



Einführung in die Programmierung

Prof. Dr. Bertrand Meyer

Lektion 3: Der Umgang mit Objekten II



Die Programmiersprache ist die Notation, welche die Syntax und die Semantik von Programmen definiert

Unsere Programmiersprache ist Eiffel

Es gibt viele Programmiersprachen! Manche sind allgemeiner, manche spezifischer

Programmiersprachen sind **künstliche Notationen**, gestaltet für einen spezifischen Zweck (das Programmieren)

Objekttechnologie

Herkunft: die Simula 67-Sprache, Oslo, Mitte der 60er. Verbreitete sich *sehr* langsam in den 70ern



Die Sprache Smalltalk (Xerox PARC, 1970er) machte Objektorientierung (O-O) "hip", indem sie es mit visuellen Technologien kombinierte

Die erste OOPSLA anno 1986 stellte O-O der breiten Masse vor. O-O verbreitete sich schnell in den 90ern durch

- O-O Sprachen: Objective C, C++, Eiffel, Java, C#...
- O-O Werkzeuge, O-O Datenbanken, O-O Analyse...

Heute ist O-O grösstenteils akzeptiert

Nicht- O-O Ansätze heissen auch "prozedural"

Über Eiffel



Erste Version 1985, seither regelmässig verbessert

Fokus: Einfachheit und Softwarequalität, im Speziellen
Verlässlichkeit, Erweiterbarkeit, Wiederverwendbarkeit

Eiffel basiert auf dem Konzept "Design by Contract" ("Entwurf
gemäss Vertrag")

Implementationen: z.B. EiffelStudio (von Eiffel Software), als
open-source (GPL) verfügbar

Internationale Standards: ECMA und ISO (International
Standards Organization), 2006

Einige Eiffel-basierte Projekte



Axa Rosenberg
Vermögensverwaltung
3 Millionen Codezeilen

Chicago Board of Trade:
Kurs-Anzeigesystem
Eiffel + CORBA +
Solaris + Windows + ...

Northrop-Grumman
Grossmasstäbliche
Simulationen für die
Raketenverteidigung

Schwedische Sozialversicherung:
Unfallberichte & -management, etc...



Weshalb benutzen wir Eiffel?



- Ein einfaches, sauberes O-O Modell
- Erlaubt es Ihnen, sich auf die Konzepte und nicht auf die Sprache zu konzentrieren
- Kleine "Sprachlast"
- Programmierumgebung (EiffelStudio)
- Portabilität: Windows / Linux / VMS & andere
- Realismus: keine "akademische" Sprache

Es bereitet Sie darauf vor, andere O-O Sprachen zu lernen, z.B. C++, Java, C#.

Kursreihe (ab dem dritten Studienjahr): "Languages in Depth". Momentan Java, C# und Eiffel.



```
class First {  
    public static void main(String args[])  
    {  
        System.out.println("Hello World!");  
    }  
}
```

Drei grundlegende Unterscheidungen



Befehl / Abfrage

Instruktion / Ausdruck

Syntax / Semantik

Instruktionen (instructions)



Die Basisoperationen eines Computers oder eines Programms heissen **Instruktionen**

Unser erstes Beispiel hatte **fünf** Instruktionen:

Central_view.highlight

Polyterasse_view.highlight

Polybahn.add_transport

Zurich_map.animate

console.output (Polybahn.west_terminal)

Aufeinanderfolgende Instruktionen



Sie können mehrere Instruktionen hintereinander schreiben, ohne sie durch ein Semikolon zu trennen:

```
Central_view.highlight  
Polyterasse_view.highlight  
Polybahn.add_transport  
Zurich_map.animate  
console.output (Polybahn.west_terminal)
```

Auch:
statement
(Anweisung)

Sie können Semikola benützen, um Instruktionen zu trennen:

```
Central_view.highlight ; Polyterasse_view.highlight ;  
Polybahn.add_transport ; Zurich_map.animate ;  
console.output (Polybahn.west_terminal)
```



Schreiben Sie eine Instruktion pro Zeile
Lassen Sie Semikola weg

Sollten Sie einmal der Meinung sein, dass mehrere Instruktionen auf einer Zeile lesbarer sind (z.B. in einem Artikel), benützen Sie Semikola:

$f(x) ; g(y)$

Ausdrücke (Expressions)



Ein **Ausdruck** ist ein Programmelement, welches mögliche Laufzeitwerte bezeichnet

Beispiele:

`console.output (Polybahn.west_terminal)`

Ein Ausdruck

Noch ein Ausdruck

Vgl.: Mathematische Ausdrücke, z.B. $a + b$



Ein Ausdruck, z.B. *Polybahn.west_terminal* ist kein Wert, sondern **bezeichnet** zukünftige Laufzeitwerte.

Eine Instruktion, z.B. *Central_view.highlight* **bezeichnet** eine Operation, die während der Laufzeit ausgeführt wird.



