



# 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 (zu) viele Programmiersprachen! Manche sind allgemeiner, manche spezifischer.

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



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".



Erste Version 1985, seither konstant überarbeitet und verbessert.

Fokus: Softwarequalität, im Speziellen Verlässlichkeit, Erweiterbarkeit, Wiederverwendbarkeit. Einfachheit steht im Vordergrund.

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

Anwendungen: auftragsentscheidende Projekte in der Industrie

Mehrere Implementationen, z.B. EiffelStudio (von Eiffel Software), welches wir benützen. EiffelStudio ist open-source.

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

# Einige Eiffel-basierte Projekte



Axa Rosenberg

Vermögensverwaltung: von 2 Milliarden \$ auf > 60 Milliarden \$

3 Millionen Codezeilen

Chicago Board of Trade:

Kurs-Anzeigesystem

Eiffel + CORBA +

Solaris + Windows + ...

Xontech (für Boeing)

Grossmassstäbliche  
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.

# Drei grundlegende Unterscheidungen

---



Befehl / Abfrage

Instruktion / Ausdruck

Syntax / Semantik

# Instruktionen (instructions)

---



Die Basisoperationen heissen **Instruktionen**.

Unser erstes Beispiel hatte **fünf** Instruktionen:

*Paris.display*

*Louvre.spotlight*

*Line8.highlight*

*Route1.animate*

*Console.show(Route1.origin)*

# Aufeinanderfolgende Instruktionen



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

*Paris.display*

*Louvre.spotlight*

*Line8.highlight*

*Route1.animate*

*Console.show (Route1.origin)*

Sie können Semikola benutzen, um Instruktionen zu trennen:

*Paris.display ; Louvre.spotlight ;*

*Line8.highlight ; Route1.animate ;*

*Console.show (Route1.origin)*

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 Report), benützen Sie Semikola:

$f(x) ; g(y)$

# Ausdrücke (Expressions)



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

Beispiele:

*Console.show(Route1.origin)*

Ein Ausdruck

Noch ein Ausdruck

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

# Syntax (syntax) und Semantik (semantics)

---



Ein Ausdruck, z.B. *Route1.origin* ist kein Wert, sondern **bezeichnet** zukünftige Laufzeitwerte.

Eine Instruktion, z.B. *Paris.display* **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.

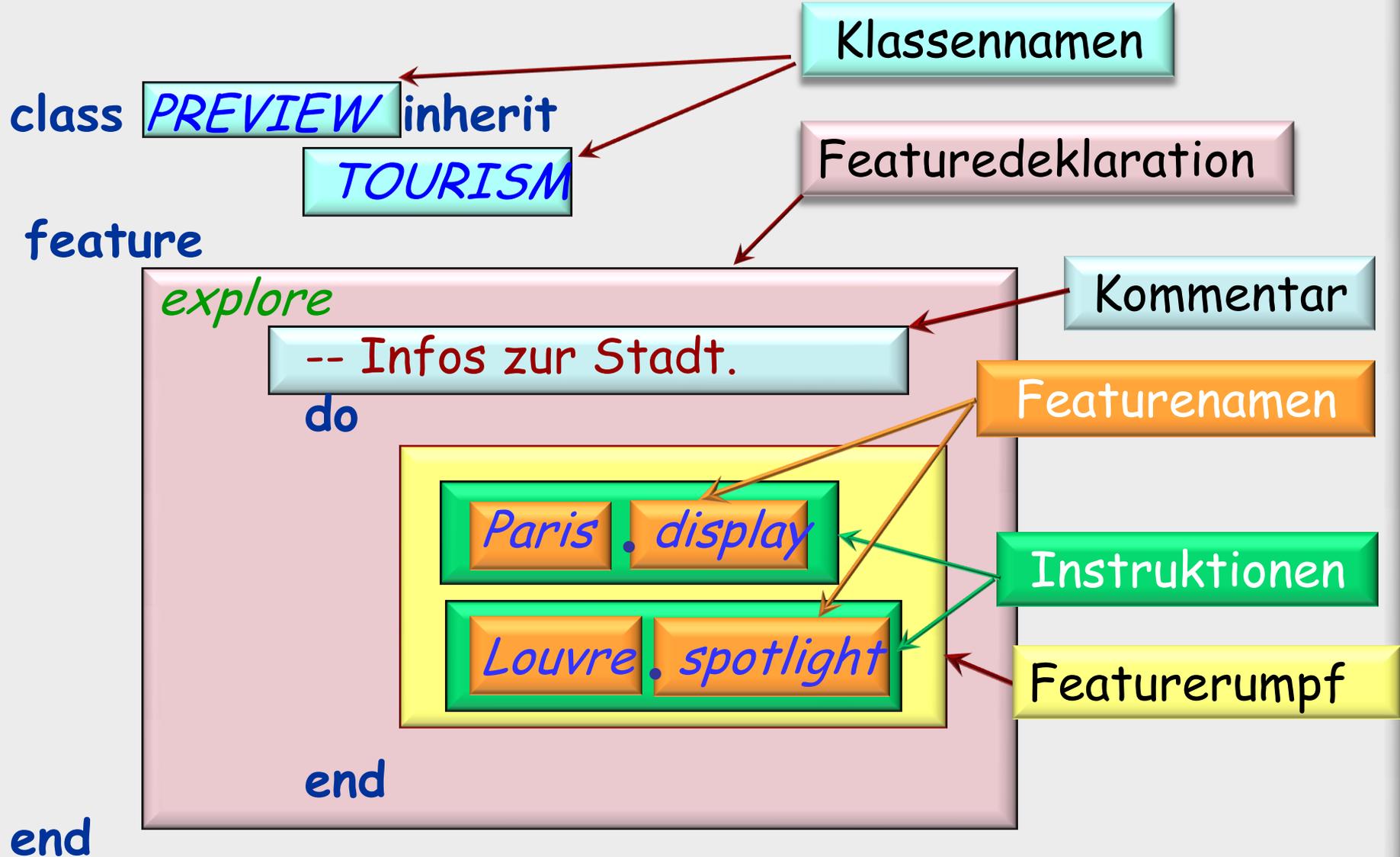
Die Syntax ist die Art, wie Sie ein Programm schreiben: Zeichen, daraus geformte Wörter und aus diesen Wörtern geformte grössere Strukturen.

