ETH Zürich
Chair of Software Engineering

Einführung in die Programmierung

Prof. Dr. Bertrand Meyer

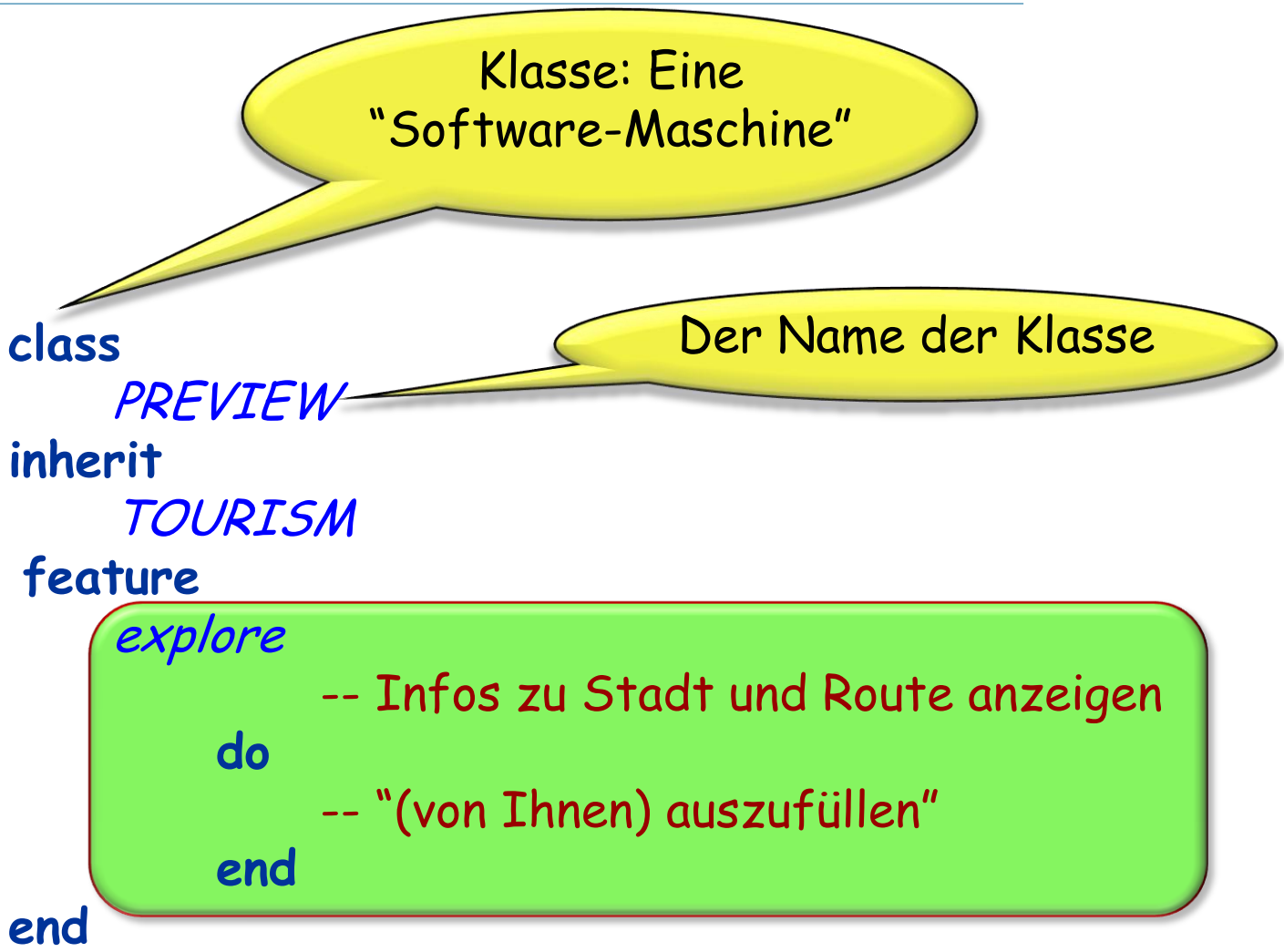
Lektion 2: Der Umgang mit Objekten I

Unser erstes Programm!

Unser Programm soll:

- Eine Karte von Paris anzeigen
- Die Position des Louvre markieren
- Die Linie 8 der Metro hervorheben
- Eine vordefinierte Route animieren

Ein Klassentext



Eine Konvention der Traffic-Bibliothek

Klassen in Traffic haben Namen der Form

TRAFFIC_ACTUAL_CLASS_NAME

In diesen Folien und im Buch lasse ich einfachheitshalber das Prefix *TRAFFIC_* weg.

Aber: Sie brauchen es, um die Klassen in der Software zu finden!