Die **Syntax** eines Programmes ist die Struktur und die Form seines (Programm-)Textes

Die **Semantik** eines Programmes ist die Menge von Eigenschaften seiner möglichen Ausführungen

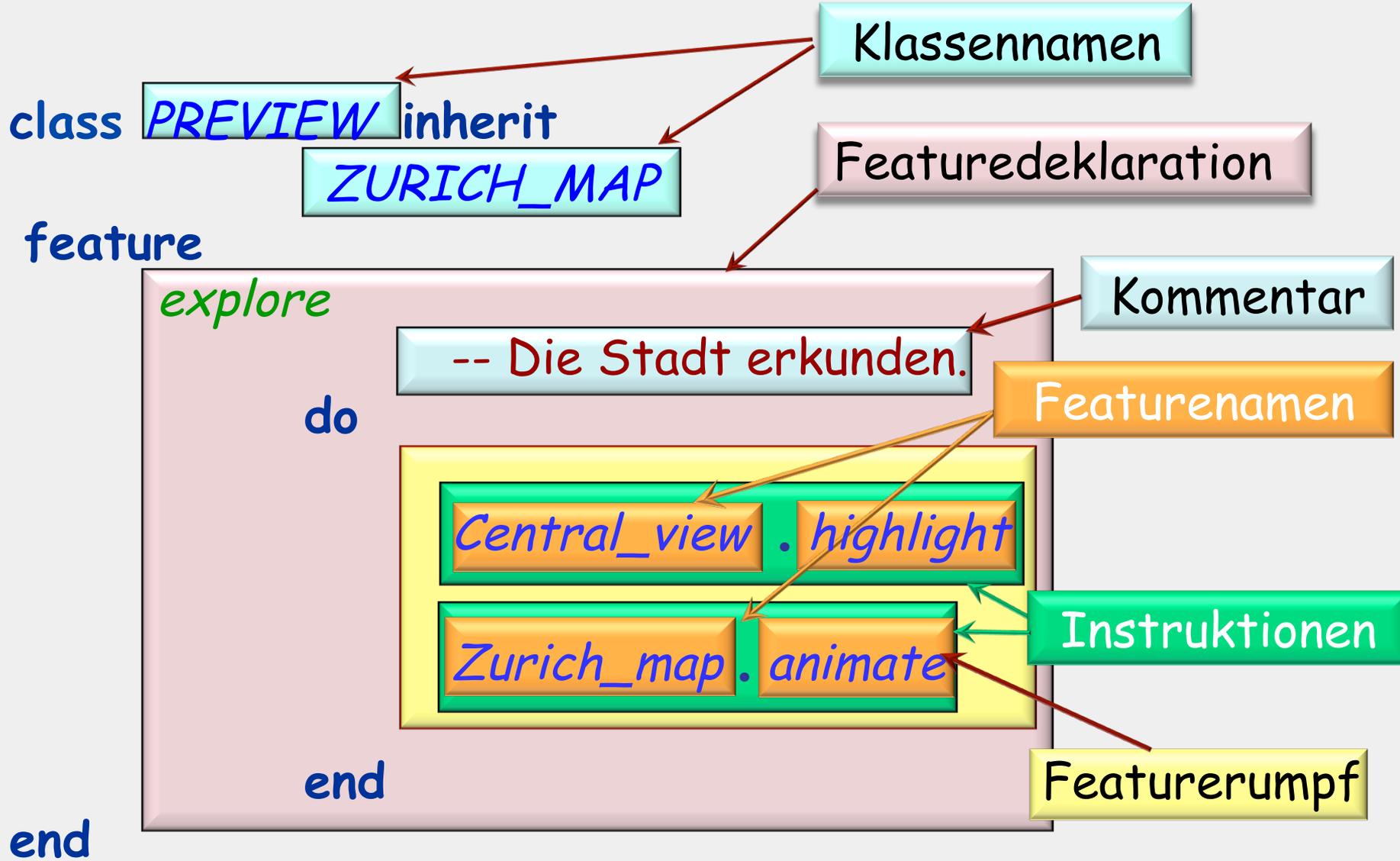
Die Syntax ist die Art, wie Sie ein Programm schreiben: Zeichen, daraus geformte Wörter und aus diesen Wörtern geformte grössere Strukturen

Die Semantik ist der Effekt, den Sie von Ihrem Programm erwarten



| Syntax | Semantik |
|-------------|----------|
| Instruktion | Befehl |
| Ausdruck | Abfrage |

Syntaxstruktur einer Klasse





- Allgemeine Form eines Textes: Abfolge von Wörtern
- Jedes Wort ist selbst eine Abfolge von Zeichen (characters)
- Unterscheidung von Syntax und Semantik
- Einige Wörter sind vordefiniert, einige benutzerdefiniert

Benutzerdefiniert Wörter



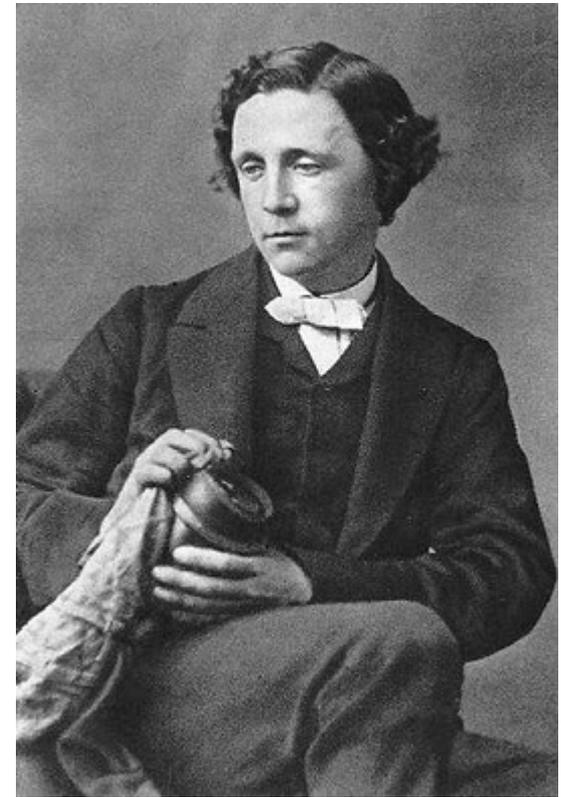
"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"

"And, has thou slain the Jabberwock?
Come to my arms, my beamish boy!
O frabjous day! Callooh! Callay!"
He chortled in his joy.

