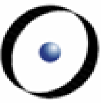
ETH Zürich
Chair of Software Engineering



Einführung in die Programmierung

Prof. Dr. Bertrand Meyer

Lektion 2: Der Umgang mit Objekten I

Unser erstes Programm!

Unser Programm soll:

- Die Position der Stationen «Central» und «Polyterrasse» auf der Karte von Zürich markieren
- Einen Wagen zur Linie 24 (Polybahn) hinzufügen
- Die Karte animieren

Ein Klassentext

Klasse: Eine "Software-Maschine"

```
class
    PREVIEW
inherit
    ZURICH_OBJECTS
feature
    explore
            -- Die Stadt erkunden.
        do
            -- "(von Uns) auszufüllen"
        end
end
```

Der Name der Klasse

Eine Konvention

Verwenden Sie für zusammengesetzte Namen "_"

ZURICH_OBJECTS
Polybahn_line_number

Wir verwenden nicht den "CamelCase" Stil:

EinKurzerAberSchwerZuLesenderName

AShortButHardToDeCipherName

sondern Unterstriche (Manchmal auch "Pascal_case" genannt):

Ein_viel_längerer_aber_immer_noch_perfekt_lesbarer_name

A_significantly_longer_but_still_perfectly_clear_name

Ein Klassentext

```
class
    PREVIEW
inherit
    ZURICH_OBJECTS
feature
    explore
            -- Die Stadt erkunden.
        do
            -- "(von Uns) auszufüllen!"
        end
end
```


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

Zauberei?

Die Klasse *ZURICH_OBJECTS* 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
    ZURICH_OBJECTS
feature
     explore
              -- Die Stadt erkunden.
        do
            Central.highlight
            Polyterrasse.highlight
            Polybahn.add_transport
            Zurich_map.animate
        end
end
```

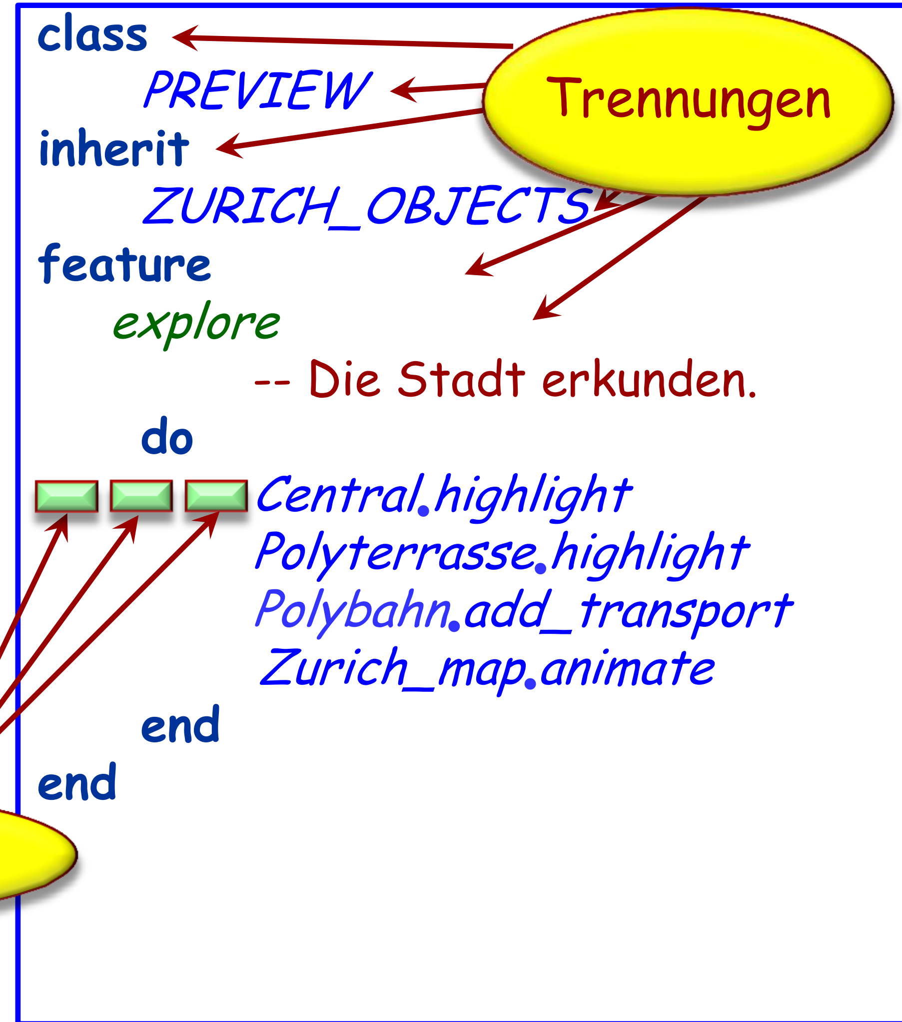
Formatierung des Programmtextes

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
	ZURICH_OBJECTS
feature
	explore
			-- Die Stadt erkunden.
		do
			Central.highlight
			Polyterrasse.highlight
			Polybahn.add_transport
			Zurich_map.animate
		end
end
```