Noch eine Konvention

Verwenden Sie für zusammengesetzte Namen "_"

TRAFFIC_STATION
Station_Paradeplatz -- oder: *Station_Parade_Platz*

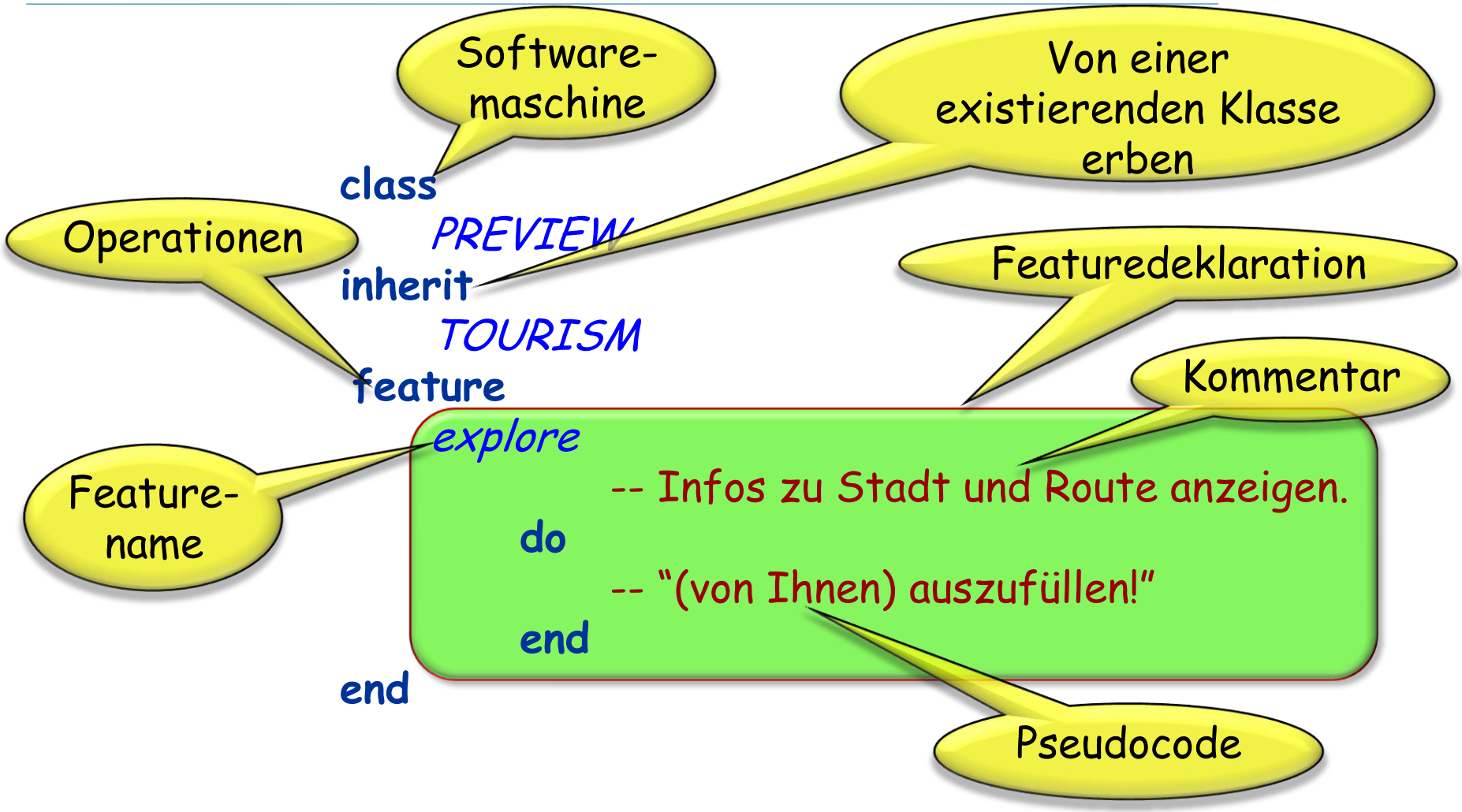
Wir verwenden nicht den "CamelCase" Stil:

AShortButHardToDeCipherName

sondern Unterstriche (Manchmal auch "Pascal_case" genannt):

A_significantly_longer_but_still_perfectly_clear_name

Ein Klassentext



Schlüsselwörter (keywords) (**class**, **inherit**, **feature**, **do**, **end**) haben eine spezielle Rolle.

Zauberei?

Die Klasse *TOURISM* ist ein Teil der unterstützenden Software.

Sie unterstützt Sie durch vordefinierte Funktionalität („Zauberei“)

Der Anteil an Zauberei wird Stück für Stück abnehmen und schlussendlich ganz verschwunden sein.

Den Featurerumpf ausfüllen

```
class
    PREVIEW
inherit
    TOURISM
feature
    explore
            -- Infos zu Stadt und Route anzeigen
        do
            Paris.display
            Louvre.spotlight
            Line8.highlight
            Route1.animate
        end
end
```