He took his vorpal sword in hand:
Long time the manxome foe he sought -
So rested he by the Tumtum tree,
And stood awhile in thought.

And, as in uffish thought he stood,
The Jabberwock, with eyes of flame,
Came whiffling through the tulgey wood,
And burbled as it came!

One, two! One, two! And through and through
The vorpal blade went snicker-snack!
He left it dead, and with its head
He went galumphing back.



Lewis Carroll



- Die **Ausdrückskraft** ist in natürlichen Sprachen viel grösser
- Die **Präzision** hingegen ist in Programmiersprachen viel höher

Programmiersprachen sind eine Erweiterung der mathematischen Notation

Kommentare sind kleine Ausschnitte aus natürlichen Sprachen, die in Programmen vorkommen

Benutzen Sie Wörter aus natürlichen Sprachen (z.B. Englisch, Deutsch) für die Namen, die Sie definieren

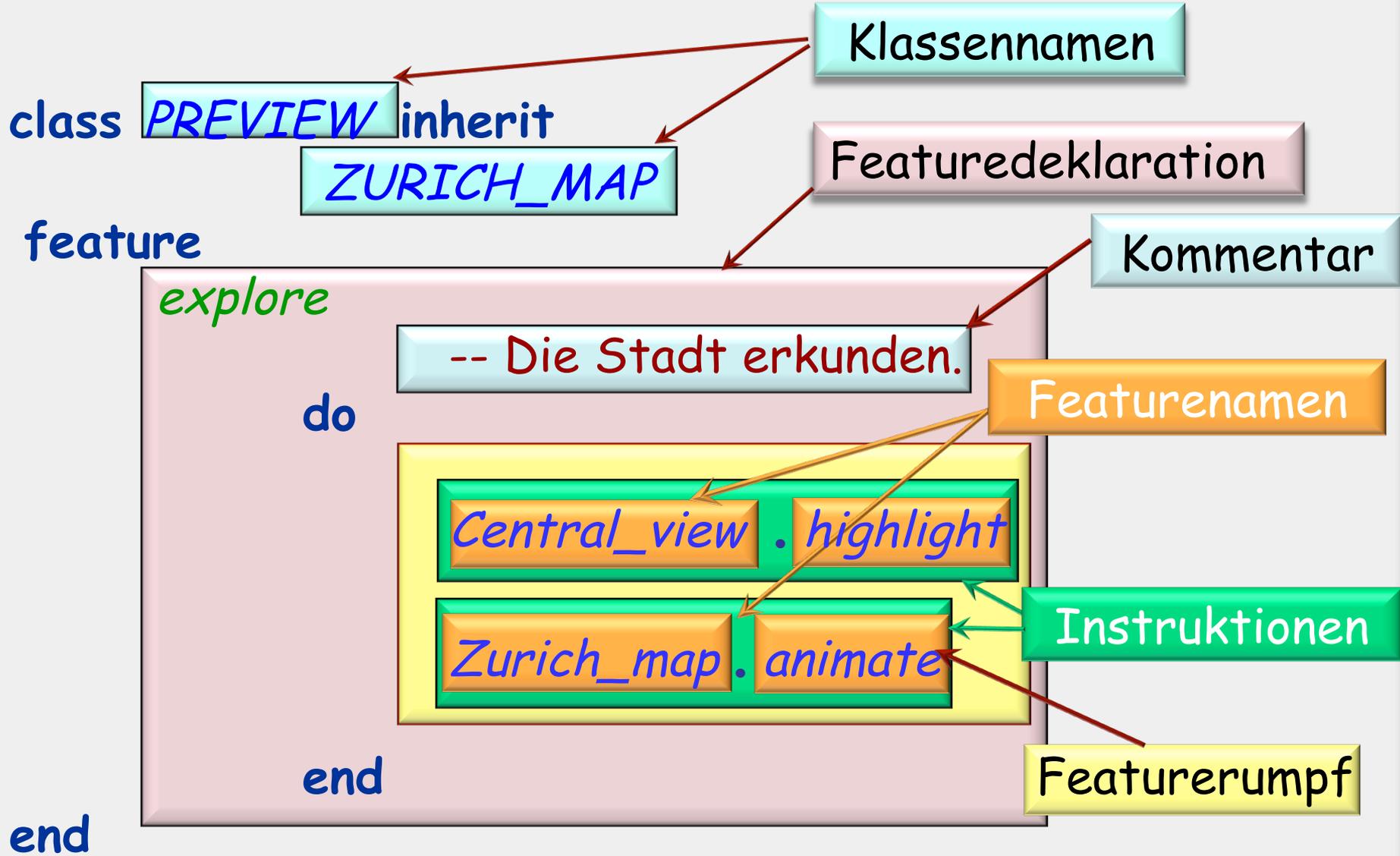
Beispiele:

- *city, station1*
- Featurenamen: *highlight, output*
- Klassennamen: *PREVIEW, CITY, STATION*
- Mit mehreren Wörtern: *add_transport, ZURICH_MAP*

Die Schlüsselwörter von Eiffel sind englische Wörter: *inherit, do, end...*

Insbesondere sind alle Schlüsselwörter **Einzelwörter**, bis auf **elseif**

Syntaxstruktur einer Klasse



Exemplar: Ein syntaktisches Element, z.B.:

- Ein Klassenname, z.B. *PREVIEW*
- Eine Instruktion, z.B. *Central_view.highlight*
- Irgendeine der Boxen der vorigen Folie
- Der gesamte Klassentext!

Exemplare können **verschachtelt** (oder **eingebettet**) sein

Delimiter, wie z.B. Schlüsselwörter (**do**, **end**, ...), Semikola, Punkte • etc. sind **keine** Exemplare



Ein **Konstrukt** ist ein gewisser Typ eines syntaktischen Elements

Jedes syntaktische Element ist ein **Exemplar** eines gewissen Konstrukts

Beispiele:

- *highlight* ist ein Exemplar des Konstrukts *Feature_name*
- Der Klassentext als Ganzes ist ein Exemplar des Konstruktes *Klasse*

Syntaktische Struktur: verschachtelte* Darstellung



class **PREVIEW** inherit

ZURICH_MAP

feature

explore

-- Die Stadt erkunden.

do

Central_view . *highlight*

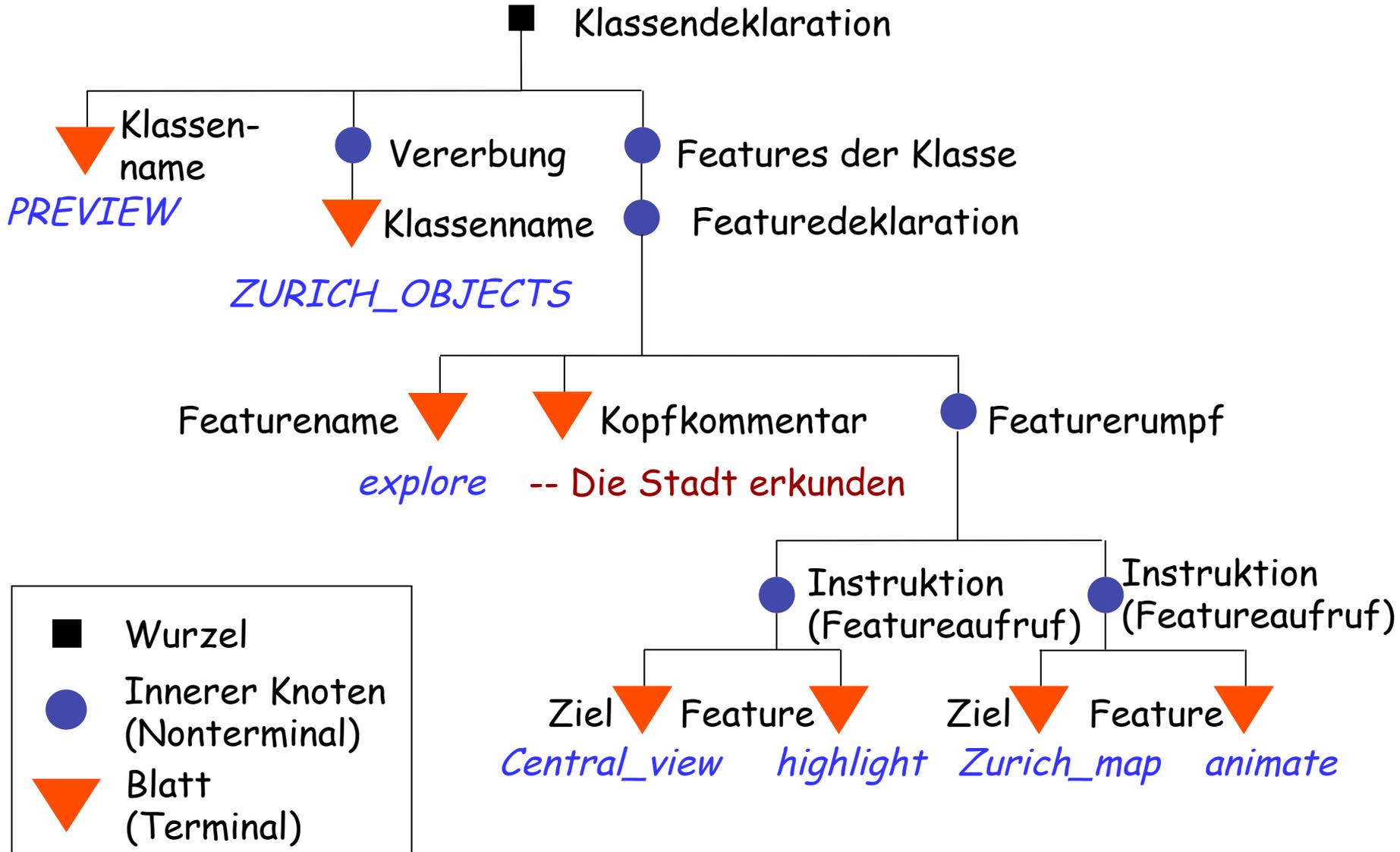
Zurich_map . *animate*

end

end

*Engl.: *nested*

Eine andere Darstellung: ein abstrakter Syntaxbaum (*)



