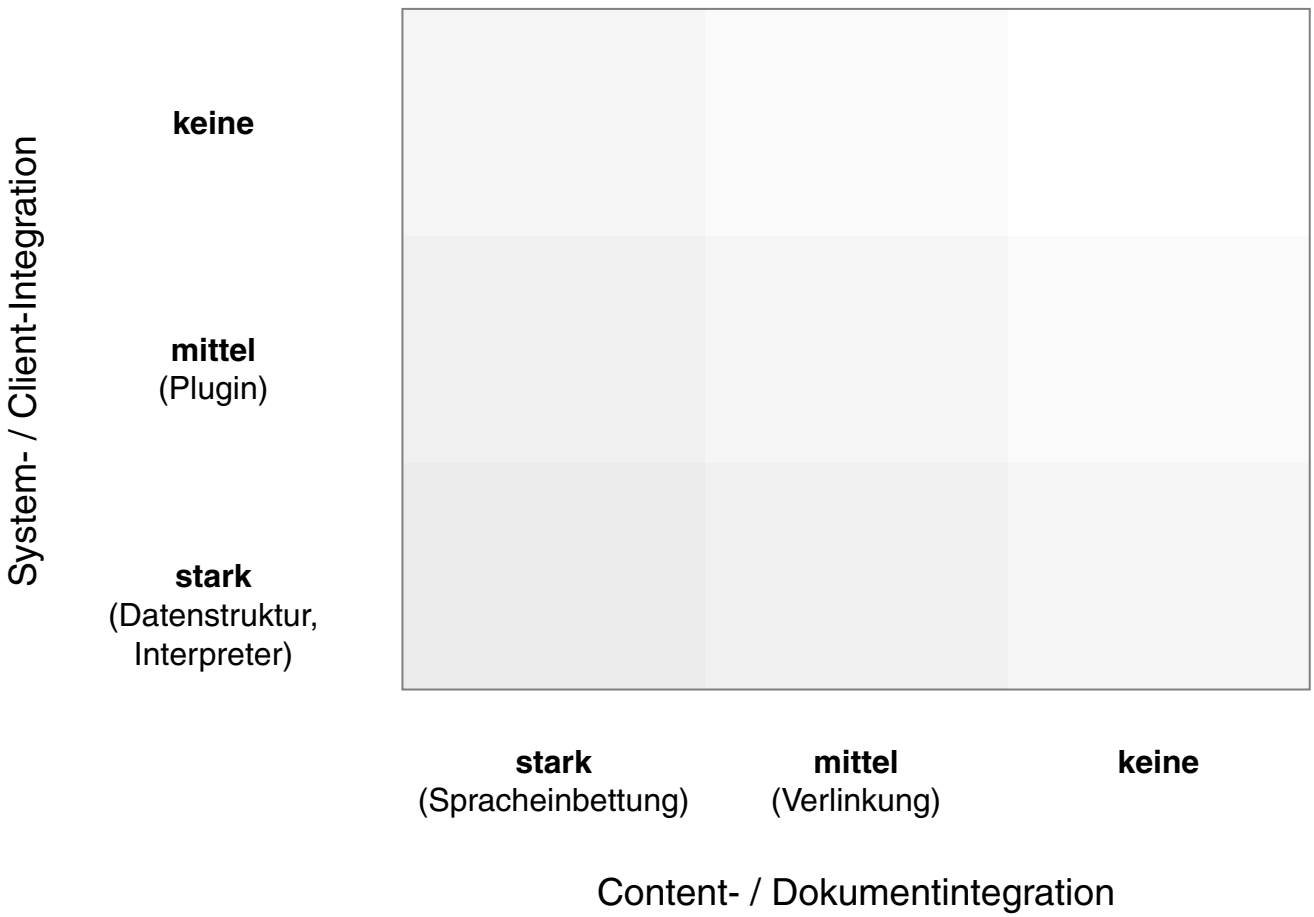
Kapitel WT:V

Web-Client

Einordnung von Client-Technologien [Stein 2012]

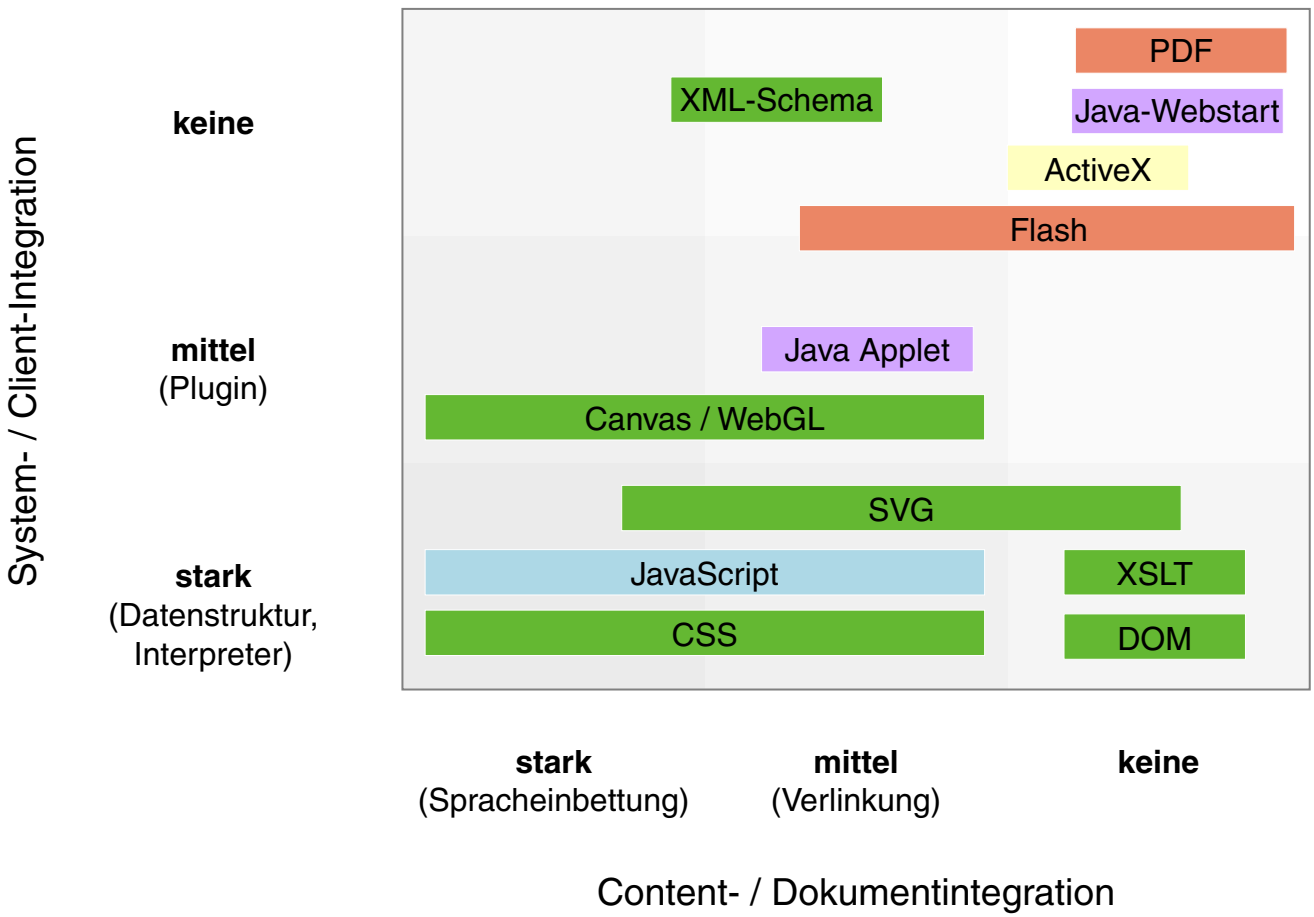
- x-Achse: Wie eng ist die Technologie mit dem dargestellten Content verwoben?
- y-Achse: Wie tief ist die Technologie in den Web-Client integriert?



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?

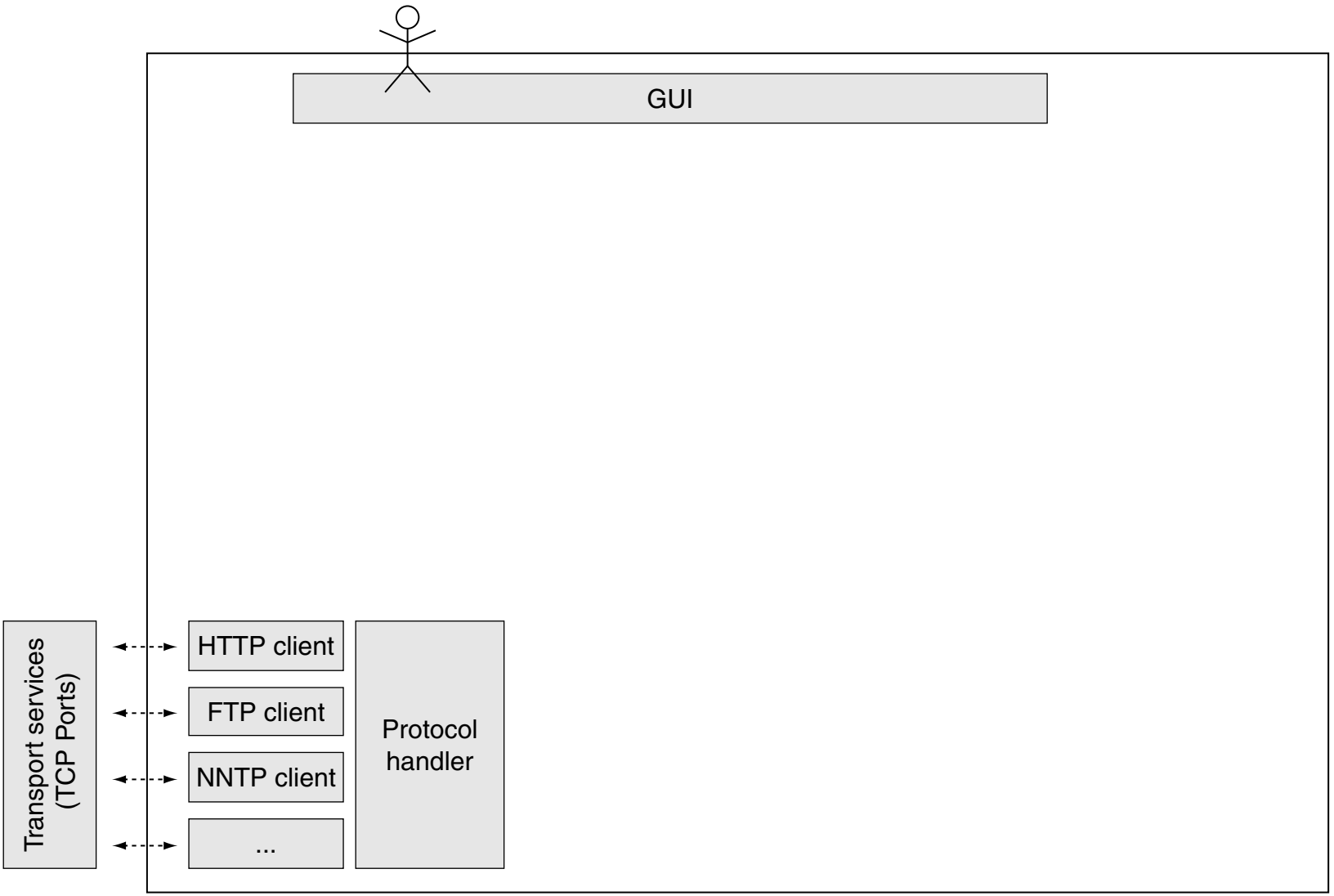


Bemerkungen:

- 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.