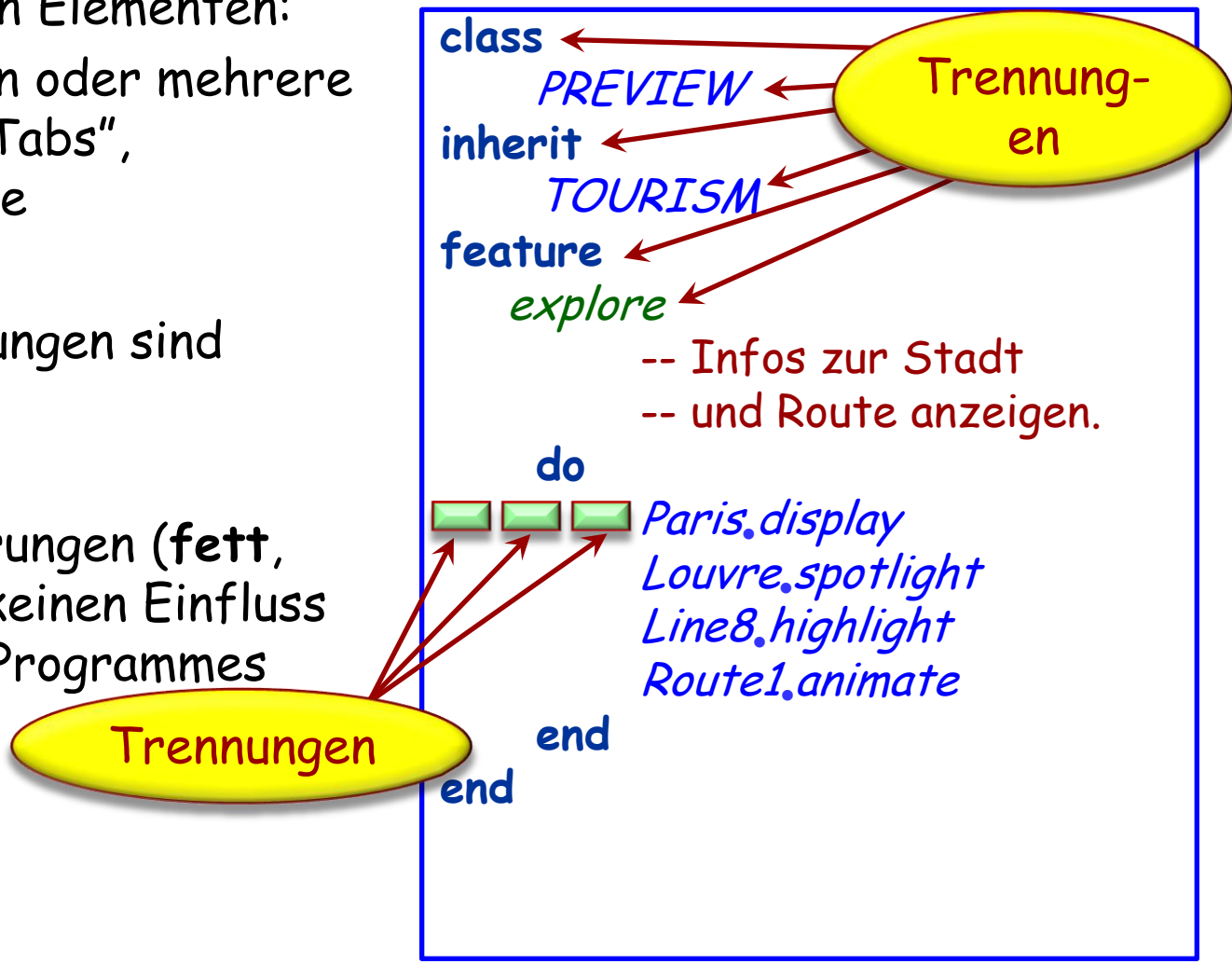
Formatierung des Programmtextes

zwischen angrenzenden Elementen:

Trennungen: ein oder mehrere Leerschläge, "Tabs", Zeilenumbrüche

Alle Arten von Trennungen sind äquivalent

Typographische Änderungen (**fett**, *kursiv*, farbig) haben keinen Einfluss auf die Semantik des Programmes



Stilregel

Verwenden Sie Tabs, um den Code einzurücken, nicht Leerschläge.

Nützen Sie Einrückungen, um die **Struktur** des Programmes hervorzuheben.


```
class
        PREVIEW
inherit
        TOURISM
feature
    explore
                -- Infos zur Stadt
                -- und Route anzeigen.
        do
                Paris.display
                Louvre.spotlight
                Line8.highlight
                Route1.animate
        end
end
```

Vordefinierte Objekte

Paris, *Louvre*, *Line8*, und *Route1* sind Namen vordefinierter Objekte .

Die Objekte sind in der Klasse *TOURISM* , der Elternklasse von *PREVIEW* , definiert.

display, *spotlight*, *highlight* und *animate* sind Features obiger Objekte, die man auf sie aufrufen kann.