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
- Unterscheidung von Syntax und Semantik
- Einige Wörter sind vordefiniert, einige benutzerdefiniert.



- Die **Macht von Ausdrücken** 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.



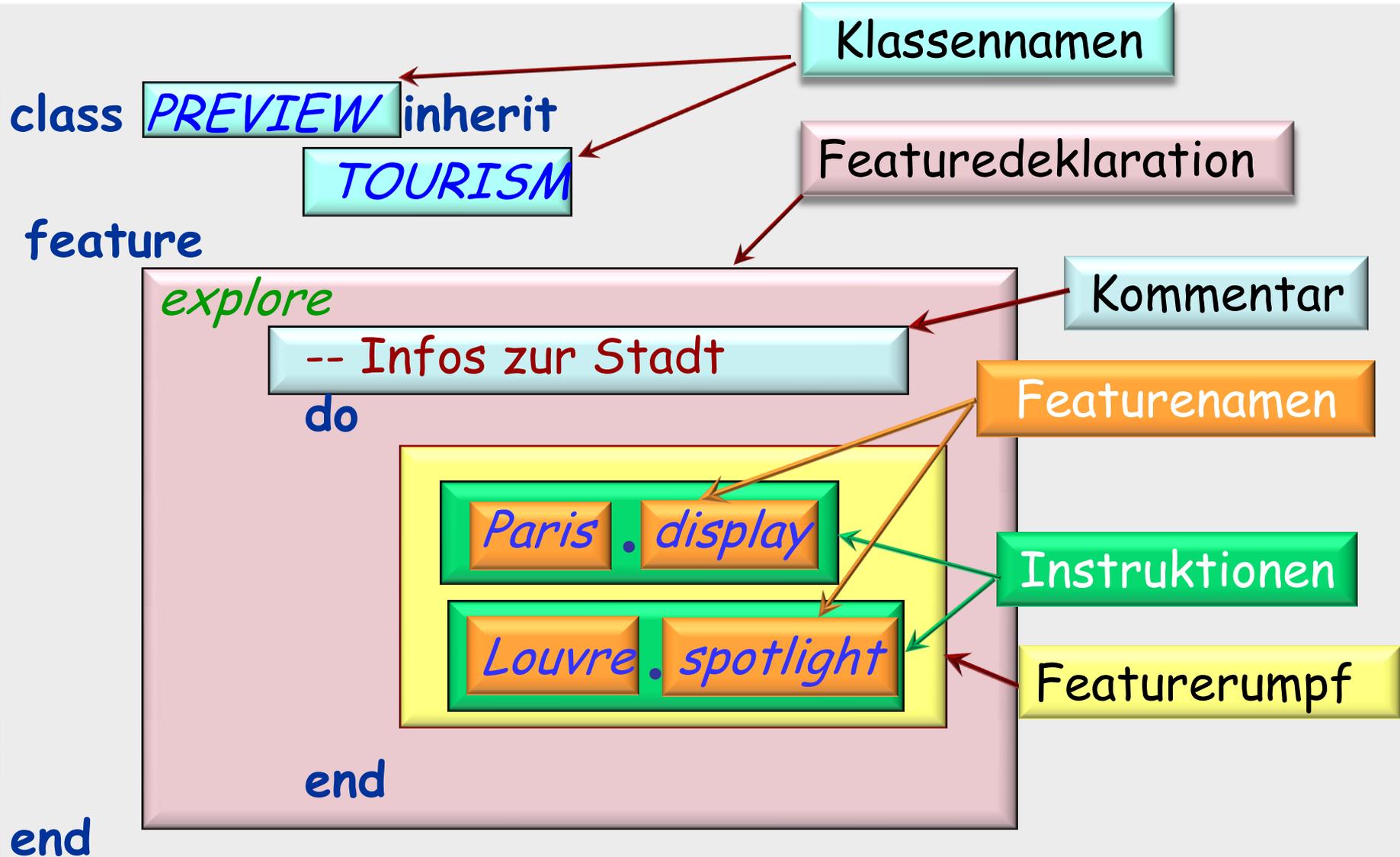
Benutzen Sie Wörter aus natürlichen Sprachen (z.B. Englisch, Deutsch) für die Namen, die Sie definieren.