(*) engl.: Abstract Syntax Tree (AST)

Abstrakter Syntaxbaum



Stellt die Syntaxstruktur dar

Nur Exemplare: keine Schlüsselwörter oder andere Delimiter
(deshalb **abstrakt**)

Benutzt den Begriff **Baum** wie Unternehmen in
organisatorischen Diagrammen

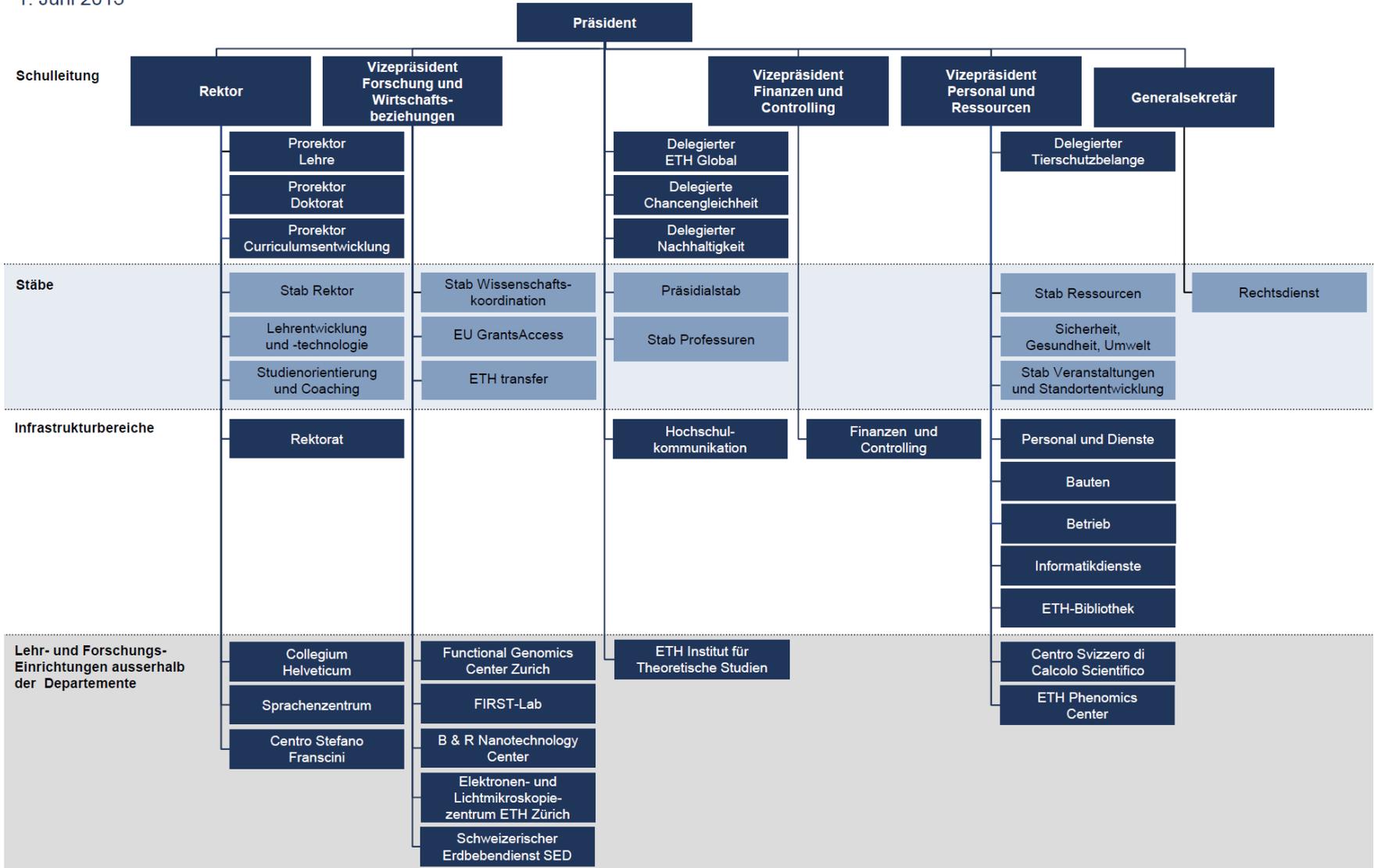
Bäume wachsen nach unten...



