

# Kapitel WT:V

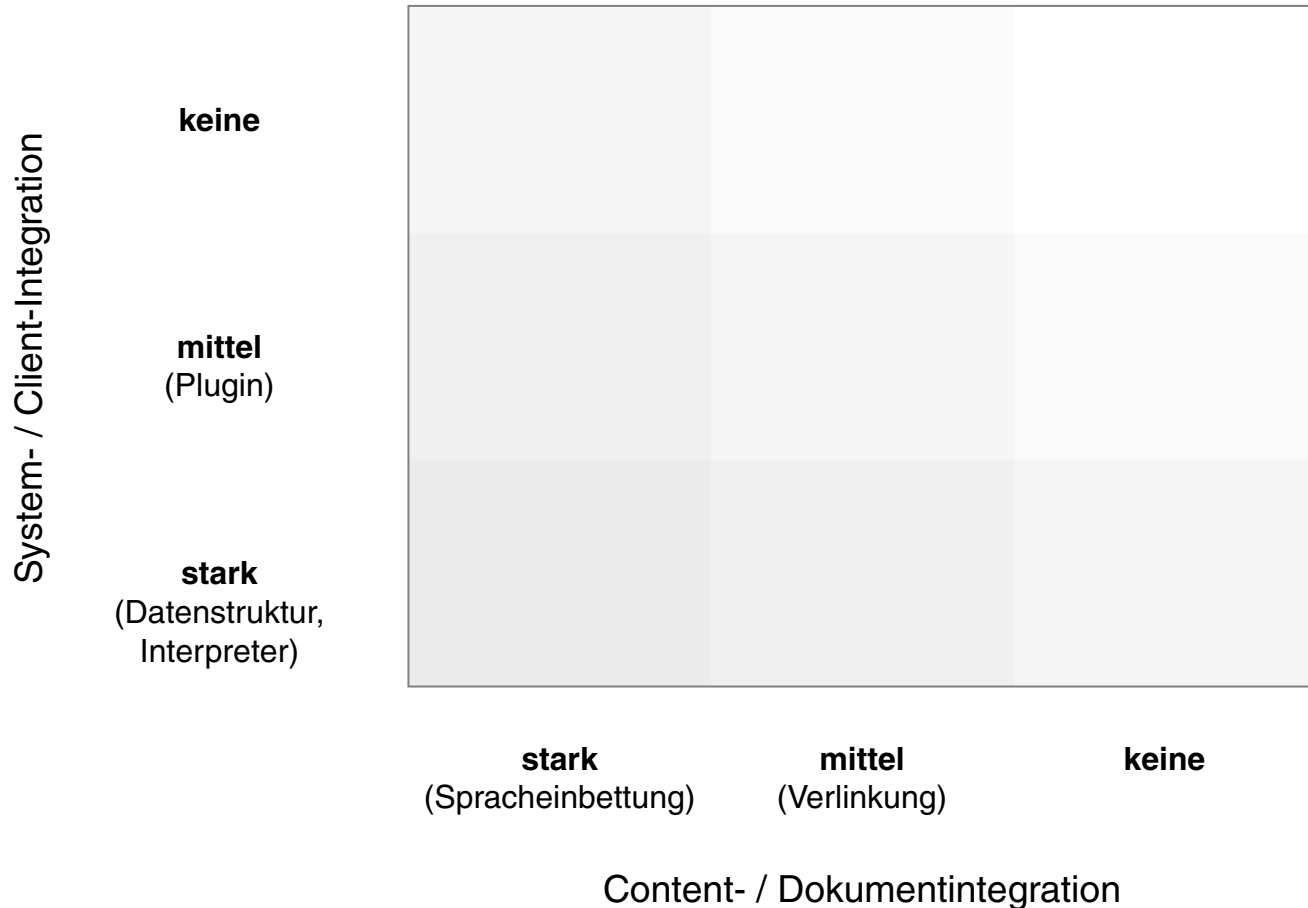
## V. Client-Technologien

- ❑ Web-Client
- ❑ Exkurs: Programmiersprachen
- ❑ JavaScript
- ❑ VBScript
- ❑ Java Applet
- ❑ Weitere Client-Technologien

# Web-Client

## Einordnung von Client-Technologien [Stein 2012]

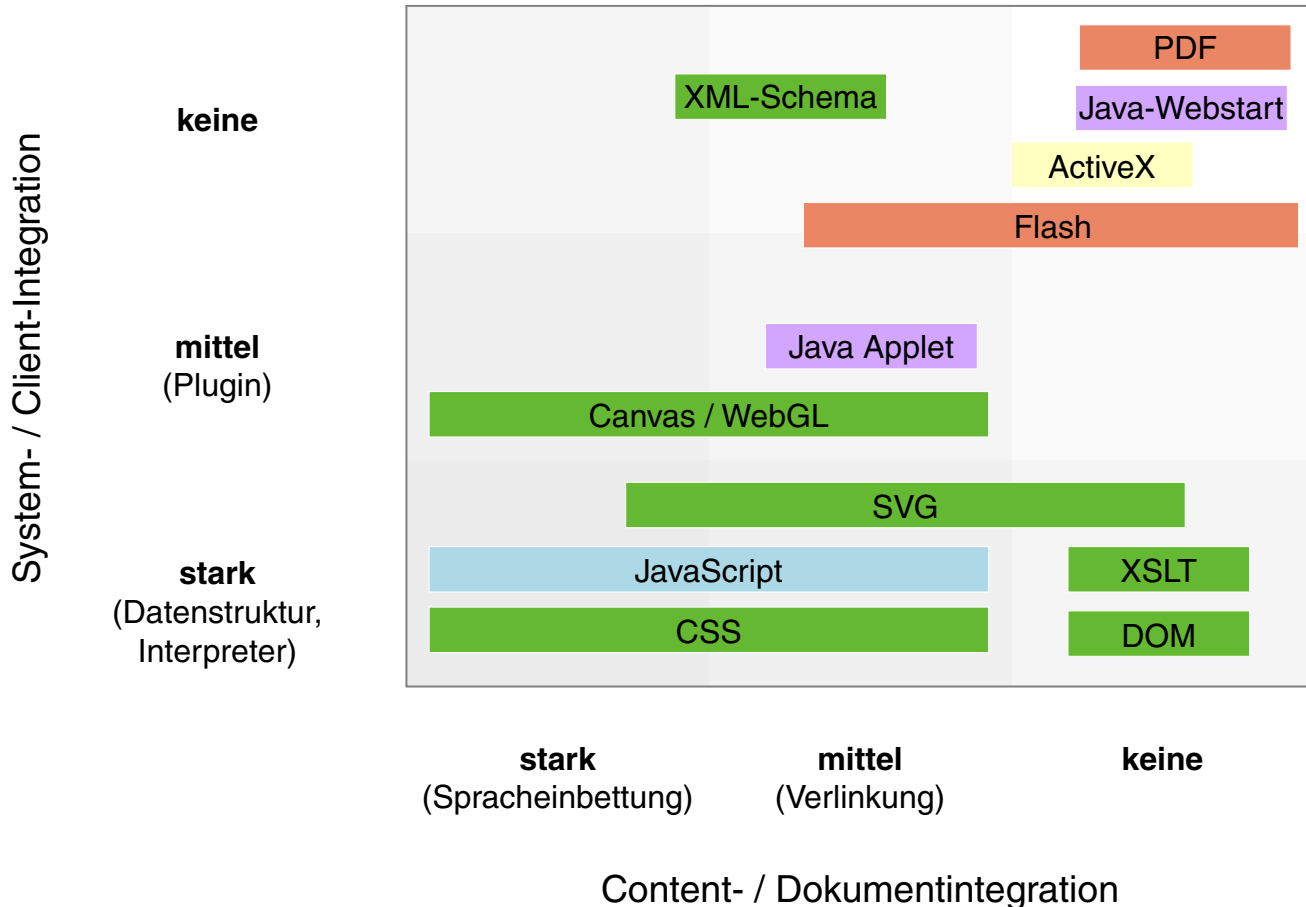
- ❑ x-Achse: Wie eng ist die Technologie mit dem dargestellten Content verwoben?
- ❑ y-Achse: Wie tief ist die Technologie in den Web-Client integriert?



# Web-Client

## Einordnung von Client-Technologien [Stein 2012]

- ❑ x-Achse: Wie eng ist die Technologie mit dem dargestellten Content verwoben?
- ❑ y-Achse: Wie tief ist die Technologie in den Web-Client integriert?

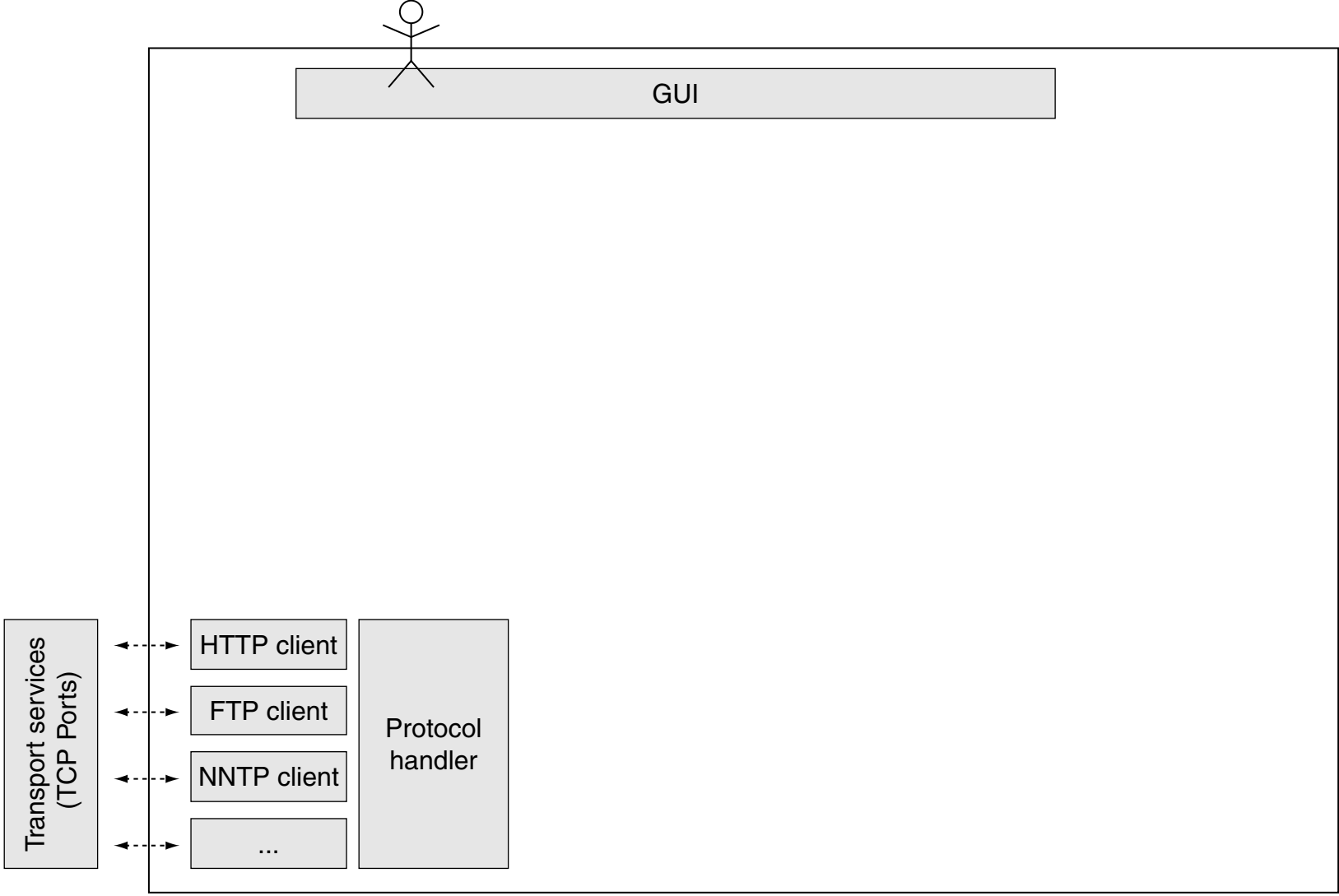


## Bemerkungen:

- ❑ Client-Technologien dienen zur Realisierung Client-seitig ablaufender Web-Anwendungen.
- ❑ Im Vergleich zu Server-seitig ablaufenden Web-Anwendungen erzeugen sie weniger Server-Last und mehr Netzlast.

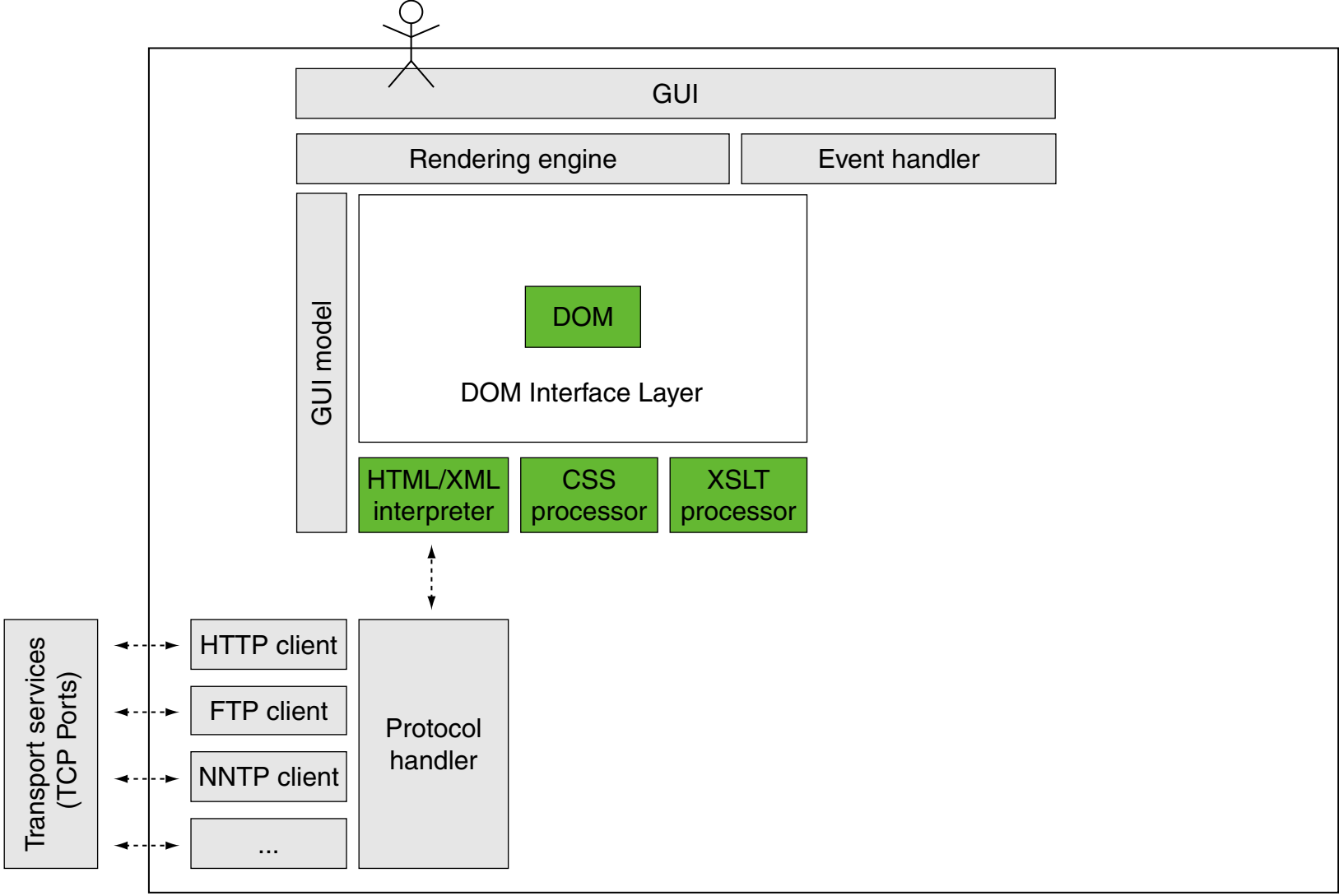
# Web-Client

Browser-Module [[MDN data flow](#)] [[W3C parsing model](#)] [[how browsers work](#)]