Tabs

Vordefinierte Objekte

Central, *Polyterrasse*, *Polybahn* und *Zurich_map* sind Namen vordefinierter Objekte

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

highlight, *add_transport* 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

```
class
    PREVIEW
inherit
    ZURICH_OBJECTS
feature
    explore
            -- Die Stadt erkunden.
        do
            Central.highlight

            Polyterrasse.highlight
            Polybahn.add_transport
            Zurich_map.animate
        end
end
```

Objekttechnologie

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

ihr_objekt.ihr_feature

Eine eigene Ausdrucksweise

Zurich_map.animate

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:

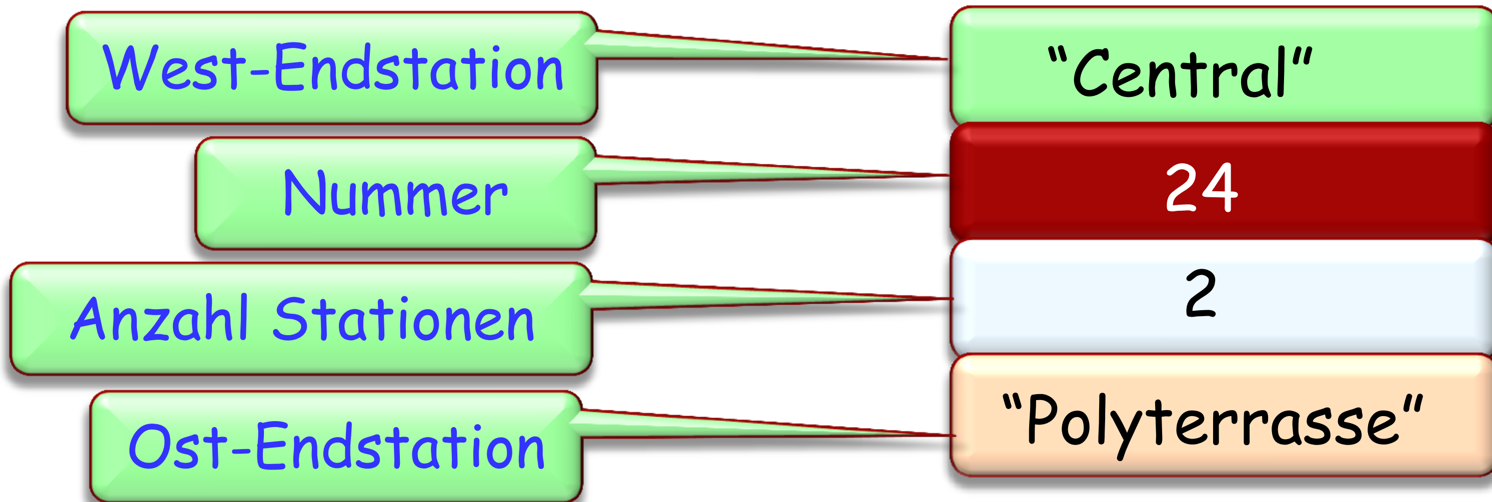
- "**Physikalische Objekte**": widerspiegeln materielle Objekte der modellierten Welt.
  Beispiele: die Polyterrasse, eine Bahn des Trams…
- "**Abstrakte Objekte**": abstrakte Begriffe aus der modellierten Welt.
  Beispiele: eine (Tram-) Linie, eine Route…
- "**Softwareobjekte**": ein reiner Softwarebegriff.
  Beispiele: "Datenstrukturen" wie Arrays oder Listen

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

Aber: 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.