Organigramm der ETH Zürich

Schulleitung und Zentrale Organe sowie Lehr- und Forschungseinrichtungen ausserhalb der Departemente

1. Juni 2013





- Repräsentieren hierarchische oder verschachtelte Strukturen
- Ähnlich wie organisatorische Diagramme (vorige Folie)
- Werden von oben nach unten oder von links nach rechts gezeichnet



Regeln:

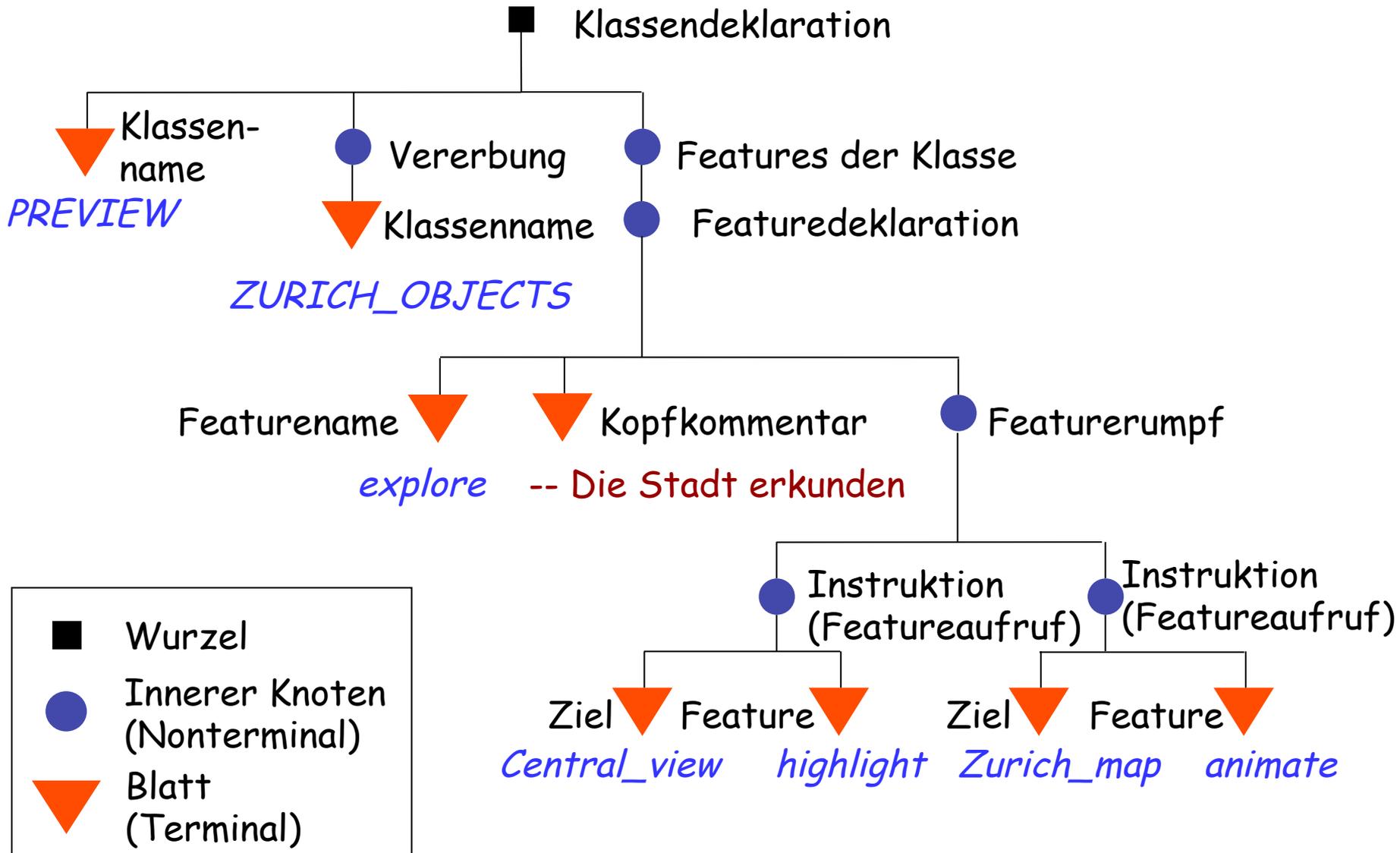
- Jeder Zweig verbindet genau zwei Knoten
- Jeder Knoten kann beliebig viele (auch keine) **abgehende** Zweige haben
- Jeder Knoten hat maximal einen **eingehenden** Zweig

Arten von Knoten:

- **Wurzel (root)**: Ein Knoten ohne eingehenden Zweig
- **Blatt (leaf)**: Ein Knoten mit keinen abgehenden Zweigen
- **Innere Knoten (internal nodes)**: weder Wurzel noch Blatt ("standard")

Ein Baum hat **genau eine Wurzel** (sonst wäre es ein **Wald**)

Eine andere Darstellung: ein abstrakter Syntaxbaum



(*) engl.: Abstract Syntax Tree (AST)



- Wurzel: repräsentiert das gesamte Exemplar (das "äusserste Rechteck")
- Innere Knoten (**Nonterminale**): repräsentieren Unterstrukturen, die wiederum Exemplare enthalten.
- Blätter (**Terminale**): repräsentieren Exemplare ohne weitere Verschachtelung

- Die Syntax einer Programmiersprache ist definiert durch eine Menge von Konstrukten sowie die (Unter-)Konstrukte dieser Konstrukte
(siehe die BNF-Vorlesung)