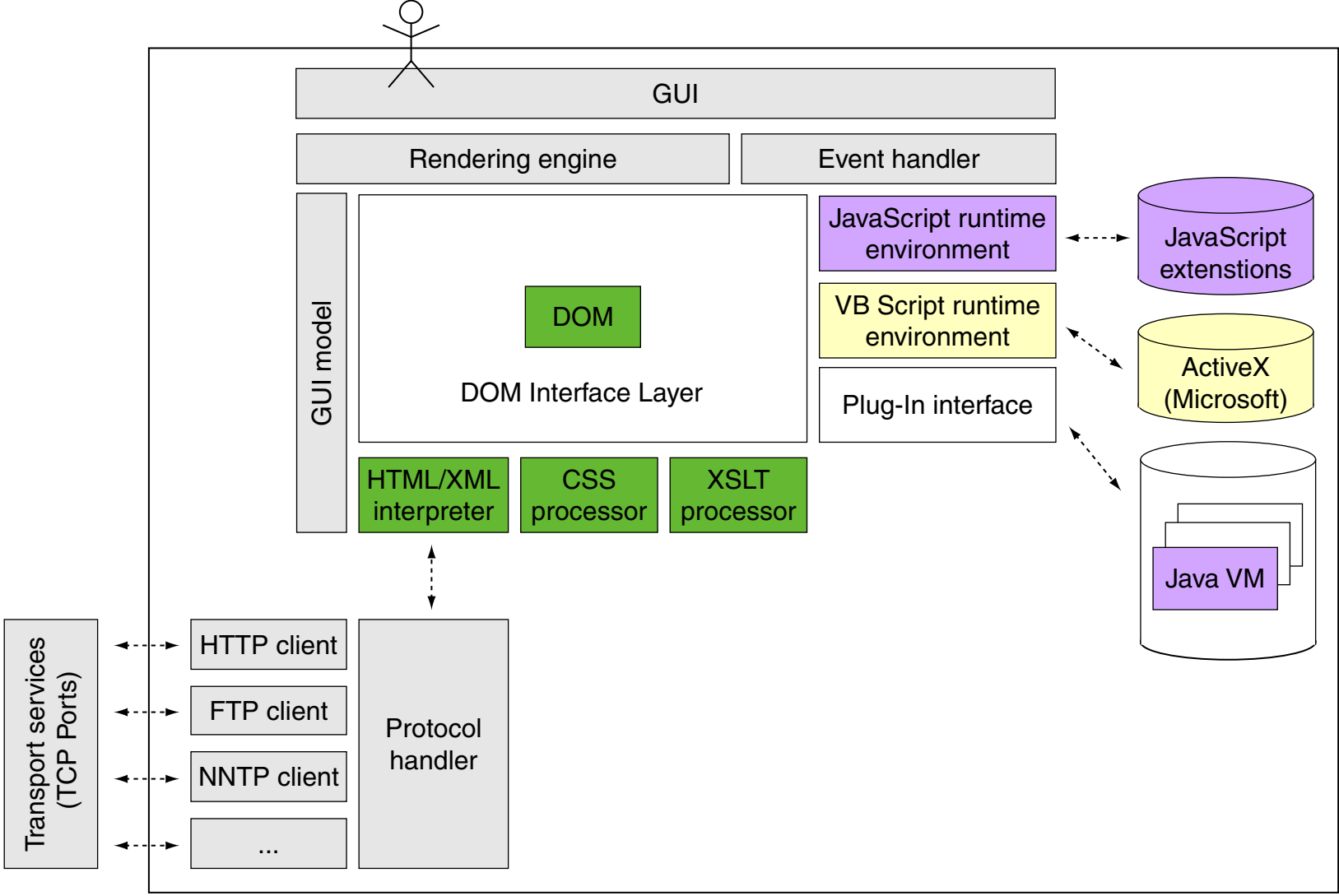
# Web-Client

Browser-Module [MDN data flow] [W3C parsing model] [how browsers work]

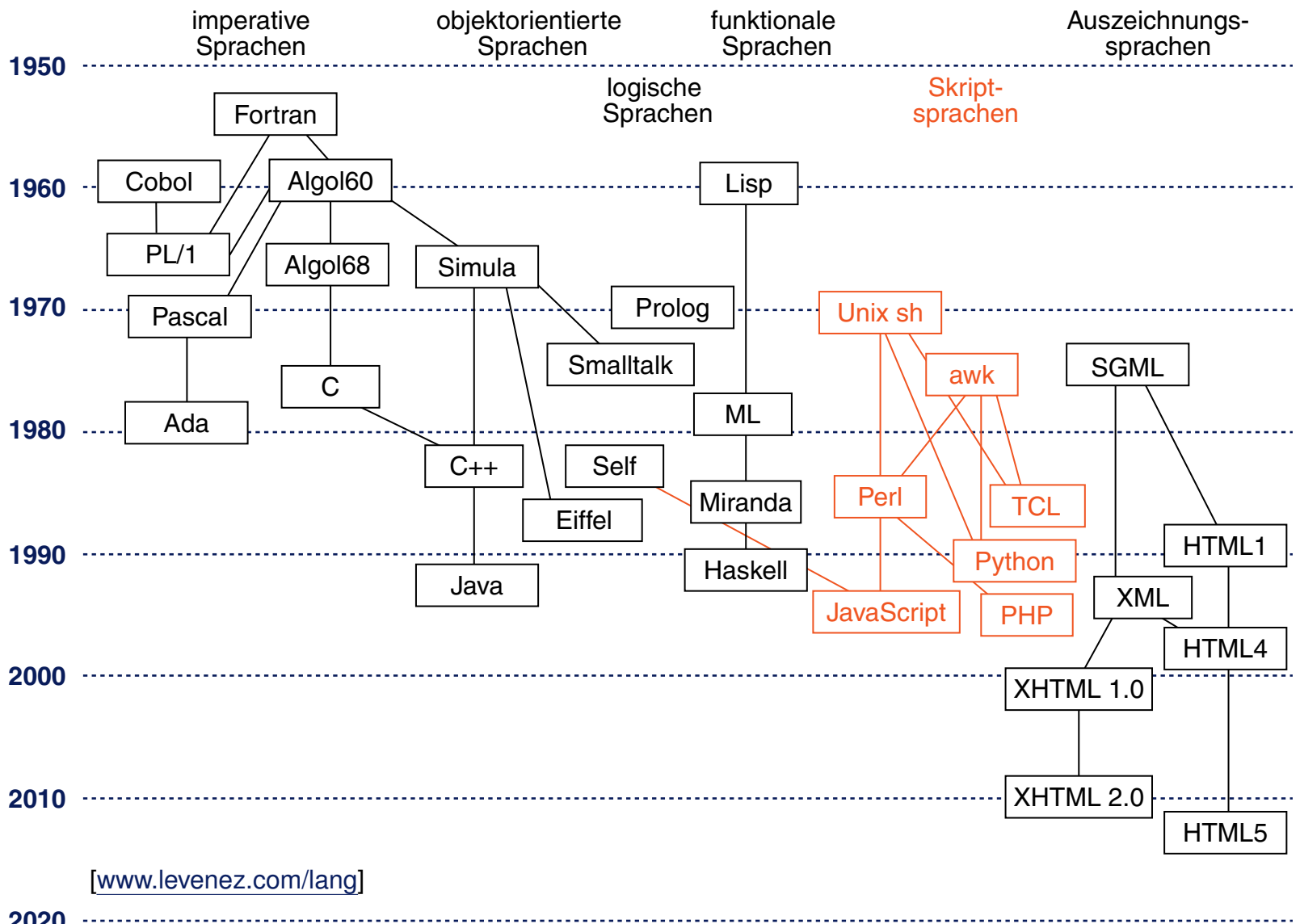


# Web-Client

Browser-Module [MDN data flow] [W3C parsing model] [how browsers work]



# Exkurs: Programmiersprachen



[[www.levenez.com/lang](http://www.levenez.com/lang)]



# Exkurs: Programmiersprachen [Kastens 2005]

## Ebenen von Spracheigenschaften

Ein Satz einer Sprache ist eine Folge von Zeichen eines gegebenen Alphabets.  
Zum Beispiel ist ein PHP-Programm ein Satz der Sprache PHP:

```
$line = fgets ( $fp , 64 ) ;
```

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebenen von Spracheigenschaften

Ein Satz einer Sprache ist eine Folge von Zeichen eines gegebenen Alphabets.  
Zum Beispiel ist ein PHP-Programm ein Satz der Sprache PHP:

```
$line = fgets ( $fp , 64 ) ;
```

Die **Struktur** eines Satzes wird auf zwei Ebenen definiert:

1. Notation von Symbolen (Lexemen, Token).
2. Syntaktische Struktur.

Die **Bedeutung** eines Satzes wird auf zwei weiteren Ebenen an Hand der Struktur für jedes Sprachkonstrukt definiert:

3. Statische Semantik.  
Eigenschaften, die *vor* der Ausführung bestimmbar sind.
4. Dynamische Semantik.  
Eigenschaften, die *erst während* der Ausführung bestimmbar sind.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 1: Notation von Symbolen

Ein Symbol wird aus einer Folge von Zeichen des Alphabets gebildet. Die Regeln zur Notation von Symbolen werden durch **reguläre Ausdrücke** definiert.

```
$line = fgets ( $fp , 64 ) ;
```

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 1: Notation von Symbolen

Ein Symbol wird aus einer Folge von Zeichen des Alphabets gebildet. Die Regeln zur Notation von Symbolen werden durch **reguläre Ausdrücke** definiert.

```
$line = fgets ($fp, 64);
```

Wichtige Symbolklassen in Programmiersprachen:

Symbolklasse	Beispiel in PHP
Bezeichner ( <i>Identifier</i> ) Verwendung: Namen für Variable, Funktionen, etc.	<code>\$line, fgets</code>
Literale ( <i>Literals</i> ) Verwendung: Zahlkonstanten, Zeichenkettenkonstanten	<code>64, "telefonbuch.txt"</code>
Wortsymbole ( <i>Keywords</i> ) Verwendung: kennzeichnen Sprachkonstrukte	<code>while, if</code>
Spezialzeichen Verwendung: Operatoren, Separatoren	<code>&lt;= = ; { }</code>

## Bemerkungen:

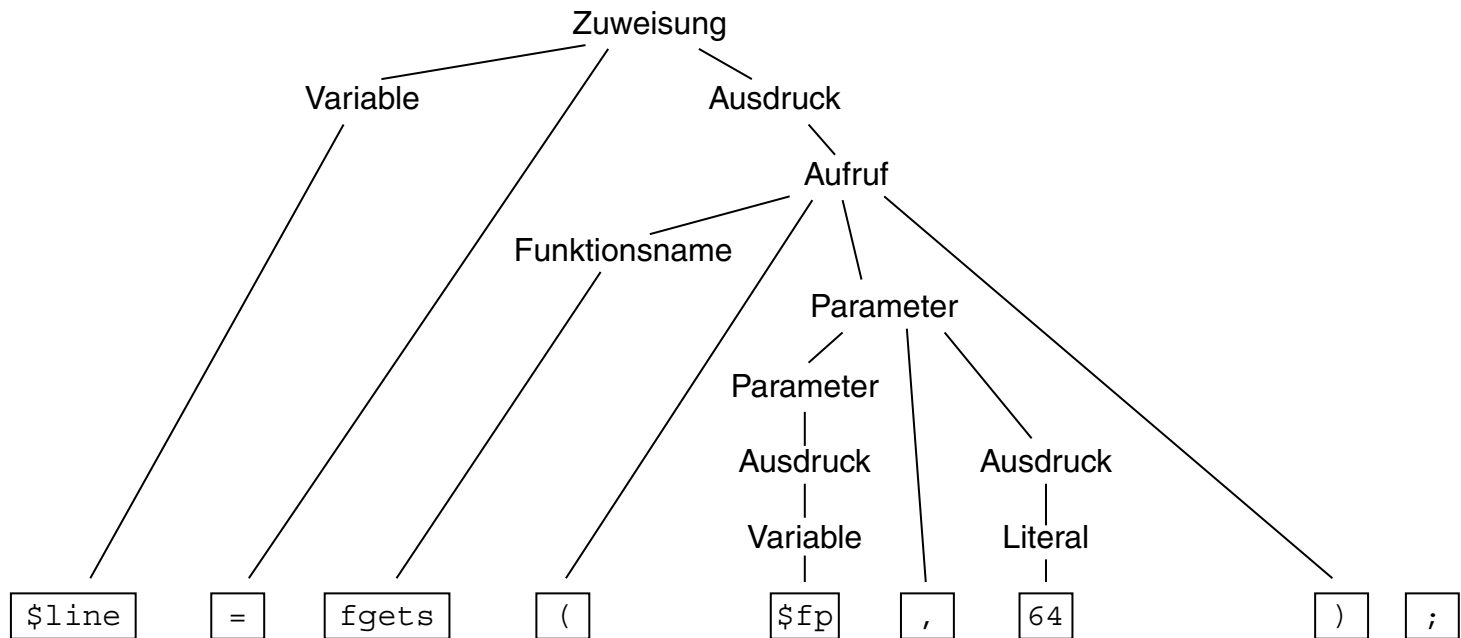
- ❑ Zwischenräume, Tabulatoren, Zeilenwechsel und Kommentare zwischen den Symbolen dienen der Lesbarkeit und sind sonst bedeutungslos.
- ❑ In Programmiersprachen bezeichnet der Begriff „Literal“ Zeichenfolgen, die zur Darstellung der Werte von Basistypen zulässig sind. Sie sind nicht benannt, werden aber über die jeweilige Umgebung ebenfalls in die Programmressourcen eingebunden. Literale können nur in rechtsseitigen Ausdrücken auftreten. Meist werden die Literale zu den Konstanten gerechnet und dann als literale Konstanten bezeichnet, da beide – im Gegensatz zu Variablen – zur Laufzeit unveränderlich sind.

Das Wort Konstante im engeren Sinn bezieht sich allerdings mehr auf in ihrem Wert unveränderliche Bezeichner, d.h., eindeutig benannte Objekte, die im Quelltext beliebig oft verwendet werden können, statt immer das gleiche Literal anzugeben. [\[Wikipedia\]](#)

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 2: Syntaktische Struktur

Ein Satz einer Sprache wird in seine Sprachkonstrukte gegliedert; sie sind meist ineinander geschachtelt. Diese syntaktische Struktur wird durch einen Strukturbaum dargestellt, wobei die Symbole durch Blätter repräsentiert sind:



Die Syntax einer Sprache wird durch eine **kontextfreie Grammatik** definiert. Die Symbole sind die Terminalsymbole der Grammatik.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 3: Statische Semantik

Eigenschaften von Sprachkonstrukten, die ihre **Bedeutung** (Semantik) beschreiben, soweit sie anhand der Programmstruktur festgestellt werden können, ohne das Programm auszuführen (= statisch).

Elemente der statischen Semantik für übersetzte Sprachen:

- Bindung von Namen.

Regeln, die einer Anwendung eines Namens seine Definition zuordnen.

Beispiel: zu dem Funktionsnamen in einem Aufruf muss es eine Funktionsdefinition mit gleichem Namen geben.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 3: Statische Semantik

Eigenschaften von Sprachkonstrukten, die ihre **Bedeutung** (Semantik) beschreiben, soweit sie anhand der Programmstruktur festgestellt werden können, ohne das Programm auszuführen (= statisch).

Elemente der statischen Semantik für übersetzte Sprachen:

- Bindung von Namen.

Regeln, die einer Anwendung eines Namens seine Definition zuordnen.

Beispiel: zu dem Funktionsnamen in einem Aufruf muss es eine Funktionsdefinition mit gleichem Namen geben.

- Typregeln.

Sprachkonstrukte wie Ausdrücke und Variablen liefern bei ihrer Auswertung einen Wert eines bestimmten Typs. Er muss im Kontext zulässig sein und kann die Bedeutung von Operationen näher bestimmen.

Beispiel: die Operanden des „\*“-Operators müssen Zahlwerte sein.



## Ebene 4: Dynamische Semantik

Eigenschaften von Sprachkonstrukten, die ihre Wirkung beschreiben und erst bei der Ausführung bestimmt oder geprüft werden können (= dynamisch).

Elemente der dynamischen Semantik:

- Regeln zur Analyse von Voraussetzungen, die für eine korrekte Ausführung eines Sprachkonstruktes erfüllt sein müssen.

Beispiel: ein numerischer Index einer Array-Indizierung, wie in `$var[$i]`, darf nicht kleiner als 0 sein.

# Exkurs: Programmiersprachen [Kastens 2005]

## Ebene 4: Dynamische Semantik

Eigenschaften von Sprachkonstrukten, die ihre Wirkung beschreiben und erst bei der Ausführung bestimmt oder geprüft werden können (= dynamisch).

Elemente der dynamischen Semantik:

- Regeln zur Analyse von Voraussetzungen, die für eine korrekte Ausführung eines Sprachkonstruktes erfüllt sein müssen.

Beispiel: ein numerischer Index einer Array-Indizierung, wie in `$var[$i]`, darf nicht kleiner als 0 sein.

- Regeln zur Umsetzung bestimmter Sprachkonstrukte.

Beispiel: Auswertung einer Zuweisung der Form

*Variable = Ausdruck*

Die Speicherstelle der Variablen auf der linken Seite wird bestimmt. Der Ausdruck auf der rechten Seite wird ausgewertet. Das Ergebnis ersetzt dann den Wert an der Stelle der Variablen. [[SELFHTML](#)]

## Bemerkungen:

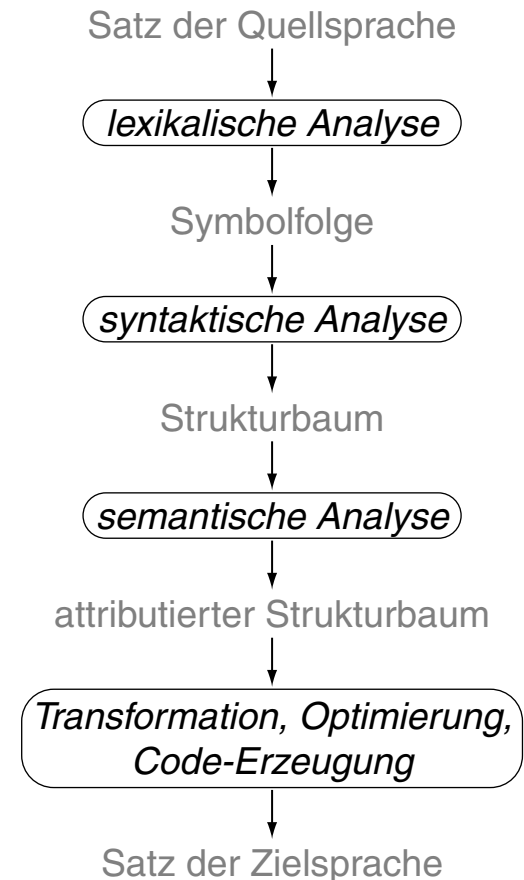
- ❑ Auf jeder der vier Ebenen gibt es also Regeln, die korrekte Sätze erfüllen müssen.
- ❑ In der Sprache PHP gehören die Typregeln zur dynamischen Semantik, da sie erst bei der Ausführung des Programms anwendbar sind.
- ❑ In der Sprache JavaScript gehören die Bindungsregeln zur statischen Semantik und die Typregeln zur dynamischen Semantik.