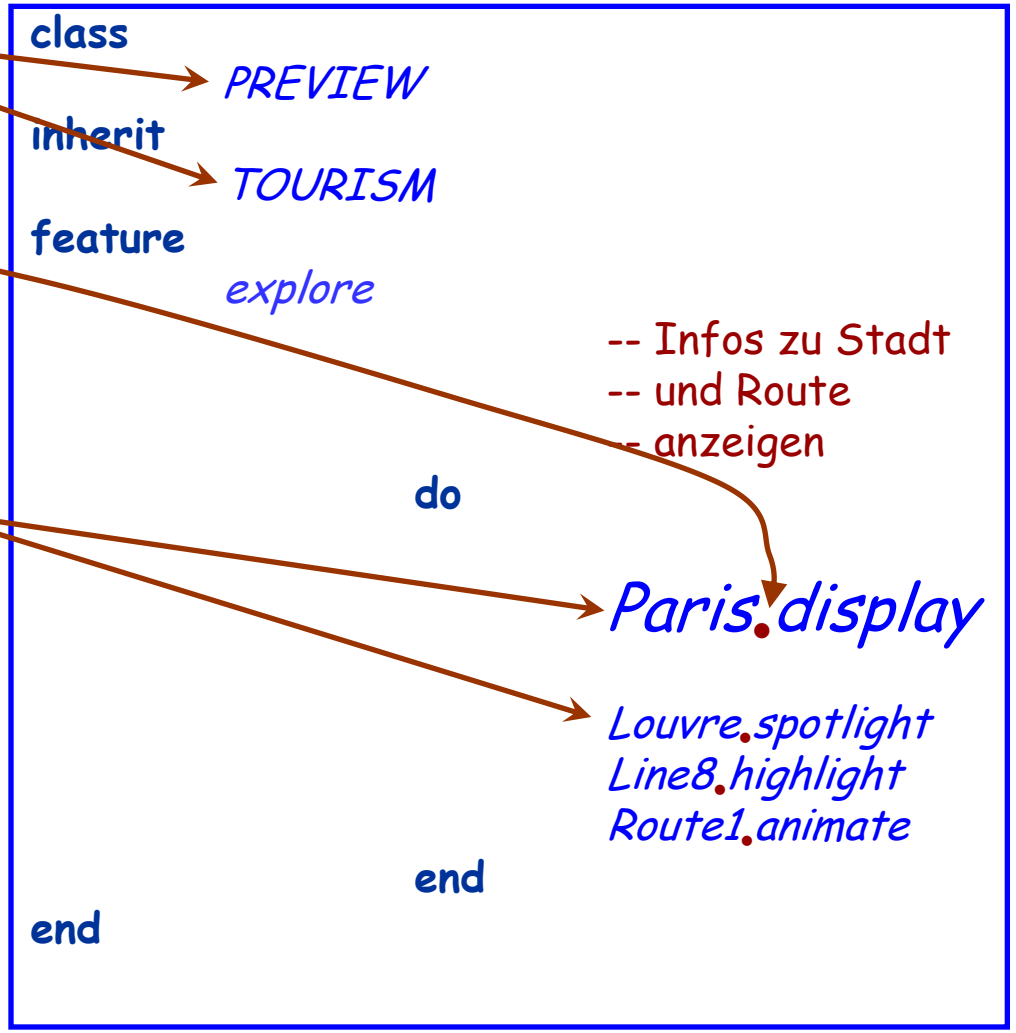
Mehr Stilregeln

Klassennamen: GROSS

Punkt des Featureaufrufs: Kein Leerschlag, weder davor noch danach.

Namen vordefinierter Objekte beginnen mit einem Grossbuchstaben.

Neue Namen (für Objekte, die Sie definieren) sind kleingeschrieben.



Objekttechnologie (object technology)

Wir arbeiten mit Objekten.

Unser Programmierstil: Objektorientierte Programmierung
Abkürzung: O-O

Allgemeiner "Objekttechnologie": Beinhaltet O-O *Datenbanken*, O-O *Analyse*, O-O *Design*...

Die Ausführung der Software besteht aus Operationen auf Objekten: feature-Aufrufen

```
your_object.your_feature
```

Eine eigene Ausdrucksweise

```
Paris.display


nächste_nachricht.send      -- next_message.send
computer.auschalten         -- computer.shut_down
telefon.läuten              -- telephone.ring
```

Objekt-Orienterte Programmierung hat einen bezeichnenden Stil.

Jede Operation wird auf **ein** Objekt (das "Ziel" (*target*) des Aufrufs) angewendet.

Was ist ein Objekt?

Softwarebegriff: Eine Maschine, definiert durch auf sie anwendbare Operationen.

Drei Arten von Objekten:

- "**Physikalische Objekte**": widerspiegeln materielle Objekte der modellierten Welt.

  Beispiele: das Louvre, Paris, eine Bahn der Metro…

- "**Abstrakte Objekte**": abstrakte Begriffe aus der modellierten Welt.

  Beispiele: eine (Metro-) Linie, eine Route…

- "**Softwareobjekte**": ein reiner Softwarebegriff.

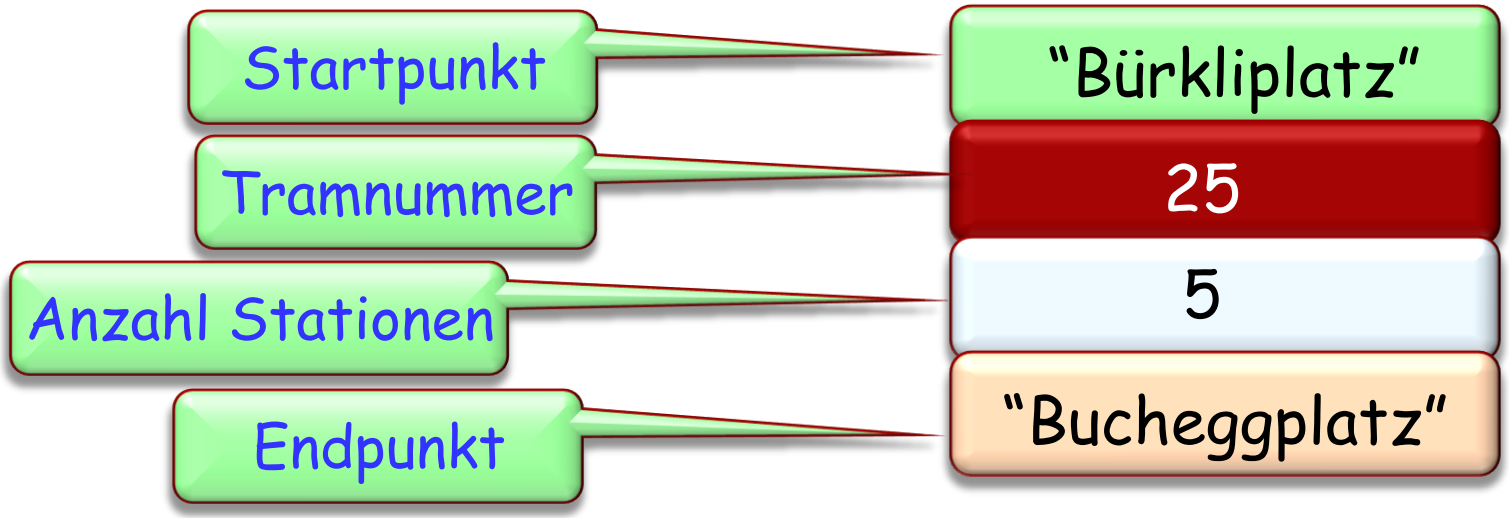
  Beispiele: "Datenstrukturen" wie Arrays oder Listen

Ein grosser Reiz der Objekttechnologie ist ihr *Modellierungsvermögen*: Verbinden von Softwareobjekten mit Objekten des Modells.