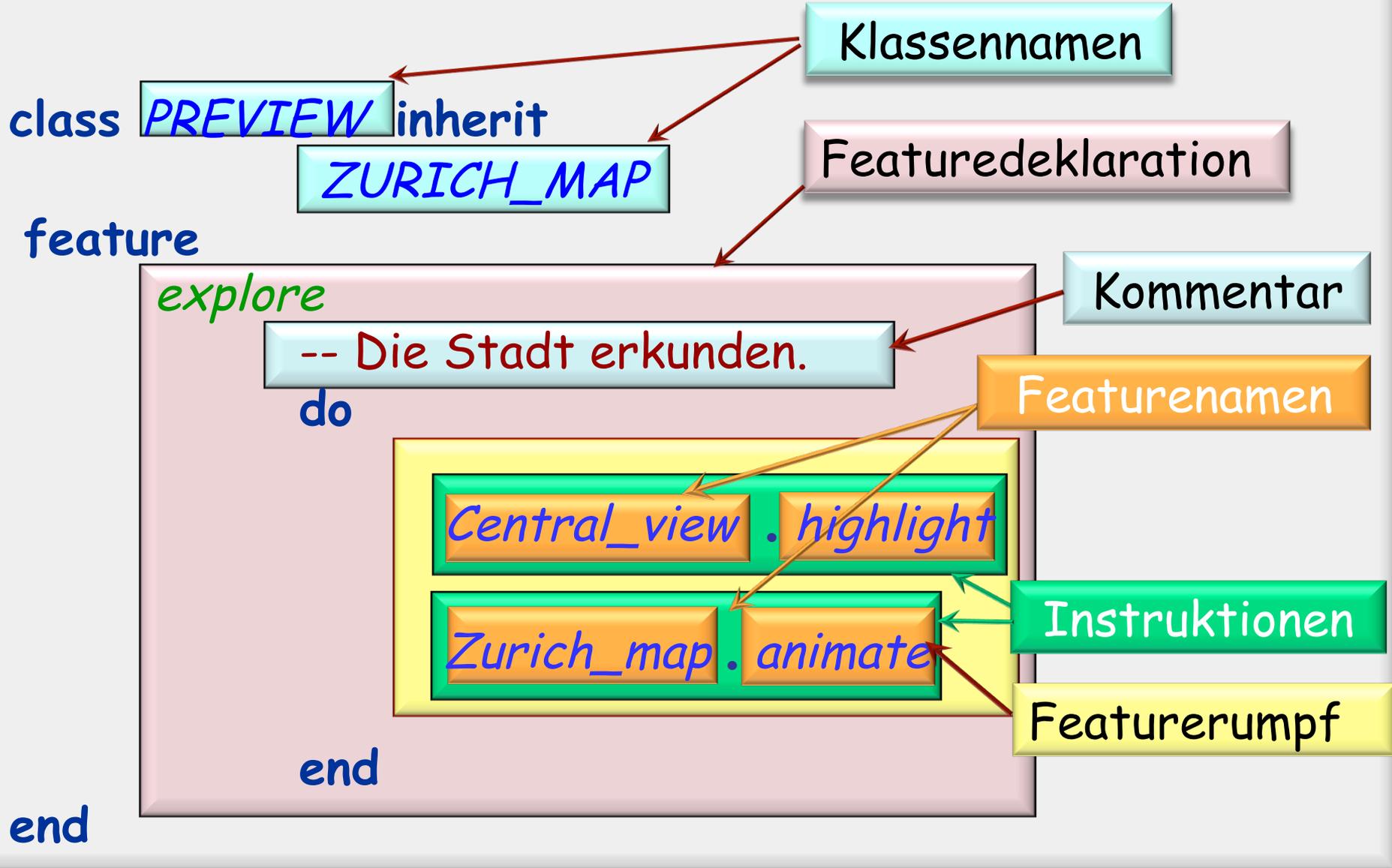


Die Grundelemente eines Programmtextes sind **Tokens**:

- **Terminale**
 - **Bezeichner (identifizier)**: durch Programmierer gewählte Namen, z.B. *Zurich_map* oder *highlight*
 - **Konstanten**: selbsterklärende Werte, z.B. *42*
- **Schlüsselwörter**, z.B. *class*
- **Spezialsymbole**: z.B. Punkt "." eines Feature-Aufrufs

Tokens definieren die **lexikalische Struktur** einer Sprache.