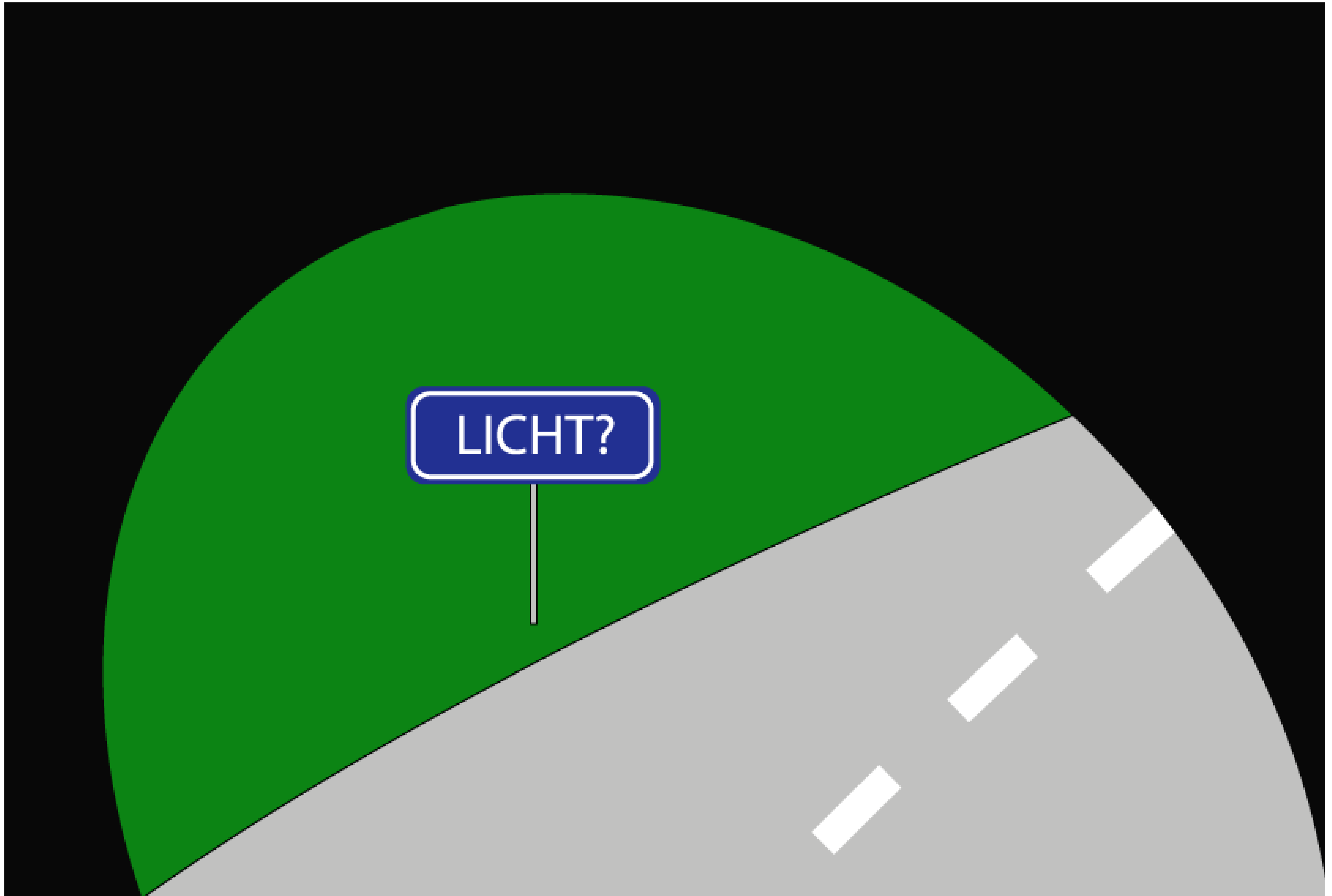
3 Arten:

- Befehl (Command)
- Abfrage (Query)
- Erzeugungsprozedur (*creation procedure*) (später studiert)

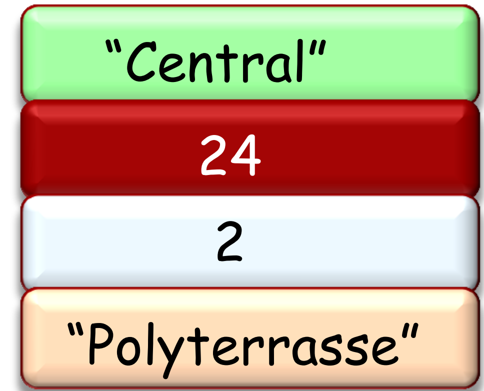
Ein Befehl

LICHT!

Eine Abfrage



Abfragen



Ziel: die Eigenschaften eines Objekts zu erhalten.

Sollte *weder das Zielobjekt noch andere Objekte ändern!*

Beispiele anhand eines "Linie" Objektes (*Polybahn*):

- Was ist die West-Endstation von *Polybahn*?
- Was ist die Ost-Endstation von *Polybahn*?
- Wieviele Stationen hat *Polybahn*?
- Welche Farbe hat *Polybahn*?

Befehle

Ziel: Ein oder mehrere Objekte zu verändern.

Beispiele anhand eines "Linie" Objektes:

- Setze die Farbe der *Polybahn*
- Füge einen neuen Wagen zur *Polybahn* hinzu

Das Befehl-Abfrage-Separationsprinzip (*)

Das Stellen einer Frage

soll die Antwort

nicht verändern

(*) engl.: Command-Query Separation principle

Ein Objekt ist eine Maschine

Ein laufendes Programm ist eine Maschine.

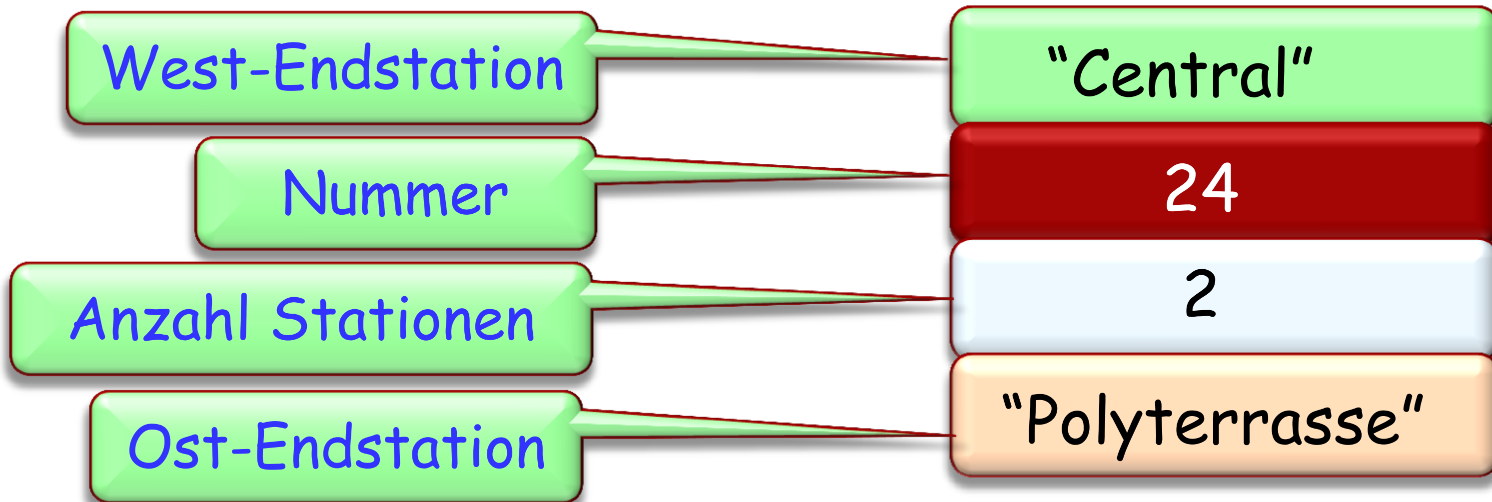
Es besteht aus kleineren Maschinen: Objekten

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. *Polybahn.west_terminal* die West-Endstation von *Polybahn* zurück

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

Features können Argumente haben...