# Exkurs: Programmiersprachen [Kastens 2005]

## Übersetzung von Sprachen

Ein **Übersetzer** transformiert jeden korrekten Satz (Programm) der Quellsprache in einen gleichbedeutenden Satz der Zielsprache.

- ❑ Die meisten Programmiersprachen zur Software-Entwicklung werden übersetzt. Beispiele: C, C++, Java, Ada, Modula.
- ❑ Zielsprache ist dabei meist eine Maschinensprache eines realen Prozessors oder einer abstrakten Maschine.
- ❑ Übersetzte Sprachen haben eine stark ausgeprägte statische Semantik.
- ❑ Der Übersetzer prüft die Regeln der statischen Semantik; viele Arten von Fehlern lassen sich vor der Ausführung finden.



# Exkurs: Programmiersprachen [Kastens 2005]

## Interpretation von Sprachen

Ein **Interpreter** liest einen Satz (Programm) einer Sprache und führt ihn aus.

Für Sprachen, die strikt interpretiert werden, gilt:

- ❑ sie haben eine einfache Struktur und keine statische Semantik
- ❑ Bindungs- und Typregeln werden erst bei der Ausführung geprüft
- ❑ nicht ausgeführte Programmteile bleiben ungeprüft

Beispiele: Prolog, interpretiertes Lisp

Moderne Interpreter erzeugen vor der Ausführung eine interne Repräsentation des Satzes; dann können auch Struktur und Regeln der statischen Semantik vor der Ausführung geprüft werden.

Beispiele: die Skriptsprachen JavaScript, PHP, Perl

## Bemerkungen:

- ❑ Es gibt auch Übersetzer für Sprachen, die keine einschlägigen Programmiersprachen sind: Sprachen zur Textformatierung ( $\text{\LaTeX} \rightarrow \text{PDF}$ ), Spezifikationssprachen ( $\text{UML} \rightarrow \text{Java}$ ).
- ❑ Interpretierer können auf jedem Rechner verfügbar gemacht werden und lassen sich in andere Software ([Web-Browser](#)) integrieren.
- ❑ Ein Interpretierer schafft die Möglichkeit einer weiteren Kapselung der Programmausführung gegenüber dem Betriebssystem.
- ❑ Interpretation kann 10-100 mal zeitaufwändiger sein, als die Ausführung von übersetztem Maschinencode.

# JavaScript

# JavaScript [Kastens 2005]

## Einführung [[Einordnung](#)]

### Charakteristika:

- ❑ abgeleitet von Perl, Notation wie C/C++ und Java, wenig Bezug zu Java
- ❑ interpretiert, dynamisch typisiert
- ❑ spezielle objektorientierte Konzepte
- ❑ eng verknüpft mit HTML, Interpretierer in [Web-Browsern](#) integriert
- ❑ Zugriff auf Elemente des dargestellten Dokuments via DOM-API

### Anwendung:

- ❑ Programme, die im Web-Browser ausgeführt werden
- ❑ Bedienoberflächen in dynamischen Web-Seiten, Animationseffekte
- ❑ Reaktion auf Ereignisse bei der Interaktion mit Web-Seiten
- ❑ Dynamische Erzeugung von Formularelementen, Eingabeüberprüfung
- ❑ Programme, die auf dem Web-Server ausgeführt werden



# JavaScript [Kastens 2005]

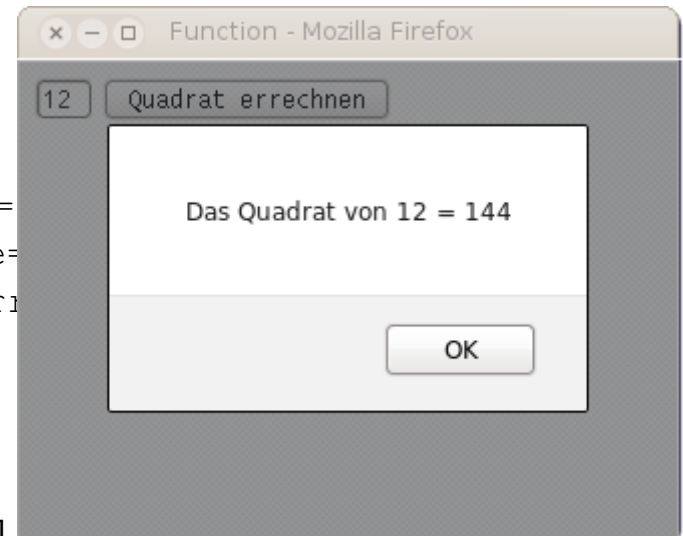
## Einführung (Fortsetzung)

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; ...">
    <title>Function</title>
    <script type="text/javascript">
      function Quadrat() {
        var Zahl = document.QuadratForm.Eingabe.value;
        var Ergebnis = Zahl * Zahl;
        alert ("Das Quadrat von " + Zahl + " = " + Ergebnis);
      }
    </script>
  </head>
  <body>
    <form name="QuadratForm" id="QF" action="">
      <input type="text" name="Eingabe" size="3">
      <input type="button" value="Quadrat errechnen" onClick="Quadrat()">
    </form>
  </body>
</html>
```

# JavaScript [Kastens 2005]

## Einführung (Fortsetzung)

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="content-type" content="text/html; ...">
    <title>Function</title>
    <script type="text/javascript">
      function Quadrat() {
        var Zahl = document.QuadratForm.Eingabe.value;
        var Ergebnis = Zahl * Zahl;
        alert ("Das Quadrat von " + Zahl + " = " + Ergebnis);
      }
    </script>
  </head>
  <body>
    <form name="QuadratForm" id="QF" action=
      <input type="text" name="Eingabe" size=
      <input type="button" value="Quadrat erre
    </form>
  </body>
</html>
```



[JavaScript-Ausführung]

## Bemerkungen:

- ❑ JavaScript kompakt:
  1. Historie
  2. Einbindung in HTML-Dokumente
  3. Grundlagen der Syntax
  4. Variablen
  5. Operatoren
  6. Datentypen
  7. Kontrollstrukturen
  8. Funktionsbibliothek
  9. Ereignisbehandlung

# JavaScript

## Historie

- 1993 NCSA Mosaic-Browser. Bilder im Fließtext, Farben für Links und Text.
- 1994 Netscape 1. Entwickelt von einer Splittergruppe des Mosaic-Teams.
- 1996 Netscape 2. Frames, JavaScript von **Brendan Eich**. JavaScript heißt zunächst LiveWire, dann LiveScript. Art des Dokumentzugriffs entspricht heutigem DOM Level 0.

# JavaScript

## Historie

- 1993 NCSA Mosaic-Browser. Bilder im Fließtext, Farben für Links und Text.
- 1994 Netscape 1. Entwickelt von einer Splittergruppe des Mosaic-Teams.
- 1996 Netscape 2. Frames, JavaScript von **Brendan Eich**. JavaScript heißt zunächst LiveWire, dann LiveScript. Art des Dokumentzugriffs entspricht heutigem DOM Level 0.
- 1996 Standardisierung der JavaScript Kernsprache durch die European Computer Manufacturers Association als **ECMAScript** in der Spezifikation **ECMA-262**. [\[MDN\]](#)
- 1997 Netscape 4. „DHTML“, basierend auf W3C CSS 1 und JavaScript 1.2.
- 1997 Internet Explorer 4. Revolutionäres Konzept für dynamische Webseiten: W3C orientiert hieran Entwicklung von DOM. Als Konkurrenz zu JavaScript entwickelt Microsoft JScript.
- 1998 Netscape-Code wird Open Source, Mozilla-Projekt wird gestartet.

# JavaScript

## Historie

- 1993 NCSA Mosaic-Browser. Bilder im Fließtext, Farben für Links und Text.
- 1994 Netscape 1. Entwickelt von einer Splittergruppe des Mosaic-Teams.
- 1996 Netscape 2. Frames, JavaScript von **Brendan Eich**. JavaScript heißt zunächst LiveWire, dann LiveScript. Art des Dokumentzugriffs entspricht heutigem DOM Level 0.
- 1996 Standardisierung der JavaScript Kernsprache durch die European Computer Manufacturers Association als **ECMAScript** in der Spezifikation **ECMA-262**. [\[MDN\]](#)
- 1997 Netscape 4. „DHTML“, basierend auf W3C CSS 1 und JavaScript 1.2.
- 1997 Internet Explorer 4. Revolutionäres Konzept für dynamische Webseiten: W3C orientiert hieran Entwicklung von DOM. Als Konkurrenz zu JavaScript entwickelt Microsoft JScript.
- 1998 Netscape-Code wird Open Source, Mozilla-Projekt wird gestartet.
- 2002 Mozilla Phoenix 0.1 (später Firefox). Open Source Rendering-Engine [Gecko](#).
- 2008 Google Chrome 1. Freie JavaScript-Engine V8 und JavaScript 1.7 ~ ECMAScript 3.
- 2010 „Letzte“ JavaScript-Version ist 1.8.5. Bezeichnung nun als ECMA-262 Editions. [\[MDN\]](#)
- 2012 Alle modernen Browser unterstützen ECMAScript 5. [\[kangax\]](#)
- 2016 Statistiken zur Verbreitung: [\[tiobe.com\]](#)

## Bemerkungen:

- ❑ Mehr zur JavaScript-Historie: [\[Tarquin\]](#) [\[SELFHTML\]](#)
- ❑ Die Entwicklung von JavaScript ist eng verknüpft mit dem „Browser-Krieg“ zwischen Microsoft und Netscape.
- ❑ Ziel der W3C-DOM-Initiative war und ist es, die Browser-Entwicklung zu vereinheitlichen. Mittlerweile ermöglichen die Browser-APIs der verschiedenen Hersteller den Zugriff auf das HTML-Dokument gemäß der DOM Level 3 Spezifikation.
- ❑ Wiederholung: W3C DOM ist nicht nur für HTML-bezogene Skriptsprachen konzipiert, sondern bezieht sich auf alle Arten von Dokumenten, die in einer SGML-basierten Sprache geschrieben sind. [\[MDN\]](#)

# JavaScript

## Einbindung in HTML-Dokumente [[SELFHTML](#)]

1. Als Script-Bereich innerhalb eines HTML-Dokuments [JavaScript-Ausführung: [1](#), [2](#)]:

```
<script type="text/javascript">  
...  
</script>
```



# JavaScript

## Einbindung in HTML-Dokumente [\[SELFHTML\]](#)

1. Als Script-Bereich innerhalb eines HTML-Dokuments [\[JavaScript-Ausführung: 1, 2\]](#) :

```
<script type="text/javascript">  
...  
</script>
```

2. Innerhalb von HTML-Tags [\[JavaScript-Ausführung\]](#) :

Verwendung im Zusammenhang mit Ereignissen (*Events*), die ein Bediener auslösen kann.

- (a) Das **Ereignis ist als Attribut** codiert; der Attributwert ist eine Anweisungsfolge, die beim Eintritt des Ereignisses ausgeführt wird:

```
<input type="button" value="..." onClick="Quadrat () ">
```

- (b) In einem Anker-Element kann – anstatt einer URL – mit `javascript:` eine Anweisungsfolge angegeben werden, die beim Klicken ausgeführt wird:

```
<a href="javascript:Quadrat () ">...</a>
```

# JavaScript

## Einbindung in HTML-Dokumente [\[SELFHTML\]](#)

1. Als Script-Bereich innerhalb eines HTML-Dokuments [\[JavaScript-Ausführung: 1, 2\]](#) :

```
<script type="text/javascript">  
...  
</script>
```

2. Innerhalb von HTML-Tags [\[JavaScript-Ausführung\]](#) :

Verwendung im Zusammenhang mit Ereignissen (*Events*), die ein Bediener auslösen kann.

- (a) Das **Ereignis ist als Attribut** codiert; der Attributwert ist eine Anweisungsfolge, die beim Eintritt des Ereignisses ausgeführt wird:

```
<input type="button" value="..." onClick="Quadrat () ">
```

- (b) In einem Anker-Element kann – anstatt einer URL – mit `javascript:` eine Anweisungsfolge angegeben werden, die beim Klicken ausgeführt wird:

```
<a href="javascript:Quadrat () ">...</a>
```

3. In einer separaten Datei [\[Source, JavaScript-Ausführung\]](#) :

```
<script src="usage.js" type="text/javascript"></script>
```

## Bemerkungen:

- ❑ Das `<script>`-Element kann mehrfach in einem HTML-Dokument verwendet werden.
- ❑ Der JavaScript-Code eines Dokuments wird beim Einlesen des Dokuments vom Browser sofort ausgeführt.
- ❑ Die Auswertung einer Funktions*definition* erzeugt keine Ausgabe und liefert auch keinen Return-Wert.
- ❑ Es gibt keine Vorschrift dafür, an welcher Stelle in einem HTML-Dokument ein JavaScript-Bereich definiert werden darf. Aus Sicht der Ladezeit kann es sinnvoll sein, diesen Bereich am Ende eines HTML-Dokuments zu platzieren.
- ❑ Eine separate Datei mit JavaScript-Code sollte die Dateinamenerweiterung `.js` besitzen; insbesondere darf diese Datei nur JavaScript-Code enthalten. Das Encoding der Datei kann im einbindenden `<script>`-Element per `charset`-Attribut spezifiziert werden.

Die Notation ähnelt in vieler Hinsicht der von C++ und Java.

### Bezeichner

- einheitliche Schreibweise für alle Arten von Bezeichnern:

```
identifizier = { letter | $ | _ } { letter | $ | _ | digit }*
```

- Groß-/Kleinschreibung wird unterschieden (*case sensitive*)

# JavaScript [Kastens 2005]

## Grundlagen der Syntax [PHP]

Die Notation ähnelt in vieler Hinsicht der von C++ und Java.

### Bezeichner

- einheitliche Schreibweise für alle Arten von Bezeichnern:

```
identifizier = { letter | $ | _ } { letter | $ | _ | digit }*
```

- Groß-/Kleinschreibung wird unterschieden (*case sensitive*)

### Anweisungen [SELFHTML]

- Ein Semikolon am Zeilenende ist möglich, kann aber entfallen.
- Zwischen Anweisungen in derselben Zeile muss ein Semikolon stehen.
- `//` kommentiert bis Zeilenende aus.
- Balancierte Kommentarklammerung: `/* Kommentar */`

- ❑ Zur Deklaration von Variablen dient das Schlüsselwort `var`.
- ❑ Eine Variable kann Werte beliebigen Typs annehmen.
- ❑ Unterscheidung von **lokalen** und **globalen** Variablen.
- ❑ Eine Variable ist lokal für eine Funktion, wenn sie innerhalb des Bindungsbereiches der Funktion mit `var` deklariert wird.
- ❑ Globale Variablen gelten im ganzen Programm – es sei denn, sie werden von einer lokalen Variable überdeckt; lokale Variablen gelten im Bindungsbereich der Funktion, in der sie deklariert sind.

- ❑ Zur Deklaration von Variablen dient das Schlüsselwort `var`.
- ❑ Eine Variable kann Werte beliebigen Typs annehmen.
- ❑ Unterscheidung von **lokalen** und **globalen** Variablen.
- ❑ Eine Variable ist lokal für eine Funktion, wenn sie innerhalb des Bindungsbereiches der Funktion mit `var` deklariert wird.
- ❑ Globale Variablen gelten im ganzen Programm – es sei denn, sie werden von einer lokalen Variable überdeckt; lokale Variablen gelten im Bindungsbereich der Funktion, in der sie deklariert sind.