Syntaxstruktur einer Klasse



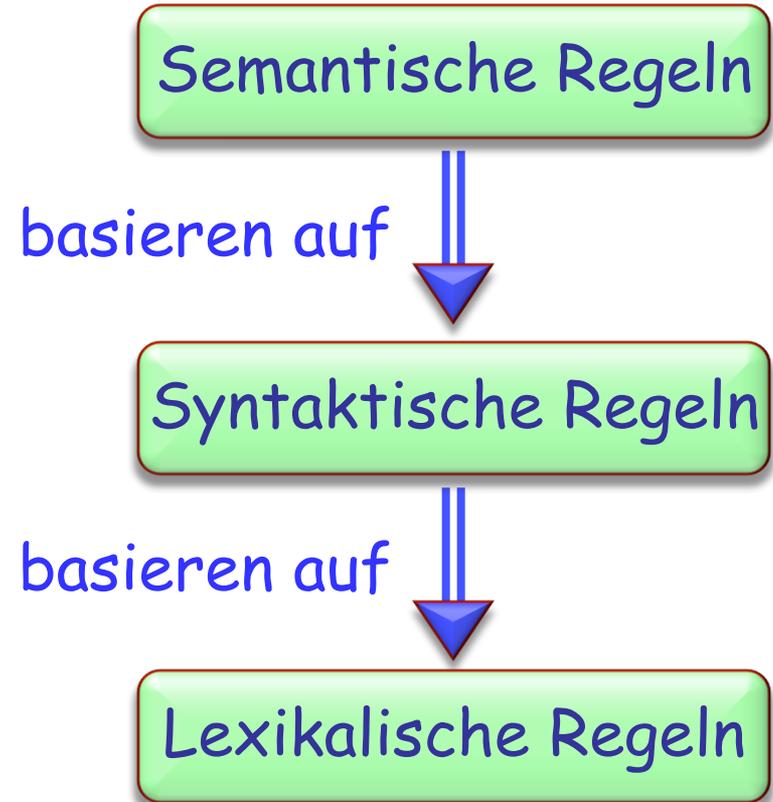
Drei Ebenen von Beschreibungen



Semantische Regeln
definieren den Effekt eines
Programms, das den
syntaktischen Regeln
genügt

Syntaktische Regeln
definieren, wie man
Exemplare aus Tokens, die
den lexikalischen Regeln
genügen, herstellt

Lexikalische Regeln
definieren, wie man aus
Zeichen Tokens macht





Bezeichner (Identifiers)

Ein Bezeichner beginnt mit einem Buchstaben, gefolgt von null oder mehr Zeichen, wovon jedes

- ein Buchstabe
 - eine Zahl (0 bis 9)
 - ein Unterstrich " _ "
- sein kann

Sie können Ihre Bezeichner (nach obigen Regeln) frei wählen, nur **Schlüsselwörter** sind verboten

- Wählen Sie Bezeichner so, dass sie klar ausdrücken, was sie tun. (z.B. *west_terminal*, *highlight*)
- Wählen Sie für Features die vollen Namen, keine Abkürzungen.
- Benutzen Sie für zusammengesetzte Bezeichner Unterstriche:

Zurich_map

- Klassennamen sollten aus Grossbuchstaben bestehen:

PREVIEW

Eine weitere Ebene



Statische Semantik definiert die Gültigkeitsregel, die durch die Syntax nicht garantiert wird.

Gültiges Beispiel:

`console.output (Polybahn.west_terminal)`

Ungültiges Beispiel:

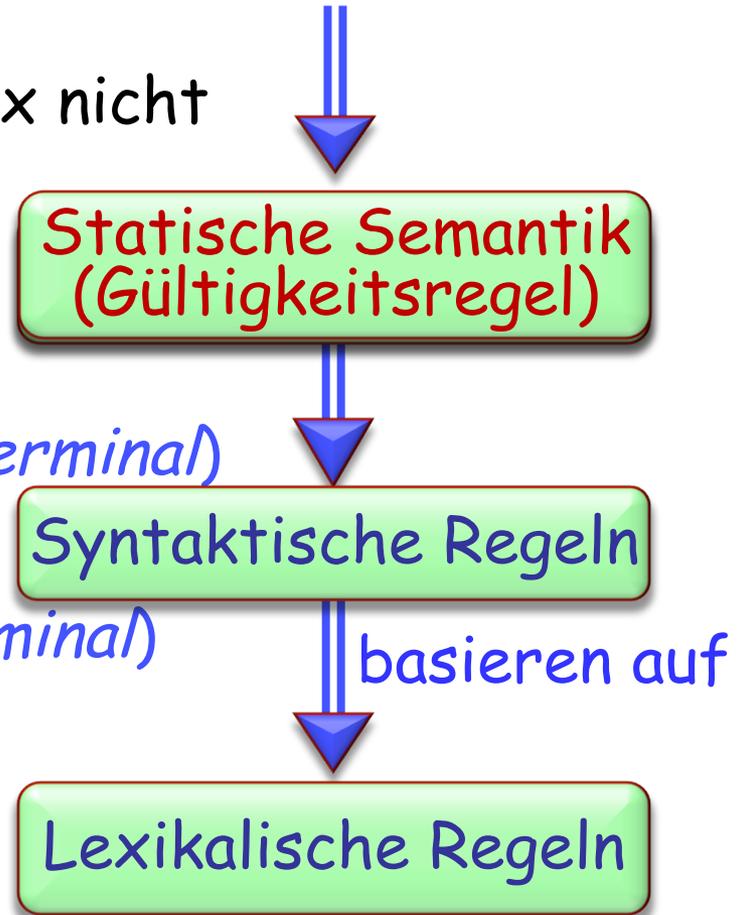
`console.output (output.west_terminal)`

(Vgl. in der deutschen Sprache:

`Ich mag meinen Computer`

Aber nicht:

`Ich mag meiner Computer)`





- Das Programmiersprachenkonzept
- Die Grundzüge von Eiffel
- Syntax (inklusive lexikalischer Ebene) vs Semantik
- Lexikalische und statische Analyseebenen
- Bäume
- Die Fachsprache der Bäume: Wurzel, innere Knoten, Blätter
- Abstrakte Syntaxbäume
- Grundlegende lexikalische Elemente
- Elementare Stilregeln

Aufgaben auf nächste Woche



Lesen Sie die Kapitel 1 bis 4 von *Touch of Class*

Stellen Sie sicher, dass Sie alle bis jetzt eingeführten **Begriffe** kennen und verstehen.