Beispiele:

- *Paris, Route1*
- Featurenamen: *show, origin*
- Klassennamen: *PREVIEW, TOURISM*
- Mit mehreren Wörtern: *show\_start, TRAFFIC\_TAXI*

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

# Syntaxstruktur einer Klasse



# Exemplare (Specimens)

---



**Exemplar:** Ein syntaktisches Element, z.B.:

- Ein Klassenname, z.B. *PREVIEW*
- Eine Instruktion, z.B. *Paris.display*
- 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.



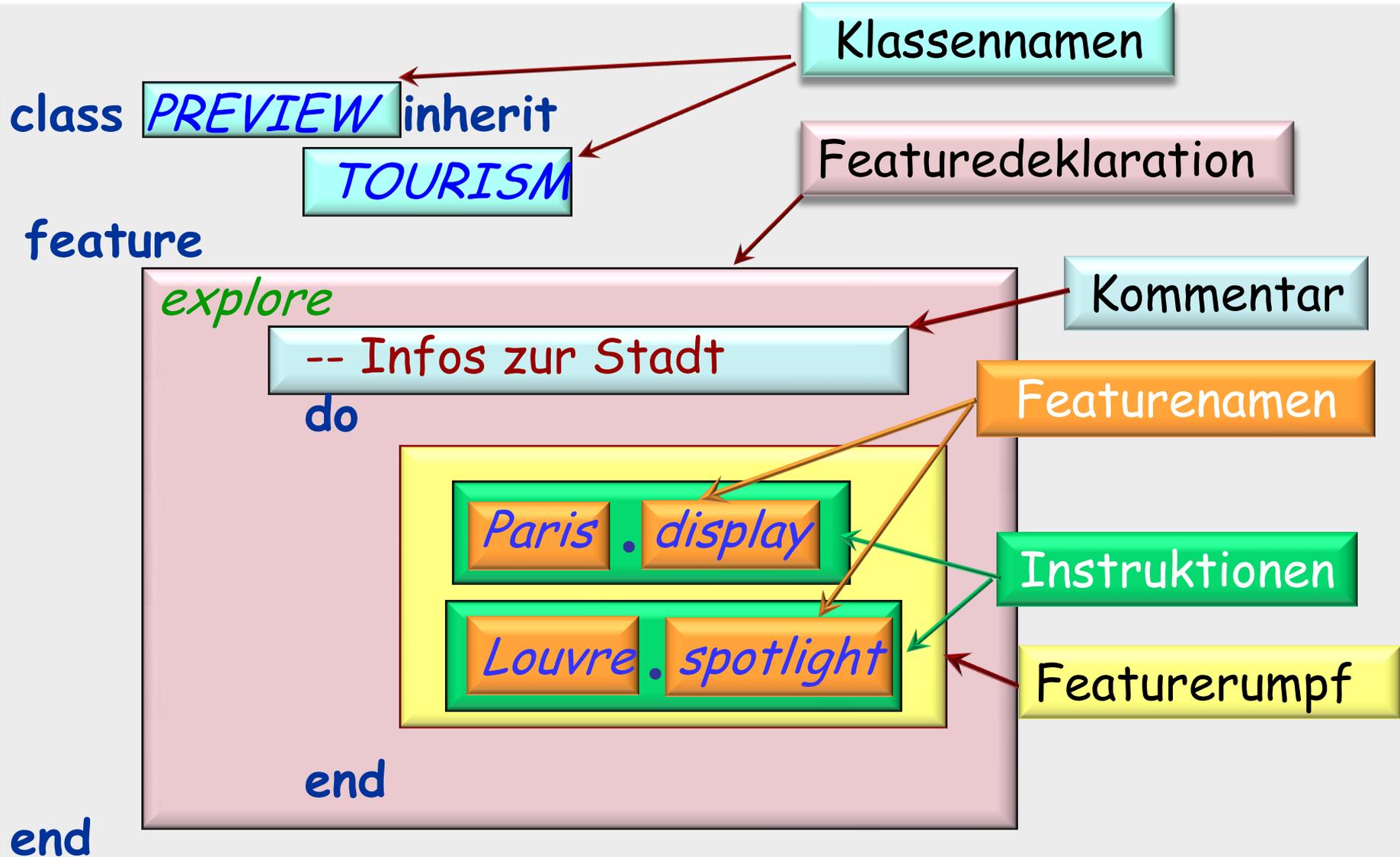
Ein **Konstrukt** ist ein gewisser Typ eines syntaktischen Elements