## Variablen: Illustration von Geltungsbereichen

```
var line;  
var sum, result;  
var col, count = 3, row;  
var minimum = count,  
    maximum = 999;
```

```
function compute (n)  
{  
  var sum = n;  
  sum = sum * col;  
  return sum;  
}
```



## Variablen: Illustration von Geltungsbereichen

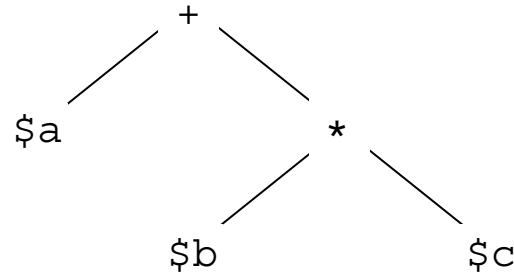
```
var line;
var sum, result;
var col, count = 3, row;
var minimum = count,
    maximum = 999;

function compute (n) // n ist lokale Variable.
{
  var sum = n;       // sum ist lokale Variable.
  sum = sum * col;   // col ist globale Variable.
  return sum;
}
```

## Operatoren: Präzedenz, Assoziativität

Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz. Durch Klammerung lässt sich die Präzedenz in Termen vorschreiben. Beispiel:

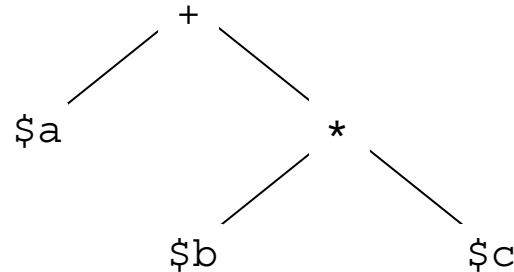
`$a + $b * $c`



## Operatoren: Präzedenz, Assoziativität

Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz. Durch Klammerung lässt sich die Präzedenz in Termen vorschreiben. Beispiel:

$\$a + \$b * \$c$

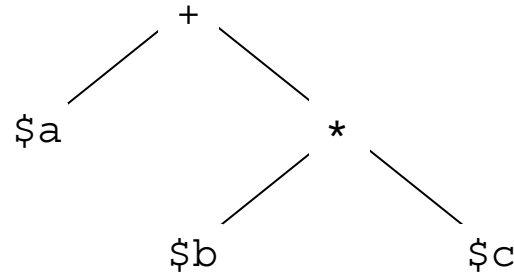


Ein Operator ist linksassoziativ (rechtsassoziativ), wenn beim Zusammentreffen von Operatoren **gleicher Präzedenz** der linke (rechte) Operator seine Operanden stärker bindet als der rechte (linke).

## Operatoren: Präzedenz, Assoziativität

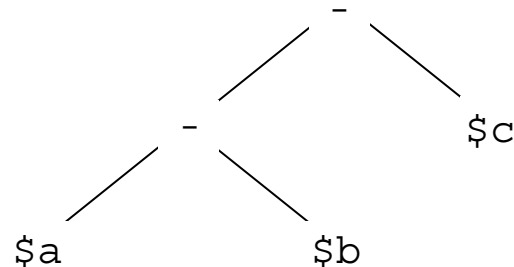
Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz. Durch Klammerung lässt sich die Präzedenz in Termen vorschreiben. Beispiel:

$\$a + \$b * \$c$



Ein Operator ist linksassoziativ (rechtsassoziativ), wenn beim Zusammentreffen von Operatoren **gleicher Präzedenz** der linke (rechte) Operator seine Operanden stärker bindet als der rechte (linke). Beispiel:

$\$a - \$b - \$c$



Präzedenz	Stelligkeit	Assoziativität	Operatoren	Erklärung
1	2	rechts	= += -=	Zuweisungsoperatoren
2	3	links	? :	bedingter Ausdruck
3	2	links		logische Disjunktion
4	2	links	&&	logische Konjunktion
5	2	links		Bitoperator
6	2	links	^	Bitoperator
7	2	links	&	Bitoperator
8	2	links	== != ===	Gleichheit, Identität
9	2	links	< <= > >=	Ordnungsvergleich
10	2	links	<< >> >>>	shift-Operatoren
11	2	links	+ -	Konkatenation, Add., Subtr.
12	2	links	* / %	Arithmetik
13	1		! - ~	Negation (logisch, arithm.)
	1		++ -	Inkrement, Dekrement
	1		typeof void	Typabfr., zu undefined konv.
14	1		() [] .	Aufruf, Index, Objektzugriff

# JavaScript [Kastens 2005]

## Datentypen: Primitive [PHP]

### number

- ❑ Keine Unterscheidung zwischen Ganzzahlen und Gleitpunktzahlen.
- ❑ Der Wert `NaN` (*not a number*) steht für ein undefiniertes Ergebnis.
- ❑ Der Wert `Infinity` steht für einen Wert, der größer als die größte repräsentierbare Zahl ist.

### string

- ❑ Zeichenkettenliterals mit einfachen oder doppelten Anführungszeichen.
- ❑ Konkatenation wie in Java: `var s = "Hello" + "world!"`
- ❑ Zeichenkettenfunktionen werden in objektorientierter Notation verwendet.  
Beispiel: `s.length`, `s.indexOf(substr)`, `s.charAt(i)`.

### boolean

- ❑ **Literale:** `true` und `false` (insbesondere nicht: `True` bzw. `False`)
- ❑ **Operatoren:** Konjunktion `&&`, Disjunktion `||`, Negation `!`

`undefined`

- ❑ Der Wert `undefined` steht dafür, dass eine Variable keinen Wert hat.
- ❑ `undefined` wird zurückgegeben, falls (a) eine Variable benutzt wird, die zwar deklariert aber der nie ein Wert zugewiesen wurde oder (b) auf eine Objektkomponente (s.u.) zugegriffen wird, die nicht existiert.

`null`

- ❑ Der Wert `null` steht dafür, dass eine Variable keinen gültigen Wert hat.
- ❑ Obwohl `null` und `undefined` verschiedene Werte sind, werden sie vom Operator `==` als gleich interpretiert.

Eine Funktion ist ein Stück ausführbarer Code, der in einem JavaScript-Programm definiert ist. Funktionen sind Objekte vom Typ `Function`.

Definition von Funktionen:

(a) Als `function`-Statement:

(b) Durch Verwendung des `function`-Statements als Funktionsliteral:

(c) Mit dem **Konstruktor** `Function()`:



Eine Funktion ist ein Stück ausführbarer Code, der in einem JavaScript-Programm definiert ist. Funktionen sind Objekte vom Typ `Function`.

Definition von Funktionen:

(a) Als `function`-Statement:

```
function f(x, y) {  
    return x * y;  
}
```

(b) Durch Verwendung des `function`-Statements als Funktionsliteral:

(c) Mit dem **Konstruktor** `Function()`:

Eine Funktion ist ein Stück ausführbarer Code, der in einem JavaScript-Programm definiert ist. Funktionen sind Objekte vom Typ `Function`.

### Definition von Funktionen:

- (a) Als `function`-Statement:

```
function f(x, y) {  
    return x * y;  
}
```

- (b) Durch Verwendung des `function`-Statements als **Funktionsliteral**:

```
var p = function (x, y) { return x * y; };    // (anonym)
```

oder

```
var p = function fn(x, y) { return x * y; }; // (benannt)
```

- (c) Mit dem **Konstruktor** `Function()`:

Eine Funktion ist ein Stück ausführbarer Code, der in einem JavaScript-Programm definiert ist. Funktionen sind Objekte vom Typ `Function`.

### Definition von Funktionen:

(a) Als `function`-Statement:

```
function f(x, y) {  
    return x * y;  
}
```

(b) Durch Verwendung des `function`-Statements als **Funktionsliteral**:

```
var p = function (x, y) { return x * y; };    // (anonym)
```

oder

```
var p = function fn(x, y) { return x * y; }; // (benannt)
```

(c) Mit dem **Konstruktor** `Function()`:

```
var p = new Function("x", "y", "return x * y;");
```

Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`.

Definition von Objekten:

- (a) Durch Verwendung des Block-Statements `{ }` als Objekliteral:
  
- (b) Mit dem **Konstruktor** `Object ()`:

Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`.

Definition von Objekten:

- (a) Durch Verwendung des Block-Statements `{ }` als **Objektliteral**:

```
var student = { matrNr:4711, name:"P. Müller" };
```

- (b) Mit dem **Konstruktor** `Object ( )`:

# JavaScript [Kastens 2005]

## Datentypen: Objekte [vordefinierte Objekte]

Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`.

### Definition von Objekten:

- (a) Durch Verwendung des Block-Statements `{ }` als **Objektliteral**:

```
var student = { matrNr:4711, name:"P. Müller" };
```

- (b) Mit dem **Konstruktor** `Object()`:

```
var student = new Object();
```

#### Hinzufügen von Komponenten:

```
student.matrNr = 4711;  
student.name = "P. Müller";
```

# JavaScript [Kastens 2005]

## Datentypen: Objekte [vordefinierte Objekte]

Objekte bestehen aus Komponenten, die jeweils einen Namen und einen Wert haben. Objekte sind vom Typ `Object`.

### Definition von Objekten:

- (a) Durch Verwendung des Block-Statements `{ }` als **Objektliteral**:

```
var student = { matrNr:4711, name:"P. Müller" };
```

- (b) Mit dem **Konstruktor** `Object()`:

```
var student = new Object();
```

Hinzufügen von Komponenten:

```
student.matrNr = 4711;  
student.name = "P. Müller";
```

### Zugriff auf Objektkomponenten mit *Objektausdruck.Name*:

`student.matrNr` liefert 4711.

## Bemerkungen:

- ❑ Objektkomponenten können Funktionen sein und heißen dann Methoden; ihre Aufrufe können die Komponenten des Objekts lesen oder verändern.
- ❑ Komponenten, die keine Methoden sind, heißen Eigenschaften.



Benutzerdefinierte Objekte können auch durch die Definition und Anwendung eines Konstruktors erzeugt werden:

```
function MyStudent(m, n) {  
  this.matrNr = m;  
  this.name = n;  
}  
  
var student = new MyStudent(4711, "P. Müller")
```

Definition von Methoden **separat** oder als **Funktionsliteral** [[JavaScript-Ausführung](#)] :

```
function MyCircle(r) {
  this.radius = r;
  this.area = getArea;
  this.circ = function() {return (Math.PI * this.radius * 2);};
}

function getArea() {
  return (Math.PI * this.radius * this.radius);
}
```

# JavaScript [Kastens 2005]

## Datentypen: Objekte (Fortsetzung)

Definition von Methoden **separat** oder als **Funktionsliteral** [[JavaScript-Ausführung](#)] :

```
function MyCircle(r) {  
    this.radius = r;  
    this.area = getArea;  
    this.circ = function() {return (Math.PI * this.radius * 2);};  
}  
  
function getArea() {  
    return (Math.PI * this.radius * this.radius);  
}
```

### Aufruf:

```
c = new MyCircle(3);  
document.writeln("Radius = " + c.radius);  
document.writeln("Area = " + c.area());  
document.writeln("Circumference = " + c.circ());
```

Definition von Methoden **separat** oder als **Funktionsliteral** [\[JavaScript-Ausführung\]](#) :

```
function MyCircle(r) {
  this.radius = r;
  this.area = getArea;
  this.circ = function() {return (Math.PI * this.radius * 2);};
}

function getArea() {
  return (Math.PI * this.radius * this.radius);
}
```

Aufruf:

```
c = new MyCircle(3);
document.writeln("Radius = " + c.radius);
document.writeln("Area = " + c.area());
document.writeln("Circumference = " + c.circ());
```

Zugriff auf eine Methode ohne „()“ liefert die Funktionsdefinition:

```
c.area -> function getArea() {return (Math.PI * this.radius * this.radius);}
c.circ -> function {return (Math.PI * this.radius * 2);}
```

## Bemerkungen:

- ❑ Die Syntax einer Konstruktordefinition folgt der Syntax einer Funktionsdefinition.
- ❑ Im Konstruktor sind die Komponenten mit dem Qualifier `this` zu versehen; er bezeichnet die jeweilige mit `new` erzeugte Objektinstanz.
- ❑ Per Konvention beginnt der Name einer Konstruktorfunktion mit einem Großbuchstaben.

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert. Arrays sind Objekte vom Typ `Array`.

Erzeugung von Arrays mit dem **Konstruktor** `Array()`:

- (a) Als Liste von Werten, indiziert von 0 an:
  
- (b) Durch Erweiterung eines leeren Arrays:
  
- (c) Als assoziatives Array:

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert. Arrays sind Objekte vom Typ `Array`.

Erzeugung von Arrays mit dem **Konstruktor** `Array()`:

- (a) Als Liste von Werten, indiziert von 0 an:

```
monatsName = new Array("", "Jan", ..., "Dez");
```

- (b) Durch Erweiterung eines leeren Arrays:

```
monatsName = new Array();  
monatsName[1] = "Jan"; monatsName[2] = "Feb"; ...
```

- (c) Als assoziatives Array:

```
monatsNr = new Array();  
monatsNr["Jan"] = 1; monatsNr["Feb"] = 2; ...
```

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert. Arrays sind Objekte vom Typ `Array`.

Erzeugung von Arrays mit dem **Konstruktor** `Array()`:

- (a) Als Liste von Werten, indiziert von 0 an:

```
monatsName = new Array("", "Jan", ..., "Dez");
```

- (b) Durch Erweiterung eines leeren Arrays:

```
monatsName = new Array();  
monatsName[1] = "Jan"; monatsName[2] = "Feb"; ...
```

- (c) Als assoziatives Array:

```
monatsNr = new Array();  
monatsNr["Jan"] = 1; monatsNr["Feb"] = 2; ...
```

Aufzählung aller Elemente **mit Schlüssel**:

```
for (mname in monatsNr) {  
    document.writeln (mname + "->" + monatsNr[mname] + "<br/>");  
}
```



# JavaScript [Kastens 2005]

## Kontrollstrukturen [\[PHP\]](#) [\[SELFHTML\]](#)

- ❑ Anweisungsfolge:
- ❑ Bedingte Anweisung:
- ❑ Return-Anweisung:
- ❑ `while`-Schleife:
- ❑ `for`-Schleife [\[JavaScript-Ausführung\]](#) :

### □ Anweisungsfolge:

```
{ var k = 42; document.writeln (5*k); }
```

Eine Anweisungsfolge definiert keinen *Scope*: eine `var`-Deklaration gilt nicht nur in der Anweisungsfolge, sondern in umgebender Funktion bzw. im umgebenden Programm.

### □ Bedingte Anweisung:

```
if (a < b) {min = a;} else {min = b;}
```

Bei einzelnen Anweisungen sind die `{}`-Klammern optional.

### □ Return-Anweisung:

```
return n*42;      return "*";
```

### □ while-Schleife:

### □ for-Schleife [\[JavaScript-Ausführung\]](#) :

### □ Anweisungsfolge:

```
{ var k = 42; document.writeln (5*k); }
```

Eine Anweisungsfolge definiert keinen *Scope*: eine `var`-Deklaration gilt nicht nur in der Anweisungsfolge, sondern in umgebender Funktion bzw. im umgebenden Programm.

### □ Bedingte Anweisung:

```
if (a < b) {min = a;} else {min = b;}
```

Bei einzelnen Anweisungen sind die `{}`-Klammern optional.

### □ Return-Anweisung:

```
return n*42;      return "*";
```

### □ while-Schleife:

```
i = 0; while (i < n) {document.write ("*"); i++;}
```

### □ for-Schleife [\[JavaScript-Ausführung\]](#) :

```
for (i = 0; i < n; i++) {document.write ("*");}
```

### □ Anweisungsfolge:

```
{ var k = 42; document.writeln (5*k); }
```

Eine Anweisungsfolge definiert keinen *Scope*: eine `var`-Deklaration gilt nicht nur in der Anweisungsfolge, sondern in umgebender Funktion bzw. im umgebenden Programm.

### □ Bedingte Anweisung:

```
if (a < b) {min = a;} else {min = b;}
```

Bei einzelnen Anweisungen sind die `{}`-Klammern optional.

### □ Return-Anweisung:

```
return n*42;      return "*";
```

### □ while-Schleife:

```
i = 0; while (i < n) {document.write ("*"); i++;}
```

### □ for-Schleife [\[JavaScript-Ausführung\]](#) :

```
for (i = 0; i < n; i++) {document.write ("*");}
```

### □ Parameterübergabe standardmäßig mittels **call-by-value**.

# JavaScript

## Funktionsbibliothek [\[PHP\]](#)

Ein Großteil der Funktionsbibliothek ist in Form von Objekten implementiert.

Unterscheidung von drei Arten vordefinierter Objekte:

1. Kernobjekte. Funktionalität unabhängig von Browser und Dokument.

Beispiele: `Array`, `Date`, `Function`, `Math`, `Object`, `RegExp` [\[MDN\]](#)

# JavaScript

## Funktionsbibliothek [\[PHP\]](#)

Ein Großteil der Funktionsbibliothek ist in Form von Objekten implementiert.  
Unterscheidung von drei Arten vordefinierter Objekte:

1. Kernobjekte. Funktionalität unabhängig von Browser und Dokument.

Beispiele: `Array`, `Date`, `Function`, `Math`, `Object`, `RegExp` [\[MDN\]](#)

2. DOM-Objekte. Repräsentation von und Zugriff auf Dokumente.

Beispiele: `Document`, `Element`, `Node` [\[MDN\]](#)

(a) DOM-Objekte nach der W3C Spezifikation. Zugriff über DOM-API.

(b) DOM-Objekte nach Netscape. Zugriff über vordefinierte Arrays. (deprecated)

(c) DOM-Objekte nach Microsoft. Zugriff über das `document.all`-Objekt. (deprecated)

# JavaScript

## Funktionsbibliothek [\[PHP\]](#)

Ein Großteil der Funktionsbibliothek ist in Form von Objekten implementiert.  
Unterscheidung von drei Arten vordefinierter Objekte:

1. Kernobjekte. Funktionalität unabhängig von Browser und Dokument.

Beispiele: `Array`, `Date`, `Function`, `Math`, `Object`, `RegExp` [\[MDN\]](#)

2. DOM-Objekte. Repräsentation von und Zugriff auf Dokumente.

Beispiele: `Document`, `Element`, `Node` [\[MDN\]](#)

(a) DOM-Objekte nach der W3C Spezifikation. Zugriff über DOM-API.