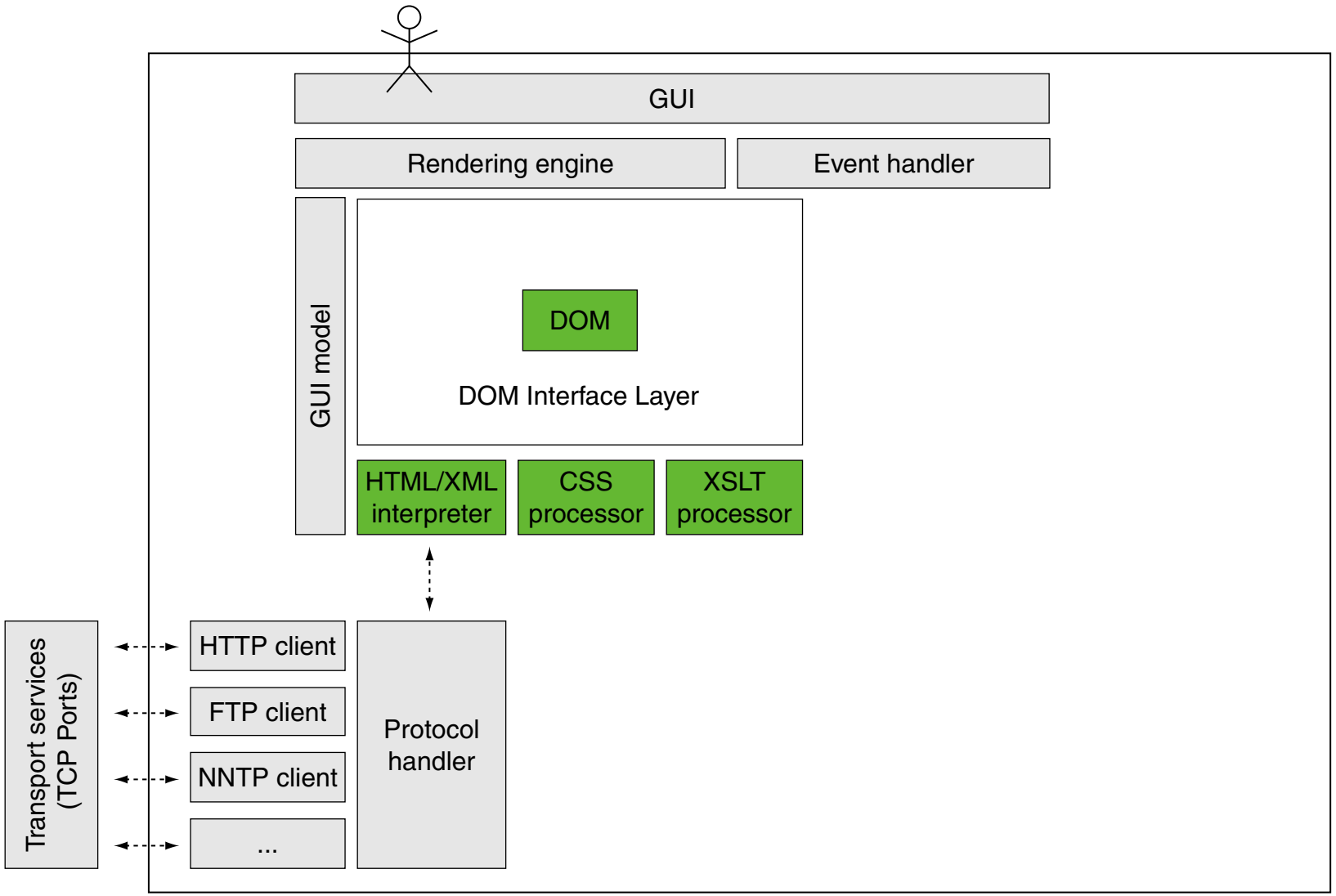
Web-Client

Browser-Module



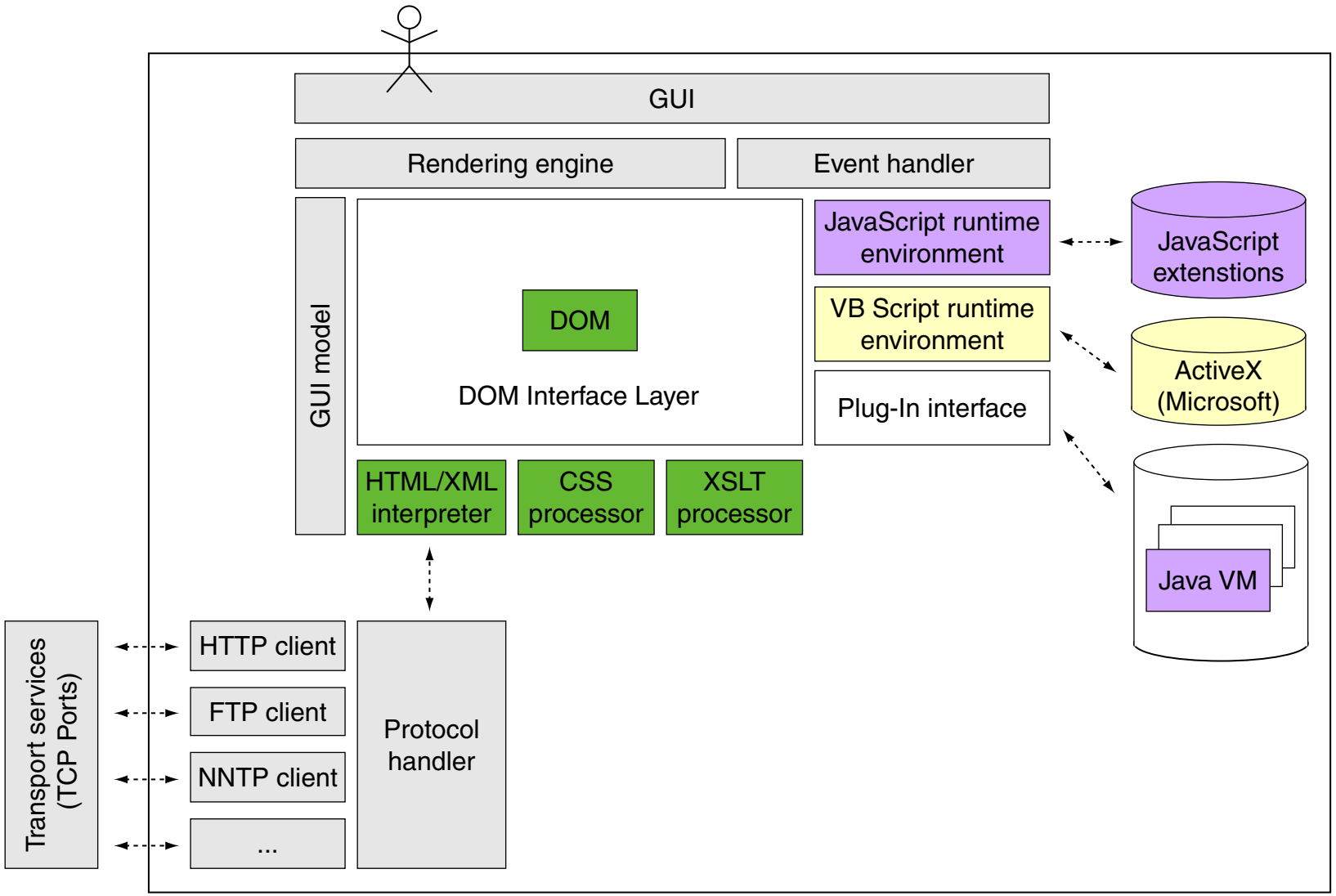
Web-Client

Browser-Module

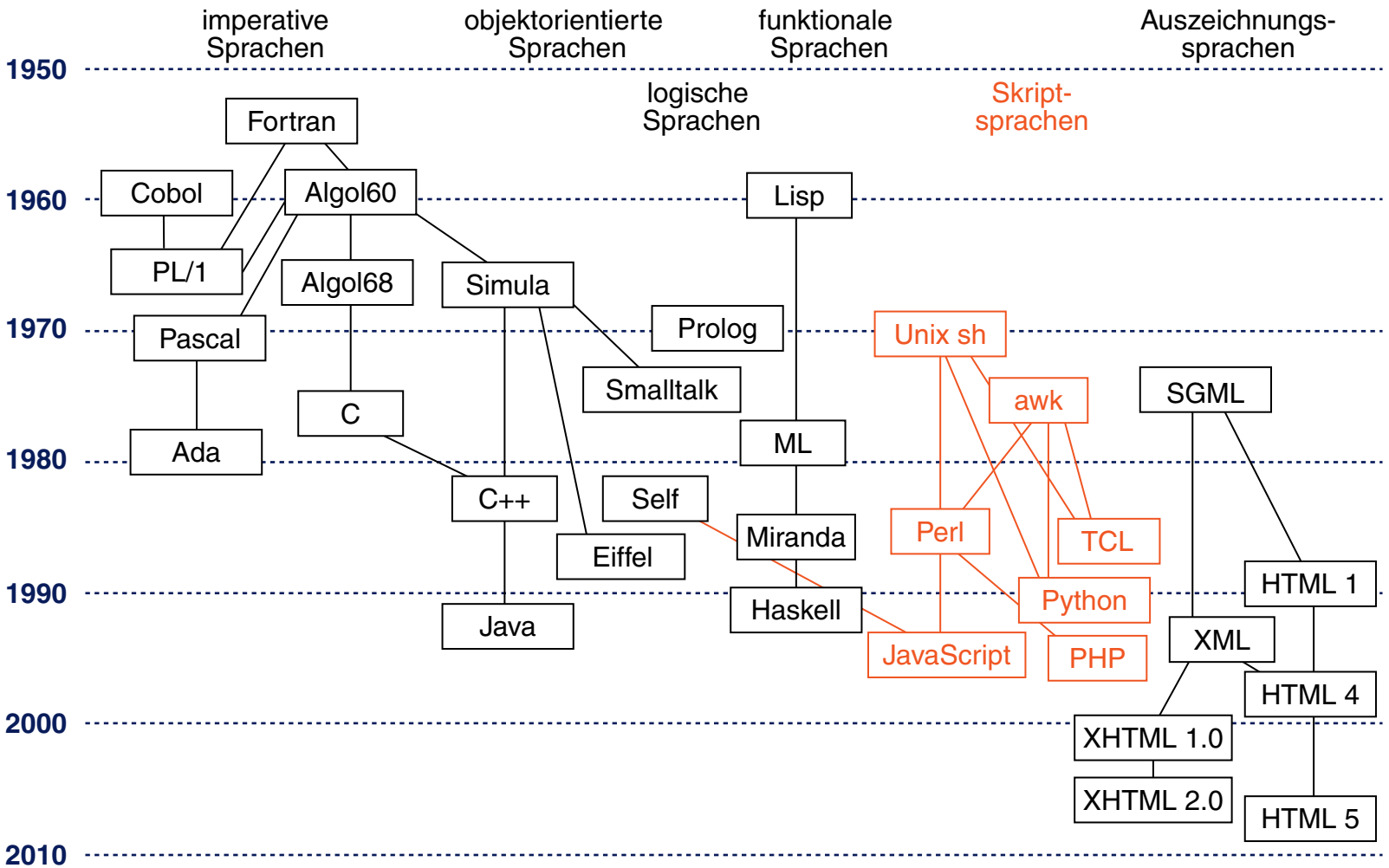


Web-Client

Browser-Module



Exkurs: Programmiersprachen [Kastens 2005]



