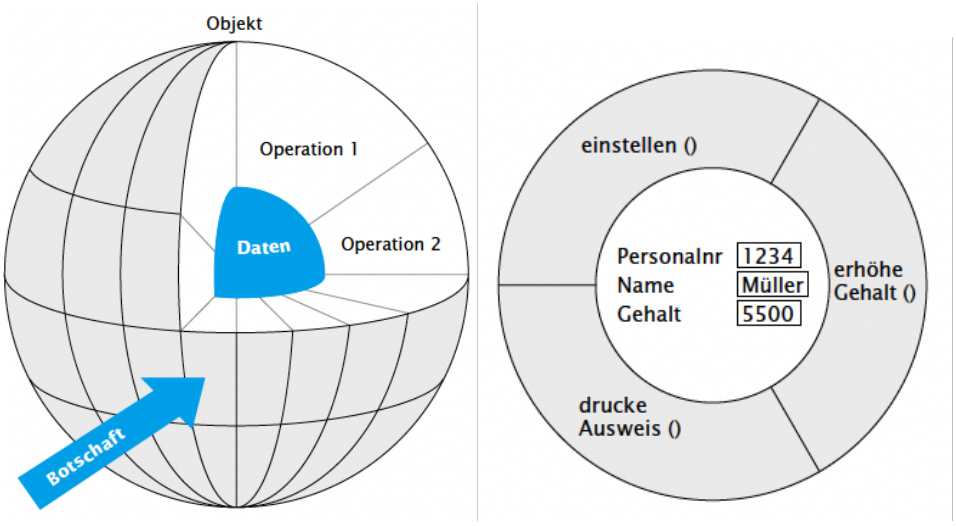
Warum Objektorientierten Programmierung?

- Modellierung der Realität durch entsprechende Objekte (Klassen).
- Möglichkeit der grafischen Darstellung von Beziehungen von Klassen untereinander (Modellierung des Problems/der Realität) mit Hilfe von UML.
- Durch das Konzept der Klassen (und Objekte) wird das Arbeiten in Teams (Kollaboratives Arbeiten) unterstützt. Dazu ist allerdings eine genaue **Schnittstellendefinition** notwendig. Diese erfolgt beispielsweise durch ein **UML-Diagramm** und Beschreibung der verwendeten Attribute und Methoden einer (Klassen-)Bibliothek. Die Schnittstellendefinition ist ein Teil der Spezifikation der Klassen und ihrer Methoden. Die Spezifikation dient damit als Beschreibung für eine meist programmiersprachenunabhängige Programmierung, die so präzise formuliert sein muss, dass sie theoretisch leicht auch von externen Programmierern ausgeführt werden kann.
- **Wartbarkeit und Erweiterbarkeit**: Die Verkapselung und das Geheimnisprinzip unterstützen die Erweiterung und Wartbarkeit der entstandenen Systeme.
- **Wiederverwendbarkeit**: Eine sorgfältig aufgebaute Vererbungshierarchie garantiert einerseits eine leichte Erweiterbarkeit und darüber hinaus ein hohes Maß an Wiederverwendbarkeit im gleichen Projekt und ggf. auch über mehrere Projekte hinweg.
- **Entwurfsmuster** (wie beispielsweise das MVC-Konzept) insbesondere eingesetzt bei ereignisorientierter Programmierung (GUI, Webdesign)) aber auch Frameworks und Klassenbibliotheken unterstützen die Lösung ähnlicher Probleme und bieten dafür eine bewährte Vorgehensweise im Entwurf (**Wiederverwendbarkeit**).
- Die objektorientierte Softwareentwicklung existiert seit ca. 1979. Sie hat sich bewährt und wird überall dort eingesetzt, wo komplexe Softwarearchitektur notwendig ist.

Geheimnisprinzip und Kapselung von Objekten

Ein wesentlicher Vorteil der OOP besteht in der Kapselung von Daten. Zugriff auf Eigenschaften (Attribute) darf normalerweise nur über Zugriffsmethoden (get- bzw set-Methoden) erfolgen. Diese Methoden können Plausibilitätstests enthalten. So kann z.B. eine Methode zum Setzen des Geburtsdatums prüfen, ob das Datum korrekt ist und sich innerhalb eines bestimmten Rahmens bewegt, z.B. Girokonto für Kinder unter 14 nicht möglich oder Kunden über 150 Jahre unwahrscheinlich.
Die Verkapselung (encapsulation) sagt aus, dass zusammengehörende Attribute und Operationen, in einer Einheit – der Klasse – verkapselt sind. Die Attribute und die Realisierung der Operationen können durchaus nach außen sichtbar sein. Beispielsweise erlaubt C++ mit seinen verschiedenen Sichtbarkeiten *public*, *protected* und *private* die Verkapselung ohne und mit Einhaltung des Geheimnisprinzips



In der Abbildung **links** realisiert beim Objekt das **Geheimnisprinzip**. Zustand und Verhalten eines Objekts bilden eine Einheit. Wir sagen auch: ein Objekt kapselt **Zustand** (Daten) und **Verhalten** (Operationen). Die Daten eines Objekts können nur mittels der Operationen (Methoden) gelesen und geändert werden (Botschaft oder Nachrichten an das Objekt). Das bedeutet, dass die Repräsentation dieser Daten nach außen verborgen sein soll. Wir sagen: ein Objekt realisiert das Geheimnisprinzip[1].

Beispiel **Datenkapselung** (Abbildung **rechts**): Ein Mitarbeiter besitzt eine Personalnummer, einen Namen und erhält ein bestimmtes Gehalt. Neue Mitarbeiter werden eingestellt, das Gehalt vorhandener Mitarbeiter kann erhöht werden und es kann ein Mitarbeiterausweis gedruckt werden. Wie die Abbildung rechts zeigt, werden die Attribute durch die Operationen vor der Außenwelt verborgen.

Die Idee dabei ist, eine Art Schale um das Objekt zu legen, so dass dieses vor direkten Zugriffen geschützt ist. Viele Programmiersprachen stellen dafür verschiedene Sichtbarkeiten für Attribute und Methoden bereit. Diese Sichtbarkeiten (meist public, protected oder private genannt) „steuern“ den Zugriff von außen.

[1] aus: Heide Balzert: Lehrbuch der Objektmodellierung. Analyse und Entwurf. Heidelberg, Berlin 1999. S. 18 und 20