Achtung: Verbinden, nicht verwechseln!

In diesem Kurs bezieht sich "*Objekt*" auf ein Softwareobjekt

Zwei Auffassungen von Objekten



Zwei Gesichtspunkte:

- 1. Ein Objekt hat Daten, abgelegt im Speicher.
- 2. Ein Objekt ist eine Maschine, die Operationen anbietet (**Features**)

Die Verbindung:

- Die Operationen (2), die die Maschine anbietet, greifen auf die Daten (1) des Objektes zu und verändern sie.

Features: Befehle und Abfragen

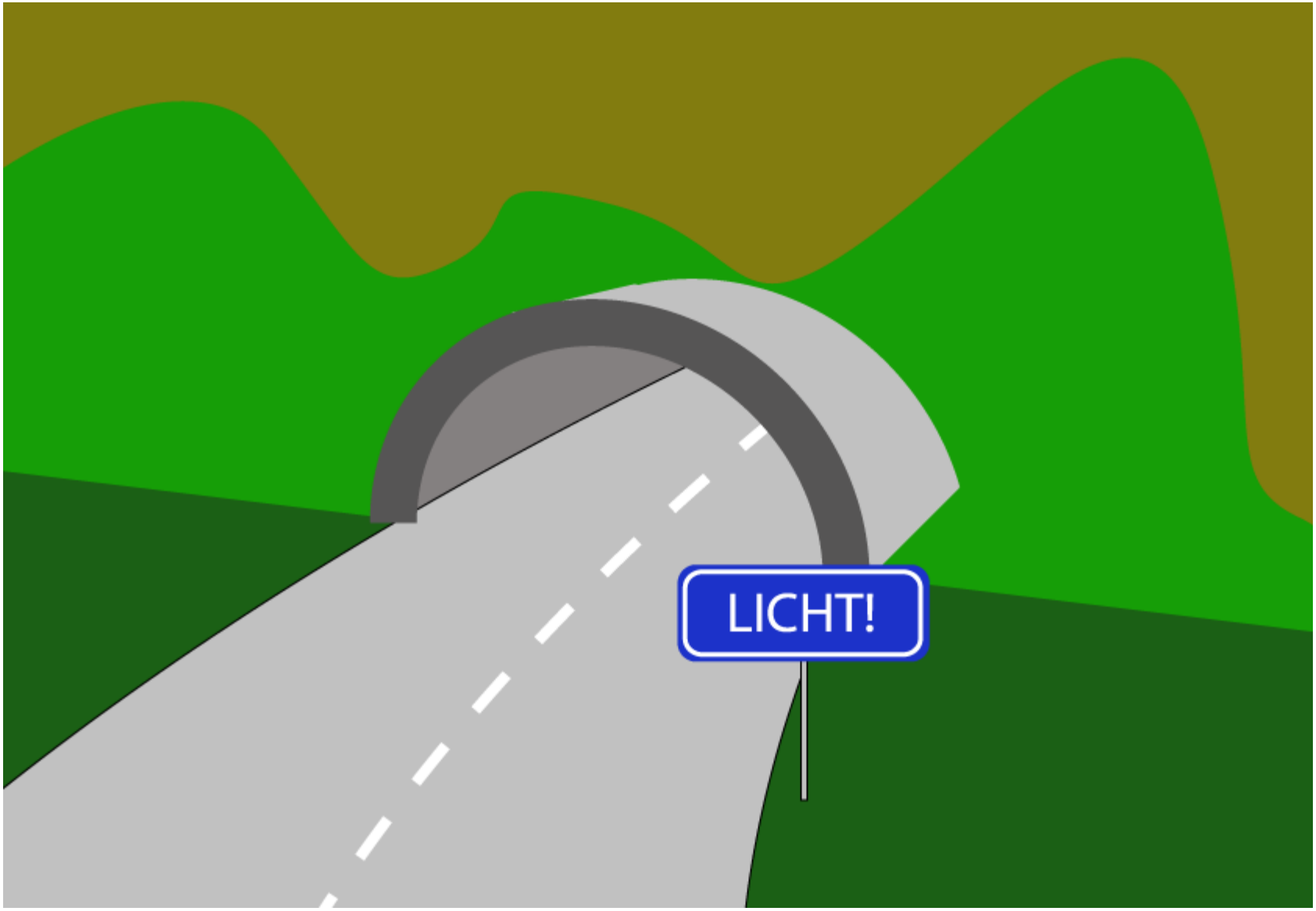
Feature: Eine Operation, die von gewissen Klassen zur Verfügung gestellt wird.