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

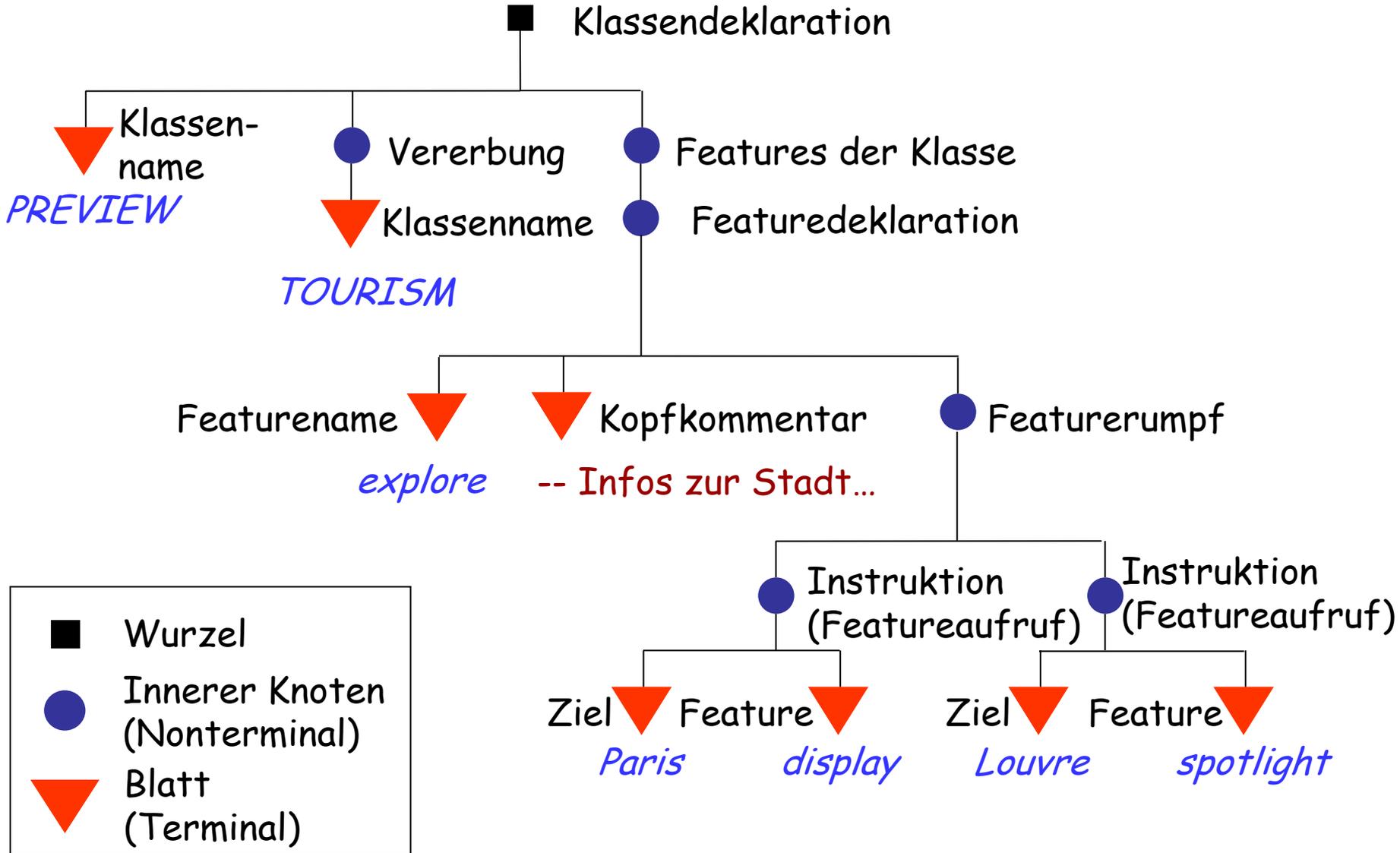
Beispiele:

- *display* ist ein Exemplar des Konstrukts *Feature\_name*
- Der Klassentext als Ganzes ist ein Exemplar des Konstruktes *Klasse*

# Syntaxstruktur einer Klasse



# 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**)

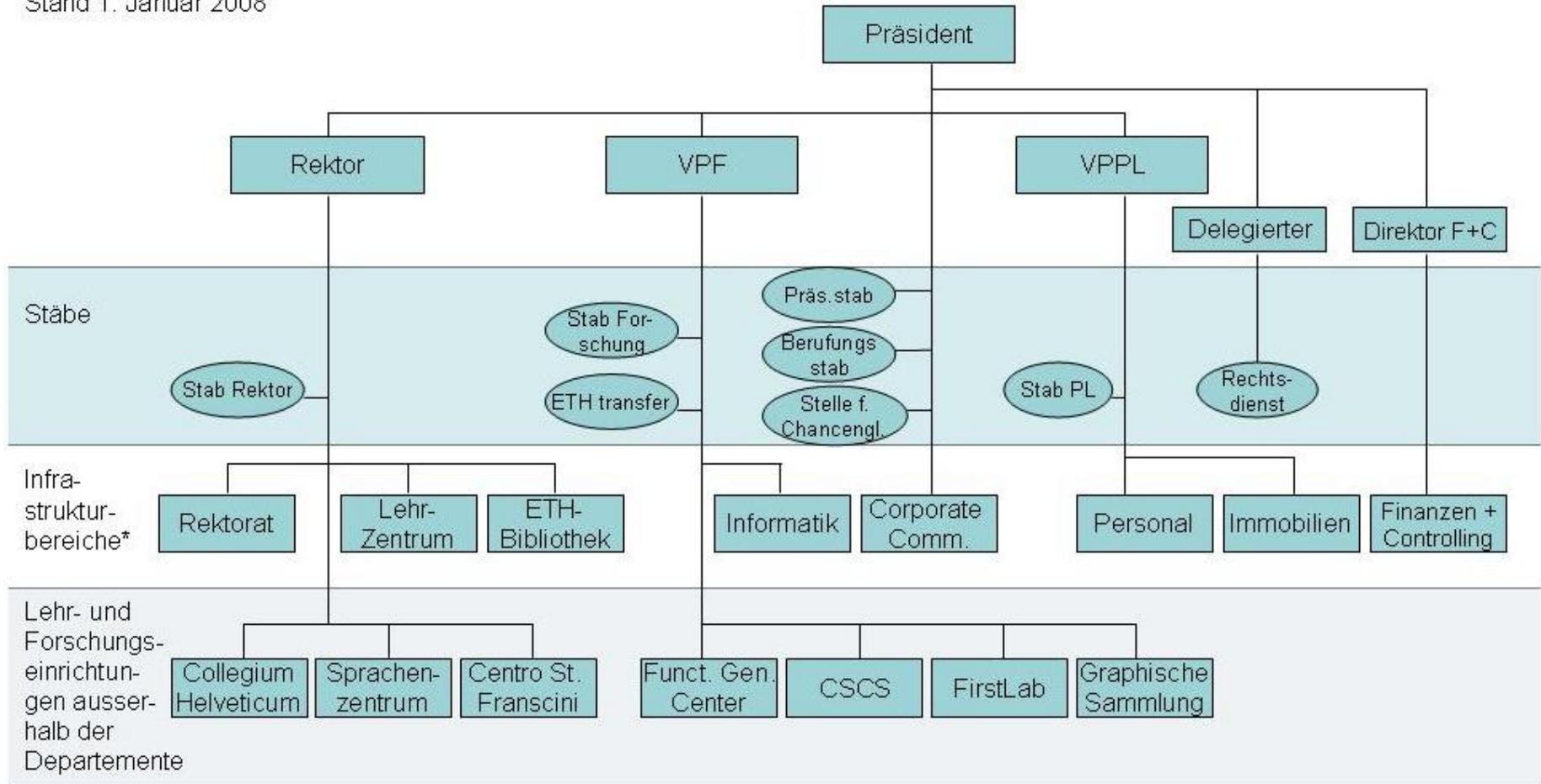
Benutzt den Begriff **Baum** wie Unternehmen in organisatorischen Diagrammen.