[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);
```

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 *(Identifier)* Verwendung: Namen für Variable, Funktionen, etc.	`$line, fgets`
Literale *(Literals)* Verwendung: Zahlkonstanten, Zeichenkettenkonstanten	`64, "telefonbuch.txt"`
Wortsymbole *(Keywords)* Verwendung: kennzeichnen Sprachkonstrukte	`while, if`
Spezialzeichen Verwendung: Operatoren, Separatoren	`<= = ; { }`

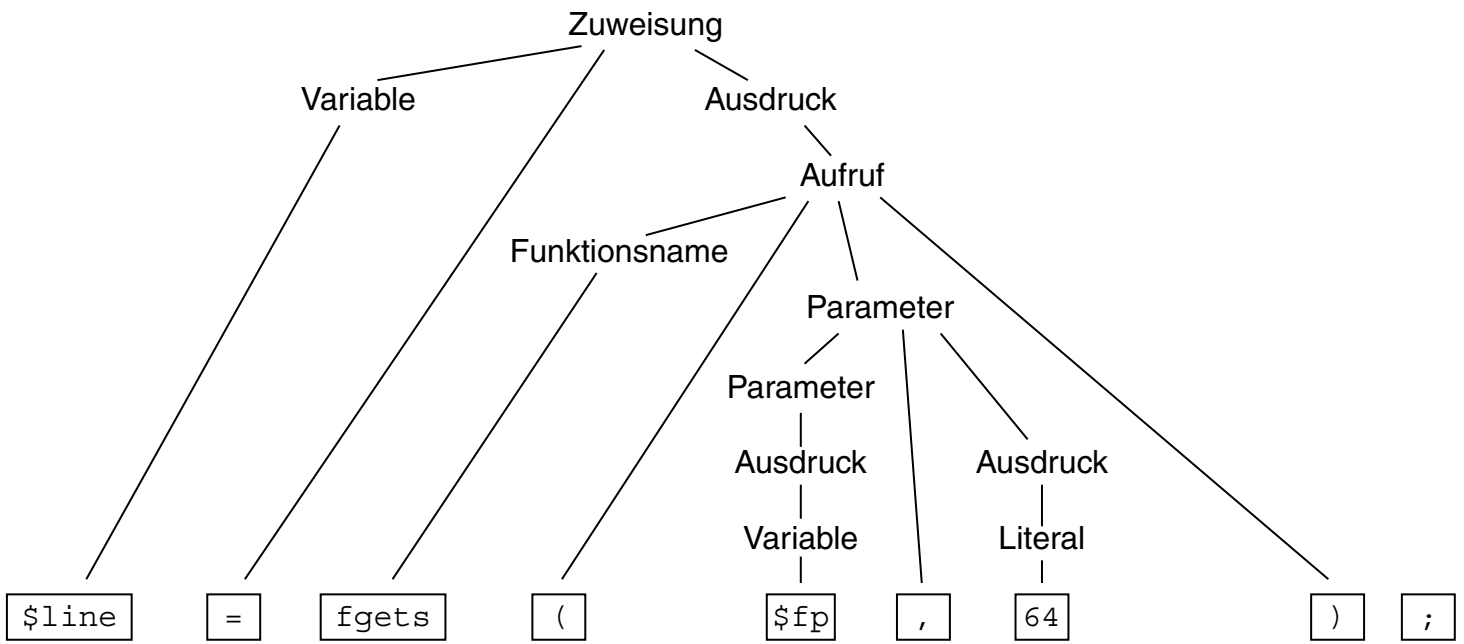
Bemerkungen:

- 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 an 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 an 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.

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.

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.

Bemerkungen:

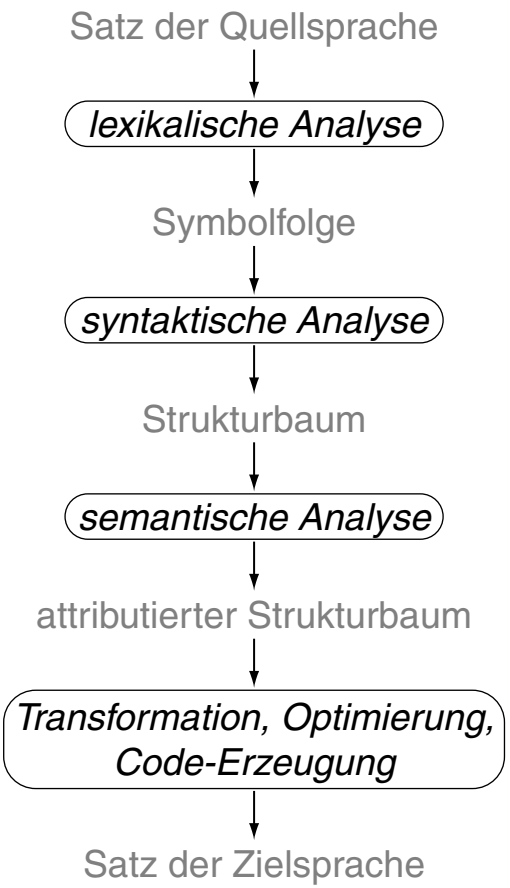
- Auf jeder der 4 Ebenen gibt es also Regeln, die korrekte Sätze erfüllen müssen.
- In der Sprache PHP gehören die Typregeln zur dynamischen Semantik, da sie erst bei der Ausführung des Programms anwendbar sind.
- In der Sprache JavaScript gehören die Bindungsregeln zur statischen Semantik und die Typregeln 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 Interpretierer 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

Neuere Interpretierer 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 (LaTeX→ PostScript), Spezifikationssprachen (UML → 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

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

JavaScript

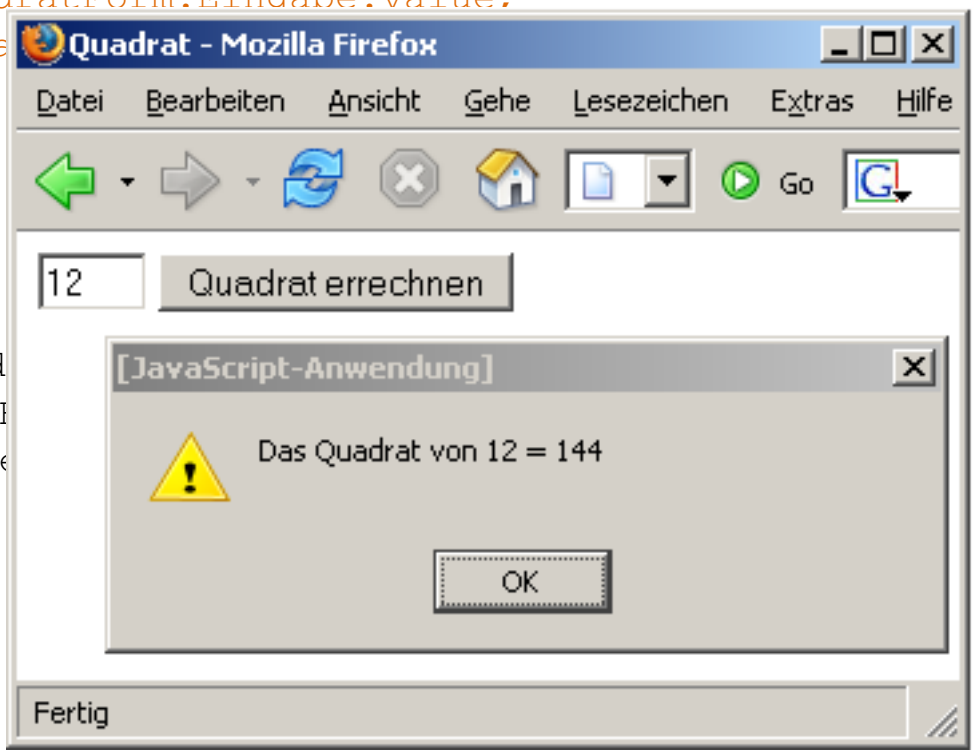
Einführung (Fortsetzung)

```
<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

Einführung (Fortsetzung)

```
<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 * Za
        alert ("Das Quadrat von
      }
    </script>
  </head>
  <body>
    <form name="QuadratForm" id
      <input type="text" name="I
      <input type="button" value
    </form>
  </body>
</html>
```



JavaScript

Historie

1993 NCSA Mosaic-Browser, mit Bildern im Fließtext und Farben für Links und Text.

1994 Netscape 1, entwickelt von einer Splittergruppe des Mosaic-Teams.

1996 Netscape 2, mit Frames und JavaScript von Brendan Eich. JavaScript heißt zunächst LiveWire, dann LiveScript. Die Art des Dokumentzugriffs heißt heute DOM Level 0.

1996 Standardisierung von JavaScript durch die European Computer Manufacturers Association, ECMA, als ECMAScript in der Spezifikation ECMA-262.

1997 Netscape 4, mit sogenanntem „DHTML“, basierend auf W3C CSS 1 und JavaScript 1.2.

1997 Internet Explorer 4, mit revolutionären Konzept für dynamische Webseiten: das W3C orientiert DOM daran. Als Konkurrenz zu JavaScript entwickelt Microsoft JScript.

1998 Der Netscape-Code wird Open Source und das Mozilla-Projekt wird gestartet.

JavaScript

Historie

1993 NCSA Mosaic-Browser, mit Bildern im Fließtext und Farben für Links und Text.

1994 Netscape 1, entwickelt von einer Splittergruppe des Mosaic-Teams.

1996 Netscape 2, mit Frames und JavaScript von Brendan Eich. JavaScript heißt zunächst LiveWire, dann LiveScript. Die Art des Dokumentzugriffs heißt heute DOM Level 0.

1996 Standardisierung von JavaScript durch die European Computer Manufacturers Association, ECMA, als ECMAScript in der Spezifikation ECMA-262.

1997 Netscape 4, mit sogenanntem „DHTML“, basierend auf W3C CSS 1 und JavaScript 1.2.

1997 Internet Explorer 4, mit revolutionären Konzept für dynamische Webseiten: das W3C orientiert DOM daran. Als Konkurrenz zu JavaScript entwickelt Microsoft JScript.

1998 Der Netscape-Code wird Open Source und das Mozilla-Projekt wird gestartet.

2002 Mozilla Phoenix 0.1 (später Firefox), mit der Open Source Rendering-Engine Gecko.

2005 Firefox 1.5, mit JavaScript 1.5 $\sim$ ECMAScript 3.

2008 Google Chrome 1, mit freier JavaScript-Engine V8 und JavaScript 1.7 $\sim$ ECMAScript 3.

2010 Firefox 4.0, mit JavaScript 1.9 $\sim$ ECMAScript 5.

2012 Firefox 13, mit experimentellem Support für ECMAScript 6.

2013 Statistiken zur Verbreitung: [tiobe.com]

Bemerkungen [Tarquin] [SELFHTML] :

- Die Entwicklung von JavaScript ist eng geknüpft an den „Browser-Krieg“ zwischen Microsoft und Netscape.
- Ab JavaScript 1.5 (mittlerweile in allen Browsern) ist DOM Level 1 realisiert.
- 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.
- Aufgrund ihrer hohen Portabilität wird die Gecko Rendering-Engine mittlerweile in 30 verschiedenen Browsern eingesetzt.
- Die W3C-DOM-Initiative entwickelt sich erfolgreich – insbesondere, was den Anspruch angeht, die Browser-Entwicklung zu vereinheitlichen: alle Browser-Hersteller streben eine Implementierung gemäß W3C DOM Level 3 an.