3 Arten:

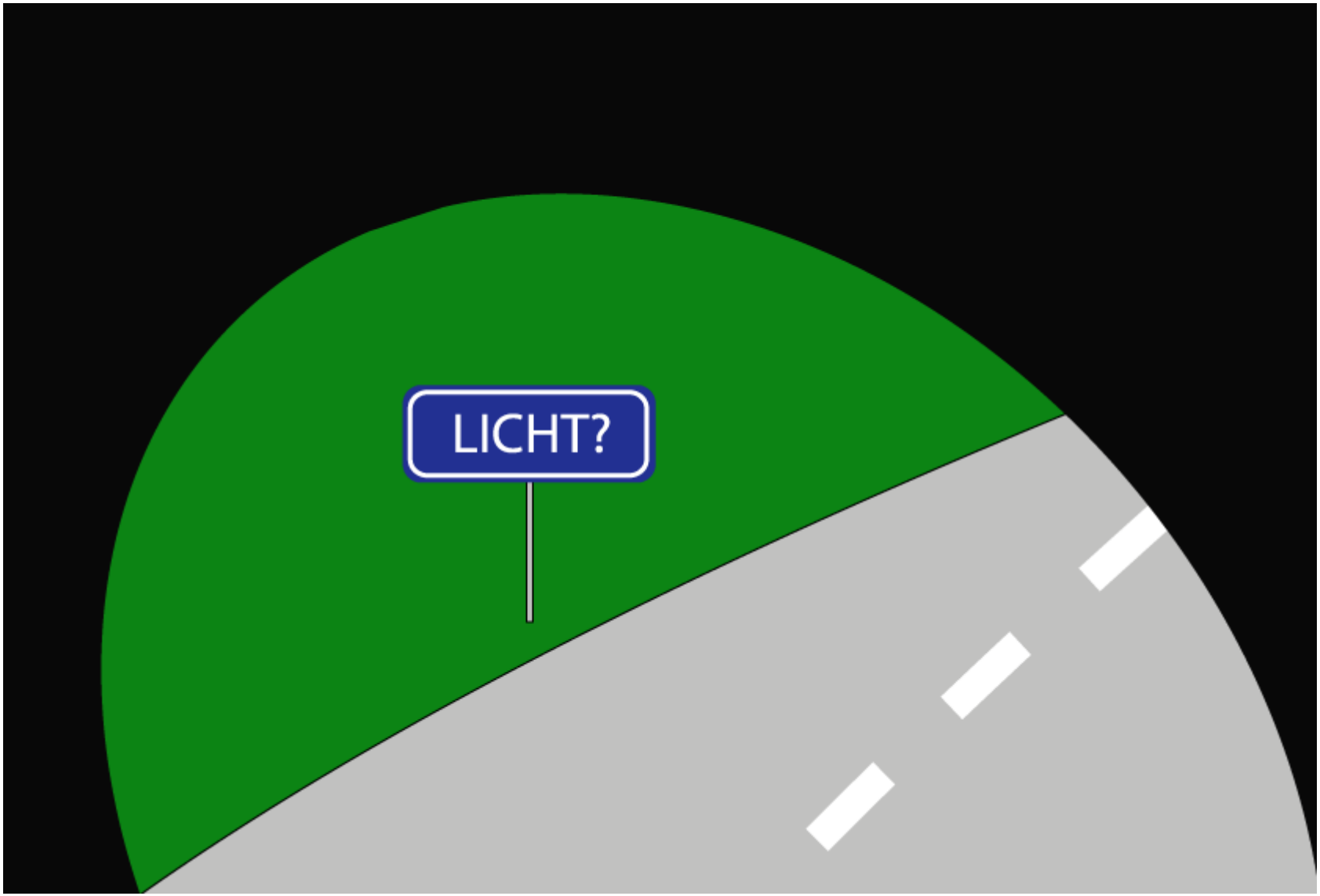
- Befehl (command)
- Abfrage (query)

- Erzeugungsprozedur (*creation procedure*) (später…)

Ein Befehl



Eine Abfrage



Abfragen

Ziel: die Eigenschaften eines Objekts zu erhalten.

Sollte *weder das Zielobjekt noch andere Objekte ändern!*

Beispiele anhand eines "route" Objektes:

- Was ist der Ursprung (die erste Station) von Route1?
- Welches ist der Endpunkt von Route1?
- Wieviele Stationen hat Route1?
- Welche Stationen besucht Route1?

Befehle

Ziel: Ein oder mehrere Objekte zu verändern.

Beispiele anhand eines "route" Objektes:

- Animiere Route1
- Füge eine neue Station zu Route1 hinzu,
  am Anfang ("prepend") bzw. am Ende ("append")

Das Befehl-Abfrage-Separationsprinzip (*)

(*) engl.: Command-Query Separation principle

Ein Objekt ist eine Maschine

Ein laufendes Programm ist eine Maschine.