# Bäume wachsen nach unten...



## Organigramm der ETH Zürich: Schulleitung, Zentrale Organe sowie Lehr- und Forschungseinrichtungen ausserhalb der Departemente

Stand 1. Januar 2008





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



## Regeln:

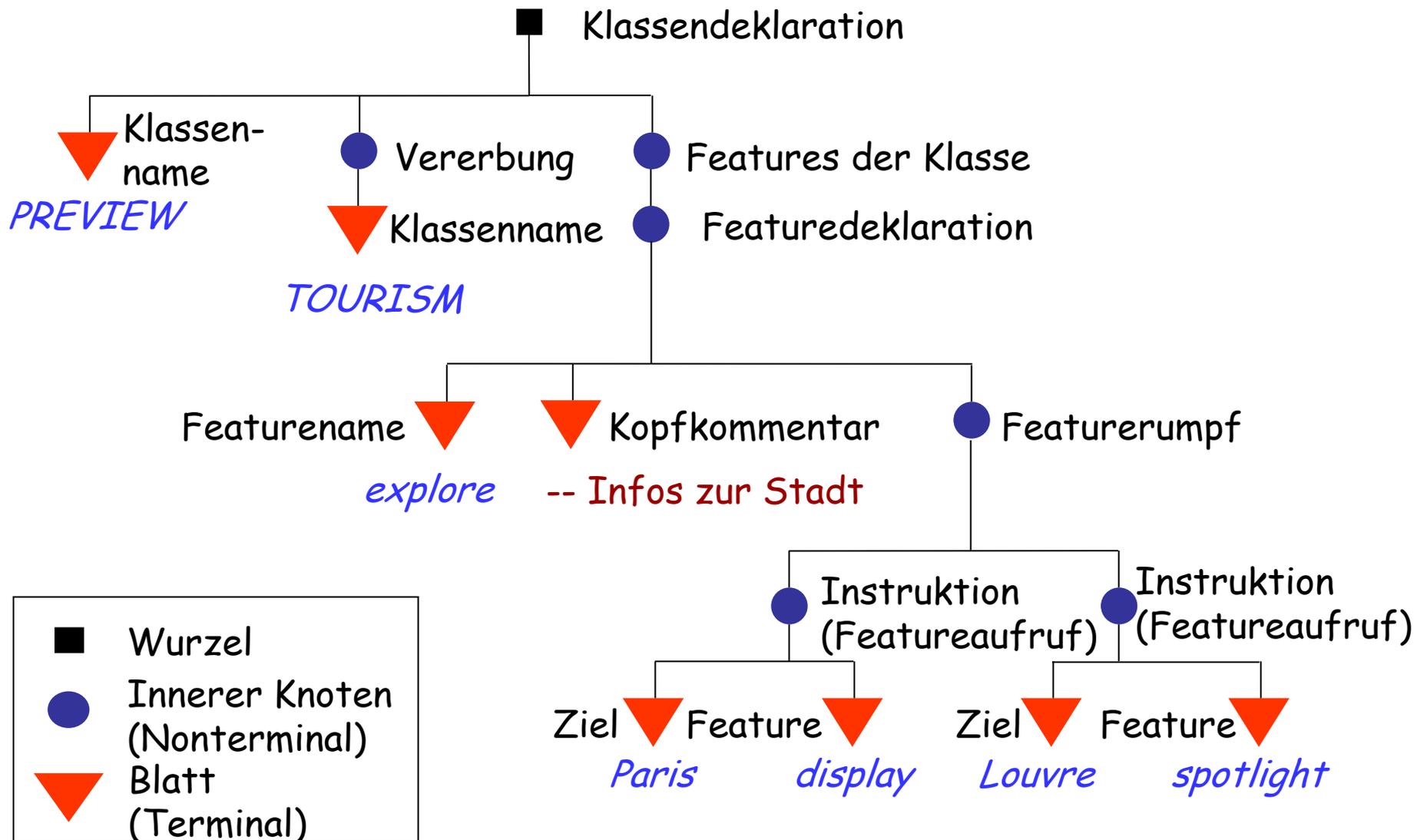
- Jeder Zweig verbindet genau zwei Knoten.
- Jeder Knoten kann beliebig viele (auch keine) **abgehende** Zweige haben
- Jeder Knoten hat maximal einen **eingehenden** Zweig.