JavaScript

Einbindung in HTML-Dokumente [SELFHTML]

1. Als Script-Bereich innerhalb eines HTML-Dokuments [Demo: 1, 2] :

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

2. Innerhalb von HTML-Tags [Demo] :

   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 angeben werden, die beim Klicken ausgeführt wird:

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

3. In einer separaten Datei [Demo] :

```
<script src="quadrat.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.
- Funktions*definitionen* erzeugen keine Ausgabe bzw. haben keinen Return-Wert.
- Es gibt keine Vorschrift dafür, an welcher Stelle in einem HTML-Dokument ein JavaScript-Bereich definiert werden darf. Es ist jedoch sinnvoll, diesen Bereich im Kopf eines HTML-Dokuments zu definieren.
- Eine separate Datei mit JavaScript-Code muss eine ASCII-Datei sein und sollte die Dateinamenerweiterung `.js` besitzen. Die Datei darf nur JavaScript-Code enthalten.

JavaScript

Grundlagen der Syntax [PHP]

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

Bezeichner

- einheitliche Schreibweise für alle Arten von Bezeichnern:

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

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

JavaScript

Grundlagen der Syntax [PHP]

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

Bezeichner

- einheitliche Schreibweise für alle Arten von Bezeichnern:

  `identifier = { 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* `*/`

JavaScript

Variablen [PHP] [SELFHTML]

- Eine Variable kann Werte beliebigen Typs annehmen.
- Es wird zwischen lokalen und globalen Variablen unterschieden.
- Eine lokale Variable erhält man durch Deklaration mit dem Schlüsselwort `var` innerhalb einer Funktion.
- Globale Variablen gelten im ganzen Programm – es sei denn, sie werden von einer lokalen Variable überdeckt; lokale Variablen gelten in ihrer Funktion.
- Variablen können durch Zuweisung eines Anfangswertes in Form eines Literals oder einer anderen Variable initialisiert werden.

JavaScript

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;
}
```

JavaScript

Variablen: Illustration von Geltungsbereichen

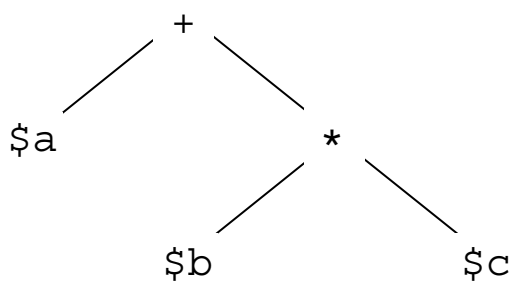
```
var line;
var sum, result;
var col, count = 3, row;
var minimum = count,
    maximum = 999;

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

JavaScript

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
```

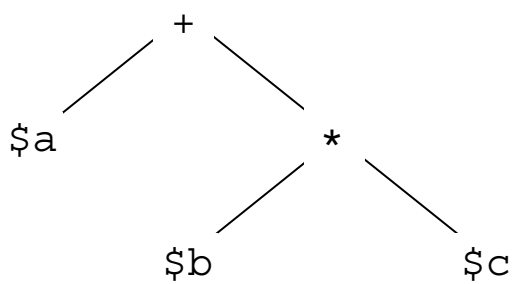


JavaScript

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
```



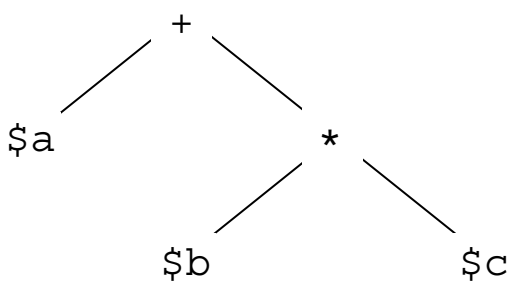
Ein Operator ist linksassoziativ (rechtsassoziativ), wenn beim Zusammentreffen von Operatoren gleicher Präzedenz der linke (rechte) Operator seine Operanden stärker bindet als der rechte (linke).

JavaScript

Operatoren: Präzedenz, Assoziativität

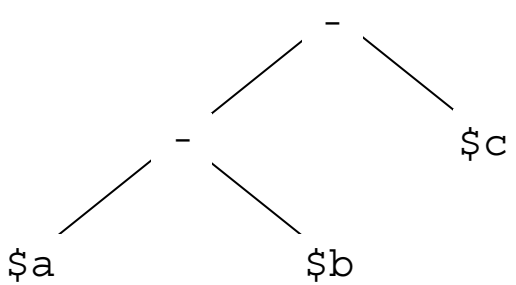
Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz. Durch Klammerung lässt sich die Präzedenz in Termen vorschreiben. Beispiel:

```
$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
```



JavaScript

Operatoren: Übersicht [SELFHTML]

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

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`

- Zeichenkettenliterale 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 `!`

JavaScript

Datentypen: Primitive (Fortsetzung)

`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 betrachtet.

JavaScript

Datentypen: Objekte [benutzerdefinierte Objekte]

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

Erzeugung von Objekten:

(a) Mit dem Konstruktor `Object()`:

(b) Durch Verwendung von einem Objektliteral:

JavaScript

Datentypen: Objekte [benutzerdefinierte Objekte]

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

Erzeugung von Objekten:

(a) Mit dem Konstruktor `Object()`:

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

Hinzufügen von Komponenten:

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

(b) Durch Verwendung von einem Objektliteral:

JavaScript

Datentypen: Objekte [benutzerdefinierte Objekte]

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

Erzeugung von Objekten:

(a) Mit dem Konstruktor `Object()`:

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

Hinzufügen von Komponenten:

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

(b) Durch Verwendung von einem Objektliteral:

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

JavaScript

Datentypen: Objekte [benutzerdefinierte Objekte]

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

Erzeugung von Objekten:

(a) Mit dem Konstruktor `Object()`:

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

Hinzufügen von Komponenten:

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

(b) Durch Verwendung von einem Objektliteral:

```
var student = { matrNr:4711, 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.

JavaScript

Datentypen: Funktionen

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

Erzeugung von Funktionen:

(a) Mit dem Konstruktor `Function()`:

(b) Als `function`-Statement:

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

JavaScript

Datentypen: Funktionen

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

Erzeugung von Funktionen:

(a) Mit dem Konstruktor `Function()`:

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

(b) Als `function`-Statement:

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

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

JavaScript

Datentypen: Funktionen

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

Erzeugung von Funktionen:

(a) Mit dem Konstruktor `Function()`:

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

(b) Als `function`-Statement:

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

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

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

JavaScript

Datentypen: benutzerdefinierte Objekte [vordefinierte Objekte]

Benutzerdefinierte Objekte können durch die Definition eines entsprechenden Konstruktors erzeugt werden:

```
function MyStudent(m, n) {
  this.matrNr = m;
  this.name = n;
}

var student = new MyStudent(4711, "P. Müller")
```

JavaScript

Datentypen: benutzerdefinierte Objekte (Fortsetzung)

Definition von Methoden als separate Funktion oder als Funktionsliteral [Demo] :

```
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

Datentypen: benutzerdefinierte Objekte (Fortsetzung)

Definition von Methoden als separate Funktion oder als Funktionsliteral [Demo] :

```
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());
```

JavaScript

Datentypen: benutzerdefinierte Objekte (Fortsetzung)

Definition von Methoden als separate Funktion oder als Funktionsliteral [Demo] :

```
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 Komponente 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.
- Üblicherweise beginnt der Name einer Konstruktorfunktion mit einem Großbuchstaben.

JavaScript

Datentypen: Arrays [PHP] [Demo]

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:

JavaScript

Datentypen: Arrays [PHP] [Demo]

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; ...
```

JavaScript

Datentypen: Arrays [PHP] [Demo]

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

Kontrollstrukturen [PHP] [SELFHTML]

- Anweisungsfolge:
- Bedingte Anweisung:
- Return-Anweisung:
- `while`-Schleife:
- `for`-Schleife [Demo] :

JavaScript

Kontrollstrukturen [PHP] [SELFHTML]

- 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 < 0) {a = b;}     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 [Demo] :

JavaScript

Kontrollstrukturen [PHP] [SELFHTML]

- 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 < 0) {a = b;}     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 [Demo] :

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

JavaScript

Kontrollstrukturen [PHP] [SELFHTML]

- 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 < 0) {a = b;}     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 [Demo] :

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

- Parameterübergabe standardmäßig mittels call-by-value.

JavaScript

Funktionsbibliothek [PHP]

Die 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` [mozilla.org]

JavaScript

Funktionsbibliothek [PHP]

Die 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` [mozilla.org]

2. DOM-Objekte. Repräsentation von und Zugriff auf Dokumente (= Inhalte im Browser-Fenster).
   (a) DOM-Objekte nach der W3C Spezifikation. Zugriff über DOM-API.
   (b) DOM-Objekte nach Netscape. Zugriff über vordefinierte Arrays. (historisch)
   (c) DOM-Objekte nach Microsoft. Zugriff über das `document.all`-Objekt. (speziell)

JavaScript

Funktionsbibliothek [PHP]

Die 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` [mozilla.org]

2. DOM-Objekte. Repräsentation von und Zugriff auf Dokumente (= Inhalte im Browser-Fenster).
   (a) DOM-Objekte nach der W3C Spezifikation. Zugriff über DOM-API.
   (b) DOM-Objekte nach Netscape. Zugriff über vordefinierte Arrays. (historisch)
   (c) DOM-Objekte nach Microsoft. Zugriff über das `document.all`-Objekt. (speziell)

3. Browser-Objekte. Repräsentation von und Zugriff auf Browser-Fenster.
   Beispiele: `History, Location, Navigator, Screen, Window` [mozilla.org]

Ergänzung durch vordefinierte Funktionen.

Beispiele: `eval, isFinite, isNaN, parseInt` [mozilla.org]

Bemerkungen:

- Kernobjekte werden auch als *Global Objects*, *Objects in the Global Scope* oder *Build-in Objects* bezeichnet. Vordefinierte Funktionen werden auch als *Global Functions* oder *Build-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 (= Wurzel*knoten* des 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 (= Wurzel*element* des XML-Dokuments bzw. Dokumentelement). [XML-Knotentypen unter dem XPath-Modell]
- 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

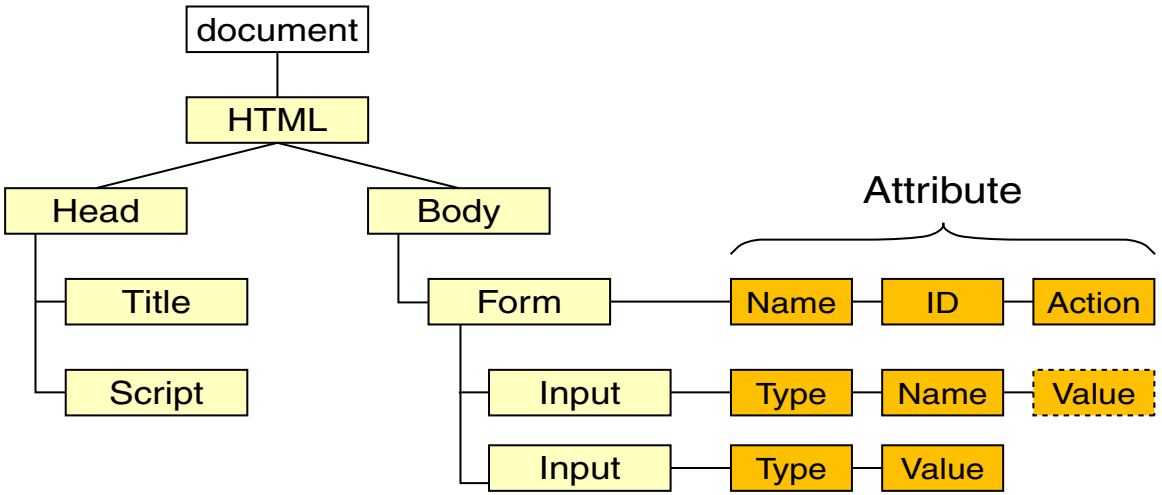
Funktionsbibliothek: DOM-Objekte [SELFHTML]

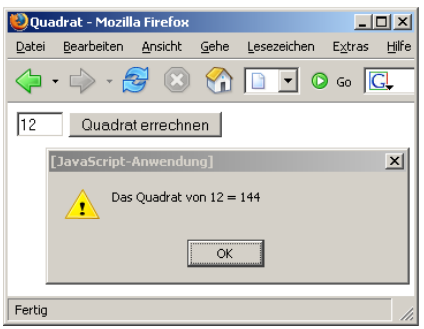
Konzepte, um auf HTML-Elementobjekte und deren Komponenten zuzugreifen:

- Eindeutig qualifizierender Name gemäß DOM-Dokumentbaum.
- DOM-API, um Elementmenge auszuwählen [2. (a)] :
  - `getElementsByName()`
  - `getElementById()`
  - `getElementsByTagname()`
  - `...`
- Vordefinierte Arrays, die alle Elemente eines Typs enthalten [2. (b)] :
  - `document.forms`
  - `document.images`
  - `...`
- Kombiniert: Auswahl von Elementmenge plus weitere Spezialisierung mittels qualifizierendem Namen.

JavaScript

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

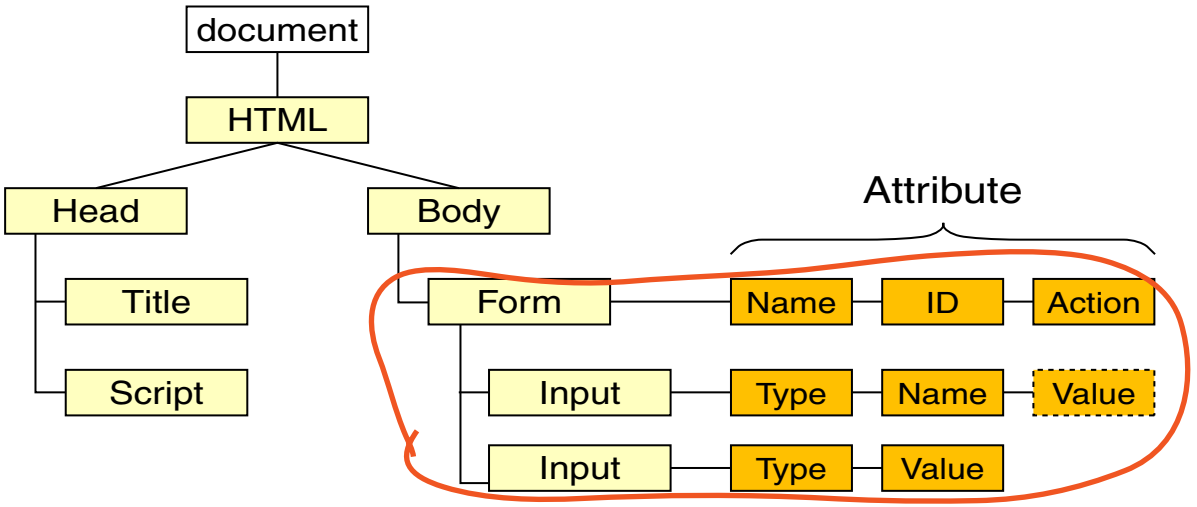





