Welt, reichen diese beiden Mechanismen aus, um Probleme zu lösen

- zum Lösen komplexer Probleme müssen in der Regel mehrere, unterschiedliche Objekte zusammenwirken
  - wie kann der Mechanismus zum Erzeugen von Objekten unterschiedliche Objekte erzeugen?
- dieses Problem kann wie in prozeduralen Sprachen gelöst werden:
  - jedes Objekt hat einen Datentyp
  - beim Erzeugen eines Objektes ist dessen Typ anzugeben
    - „erzeuge ein Objekt vom Typ `T`“

- die Datentypen von Objekten müssen allerdings komplexer sein als in prozeduralen Sprachen
  - denn Objekte haben ja nicht nur eine Gestalt, sondern auch ein Verhalten
    - dass Objekte unterschieden werden können, d.h. eine Identität haben, ergibt sich dadurch, dass jedes Objekt beim Erzeugen separat gespeichert wird
- nötig ist eine Art Bauplan, in dem festgelegt ist, welche Gestalt und welches Verhalten Objekte haben, die nach diesem Plan „gebaut“ (d.h. erzeugt) werden

- ein derartiger Bauplan, d.h. ein Datentyp von Objekten, wird Klasse genannt
- eine Klasse muss (mindestens) enthalten
  - Angaben zu den Attributen, d.h. zu den Teilen, aus denen sich die Objekte zusammensetzen, die nach diesem Plan gebaut werden
  - Angaben zu Konstruktoren, die beim Erzeugen der Objekte benutzt werden, um sie in einen sinnvollen Initialzustand zu bringen
    - es kann keinen, einen oder mehrere geben (Varianten)
  - Angaben zu den Methoden, d.h. zu den Fähigkeiten, die die Objekte haben, die nach diesem Plan gebaut werden

- Klassen veranschaulichen wir uns so:



Hinweis: seit 1997 gibt es mit der Unified Modeling Language (UML) einen Standard zur grafischen Beschreibung (nicht nur) objektorientierter Softwaresysteme

unsere vereinfachte Darstellung entspricht dem nur bedingt

- ist `T` eine Klasse, dann nennt man Objekte vom Typ `T` auch Instanzen der Klasse `T`
- Instanzen der Klasse `Ware` bestehen also aus drei Teilen (den Attributen `produktnummer`, `bezeichnung` und `preis`) und haben die Fähigkeit
  - ihre Produktnummer zu liefern
  - ihre (Waren-)Bezeichnung zu liefern
  - ihren Preis zu liefern
  - sich selbst (auf den Bildschirm) zu schreiben

- zu jeder Klasse, d.h. nach jedem Bauplan für Objekte, können beliebig viele Objekte, d.h. Instanzen, erzeugt werden
  - insbesondere können einmal erstellte Baupläne, d.h. Klassen, in beliebig vielen Programmen zum Erzeugen von Instanzen herangezogen werden
- dieser Aspekt der Wiederverwendung von Klassen ist einer der wichtigsten Gründe dafür, warum sich die Objektorientierung in der Praxis durchgesetzt hat
  - einfach aus ökonomischen Gründen!

- im Zentrum der objektorientierten Programmierung steht daher die Gestaltung von Klassen, d.h. von Bauplänen, nach denen Objekte gebaut werden
- Algorithmen spielen eine untergeordnete Rolle
  - sie sind nur noch für die Implementierung von Methoden wichtig, die jeweils genau eine Fähigkeit realisieren
- das eigentliche Erzeugen der Objekte leistet der Compiler

- das Vorgehen in der objektorientierten Programmierung ist daher signifikant verschieden von dem in der prozeduralen Programmierung:
  - zu gegebenem Problem muss erkannt werden,
    - welche Experten mit welchen Fähigkeiten nötig sind, um das Problem zu lösen, d.h. welche Objekte erzeugt werden müssen
    - wie die Zusammenarbeit der Objekte zu koordinieren ist
  - wenn es bereits Baupläne, d.h. Klassen gibt, nach denen die (oder einige der) benötigten Objekte gebaut werden können, sind diese wiederzuverwenden

  - für Objekte, für die es noch keine Baupläne gibt, sind Baupläne, d.h. Klassen, zu erstellen
    - sie sollten so allgemein wie möglich sein, damit die nach ihnen gebauten Objekte nicht nur zur Lösung des gegebenen Problems beitragen, sondern auch in anderen Programmen wiederverwendet werden können
- objektorientierte Programmierung hat demnach
  - eine planerische Komponente
    - welche Experten (= Objekte) brauche ich?
    - wie sind sie zu koordinieren?
  - und eine gestalterische Komponente
    - wie müssen die Baupläne (= Klassen) für (noch nicht vorhandene) Experten gestaltet werden?

- im Idealfall gibt es in objektorientiert geschriebenen Programmen keine anderen Daten als Objekte, d.h. Instanzen von Klassen
  - Programmiersprachen, die das erzwingen, werden rein objektorientiert genannt
    - dazu gehören Smalltalk-80, Eiffel, Ruby und JADE
- die meisten Programmiersprachen, die die objektorientierte Programmierung unterstützen, sind aber nicht so streng
  - bei ihnen kommt es auf die Programmierer und Programmiererinnen an, ob sie die Grundidee der Objektorientierung richtig umsetzen

  - Java erzwingt immerhin, dass ein Programm ausschließlich aus (der Definition von) Klassen besteht
    - die aber – aus der Sicht der Objektorientierung – falsch gestaltet sein können
  - in C++ gibt es erheblich mehr Freiheiten
    - weshalb das objektorientierte Programmieren mit C++ sehr viel Verständnis und Selbstdisziplin erfordert