## 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**).

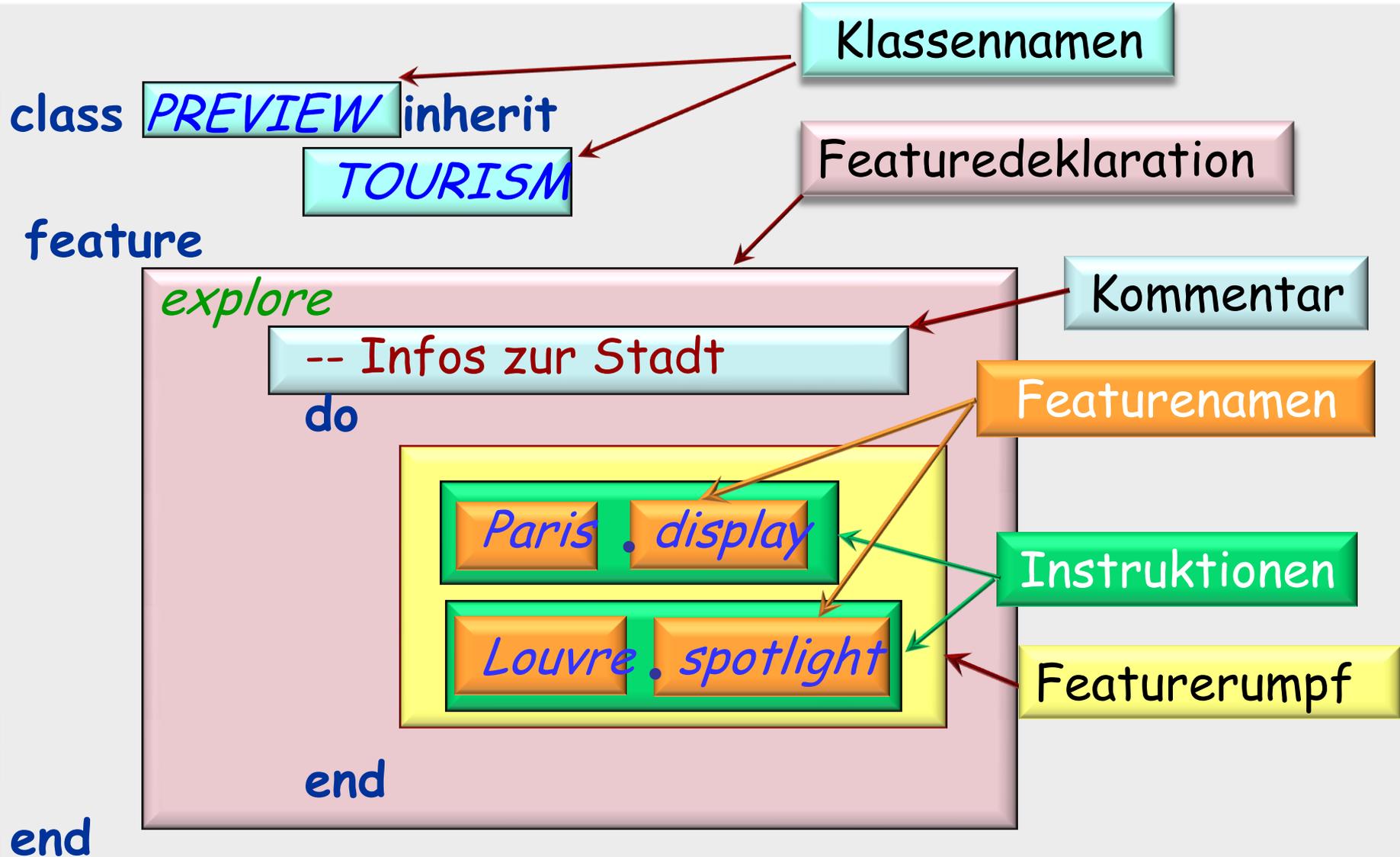
# Eine andere Darstellung: ein abstrakter Syntaxbaum





- 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.

# Syntaxstruktur einer Klasse





Die Grundelemente eines Programmtextes sind **Tokens**:

➤ **Terminale**

- **Bezeichner (identifizier)**: durch den/die Programmierer/in gewählte Namen, z.B. *Paris* oder *display*
- **Konstanten**: selbsterklärende Werte, z.B. **42**