```
<html>
  <head>
    ...
    <script type="text/javascript">
      function Quadrat() {...}
    </script>
  </head>
  <body>
    <form name="QuadratForm" id="QF" action="">
      <input type="text" name="Eingabe" size="3">
      <input type="button" value="Quadrat errechnen" onClick="Quadrat()">
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

JavaScript

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



Zugriffsmöglichkeiten auf das erste Formelement:

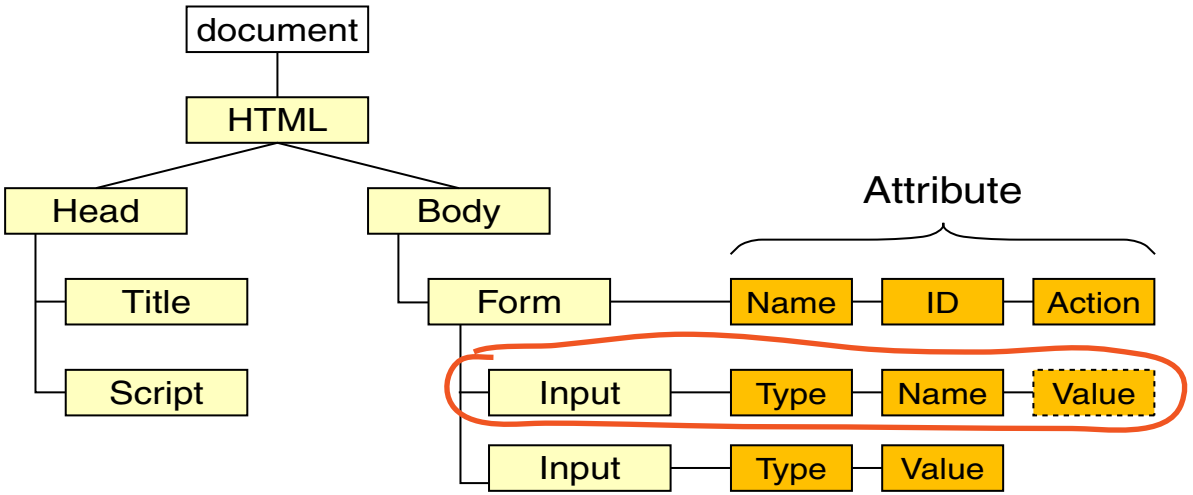
`document.QuadratForm`	qualifizierender Name
`document.getElementsByTagName("form")[0]`	DOM-API
`document.getElementById("QF")[0]`	DOM-API
`document.forms[0]`	vordefiniertes Array

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:

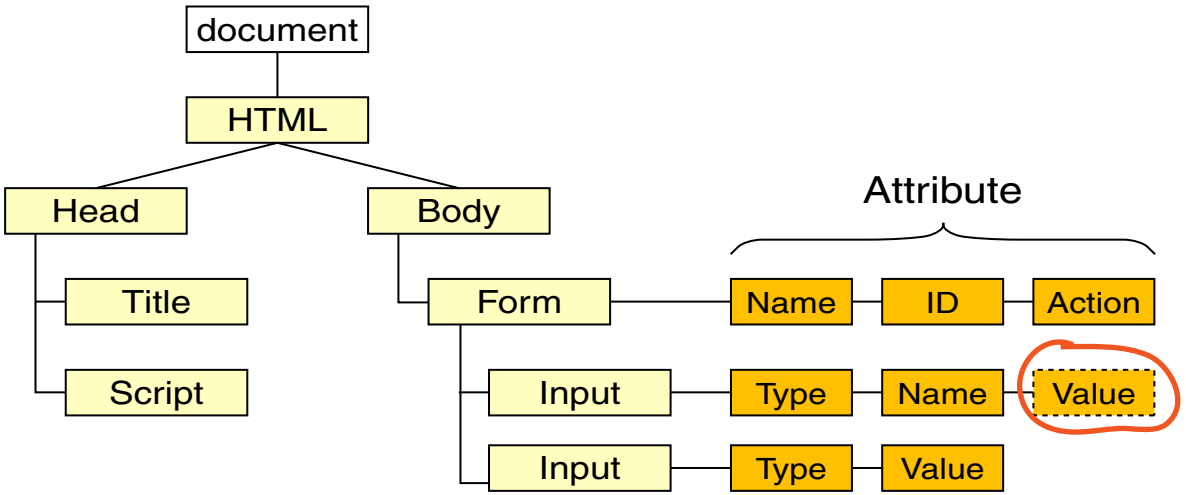
`document.QuadratForm.Eingabe`	qualifizierender Name
`document.getElementsByTagName("form")[0].Eingabe`	DOM-API
`document.getElementsByName("Eingabe")[0]`	DOM-API
`document.forms[0].Eingabe`	vordefiniertes Array

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:

- Die Verwendung von Netscapes vordefinierten Arrays ist – genauso wie Microsofts `all`-Objekt – eine Browser-spezifische Lösung; sie hat größtenteils historische Gründe. Langfristig wird sich die Objektphilosophie des W3C DOM durchsetzen. Das heißt, die Zugriffe auf Dokumentbestandteile erfolgen gemäß der DOM-API des DOM Core- und des DOM HTML-Moduls – 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 `submit()` und `reset()`. [SELFHTML]

JavaScript

Funktionsbibliothek: DOM-Objekte (Fortsetzung)

HTML-Element-Objekte. [W3C] [w3schools] :

- Anchor
- Area
- Base
- Body
- Button
- Form
- Frame
- Frameset
- IFrame
- Image
- Input Button
- Input Checkbox
- Input File
- Input Hidden
- Input Password
- Input Radio
- Input Reset
- Input Submit
- Input Text
- Link
- Meta
- Object
- Option
- Select
- Style
- Table
- td, th, tr
- Textarea

JavaScript

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:

```
<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>
```

JavaScript

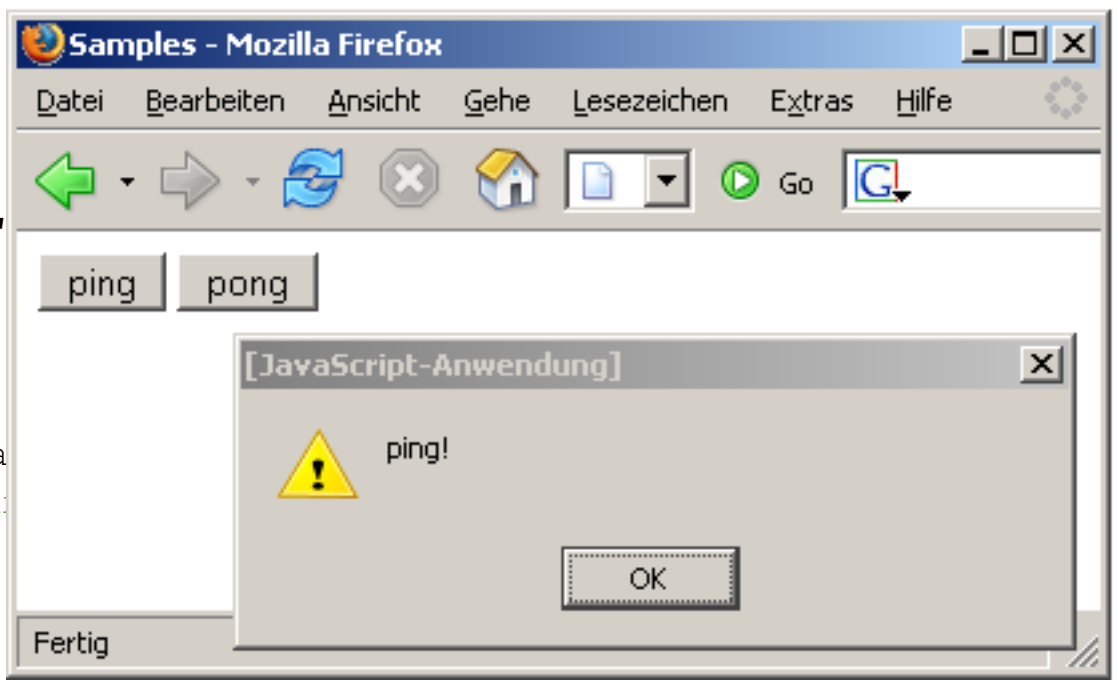
Ereignisbehandlung

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:

```
<html>
  <head>
    <title>Event</title>
  </head>
  <body>
    <form name="testForm"
      <input type="button
      <input type="button
    </form>

    <script type="text/ja
      document.testForm.K
    </script>
  </body>
<html>
```

[Demo]



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]

JavaScript

Ereignisbehandlung: wichtige Ereignisse

Event-Handler	HTML-Elemente	Semantik
`onclick`	Knopf, Checkbox, Anker	Element wird angeklickt
`onchange`	Textfeld, Textbereich, Auswahl	Wert wird geändert

`onkeydown`
`onkeyup`
`onkeypress`
	Dokument, Bild, Anker,Textfeld	Taste gedrückt/losgelassen

`onload`	Body	Beim Laden eines Dokuments
`onmousedown` `onmouseup`	Dokument, Knopf, Anker	Maustaste gedrückt/losgelassen
`onmouseout`	Bereiche, Anker	Mauszeiger verlässt einen Bereich
`onmouseover`	Anker	Mauszeiger über Anker
`onreset`, `onsubmit`	Formular	Reset/Submit für ein Formular
`onselect`	Textfeld, Textbereich	Element wird ausgewählt
`onfocus` `onblur`	Fenster, alle Formularelemente	Eingabefokus wird dem Element gegeben/entzogen

JavaScript

Quellen zum Nachlernen und Nachschlagen im Web

- S. Münz. *SELFHTML: JavaScript*
  de.selfhtml.org/javascript
- W3 Schools. *JavaScript Tutorial.*
  www.w3schools.com/js
- M. Wilton-Jones. *JavaScript Tutorial.*
  www.howtocreate.co.uk/tutorials/javascript
- M. Schäfer. *Einführung in JavaScript.*
  molily.de/js
- mozilla.org. *JavaScript.*
  https://developer.mozilla.org/en/JavaScript

- M. Miller, W. Horwat, M. Samuel. *Changes to JavaScript, Part 1: EcmaScript 5.*
  GoogleTechTalk on www.youtube.com
- ECMA International. *ECMAScript Language Specification, 5th Edition Dec. 2009.*
  www.ecma-international.org/publications/standards/ECMA-262