(b) DOM-Objekte nach Netscape. Zugriff über vordefinierte Arrays. (deprecated)

(c) DOM-Objekte nach Microsoft. Zugriff über das `document.all`-Objekt. (deprecated)

3. Browser-Objekte. Repräsentation von und Zugriff auf Browser-Fenster.

Beispiele: `History`, `Location`, `Navigator`, `Screen`, `Window` [\[MDN\]](#)

# JavaScript

## Funktionsbibliothek [\[PHP\]](#)

Ein Großteil der Funktionsbibliothek ist in Form von Objekten implementiert.

Unterscheidung von drei Arten vordefinierter **Objekte**:

1. Kernobjekte. Funktionalität unabhängig von Browser und Dokument.

Beispiele: `Array`, `Date`, `Function`, `Math`, `Object`, `RegExp` [\[MDN\]](#)

2. DOM-Objekte. Repräsentation von und Zugriff auf Dokumente.

Beispiele: `Document`, `Element`, `Node` [\[MDN\]](#)

(a) **DOM-Objekte nach der W3C Spezifikation. Zugriff über DOM-API.**

(b) DOM-Objekte nach Netscape. Zugriff über vordefinierte Arrays. (deprecated)

(c) DOM-Objekte nach Microsoft. Zugriff über das `document.all`-Objekt. (deprecated)

3. Browser-Objekte. Repräsentation von und Zugriff auf Browser-Fenster.

Beispiele: `History`, `Location`, `Navigator`, `Screen`, `Window` [\[MDN\]](#)

Ergänzung durch vordefinierte **Funktionen**.

Beispiele: `eval`, `isFinite`, `isNaN`, `parseInt` [\[MDN\]](#)



## Bemerkungen:

- ❑ Mehr zu JavaScript-Objekten: [\[w3schools\]](#)
- ❑ Kernobjekte werden auch als *Global Objects*, *Objects in the Global Scope* oder *Built-in Objects* bezeichnet. Vordefinierte Funktionen werden auch als *Global Functions* oder *Built-in Functions* bezeichnet.
- ❑ DOM-Objekte und Browser-Objekte sind Teil derselben Hierarchie. Das höchste Objekt ist das `window`-Objekt; eines seiner Kindobjekte ist das `document`-Objekt (= *Wurzelknoten* des HTML/XML-Dokuments). [\[SELFHTML\]](#)
- ❑ Das `document`-Objekt bildet das oberste Ausgangsobjekt für das Document Object Model, DOM. Eines der Kindobjekte des `document`-Objekts wiederum ist das `documentElement`-Objekt (= *Wurzelement* des HTML/XML-Dokuments). [\[SELFHTML\]](#)
- ❑ Streng genommen müsste statt `document.write()` ausführlich `window.document.write()` bzw. allgemein `window.document.component()` notiert werden. Dass die erste Notationsvariante funktioniert, zeigt, dass per Default die DOM-Hierarchie den Ausgangspunkt darstellt.
- ❑ Dadurch, dass der Wert einer Eigenschaft wiederum ein Objekt sein darf, kann der Zugriff auf einen bestimmten Wert eines „innenliegenden“ Objekts tief geschachtelt sein:  
`window.document.images.length`
- ❑ HTML 5 enthält verbindliche Standards für Browser-Objekte wie das `window`-Objekt. [\[W3C\]](#)

# JavaScript

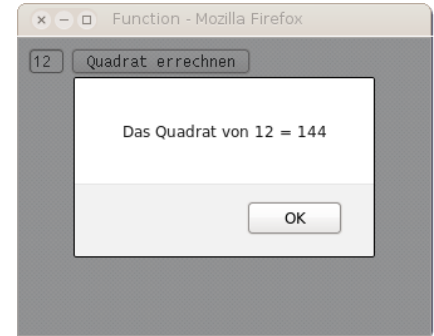
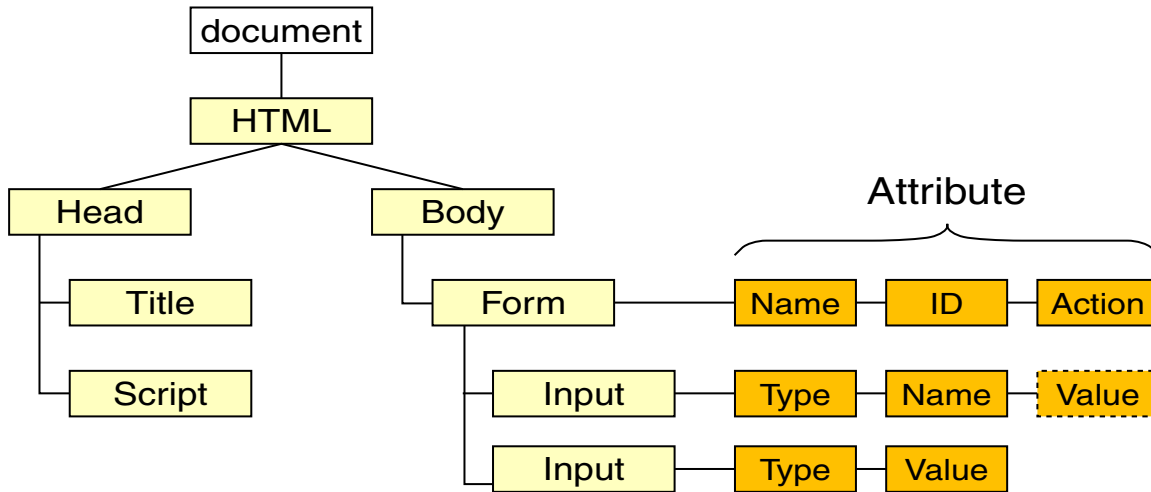
## Funktionsbibliothek: DOM-Objekte

Konzepte, um auf HTML-Elementobjekte und deren Komponenten zuzugreifen:

- ❑ Eindeutig qualifizierender Name gemäß DOM-Dokumentbaum.
- ❑ DOM-API, um Elementmenge auszuwählen [\[\(a\)\]](#) :
  - `getElementByName()`
  - `getElementById()`
  - `getElementsByTagName()`
  - ...
- ❑ Vordefinierte Arrays, die alle Elemente eines Typs enthalten [\[\(b\)\]](#) :
  - `document.forms`
  - `document.images`
  - ...
- ❑ Kombiniert: Auswahl von Elementmenge plus weitere Spezialisierung mittels qualifizierendem Namen.

# JavaScript

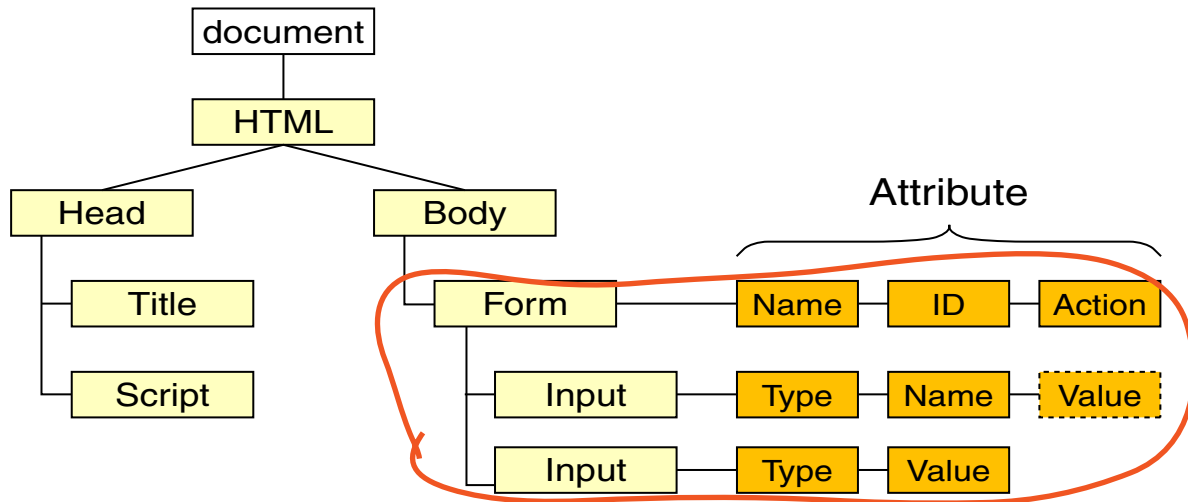
## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [[JavaScript-Einführungsbeispiel](#)]



```
<!DOCTYPE html>
<html>
  <head>
    ...
    <script type="text/javascript">
      function Quadrat() {...}
    </script>
  </head>
  <body>
    <form name="QuadratForm" id="QF" action="">
      <input type="text" name="Eingabe" size="3">
      <input type="button" value="Quadrat errechnen" onClick="Quadrat()">
    </form>
  </body>
</html>
```

# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [\[JavaScript-Einführungsbeispiel\]](#)



### Zugriffsmöglichkeiten auf das erste **Formelement**:

`document.QuadratForm`

`document.getElementsByTagName("form")[0]`

`document.getElementById("QF")`

`document.forms[0]`

qualifizierender Name

DOM-API

DOM-API

vordefiniertes Array

### Kontrollausgaben:

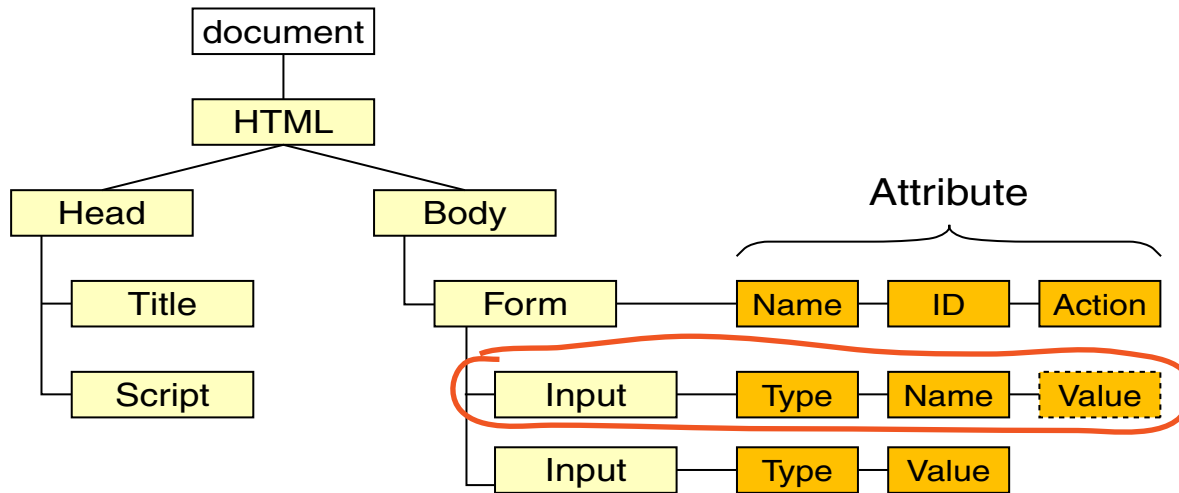
`document.writeln(document.QuadratForm) -> [object HTMLFormElement]`

`document.writeln(document.getElementsByTagName("form")) -> [object HTMLCollection]`

`document.writeln(document.getElementsByTagName("form")[0]) -> [object HTMLFormElement]`

# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [[JavaScript-Einführungsbeispiel](#)]



### Zugriffsmöglichkeiten auf das erste Eingabeelement:

```
document.QuadratForm.Eingabe
```

```
document.getElementsByTagName("form")[0].Eingabe
```

```
document.getElementsByName("Eingabe")[0]
```

```
document.forms[0].Eingabe
```

qualifizierender Name

DOM-API

DOM-API

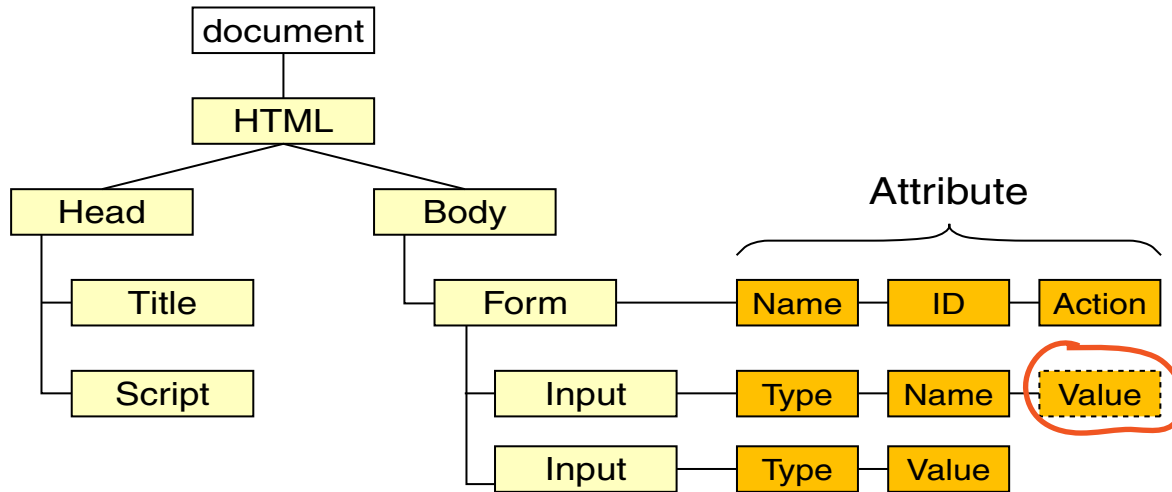
vordefiniertes Array

### Kontrollausgaben:

```
document.writeln(document.QuadratForm.Eingabe) -> [object HTMLInputElement]
```

# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [[JavaScript-Einführungsbeispiel](#)]



Zugriffsmöglichkeiten auf das **Eingabefeld** im ersten Eingabeelement:

```
document.QuadratForm.Eingabe.value  
document.getElementsByTagName("form")[0].Eingabe.value  
document.getElementsByName("Eingabe")[0].value  
document.forms[0].Eingabe.value
```

Kontrollausgaben:

```
document.writeln(document.QuadratForm.Eingabe.value) -> 11
```

# JavaScript

## Funktionsbibliothek: DOM-Objekte (Fortsetzung) [[JavaScript-Einführungsbeispiel](#)]

```
<script type="text/javascript">
  function Quadrat() {
    var Zahl = document.QuadratForm.Eingabe.value;
    var Ergebnis = Zahl * Zahl;
    alert ("Das Quadrat von " + Zahl + " = " + Ergebnis);
  }
</script>

<form name="QuadratForm" id="QF" action="">
  <input type="text" name="Eingabe" size="3">
  <input type="button" value="Quadrat errechnen" onClick="Quadrat()">
</form>
```

Genauso wie das Abfragen ist auch das Setzen von Werten möglich:

```
document.QuadratForm.Eingabe.value = 12;
```

## Bemerkungen:

- ❑ Zugriffe auf Dokumentbestandteile erfolgen gemäß der DOM-API – entweder über Methoden des generischen `Node`-Interfaces wie `getChildNodes()` oder über Methoden des `Document`-Interfaces wie `getElementsByName()`, `getElementsByTagName()` oder `getElementById()`. [\[SELFHTML\]](#)
- ❑ Jedes erlaubte Attribut eines HTML-Elements steht als Eigenschaft im entsprechenden DOM-Objekt zur Verfügung. So hat beispielsweise das HTML-Element `<input>` ein erlaubtes Attribut `value`. In der DOM-Hierarchie gibt es somit ein `input`-Elementobjekt mit der Eigenschaft `value`.
- ❑ DOM definiert für einige der HTML-Element-Objekte auch Methoden. So gibt es für das DOM-Objekt des HTML-Elements `<form>` die Methoden `reset()` und `submit()`. [\[SELFHTML\]](#)



## Ereignisbehandlung

Ein Ereignis (*Event*) ist die Wahrnehmung einer Zustandsänderung. Die ereignisgetriebene Programmierung ordnet den Ereignissen Operationen zu.

## Ereignisbehandlung

Ein Ereignis (*Event*) ist die Wahrnehmung einer Zustandsänderung. Die ereignisgetriebene Programmierung ordnet den Ereignissen Operationen zu.