➤ **Schlüsselwörter**, z.B. **class**

➤ **Spezialsymbole**: z.B. Punkt "." eines Feature-Aufrufs

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

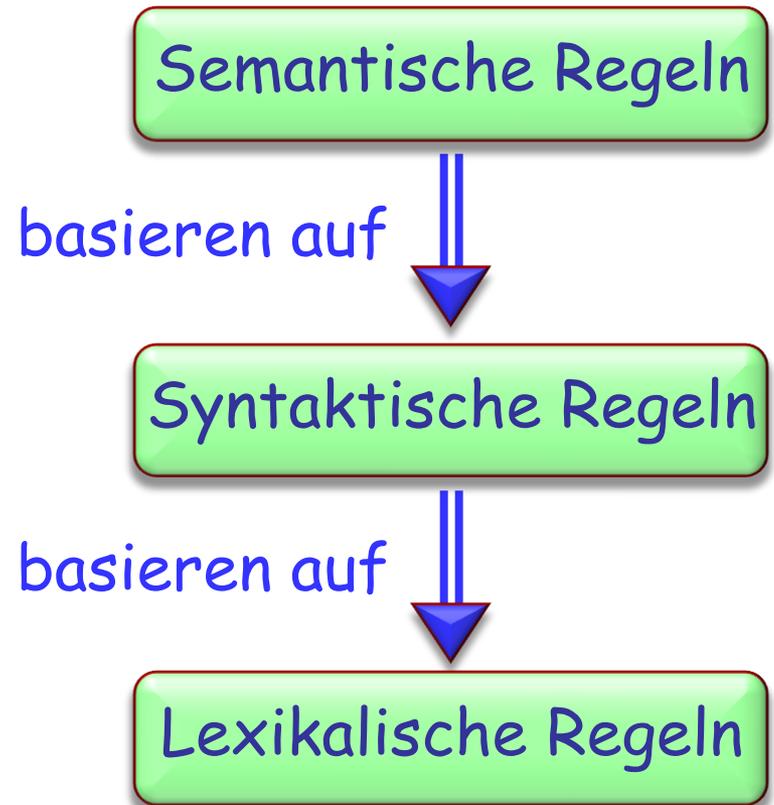
# Drei Ebenen von Beschreibungen



**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. *display*, *origin*)
- Wählen Sie für Features die vollen Namen, keine Abkürzungen.
- Benutzen Sie für zusammengesetzte Bezeichner Unterstriche:

*bus\_station*

- Klassennamen sollten aus Grossbuchstaben bestehen:

*PREVIEW*

# Eine weitere Ebene



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

Gültiges Beispiel:

*Console.show(Route1.origin)*

Ungültiges Beispiel:

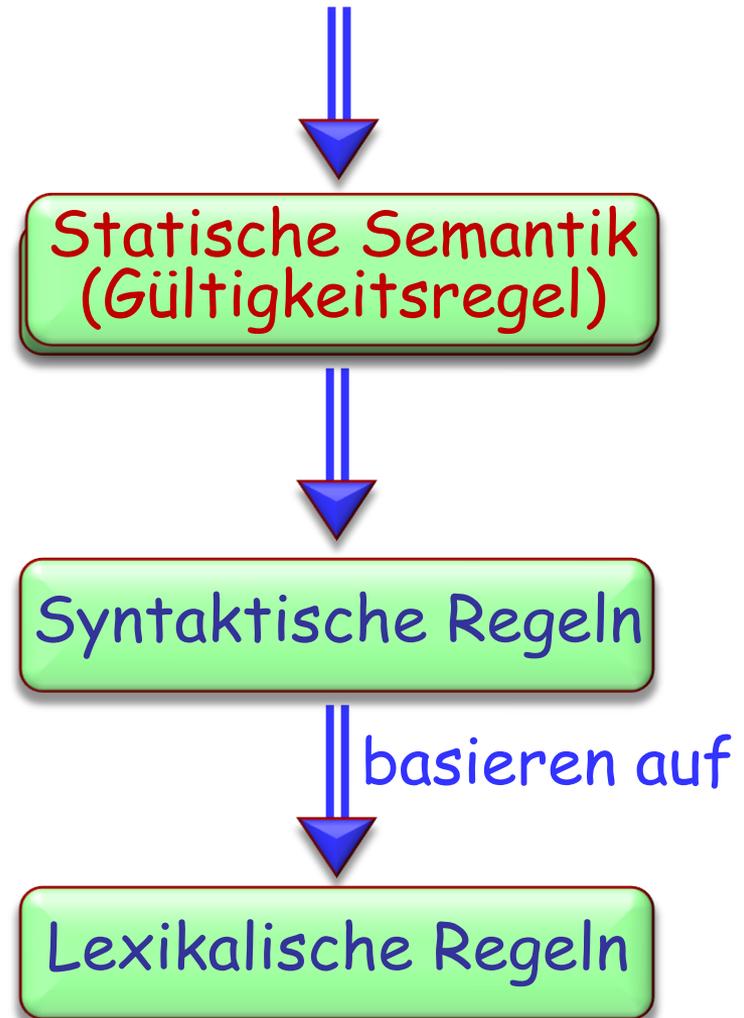
*Console.show(show.origin)*

(Vgl. in der deutschen Sprache:

*Ich mag meinen Computer.*

Aber nicht:

*Ich Computer meinen mag.)*





- 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 5 von *Touch of Class*

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