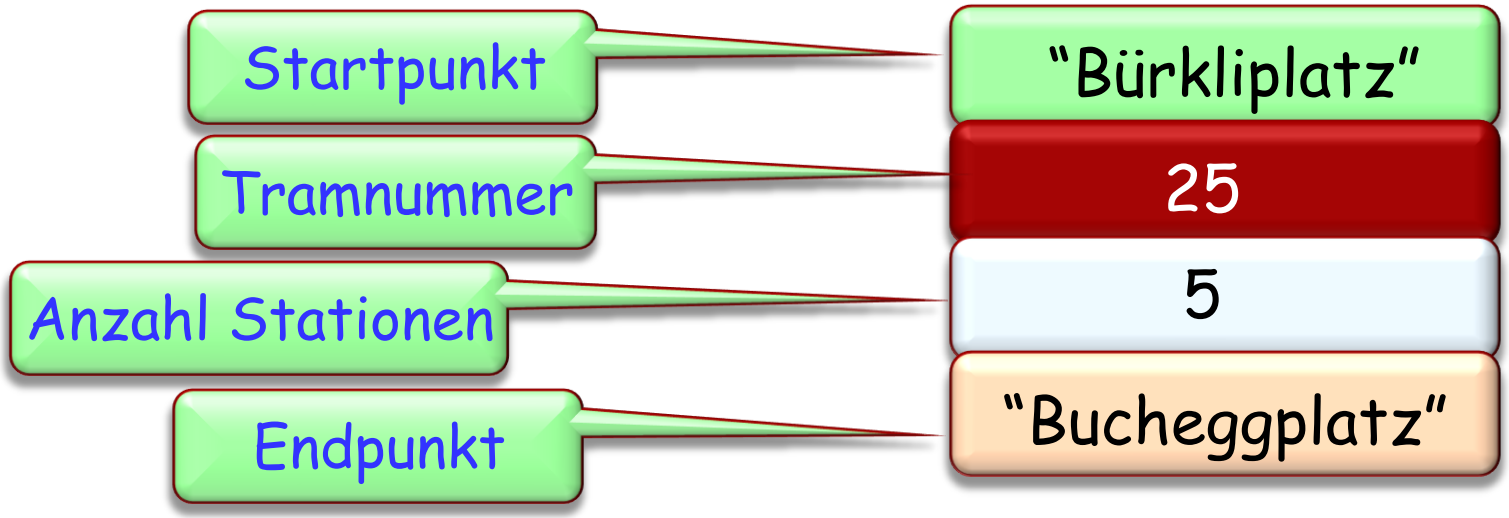
Es besteht aus kleineren Maschinen: Objekten

Während einer Programmausführung können sehr viele Objekte zum Einsatz kommen (auch mehrere Millionen!)

Ein Objekt ist eine Maschine

Eine Maschine, Hardware oder Software, ist charakterisiert durch die Operationen ("Features"), die ein Benutzer auf sie anwenden kann.

Zwei Auffassungen von Objekten



Zwei Gesichtspunkte:

- 1. Ein Objekt hat Daten, abgelegt im Speicher.
- 2. Ein Objekt ist eine Maschine, die Operationen anbietet (**Features**: Befehle und Abfragen)

Die Verbindung:

- Die Operationen (2), die die Maschine anbietet, greifen auf die Daten (1) des Objektes zu und verändern sie.

Objekte: eine Definition

Ein Objekt ist eine Softwaremaschine, die es Programmen erlaubt, auf eine Ansammlung von Daten zuzugreifen und diese zu verändern.

Definition und Klassifizierung von Features

Ein Feature ist eine Operation, die Programme auf bestimmte Arten von Objekten aufrufen können.

- Ein feature, welches (nur) auf ein Objekt zugreift, ist eine Abfrage.

- Ein feature, welches ein Objekt modifizieren kann, ist ein Befehl.

Der Gebrauch von Abfragen

Abfragen sind genauso wichtig wie Befehle!

Abfragen "machen" nichts, aber sie geben einen Wert zurück. So gibt z.B. *Route1.origin* die Startstation von *Route1* zurück.

Sie dürfen mit den Rückgabewerten von Abfragen arbeiten, z.B. die Startstation bestimmen und anschliessend auf dem Bildschirm hervorheben.

Features können Argumente (arguments) haben…

Aufgabe:

- Geben Sie die Startstation von *Route1* auf dem "Konsolenfenster" aus.

Sie brauchen:

- Das vordefinierte Objekt *Console*
- Das auf *Console* aufrufbare Feature *show*
- Das Objekt *Route1*
- Das auf *Route1* aufrufbare Feature *origin*, welches die Startstation zurückgibt

```
Console.show (Route1 .origin)
```

Den Featurerumpf ausbauen

```
class PREVIEW
    inherit TOURISM
feature
    explore
            -- Infos zu Stadt und Route sowohl den
            -- Ursprung der Route anzeigen.
        do
            Paris.display
            Louvre.spotlight
            Line8.highlight
            Console.show (Route1.origin)

        end
end
```

Features mit Argumenten

```
your_object.your_feature (some_argument)
```

some_argument ist ein Wert, welcher *your_feature* braucht.

Beispiel: Feature *show* muss wissen, was es anzeigen soll.

Es ist das gleiche Konzept wie Argumente in der Mathematik:

cos (x)

Features können mehrere Argumente haben:

x.f (a, b, c, d) -- Getrennt durch Kommas

In gut geschriebener O-O software haben die meisten features gar kein oder 1 Argument.

Eine eigene Ausdrucksweise

Paris.display

next_message.send
computer.shut_down
telephone.ring

Jede Operation wird auf ein Objekt angewendet.

Eine eigene Ausdrucksweise

```
Paris.display

next_message.send_to (recipient)
computer.shut_down_after (3)
telephone.ring_several (10, Loud)
```

Jede Operation wird auf ein Objekt angewendet
und kann Argumente benötigen.