JavaScript

Quellen zum Nachlernen und Nachschlagen im Web

- mozilla.org. *Firebug Firefox Extension.*
  addons.mozilla.org/en-US/firefox/addon/firebug
- mozilla.org. *Venkman JavaScript Debugger.*
  developer.mozilla.org/en/Venkman

Kapitel WT:V (Fortsetzung)

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 Softwarekomponenten

Anwendung [Demo] :

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

Java Applet

Einführung (Fortsetzung)

Ein einfaches Applet:

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

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

Java Applet

Einführung (Fortsetzung)

Ein einfaches Applet:

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

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

Eine HTML-Seite, die das Applet einbindet:

```
<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="." >
    </object>
  </body>
</html>
```

Java Applet

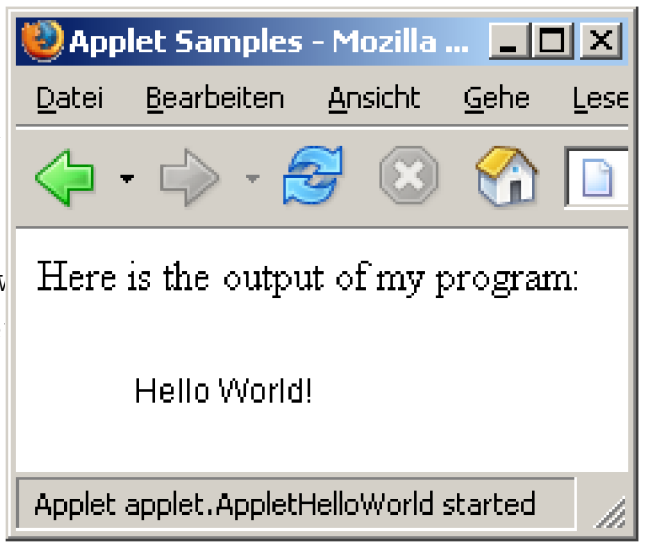
Einführung (Fortsetzung)

Ein einfaches Applet:

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

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

Eine HTML-Seite, die das Applet einbindet:

```
<html>
  <head> <title>Applet Sample</title> </head>
  <body>
    <p>Here is the output of my program:</p>
    <object type="application/x-java-applet" v
       <param name="code" value="applet.Apple
       <param name="codebase" value="." >
    </object>
  </body>
</html>
```



[Demo]

Java Applet

Einführung (Fortsetzung)

Ein einfaches Applet:

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

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

Eine vergleichbare Java-Anwendung:

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

public class ApplicationHelloWorld extends Frame{

  public ApplicationHelloWorld(){
    Label l=new Label("Hello World!");
    add(l);}

  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="appletinstancename"] >
  <param name="code" value="appletfile">
 [<param name="archive" value="file1.jar, file2.jar">]
 [<param name="codebase" value="codebaseurl">]
 [<param name="appletparameter_1" value="value_1">]
  ...
 [alternatehtml]
</object>
```

Java Applet

Einbindung in HTML-Dokumente

```
<object
  type="application/x-java-applet"
  width="pixels"
  height="pixels"
 [name="appletinstancename"] >
  <param name="code" value="appletfile">
 [<param name="archive" value="file1.jar, file2.jar">]
 [<param name="codebase" value="codebaseurl">]
 [<param name="appletparameter_1" value="value_1">]
  ...
 [alternatehtml]
</object>
```

Beispiel:

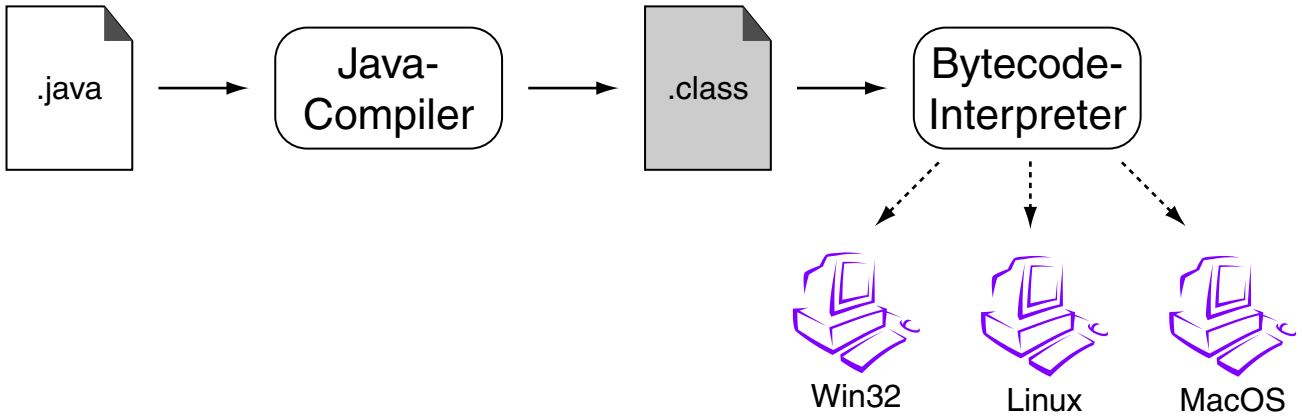
```
<object type="application/x-java-applet" width="500" height="20">
  <param name="code" value="aisearch/client/Client.class">
  <param name="archive" value="engine/aisearch.jar">
  <param name="imagesource" value="images/picture1.jpg">
  <param name="backgroundcolor" value="0xc0c0c0">
  ...
```

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.
- Die Einbindung via `applet`-Tag ist deprecated, funktioniert aber noch in den gängigen Browsern.

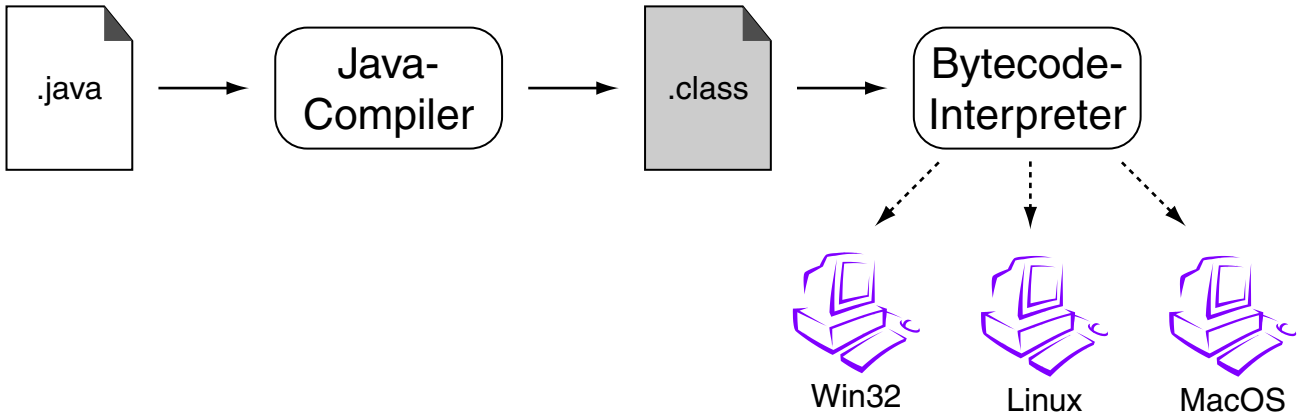
Java Applet

Java-Plattform



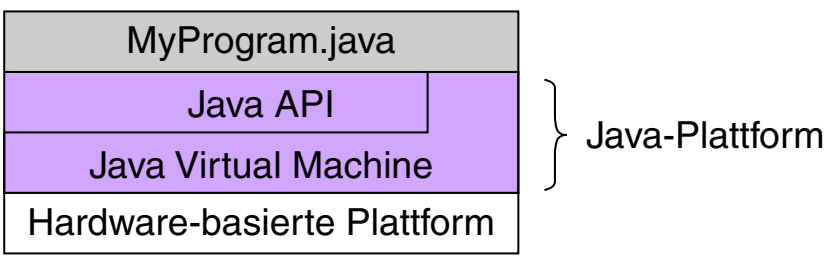
Java Applet

Java-Plattform



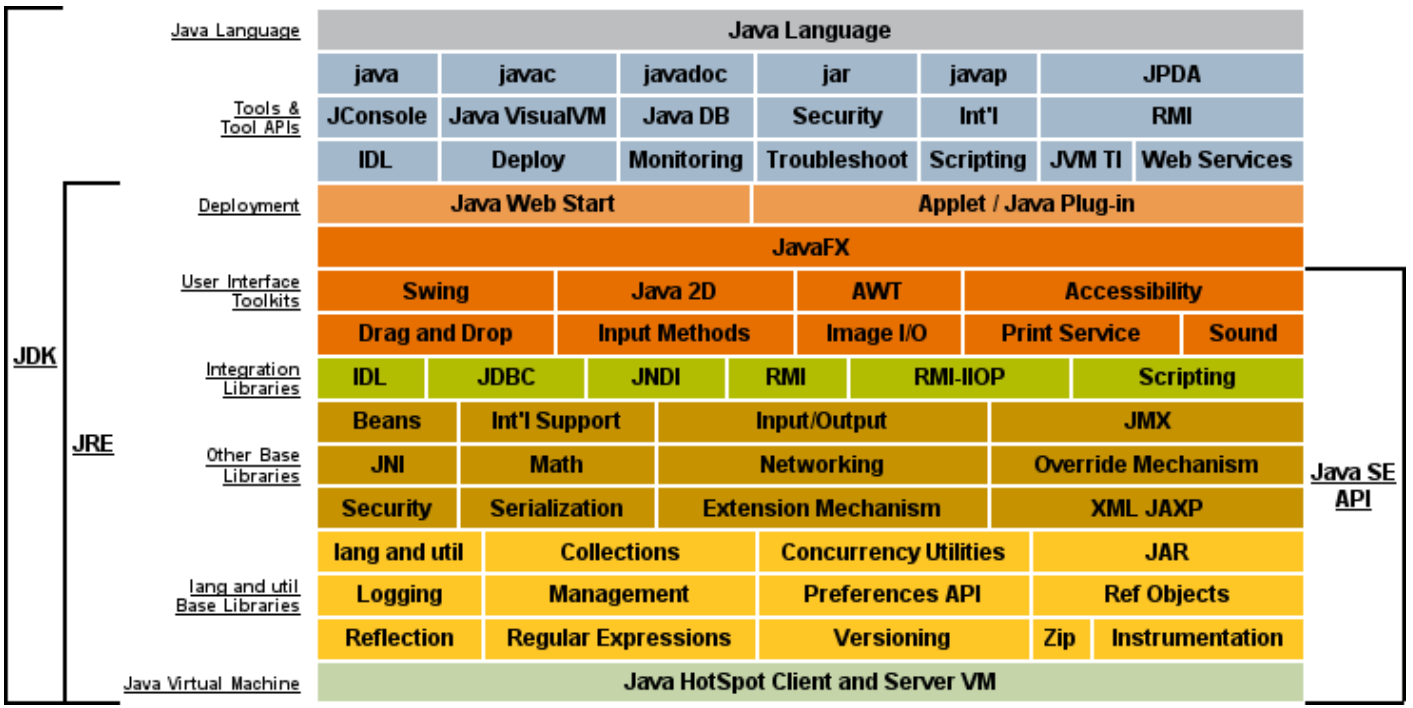
“A platform is the hardware or software environment in which a program runs. [...] Most platforms can be described as a combination of the operating system and underlying hardware. The Java platform differs from most other platforms in that it’s a software-only platform that runs on top of other hardware-based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

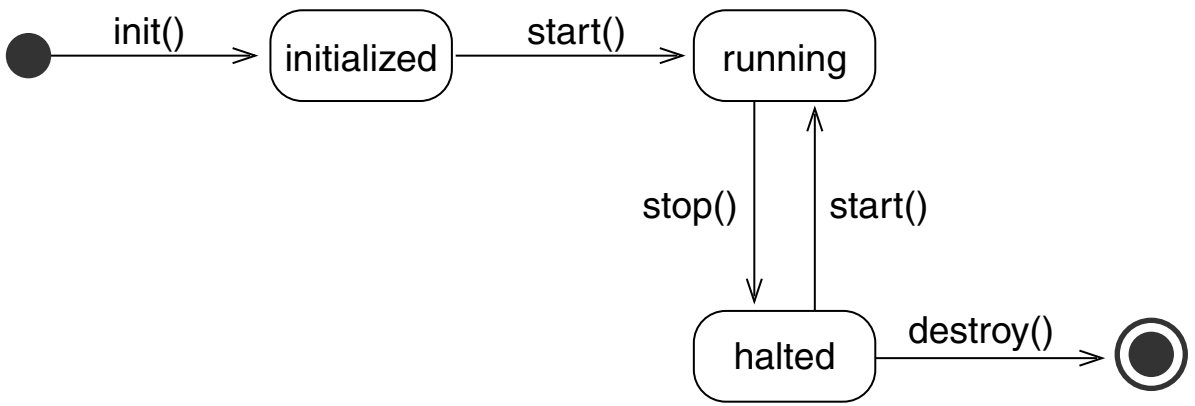
Die entsprechenden Java-Befehle:

```
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
  ...
}
```

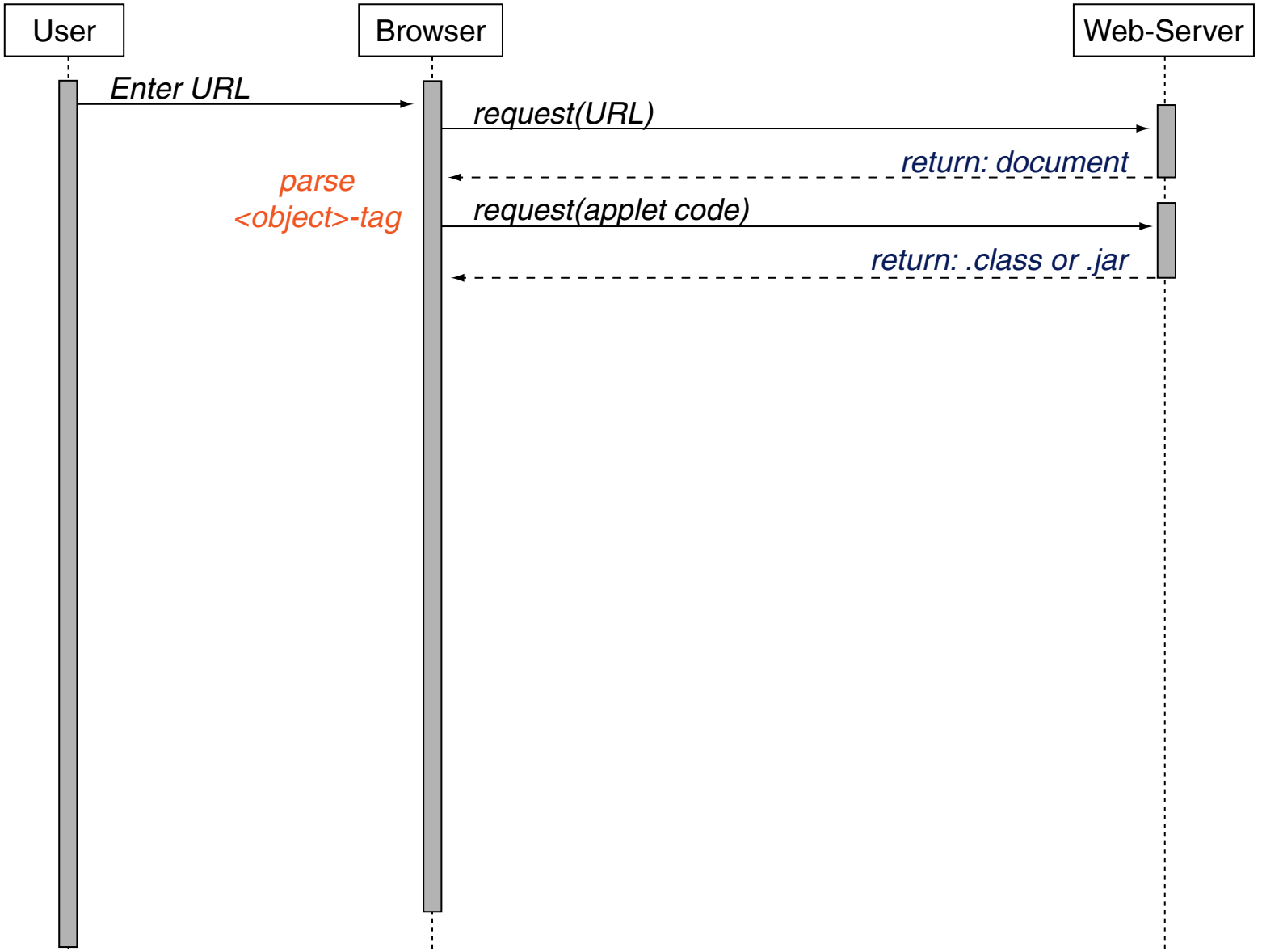
Java Applet

Applet-Lebenszyklus (Fortsetzung)

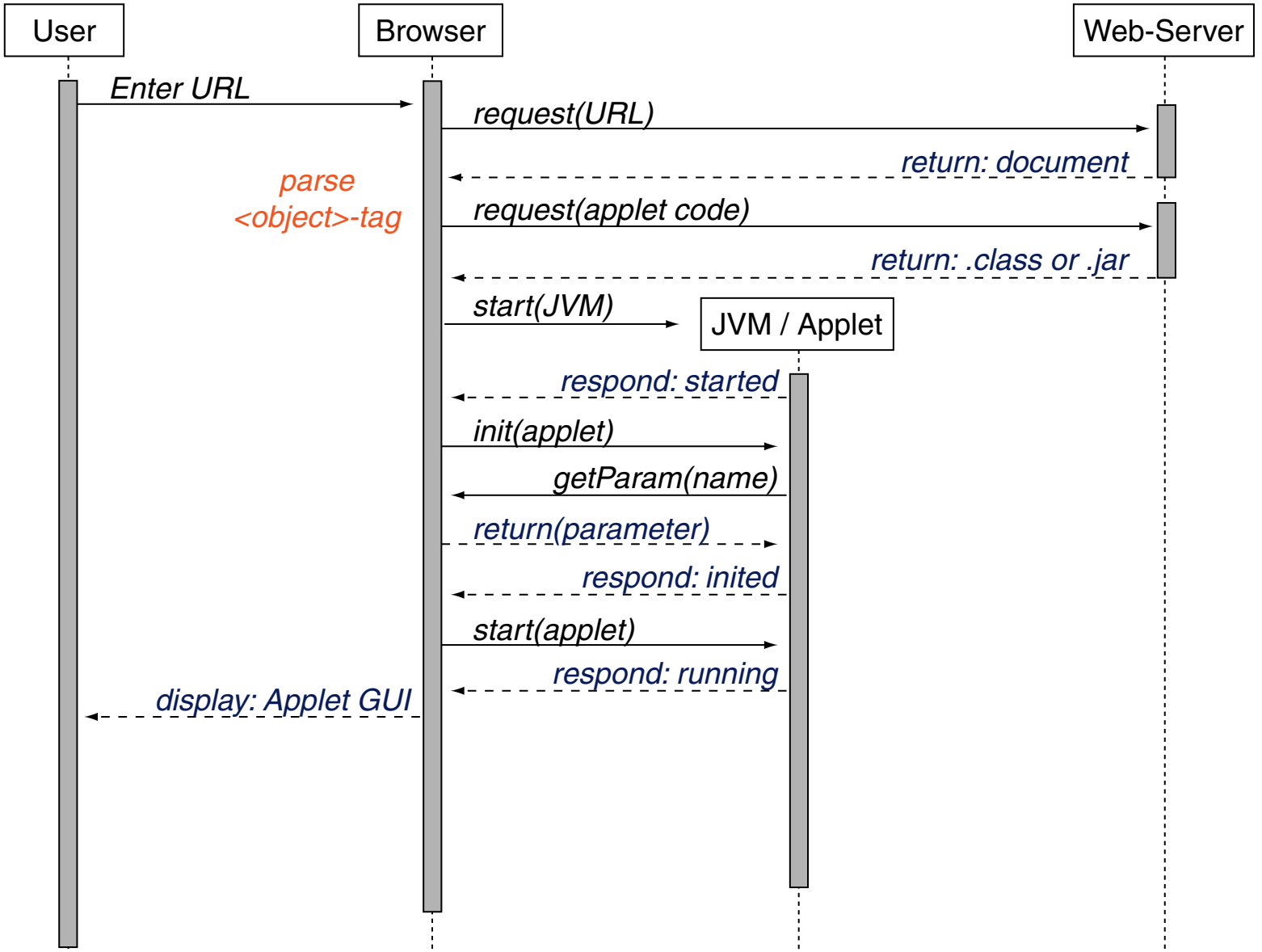
Nach dem Laden instantiiert der Browser die Applet-Klasse, ruft dann die `init()`-Methode und danach die `start()`-Methode auf.



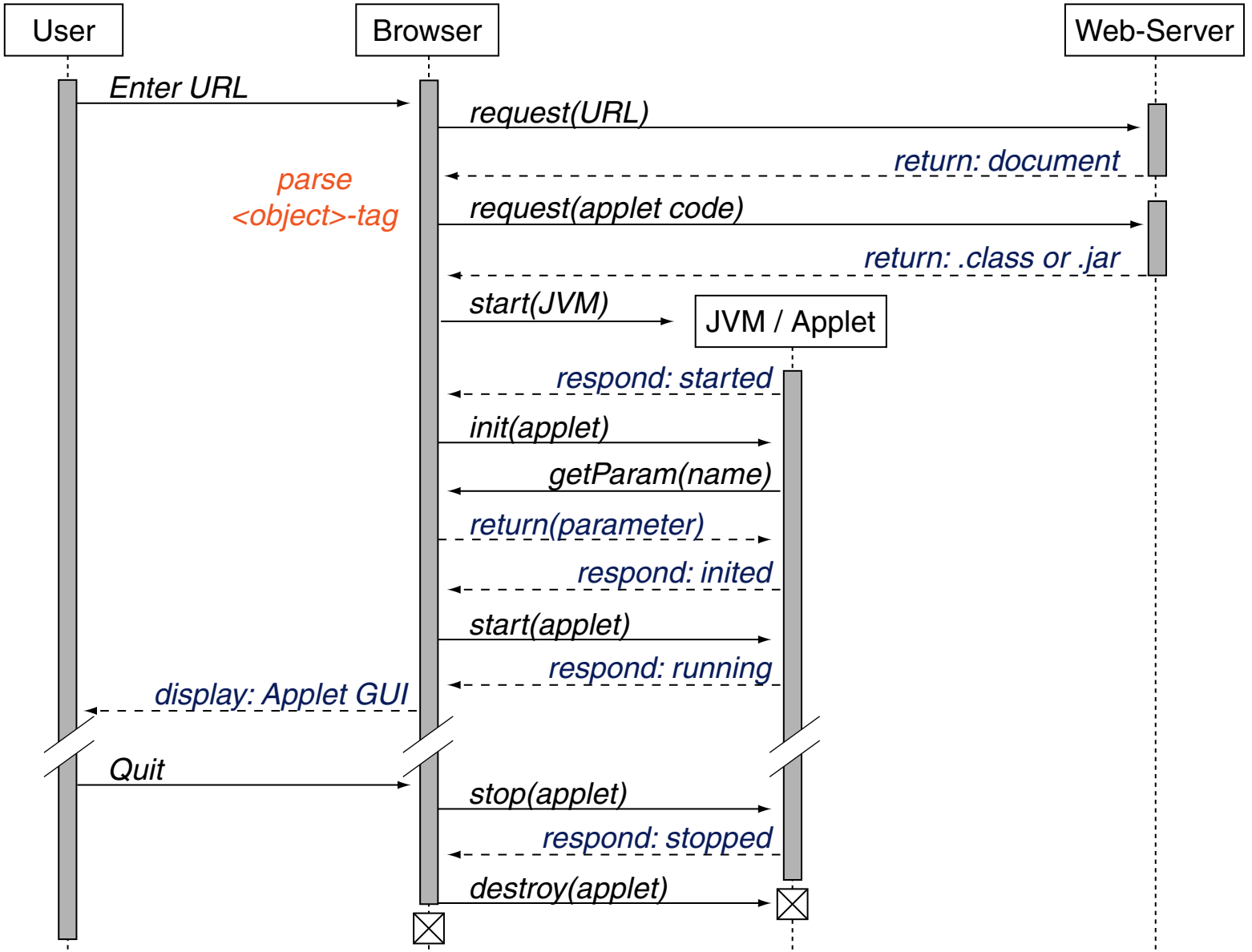
Java Applet



Java Applet



Java Applet



Java Applet

Beispiel: Anfordern von Web-Dokumenten

[Demo]

Java Applet

Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

Organisiert als Frameset in drei HTML-Dokumenten:

```
<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>
      <p>Your browser cannot display frames.<p>
    </noframes>
  </frameset>
</html>
```

Java Applet

Beispiel: Anfordern von Web-Dokumenten (Fortsetzung)

HTML-Dokument AppletWindowControl.html:

```
<html>
    <title>Applet Window Control</title>
  <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="applet-window-control.jar">
      <h1>Please install Java</h1>
    </object>
  </body>
</html>
```

HTML-Dokument AppletWindowContent.html:

```
<html>
    <title>Applet Window Content</title>
  <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
WindowName	Anzeige in dem (eventuell neu erzeugten) Fenster *WindowName*.
`"_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

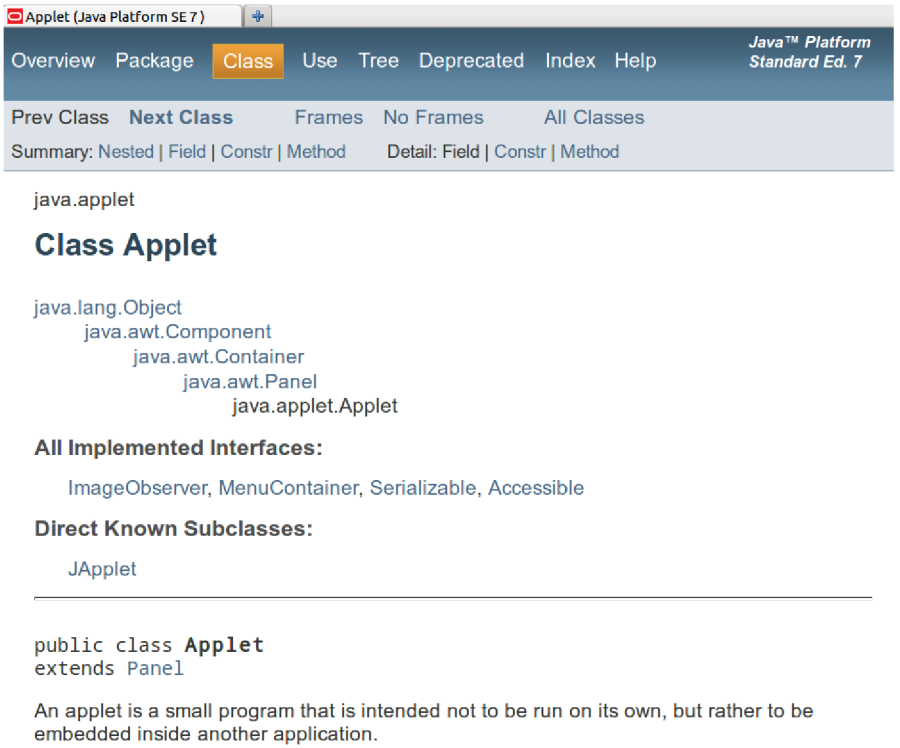
Applet-Programmierung heißt oft Oberflächenprogrammierung. Das JDK stellt verschiedene Klassen zur Realisierung von Benutzer-Interfaces bereit:

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

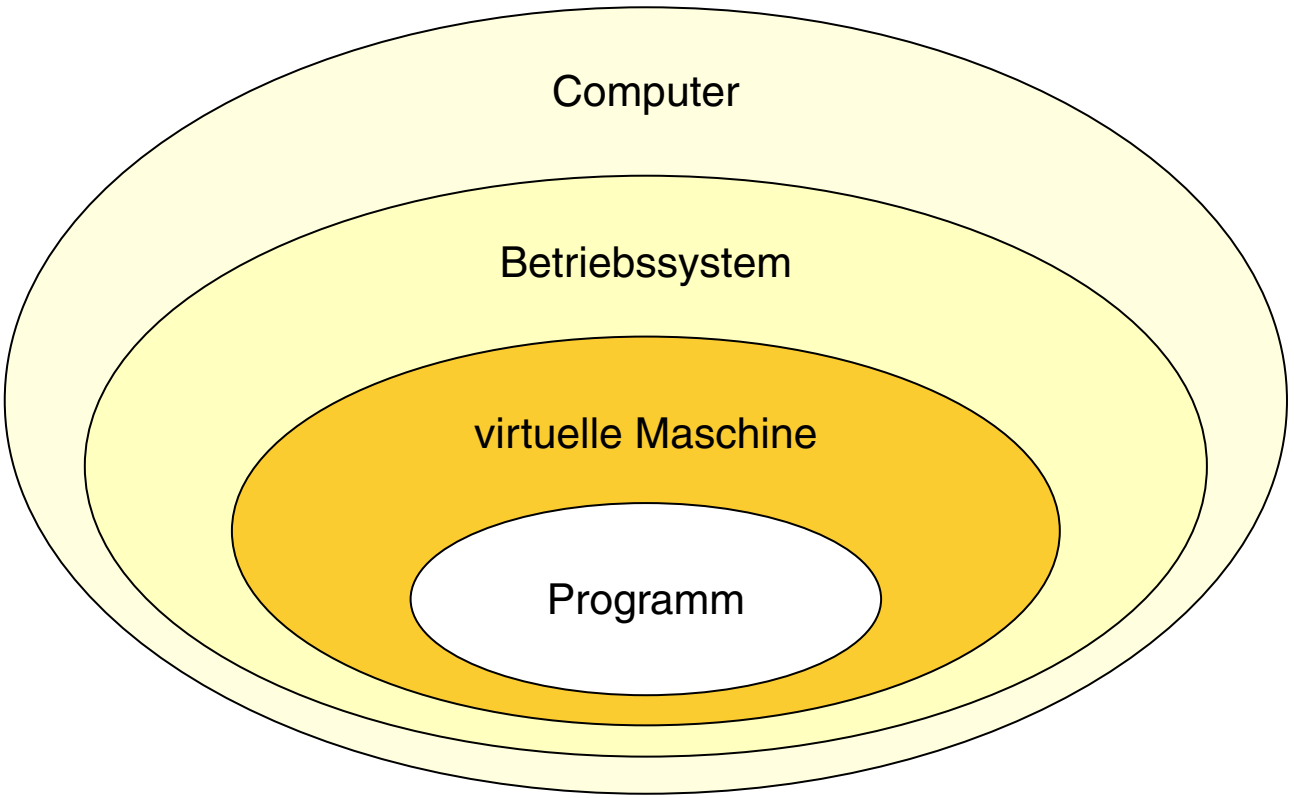
Bemerkungen:

- 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 unbekannten 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:

$$e_k : P \to C$$
$$d_k : C \to P$$

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:

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

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)

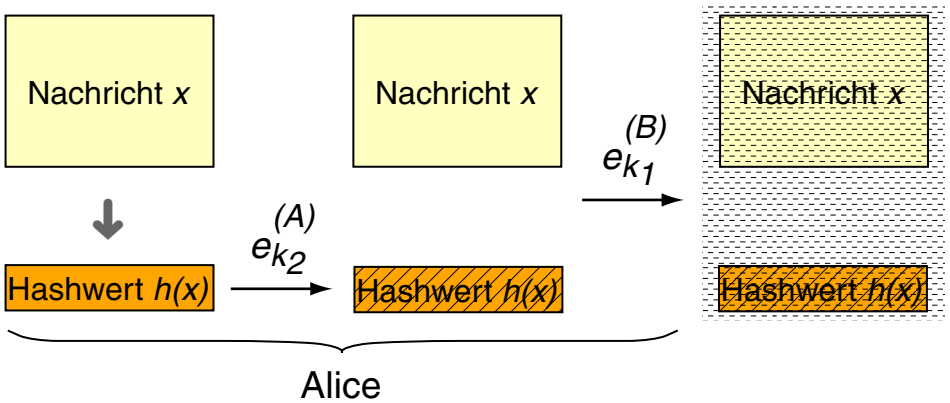
Woher weiß Bob, dass Alice und nicht Eve der Autor von Nachricht x ist?

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:

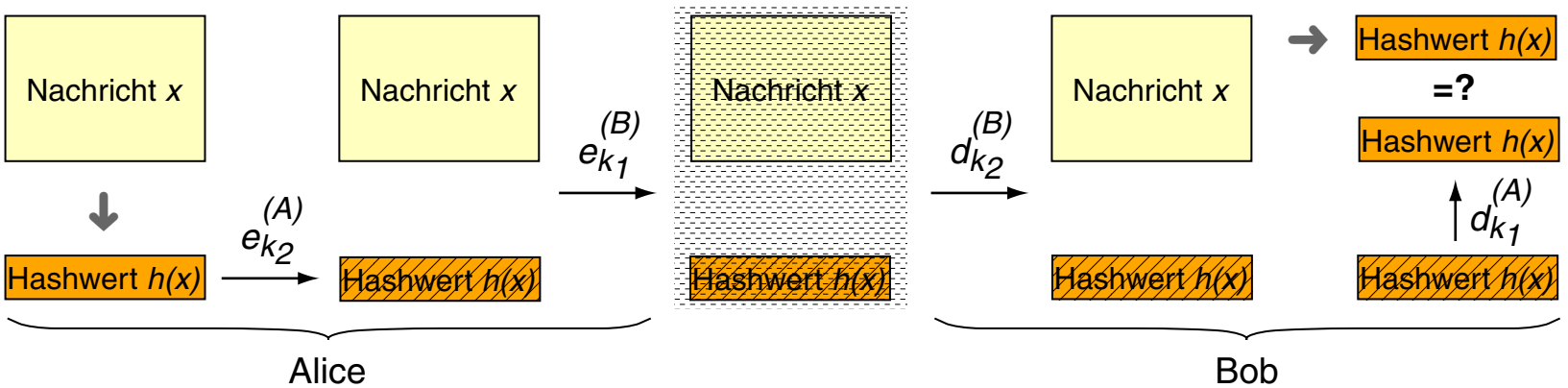
1. Alice berechnet für Nachricht x den Hashwert $h(x)$.
2. Alice verschlüsselt $h(x)$ als $y_h = e_{k_2}^{(A)}(h(x))$.
3. Alice versendet Nachricht $x + y_h$ als $e_{k_1}^{(B)}(x + y_h)$ zu Bob.

Java Applet

Signierung (Fortsetzung)

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

Sei $h : P \to N$ 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

- J. Gosling, H. McGilton. *The Java Language Environment.*
  www.oracle.com/technetwork/java/
- ORACLE. *Learning the Java Language.*
  docs.oracle.com/javase/tutorial/java/
- ORACLE. *Signing Code and Granting It Permissions.*
  docs.oracle.com/javase/tutorial/security/toolsign/
- ORACLE. *Applets.*
  docs.oracle.com/javase/tutorial/deployment/applet/
- ORACLE. *How to Make Applets. (Swing)*
  docs.oracle.com/javase/tutorial/uiswing/components/applet.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.

Weitere Client-Technologien

Vergleich der Technologien

	Programmier-aufwand	Anwendungen	Abhängigkeit vom Browser	Abhängigkeit vom Betriebssystem
Hilfsprogramm		Darstellung besonderer Medientypen	keine	hoch
Plugin	sehr hoch	"	hoch	hoch
Skript	mittel	Eingabevalidierung, dynamische Änderung von Dokumenten	mittel	keine
Java Applet	hoch	komplexere Anwendungen mit grafischer Oberfläche	gering	gering
ActiveX	hoch	"	hoch	extrem

[Turau 1999]