Möglichkeiten für die Behandlung des Ereignisses „Mausklick“ in

<form>-Elementen:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Event</title>
  </head>
  <body>
    <form name="testForm">
      <input type="button" value="ping" onclick='alert("ping!");'>
      <input type="button" name="Knopf" value="pong">
    </form>
    <script type="text/javascript">
      document.testForm.Knopf.onclick = function() { alert("pong!"); };
    </script>
  </body>
</html>
```



## Bemerkungen:

- ❑ Beachte die unterschiedliche Art und Weise, wie die Funktionen als Wert des `onclick`-Attributes zugewiesen werden.
- ❑ Mittlerweile können viele Mouse-Events mittels CSS realisiert werden. [\[webis\]](#)

## Ereignisbehandlung: wichtige Ereignisse

---

<b>Event-Handler</b>	<b>HTML-Elemente</b>	<b>Semantik</b>
<code>onclick</code>	Knopf, Checkbox, Anker	Element wird angeklickt
<code>onchange</code>	Textfeld, Textbereich, Auswahl	Wert wird geändert
<code>onkeydown</code> <code>onkeyup</code> <code>onkeypress</code>	Dokument, Bild, Anker, Textfeld	Taste gedrückt/losgelassen
<code>onload</code>	Body	Beim Laden eines Dokuments
<code>onmousedown</code> <code>onmouseup</code>	Dokument, Knopf, Anker	Maustaste gedrückt/losgelassen
<code>onmouseout</code>	Bereiche, Anker	Mauszeiger verlässt einen Bereich
<code>onmouseover</code>	Anker	Mauszeiger über Anker
<code>onreset</code> , <code>onsubmit</code>	Formular	Reset/Submit für ein Formular
<code>onselect</code>	Textfeld, Textbereich	Element wird ausgewählt
<code>onfocus</code> <code>onblur</code>	Fenster, alle Formularelemente	Eingabefokus wird dem Element gegeben/entzogen

---

# JavaScript

## Quellen zum Nachlernen und Nachschlagen im Web

- ❑ ECMA. *Standard ECMA-262: ECMAScript Language Specification*.  
[www.ecma-international.org/publications/standards/Ecma-262](http://www.ecma-international.org/publications/standards/Ecma-262)
- ❑ Kastens. *Einführung in Web-bezogene Sprachen*.  
Vorlesung WS 2005/06, Universität Paderborn.
- ❑ MDN. *JavaScript*.  
[developer.mozilla.org/en-US/docs/Web/JavaScript](http://developer.mozilla.org/en-US/docs/Web/JavaScript)
- ❑ Miller/Horwat/Samuel. *Changes to JavaScript, Part 1: EcmaScript 5*.  
[GoogleTechTalk on www.youtube.com](http://GoogleTechTalk.com)
- ❑ SELFHTML e.V. *JavaScript*.  
[wiki.selfhtml.org/wiki/JavaScript](http://wiki.selfhtml.org/wiki/JavaScript)
- ❑ Tarquin. *JavaScript Tutorial*.  
[www.howtcreate.co.uk/tutorials/javascript](http://www.howtcreate.co.uk/tutorials/javascript)
- ❑ W3 Schools. *JavaScript Tutorial*.  
[www.w3schools.com/js](http://www.w3schools.com/js)

# JavaScript

## Quellen zum Nachlernen und Nachschlagen im Web

- ❑ Mozilla. *Firefox Add-on: Firebug*.  
[addons.mozilla.org/en-US/firefox/addon/firebug](https://addons.mozilla.org/en-US/firefox/addon/firebug)
- ❑ Mozilla. *Firefox Add-on: JavaScript Debugger*.  
[addons.mozilla.org/en-US/firefox/addon/javascript-debugger](https://addons.mozilla.org/en-US/firefox/addon/javascript-debugger)

## V. Client-Technologien

- ❑ Einführung
- ❑ Exkurs: Programmiersprachen
- ❑ JavaScript
- ❑ VBScript
- ❑ Java Applet
- ❑ Weitere Client-Technologien



# Java Applet

## Einführung [\[Einordnung\]](#)

*“A Java applet is a special kind of **Java program** that a browser enabled with Java technology can download from the internet and run.”*

[\[Oracle\]](#)

## Charakteristika:

- ❑ programmiert in der Multi-Purpose-Programmiersprache Java
- ❑ in HTML-Dokumente eingebettete Software**komponenten**

## Anwendung [\[Oracle Demos\]](#) :

- ❑ leistungsfähige grafische Oberflächen
- ❑ hohe Interaktivität zwischen Anwender und Software
- ❑ Netzwerkkommunikation kann beliebige Protokolle implementieren
- ❑ Präsentationsschicht für komplexe (n-Tier-)Architekturen

# Java Applet

## Einführung (Fortsetzung)

### Ein einfaches Applet:

```
package applet;
import java.applet.*;
import java.awt.*;

public class AppletHelloWorld extends Applet{
    public void init(){
        add(new Label("Hello World!"));    }
}
```

# Java Applet

## Einführung (Fortsetzung)

### Ein einfaches Applet:

```
package applet;
import java.applet.*;
import java.awt.*;

public class AppletHelloWorld extends Applet{
    public void init(){
        add(new Label("Hello World!"));    }
}
```

### Eine HTML-Seite, die das Applet einbindet:

```
<!DOCTYPE html>
<html>
  <head> <title>Applet Sample</title> </head>
  <body>
    <p>Here is the output of my program:</p>
    <object type="application/x-java-applet" width="150" height="25">
      <param name="code" value="applet.AppletHelloWorld">
      <param name="codebase" value=".">
      <param name="archive" value="part-client-...-code-java.jar">
    </object>
  </body> </html>
```

# Java Applet

## Einführung (Fortsetzung)

### Ein einfaches Applet:

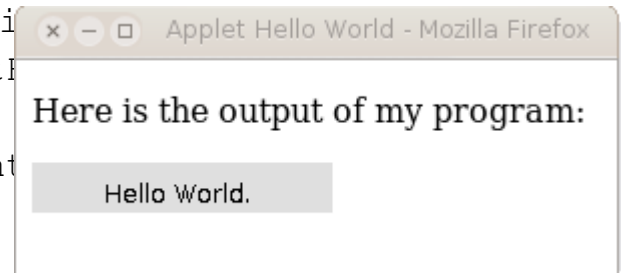
```
package applet;
import java.applet.*;
import java.awt.*;

public class AppletHelloWorld extends Applet{
    public void init(){
        add(new Label("Hello World!"));    }
}
```

### Eine HTML-Seite, die das Applet einbindet:

```
<!DOCTYPE html>
<html>
  <head> <title>Applet Sample</title> </head>
  <body>
    <p>Here is the output of my program:</p>
    <object type="application/x-java-applet" width="300" height="100">
      <param name="code" value="applet.AppletHelloWorld.class">
      <param name="codebase" value=".">
      <param name="archive" value="part-client.jar">
    </object>
  </body> </html>
```

[Applet-Ausführung]



# Java Applet

## Einführung (Fortsetzung)

### Ein einfaches Applet:

```
package applet;
import java.applet.*;
import java.awt.*;

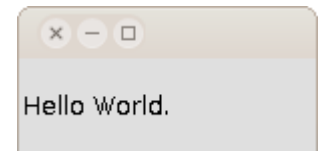
public class AppletHelloWorld extends Applet{
    public void init(){
        add(new Label("Hello World!"));    }
}
```

### Eine vergleichbare Java-Anwendung:

```
package applet;
import java.awt.*;

public class ApplicationHelloWorld extends Frame{
    public ApplicationHelloWorld(){
        add(new Label("Hello World!"));    }

    public static void main(String[] args){
        ApplicationHelloWorld hwa = new ApplicationHelloWorld();
        hwa.setSize(150,75);
        hwa.setVisible(true);    }
}
```



# Java Applet

## Einbindung in HTML-Dokumente

```
<object
  type="application/x-java-applet "
  width="pixels"
  height="pixels"
  [name="applet_instance_name" ] >

  <param name="code" value="applet_file" >
  [<param name="archive" value="file1.jar, file2.jar" >]
  [<param name="codebase" value="codebase_url" >]
  [<param name="applet_parameter1" value="value1" >]
  [<param name="applet_parameter2" value="value2" >]
  . . .
  [alternate_html]
</object>
```

# Java Applet

## Einbindung in HTML-Dokumente

```
<object
  type="application/x-java-applet "
  width="pixels"
  height="pixels"
  [name="applet_instance_name" ] >

  <param name="code" value="applet_file">
  [<param name="archive" value="file1.jar, file2.jar">]
  [<param name="codebase" value="codebase_url">]
  [<param name="applet_parameter1" value="value1">]
  [<param name="applet_parameter2" value="value2">]
  ...
  [alternate_html]
</object>
```

### Beispiel:

```
<object type="application/x-java-applet" width="500" height="20">
  <param name="code" value="aisearch/client/Client.class">
  <param name="archive" value="engine/aisearch.jar">
  <param name="imagesource" value="images/picture1.jpg">
  <param name="backgroundcolor" value="0xc0c0c0">
  ...
```

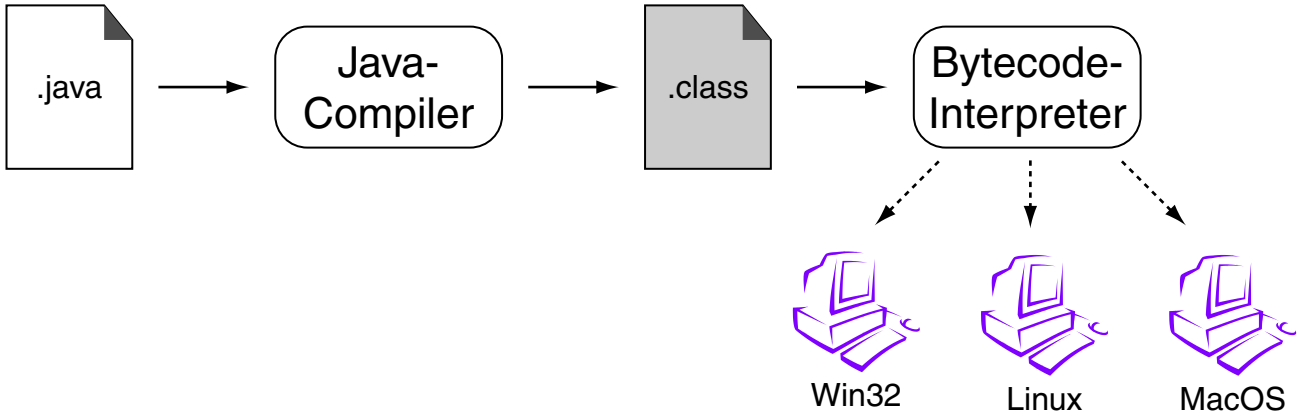
## Bemerkungen:

- ❑ Der Parameter `code` spezifiziert die Applet-Klasse, die vom Browser instantiiert wird.
- ❑ Die Wertzuweisung bei den Parametern `code`, `codebase` und `archive` zeigt, dass – wie in Java üblich – die Paketstruktur der Anwendung zu berücksichtigen ist. Mittels `codebase` wird der Java-Classpath gesetzt.



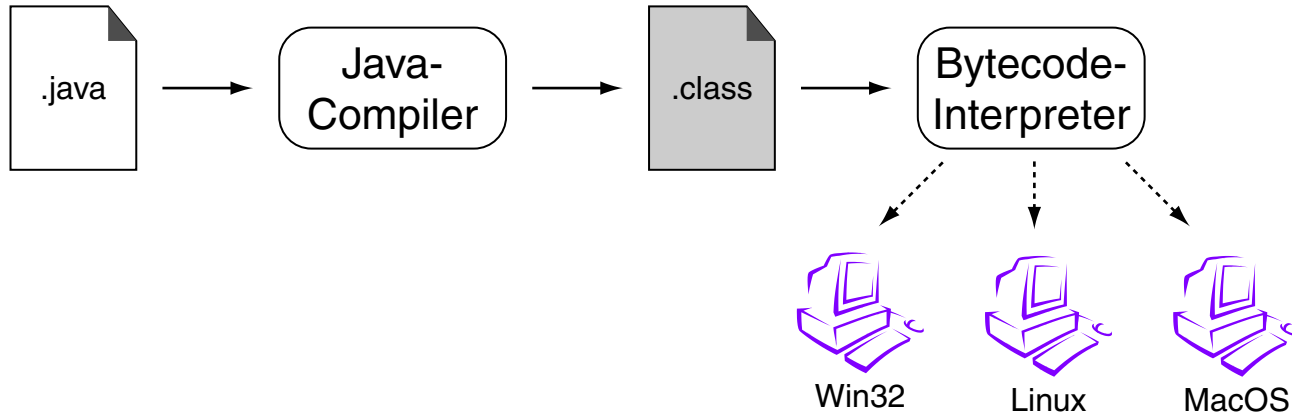
# Java Applet

## Java-Plattform



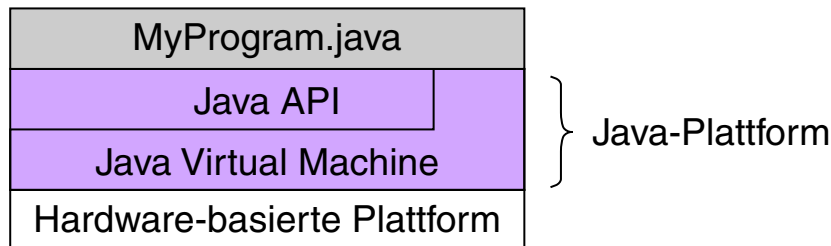
# Java Applet

## Java-Plattform



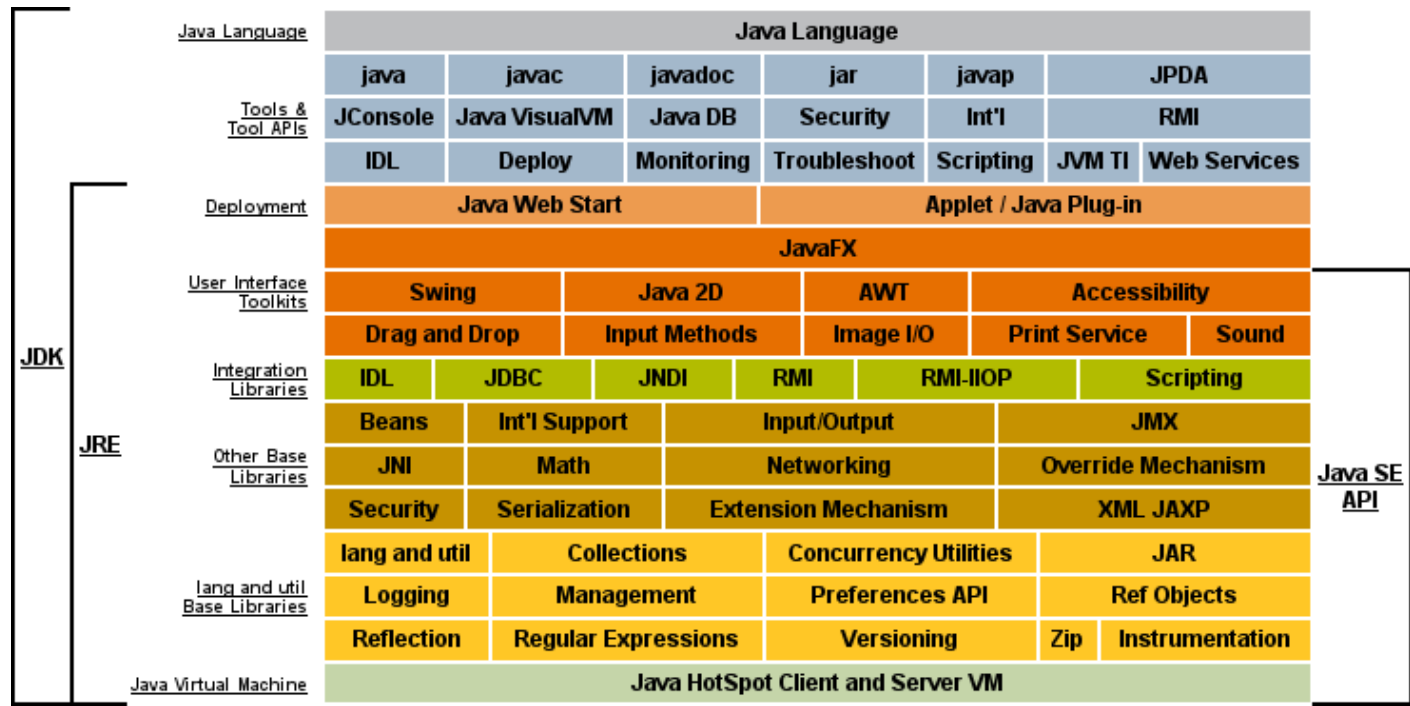
*“A platform is the hardware or software environment in which a program runs. [...] Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it’s a software-only platform that runs on top of other hardware-based platforms.”*

[Oracle]



# Java Applet

## Java-Plattform (Fortsetzung)



[Oracle]

# Java Applet

## Applet-Lebenszyklus

Applets können

1. initialisiert,
2. gestartet,
3. gestoppt und
4. aus dem Speicher entfernt werden.

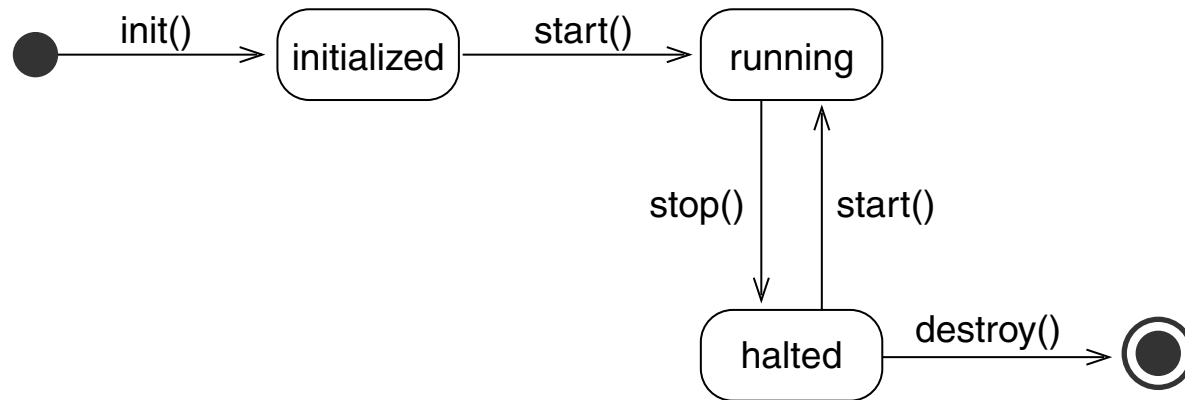
Die entsprechenden Java-Befehle:

```
public class myApplet extends Applet {  
    ...  
    public void init() {...}           // prepare variables, GUI  
    public void start() {...}         // start running  
    public void stop() {...}          // stop running  
    public void destroy() {...}       // cleanup  
    ...  
}
```

# Java Applet

## Applet-Lebenszyklus (Fortsetzung)

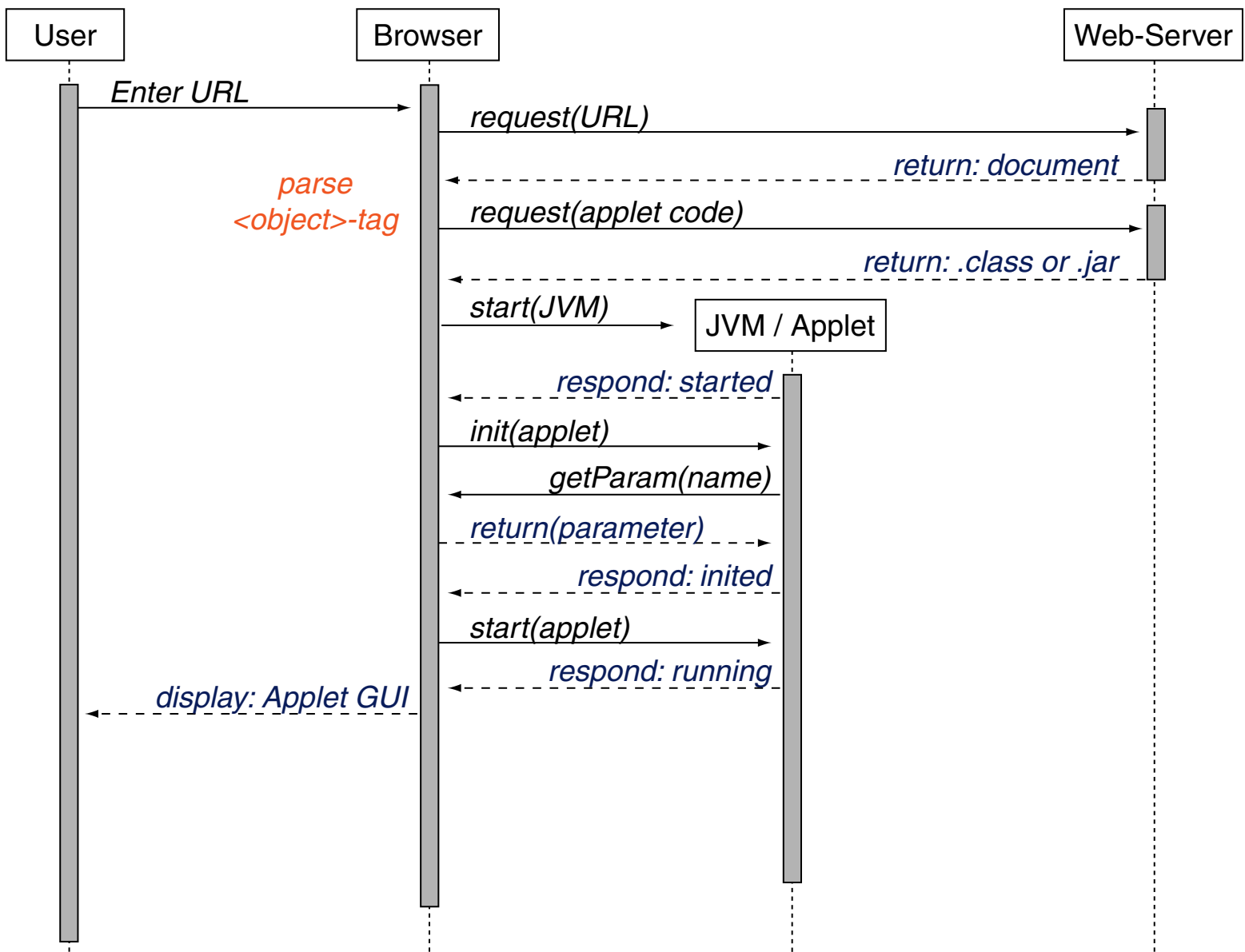
Nach dem Laden instantiiert der Browser die Applet-Klasse, ruft dann die `init()`-Methode und danach die `start()`-Methode auf.



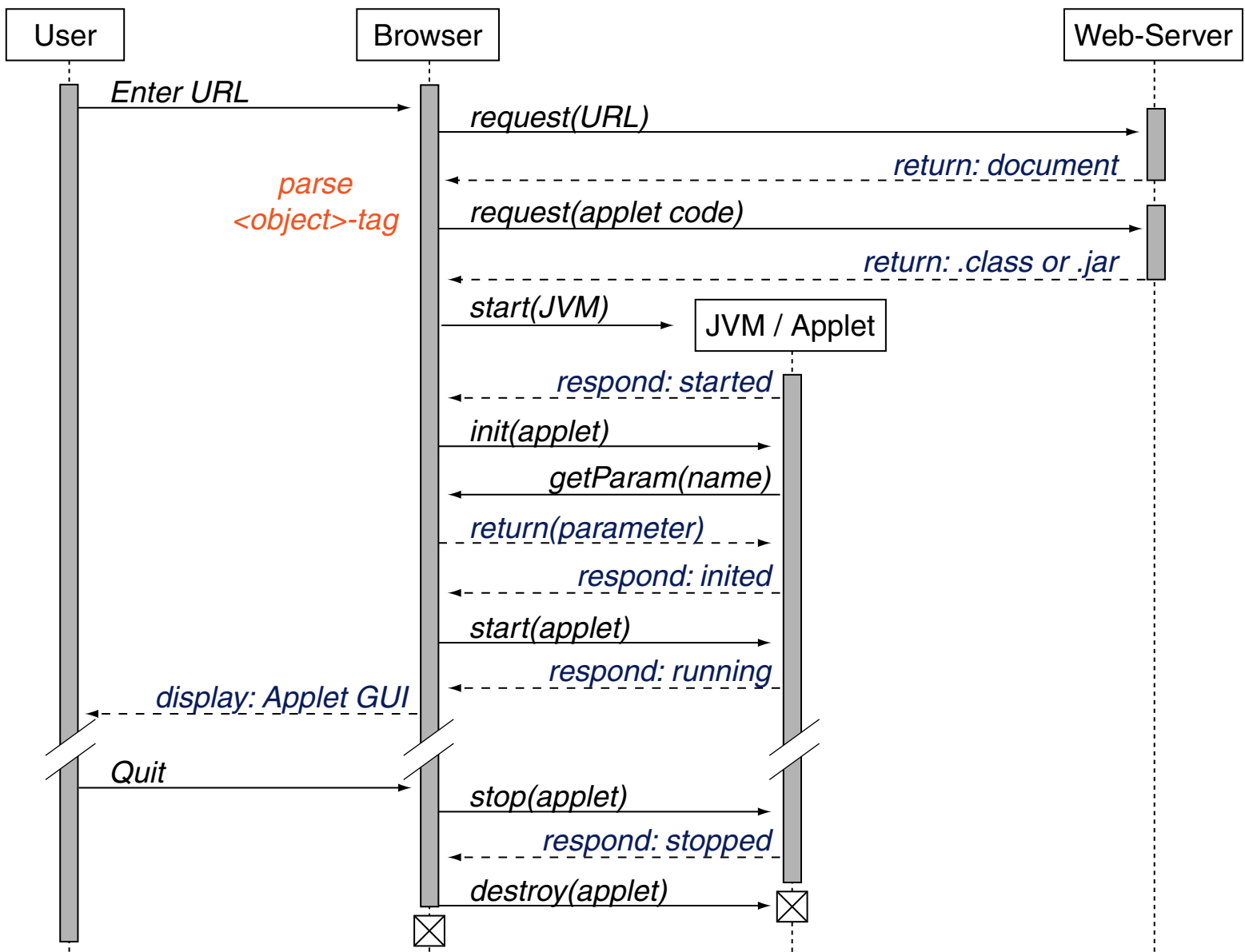
# Java Applet



# Java Applet



# Java Applet





# Java Applet

## Beispiel: Anfordern von Web-Dokumenten

Applet Window Control Example - Mozilla Firefox

Alsearch  
ChatNoir  
Netspeak

human interaction Alsearch!

behavioural economics tea flavour human interaction

ROBOT PHYSICAL [6]  
INSTITUTE CMU [5]  
INTERFACE USER [2]  
PROGRAMMING DEGREE [6]  
INFORMATION TERM [3]  
VIRTUAL LAB [4]  
SIGCHI CONFERENCE [4]  
BOOKS [4]  
MANAGEMENT PROCESS

portr. priv  portr. non-priv  article  shop  link list  help

de.aisearch. by benno stein and sven meyer zu eissen.

jece/CCCE  
/Human\_Computer\_Interaction.pdf  
(Yahoo)

[35. HCII Home | Human-Computer Interaction Institute](#)  
HCII Home | Human-Computer Interaction Institute  
Study how people design, implement and use interactive computer systems, and how ...  
CMU and SCS ceremonies are Sunday, May 17 2009 ...  
<http://www.hcii.cmu.edu/>  
(Yahoo)

[36. Human Interaction Research Institute](#)  
Human Interaction Research Institute HIRI has helped nonprofits, communities and funders meet the challenges of ... The Human Interaction Research Institute (HIRI) helps nonprofits, funders and ...  
<http://www.humaninteract.org/>  
(Yahoo)

PROGRAMMING DEGREE

[37. Human Interactions in](#)

[Applet-Ausführung]

# Java Applet

## Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

Organisiert als Frameset in drei HTML-Dokumenten:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Applet Window Control Example</title>
  </head>
  <frameset cols="150,*" border="0" >
    <frame src="AppletWindowControl.html" name="control"/>
    <frame src="AppletWindowContent.html" name="content"/>
  <noframes>
    <body>
      <p>Your browser cannot display frames.</p>
    </body>
  </noframes>
</frameset>
</html>
```

# Java Applet

## Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

### HTML-Dokument AppletWindowControl.html:

```
<!DOCTYPE html>
<html>
  <head> <title>Applet Window Control</title> </head>
  <body>
    <object type="application/x-java-applet" width="120" height="75" >
      <param name="code" value="applet.AppletWindowControl">
      <param name="codebase" value=".">
      <param name="archive"
        value="part-client-technologies-code-java.jar">
      <h1>Please install Java.</h1>
    </object>
  </body> </html>
```

### HTML-Dokument AppletWindowContent.html:

```
<!DOCTYPE html>
<html>
  <head> <title>Applet Window Content</title> </head>
  <body>
    <p>Please make your choice on the left side.</p>
  </body> </html>
```

# Java Applet

## Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

```
package applet;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.MalformedURLException;
import java.net.URL;

import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JPanel;

public class AppletWindowControl extends JApplet implements ActionListener {

    ...

    public void init() {...

    private JButton makeButton(String name) {...
        // Event handler.
    public void actionPerformed(ActionEvent event) {...
}
}
```

# Java Applet

## Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

```
package applet;

import java.awt.BorderLayout;...

public class AppletWindowControl extends JApplet implements ActionListener {

    JButton aisearchButton;
    JButton microsoftButton;
    JButton googleButton;

    String targetWindow = "content";
    String aisearchString = "Aisearch";
    String chatnoirString = "ChatNoir";
    String netspeakString = "Netspeak";

    URL aisearchURL = null;
    URL chatnoirURL = null;
    URL netspeakURL = null;

    public void init() {...

    private JButton makeButton(String name) {...
    // Event handler.
    public void actionPerformed(ActionEvent event) {...
}
```

# Java Applet

## Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

```
public void init() {  
  
    // Create a panel for the buttons.  
    JPanel buttonPanel = new JPanel();  
    buttonPanel.setLayout(new GridLayout(3, 1));  
  
    // Add the buttons to the panel.  
    aisearchButton = makeButton(aisearchString);  
    chatnoirButton = makeButton(chatnoirString);  
    netspeakButton = makeButton(netspeakString);  
    buttonPanel.add(aisearchButton);  
    buttonPanel.add(chatnoirButton);  
    buttonPanel.add(netspeakButton);  
  
    // Add the panel to the applet.  
    this.getContentPane().add(buttonPanel, BorderLayout.CENTER);  
  
    // Create URLs.  
    try {  
        aisearchURL = new URL("http://www.aisearch.de");  
        chatnoirURL = new URL("http://chatnoir.webis.de");  
        netspeakURL = new URL("http://www.netspeak.org");  
    } catch (MalformedURLException mue) {  
        System.out.println(mue.getMessage());  
    }  
}
```

# Java Applet

## Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

```
private JButton makeButton(String name) {  
  
    JButton button = new JButton(name);  
    button.addActionListener(this);  
    return button;  
}
```

```
// Event handler.
```

```
public void actionPerformed(ActionEvent event) {  
  
    if (aisearchString.equals(event.getActionCommand())) {  
        this.getAppletContext().showDocument(aisearchURL, targetWindow);  
    }  
    if (chatnoirString.equals(event.getActionCommand())) {  
        this.getAppletContext().showDocument(chatnoirURL, targetWindow);  
    }  
    if (netspeakString.equals(event.getActionCommand())) {  
        this.getAppletContext().showDocument(netspeakURL, targetWindow);  
    }  
}
```

# Java Applet

## Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

Die Methode `showDocument()` des Interfaces `AppletContext` [\[Javadoc\]](#):

```
public void showDocument(java.net.URL url)
```

```
public void showDocument(java.net.URL url, String targetWindow)
```

Optionen für `targetWindow`:

---

Option	Beschreibung
<i>WindowName</i>	Anzeige in dem (eventuell neu erzeugten) Fenster <i>WindowName</i> .
"_blank"	Anzeige in neuem, unbenanntem Fenster.
"_self"	Anzeige in dem Frame des Applet-Fensters.
"_parent"	Anzeige im Parent-Frame des Applet-Fensters.
"_top"	Anzeige im Top-Level-Frame des Applet-Fensters.

---



# Java Applet

## GUI-Programmierung

Applet-Programmierung heißt oft Oberflächenprogrammierung. Das JDK stellt verschiedene Klassen zur Realisierung von Benutzer-Interfaces bereit:

---

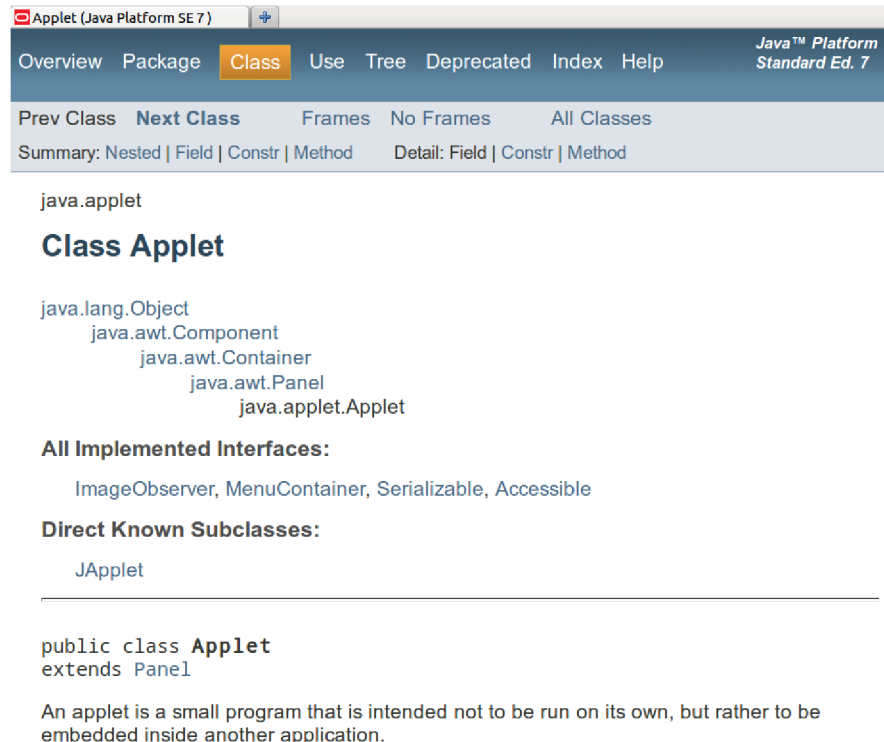
<b>Java-Klasse</b>	<b>GUI-Element, Widget</b>
<code>java.awt.Button</code>	Buttons
<code>java.awt.Checkbox</code>	Checkboxen
<code>java.awt.TextField</code>	einzeilige Textfelder
<code>java.awt.TextArea</code>	größere Textbereiche und Editierfelder
<code>java.awt.Label</code>	Labels
<code>java.awt.List</code>	Listen
<code>java.awt.Choice</code>	Pop-up- und Auswahllisten
<code>java.awt.Scrollbar</code>	Schieberegler und Scrollbars
<code>java.awt.Canvas</code>	Zeichenflächen
<code>java.awt.Menu,</code> <code>java.awt.MenuItem,</code> <code>java.awt.CheckboxMenuItem</code>	Menüs
<code>java.awt.Panel,</code> <code>java.awt.Window</code>	Container

---

# Java Applet

## GUI-Programmierung (Fortsetzung)

Die Vererbungshierarchie der Klasse Applet zeigt den starken grafischen Bezug der Applet-Programmierung [[Javadoc](#)]:



Applet (Java Platform SE 7)

Overview Package **Class** Use Tree Deprecated Index Help *Java™ Platform Standard Ed. 7*

Prev Class **Next Class** Frames No Frames All Classes

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.applet

### Class Applet

java.lang.Object  
  java.awt.Component  
    java.awt.Container  
      java.awt.Panel  
        java.applet.Applet

**All Implemented Interfaces:**

ImageObserver, MenuContainer, Serializable, Accessible

**Direct Known Subclasses:**

JApplet

---

```
public class Applet
extends Panel
```

An applet is a small program that is intended not to be run on its own, but rather to be embedded inside another application.