Aufgabe:

- Geben Sie die West-Endstation von *Polybahn* auf dem "Konsolenfenster" aus.

Sie brauchen:

- Das Objekt *console*
- Das auf *console* aufrufbare Feature *output*
- Das Objekt *Polybahn*
- Das auf *Polybahn* aufrufbare Feature *west_terminal*, welches die West-Endstation zurückgibt

console.output (Polybahn.west_terminal)

Den Featurerumpf ausbauen

```
class PREVIEW
inherit ZURICH_OBJECTS
 feature
      explore
              -- Die Stadt erkunden und die
              -- West-Endstation der Polybahn anzeigen.
          do
              Central.highlight
              Polyterrasse.highlight
              Polybahn.add_transport
              Zurich_map.animate

              console.output (Polybahn.west_terminal)

      end
end
```

Features mit Argumenten

ihr_objekt.ihr_feature (ein_argument)

ein_argument ist ein Wert, welcher *ihr_feature* braucht

Beispiel: Feature *output* 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

Zurich_map.animate

next_message.send
computer.shut_down
telephone.ring

Jede Operation wird auf ein Objekt angewendet

Eine eigene Ausdrucksweise

Zurich_map.animate

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.

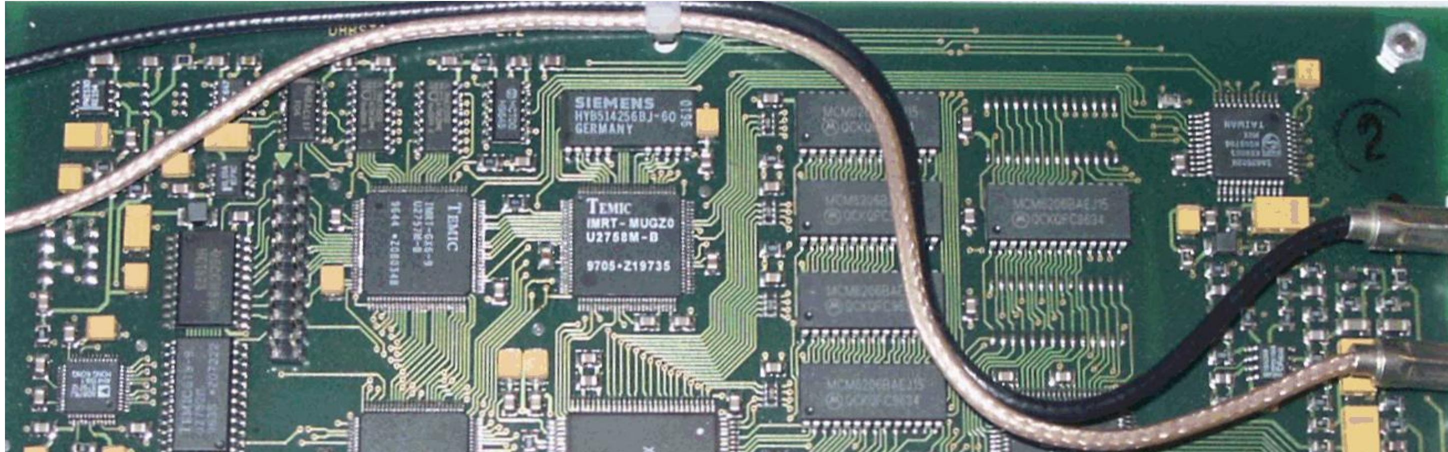
Skalierbarkeit

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

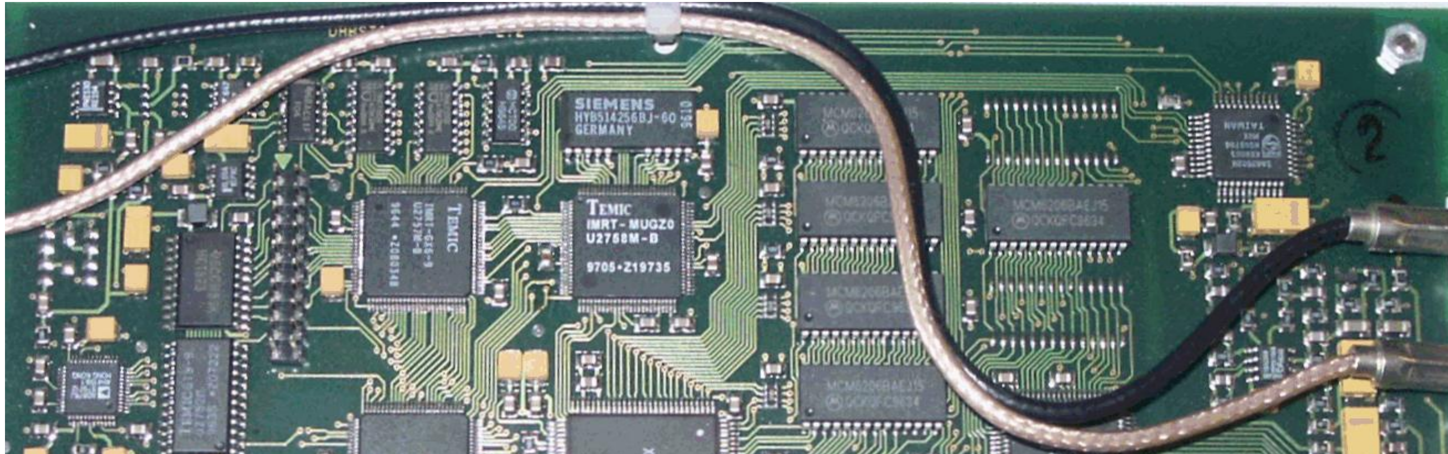
Genau das ist das Ziel für die Techniken, die wir in diesem Kurs lehren

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.

Mehr über unseres erste Beispiel

```
class
    PREVIEW
inherit
    ZURICH_OBJECTS
feature
    explore
            -- Die Stadt erkunden.
        do
            Central.highlight
            Polyterrasse.highlight
            Polybahn.add_transport
            Zurich_map.animate