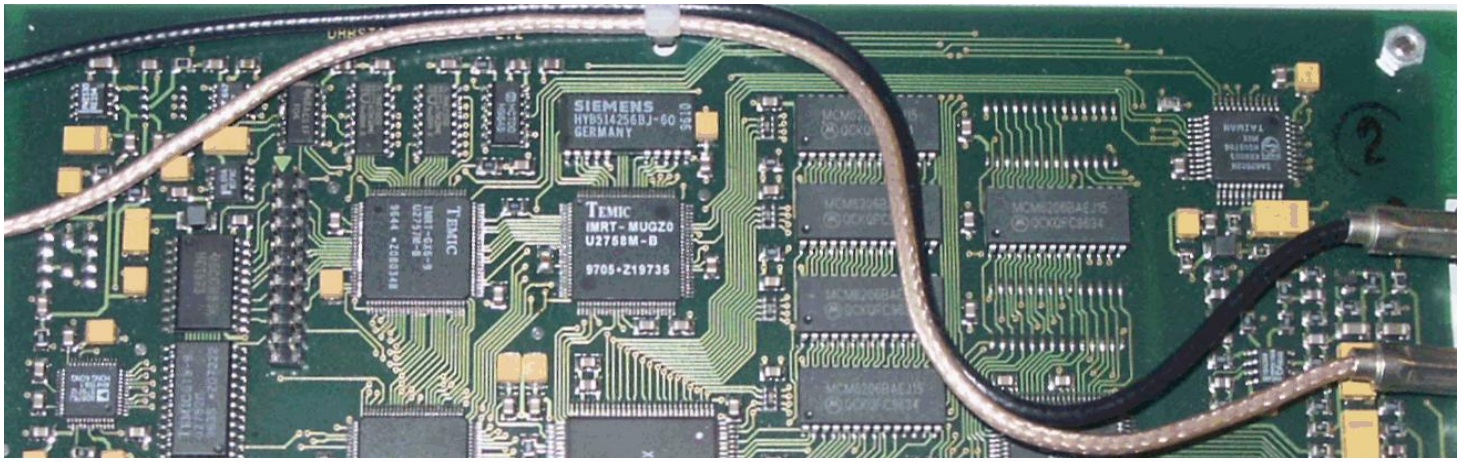
Skalierbarkeit

Eine der schwierigesten Aufgaben im Lernen von Software ist das Finden von guten Lösungen, die sowohl im Kleinen als auch im Grossen gut funktionieren.

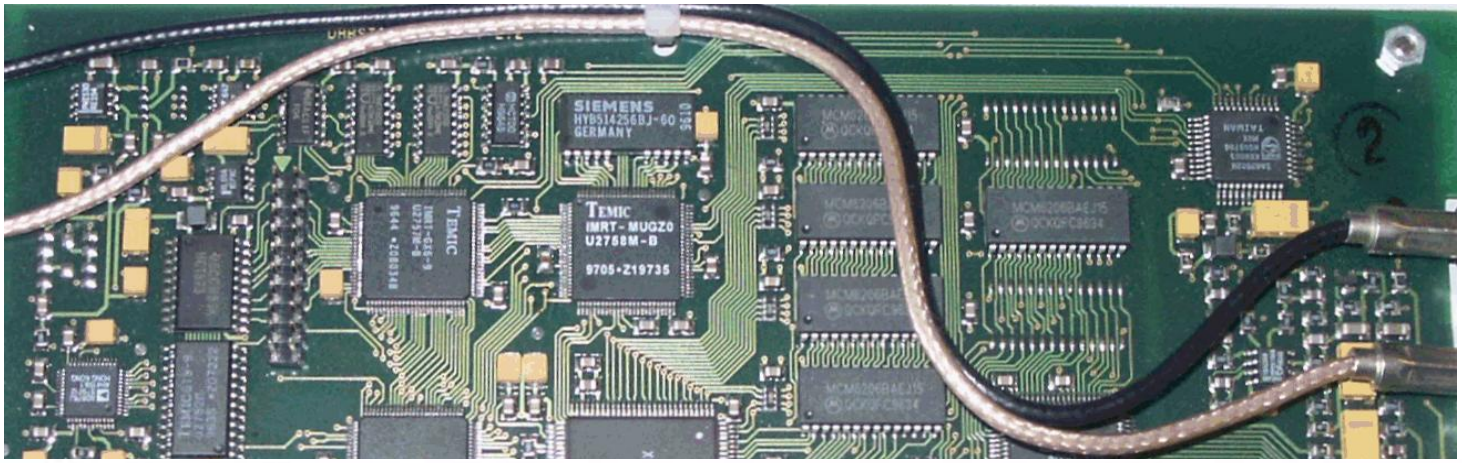
Genau das ist das Ziel für die Techniken, die wir in diesem Kurs lehren.

Ein Objekt hat eine **Schnittstelle (interface)**

Ein Objekt hat eine Implementation



Das Geheimnisprinzip (Information Hiding)



Das Geheimnisprinzip (Information Hiding)

Der Designer jedes Moduls muss spezifizieren, welche Eigenschaften für Clients abrufar sind (**öffentlich**) und welche intern (**geheim**) sind.

Die Programmiersprache muss sicherstellen, dass Kunden nur öffentliche Eigenschaften nützen können.

Was bisher geschah…

Grundkonzepte und -konstruktionen der Objekttechnologie:

- Klassen (eine erste Sicht)
- Grundstruktur von Programmtext
- Objekte
- Features
- Befehle und Abfragen
- Featureaufrufe
- Features mit Argumenten

Methodologische Prinzipien:

- Befehl-Abfrage-Separation
- Geheimnisprinzip (Information Hiding)

Bis nächste Woche

Lesen Sie Kapitel 1 und 2 von *Touch of Class*

Schauen Sie sich die Folien der nächsten zwei Vorlesungen (2 und 3) an