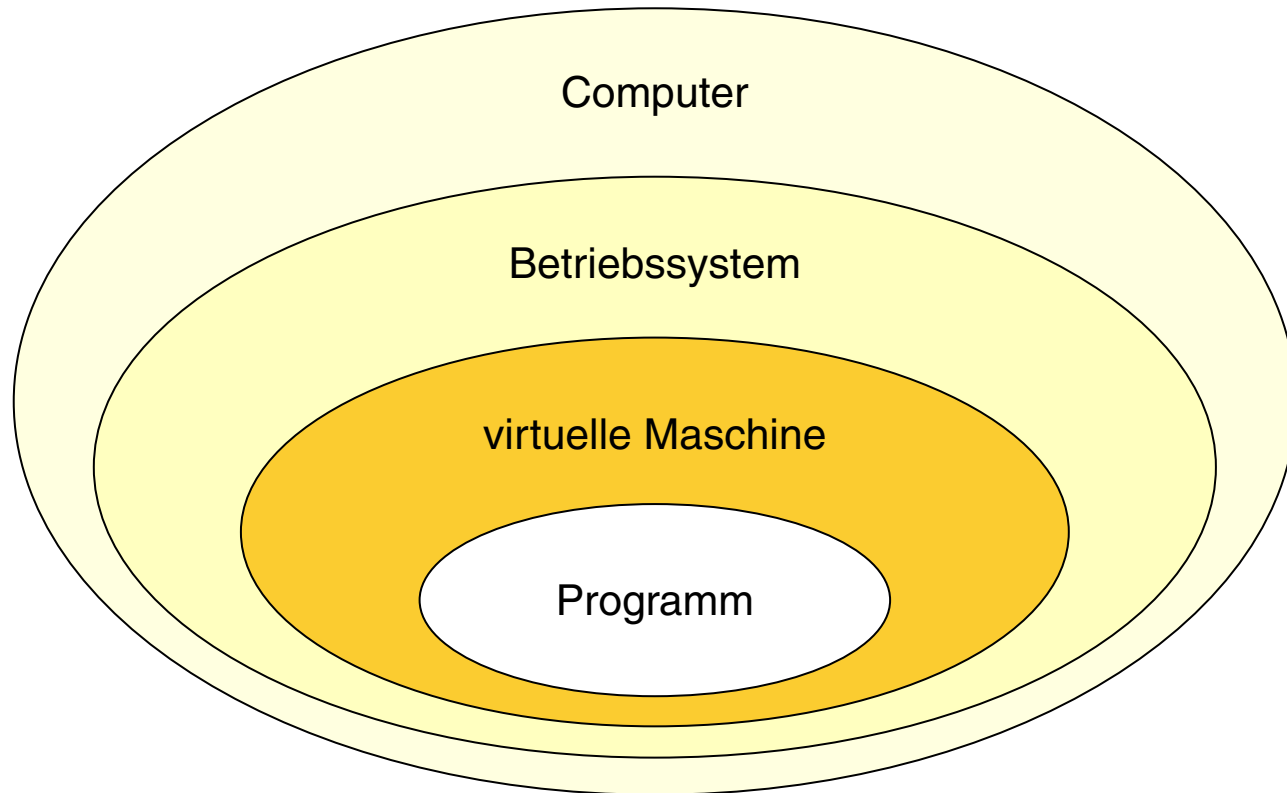
## Bemerkungen:

- ❑ Alle Swing-Komponenten sind verwendbar, wenn ein Applet von der Klasse `javax.swing.JApplet` anstatt von der Klasse `java.applet.Applet` erbt.  
[Javadoc: [1](#), [2](#)]

# Java Applet

## Sandbox-Prinzip

Die Java VM kapselt die Ausführung von Java-Programmen gegenüber dem Betriebssystem:



# Java Applet

## Sandbox-Prinzip (Fortsetzung)

### Beschränkungen von Applets:

- ❑ Systembibliotheken dürfen nicht geladen, „Native-Methoden“ nicht definiert werden.
- ❑ Netzwerk-Verbindungen zu beliebigen Hosts sind nicht erlaubt. Nur das Anfordern von Web-Dokumenten ist möglich.
- ❑ Auf dem Applet-**ausführenden** Host sind nicht erlaubt:
  - gewöhnliche Lese- und Schreibzugriffe
  - das Starten von Programmen
  - das Abfragen von Systemeigenschaften

# Java Applet

## Sandbox-Prinzip (Fortsetzung)

### Möglichkeiten von Applets:

- ❑ Zum Applet-ausliefernden Host (Web-Server) dürfen Netzwerkverbindungen initiiert werden.
- ❑ HTML-Dokumente dürfen von beliebigen Hosts angefordert werden.
- ❑ Mit `public` deklarierte Methoden anderer Applets derselben HTML-Seite dürfen aufgerufen werden.
- ❑ Auf eine `public` deklarierte Methode `method` eines Applets `applet` ist der Zugriff mit JavaScript möglich: `document . applet.method`
- ❑ Applets können weiterlaufen, auch wenn der Browser die zugehörige HTML-Seite verwirft.
- ❑ Applets, die nicht über das Web mit dem Browser-Plugin, sondern mit dem Java Runtime Environment (JRE) gestartet wurden, haben die Einschränkungen nicht.
- ❑ Der Anwender kann die **Beschränkungen für Applets aufheben**.

## Bemerkungen:

- ❑ Es bleibt dem Applet-ausliefernden Host natürlich vorbehalten, Netzwerkverbindungen von außen zu akzeptieren. Der entscheidende Punkt hier ist, *wo* die Restriktion der Verbindungserstellung auferlegt wird: durch die JVM, die das Applet ausführt, oder durch einen Host im Internet.
- ❑ Der Anwender gibt durch seine Zustimmung zur Aufhebung der Beschränkungen dem Applet die gleichen Rechte, die er auf seinem System besitzt. Die Gefahr dabei ist essentiell dieselbe wie beim Herunterladen ausführbarer Programme aus dem Netz, dem Öffnen von unbekanntem E-Mail-Anhängen etc.

# Java Applet

## Signierung

Welche Grundvoraussetzung sollte erfüllt sein, damit ein Anwender einem Applet relativ gefahrlos mehr Rechte einräumen kann?



# Java Applet

## Signierung

Welche Grundvoraussetzung sollte erfüllt sein, damit ein Anwender einem Applet relativ gefahrlos mehr Rechte einräumen kann?

Bei der Übertragung eines Applets (allgemein: Nachricht) kann „viel passieren“: sie kann abgefangen, umgeleitet oder ausgetauscht werden.

Standardszenario aus der Kryptografie:



Wichtige Aspekte in diesem Zusammenhang sind:

- ❑ Vertraulichkeit: Geheimhaltung des Inhalts
- ❑ Integrität: Aufdeckung von Inhaltsveränderungen
- ❑ Authentizität: Garantie der Urheberschaft

# Java Applet

## Signierung (Fortsetzung)

Sei  $P$  die Menge aller Texte (*plain texts*),  $K$  die Menge aller Schlüssel (*keys*),  $C$  die Menge aller verschlüsselten Texte (*cipher texts*) und  $e_k, d_k$  zwei Funktionen:

$$\begin{array}{l} e_k : P \rightarrow C \\ d_k : C \rightarrow P \end{array} \quad \text{mit} \quad d_k(e_k(x)) = x, \quad x \in P, k \in K$$

Protokoll einer symmetrischen Verschlüsselung:

1. Alice und Bob wählen einen gemeinsamen Schlüssel  $k \in K$ .
2. Alice versendet Nachricht  $x$  als  $y = e_k(x)$  zu Bob.
3. Bob entschlüsselt  $y$  und erhält  $x = d_k(y)$ .

# Java Applet

## Signierung (Fortsetzung)

Sei  $P$  die Menge aller Texte (*plain texts*),  $K$  die Menge aller Schlüssel (*keys*),  $C$  die Menge aller verschlüsselten Texte (*cipher texts*) und  $e_k, d_k$  zwei Funktionen:

$$\begin{aligned} e_k &: P \rightarrow C \\ d_k &: C \rightarrow P \end{aligned} \quad \text{mit} \quad d_k(e_k(x)) = x, \quad x \in P, k \in K$$

Idee der **asymmetrischen** Public-Key-Kryptografie: Alice und Bob haben je zwei Schlüssel  $k_1$  (öffentlich) und  $k_2$  (privat) mit  $d_{k_1}(e_{k_2}(x)) = d_{k_2}(e_{k_1}(x)) = x$ .

Protokoll einer asymmetrischen Verschlüsselung:

1. Alice und Bob wählen jeder für sich die Schlüssel  $k_1^{(A)}, k_2^{(A)}$  und  $k_1^{(B)}, k_2^{(B)}$ .
2. Beide veröffentlichen ihren Schlüssel  $k_1$ .
3. Alice versendet Nachricht  $x$  als  $y = e_{k_1}^{(B)}(x)$  zu Bob.
4. Bob entschlüsselt  $y$  und erhält  $x = d_{k_2}^{(B)}(y)$ .

## Bemerkungen:

- ❑ Bekannte Verfahren zur symmetrischen Verschlüsselung sind der Data Encryption Standard, DES, der Advanced Encryption Standard, AES, und der International Data Encryption Standard, IDEA.
- ❑ Ein bekanntes Verfahren zur asymmetrischen Verschlüsselung ist RSA, unter anderem implementiert in PGP und GnuPG.

# Java Applet

## Signierung (Fortsetzung)

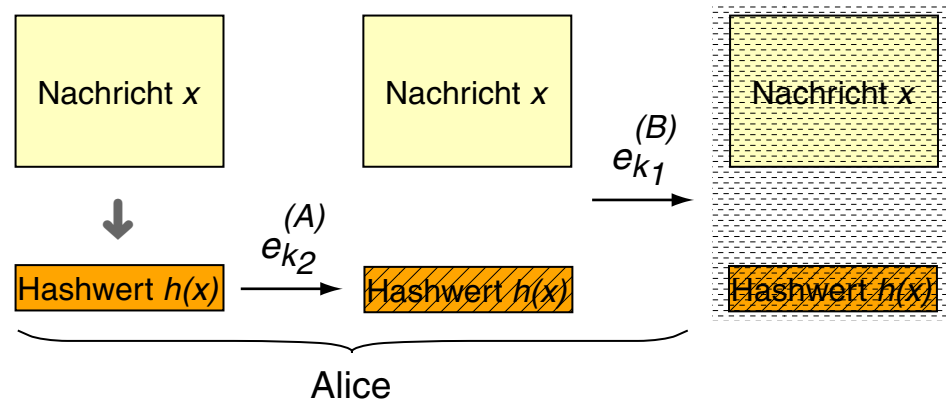
Woher weiß Bob, dass Alice und nicht Eve der Autor von Nachricht  $x$  ist?

# Java Applet

## Signierung (Fortsetzung)

Woher weiß Bob, dass Alice und nicht Eve der Autor von Nachricht  $x$  ist?

Sei  $h : P \rightarrow N$  eine Hashfunktion, die mit extrem hoher Wahrscheinlichkeit eine eindeutige Charakterisierung  $h(x)$  einer Nachricht  $x$  berechnet.



Protokoll zum digitalen Signieren:

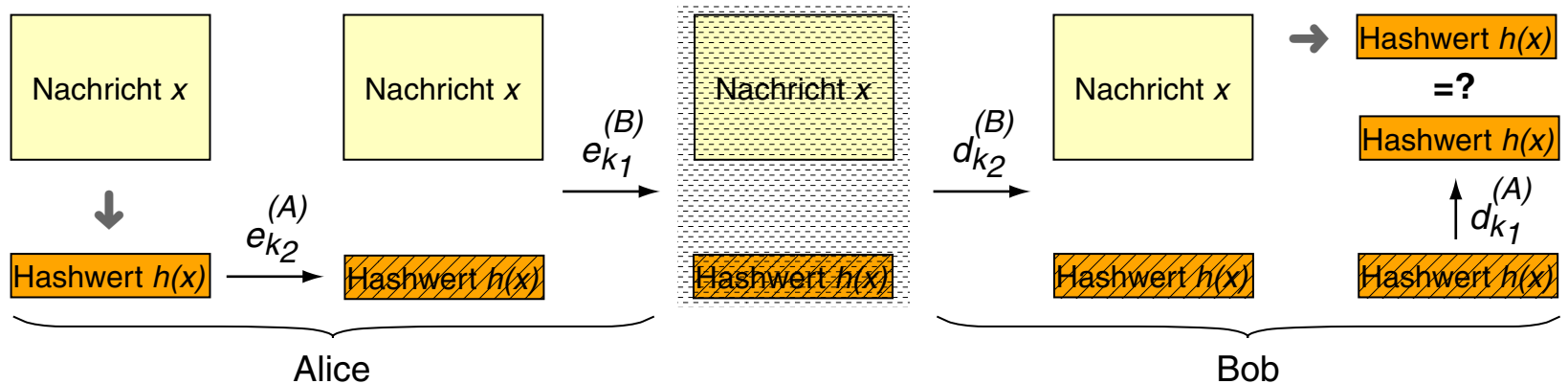
1. Alice berechnet für Nachricht  $x$  den Hashwert  $h(x)$ .
2. Alice verschlüsselt  $h(x)$  als  $y_h = e_{k_2}^{(A)}(h(x))$ .
3. Alice versendet Nachricht  $x + y_h$  als  $e_{k_1}^{(B)}(x + y_h)$  zu Bob.

# Java Applet

## Signierung (Fortsetzung)

Woher weiß Bob, dass Alice und nicht Eve der Autor von Nachricht  $x$  ist?

Sei  $h : P \rightarrow N$  eine Hashfunktion, die mit extrem hoher Wahrscheinlichkeit eine eindeutige Charakterisierung  $h(x)$  einer Nachricht  $x$  berechnet.



Protokoll zum Verifizieren:

1. Bob entschlüsselt mittels  $d_{k_2}^{(B)}$  die Nachricht und erhält  $x + y_h$ .
2. Bob berechnet für Nachricht  $x$  den Hashwert  $h(x)$ .
3. Bob berechnet  $d_{k_1}^{(A)}(y_h)$  und vergleicht den Wert mit  $h(x)$ .

## Bemerkungen:

- ❑ Der Vergleich von digitalen Signaturen mit realen Unterschriften ist gerechtfertigt, da Bob durch Abgleich der Signatur von Alice ihre Identität zu verifizieren sucht. Entscheidender Punkt bei digitalen Signaturen ist die Verhinderung der Trennung von Nachricht und Signatur.
- ❑ Falls Alice und Bob sich nicht kennen, kann digitale Signierung durch Zwischenschaltung eines gemeinsamen Vertrauensgebers, einer sogenannten Zertifizierungsinstanz, geschehen.
- ❑ Tutorial zur Code-Signierung in Java. [[Oracle](#)]



# Java Applet

## Quellen zum Nachlernen und Nachschlagen im Web

- ❑ Gosling, McGilton. *The Java Language Environment*.  
[www.oracle.com/technetwork/java/](http://www.oracle.com/technetwork/java/)
- ❑ Oracle. *Learning the Java Language*.  
[docs.oracle.com/javase/tutorial/java/](http://docs.oracle.com/javase/tutorial/java/)
- ❑ Oracle. *Signing Code and Granting It Permissions*.  
[docs.oracle.com/javase/tutorial/security/toolsign/](http://docs.oracle.com/javase/tutorial/security/toolsign/)
- ❑ Oracle. *Applets*.  
[docs.oracle.com/javase/tutorial/deployment/applet/](http://docs.oracle.com/javase/tutorial/deployment/applet/)
- ❑ Oracle. *How to Make Applets. (Swing)*  
[docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html](http://docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html)

# Weitere Client-Technologien

# Weitere Client-Technologien

## Java Web Start [\[Einordnung\]](#)

Idee: Realisierung bestimmter Aspekte reiner Web-basierter Software für beliebige Anwendungssoftware.

### Web-basierte Software:

- ❑ Download und Ausführung auf dem Client per Maus-Click
- ❑ Software zentral und immer aktuell auf dem Server
- ❑ keine Installation und keine Administrationsproblematik auf dem Client

### Java Web Start [\[Oracle\]](#) :

- ❑ Installation und Ausführung auf dem Client per Maus-Click
- ❑ bei Programmstart Kontaktierung des Servers und evtl. Aktualisierung
- ❑ skalierbare Ausführungsrechte: von Sandbox-gesichert bis unbeschränkt

## Bemerkungen:

- ❑ Entsprechende Technologien von Microsoft sind ActiveX und die Common Language Runtime, CLR.
- ❑ Das Microsoft-Konzept des Ladens und Ausführens von ActiveX-Komponenten ist deutlich weniger restriktiv als das Applet-Konzept.