        end
end
```

Drei Arten von Objekten (Erinnerung)

Drei Arten von Objekten:

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

  Beispiele: die Polyterrasse, eine Bahn des Trams…

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

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

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

  Beispiele: "Datenstrukturen" wie Arrays oder Listen

Mehr über unseres erste Beispiel

```
class
    PREVIEW
inherit
    ZURICH_OBJECTS
feature
    explore
                -- Die Stadt erkunden.
        do
            Central.highlight
            Polyterrasse.highlight
            Polybahn.add_transport
            Zurich_map.animate
        end
end
```

Eine bessere Version

```
class
    PREVIEW
inherit
    ZURICH_OBJECTS
feature
    explore
            -- Die Stadt erkunden.
        do
            Central_view.highlight
            Polyterrasse_view.highlight
            Polybahn.add_transport
            Zurich_map.animate
        end
end
```

Modell und Präsentation

(Model and View)

Modell-Objekte beschreiben Elemente von einem Modell der externen Welt

- Beispiel: *Polyterrasse*

View-Objekte beschreiben Elemente von der Benutzer-Schnittstelle

- Beispiel: *Polyterrasse_view*

Model-View-Controller (Modell/Präsentation/Steuerung)

(Trygve Reenskaug, 1979)

MVC structure

updates

MODEL

VIEW$_i$

represents

CONTROLLER

updates

...

(Other views)

GUI tools

sees

thinks in terms of

interacts with

User

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)
- Modell-Präsentation-Trennung (Model-View Separation)

Bis nächste Woche

Lesen Sie Kapitel 1 bis 3 von *Touch of Class*

Schauen Sie sich die Folien der nächsten zwei Vorlesungen an