

Web programiranje

HTML

World Wide Web

- WWW je osmišljen u CERN-u, početkom 90-tih godina prošlog veka
- Klijent-server arhitektura
 - Klijenti su čitači (browser)
 - Serveri su web serveri
- HTTP protokol – protokol za komunikaciju između web čitača i servera
 - Klijent traži neki resurs od servera
 - Server isporučuje resurs klijentu
- Resurs može biti
 - HTML datoteka
 - slika
 - film
 - itd.
- Za WWW nije neophodan internet i web server
 - možemo da učitamo HTML datoteku sa diska u čitač

HTTP protokol

HTTP
klijent

HTTP
server

```
GET /docs.html HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
Accept-Cookies: yes
Host: pipin.tmd.ns.ac.yu
...
```

HTTP
klijent

HTTP
server

```
HTTP/1.0 200 OK
Content-Type: text/html

<HTML>
<HEAD>
...
```

Šta je HTML?

- HTML je skraćenica od Hyper Text Markup Language
 - markap jezik koji opisuje stranice koje imaju strukturu i izgled
 - markap jezik označava tekst
 - dodatnim oznakama (tagovima)
- Tekstualna datoteka
 - sastoji se iz markap tagova
 - tagovi
 - definišu strukturu HTML dokumenta
 - objašnjavaju čitaču kako da prikaže stranicu
 - ekstenzija je html ili htm
 - svi whitespace-ovi (razmak, tab, enter) se svode na jedan

Primer

```
<html>
```

```
<head>
```

```
  <title>Naslov</title>
```

```
</head>
```

```
<body>
```

```
  <p>Moj prvi HTML dokument. <b>Ovaj  
  tekst je podebljan.</b></p>
```

```
</body>
```

```
</html>
```

HTML tagovi

- Označavaju HTML elemente
- HTML element se sastoji iz
 - početnog (otvarajućeg) taga <tag>
 - sadržaja
 - zatvarajućeg taga </tag>
- Tag počinje znakom <
- Tag se završava znakom >
- Vrste:
 - složeni (imaju i otvarajući i zatvarajući tag)
 - prazni (prosti) tagovi, gde nema zatvarajućeg taga i sadržaja
 - primer:

- Tagovi ne smeju da se preklapaju

Primer

- Jednostavan element

This text is bold

- Element koji sadrži drugi

<body>

**This is my first
homepage.This text is
bold. This text is not
bold.**

</body>

HTML tagovi

- Nisu osetljivi na mala i velika slova
- Preporuka je da se pišu malim slovima
- Potrebno za XHTML (HTML sledeće generacije)

Osnovni tagovi

- `<html>`
 - obuhvata ceo HTML dokument
- `<head>`
 - zaglavlje HTML dokumenta
- `<body>`
 - telo HTML dokumenta

Osnovni tagovi

- Tag **<html>** je okvir u kom se nalaze svi ostali tagovi
 - HTML dokument uvek počinje tagom `<html>`, a završava se tagom `</html>`.
- Tag **<head>** uokviruje zaglavlje u kom se nalaze informacije o samom dokumentu (naslov, opis, ključne reči, ime autora, itd.); opcion je.
- Sve ono što vidimo u prozoru browser-a, tj. sadržaj stranice, nalazi se u telu dokumenta koje uokviruje element **<body>**
 - u dokumentu sme da postoji samo jedan par tagova `<body></body>`.

Tag <head>

- Browser ne prikazuje informacije koje se nalaze između tagova **<head>** i **</head>**, osim sadržaja taga **<title>**.
- Sadržaj taga **<title>** je naslov HTML dokumenta koji će se pojaviti u naslovnoj liniji web čitača.

Tag <body>

- Ovaj element specificira glavni sadržaj dokumenta
- Početni tag **<body>** može da ima attribute koji omogućavaju da se specificiraju karakteristike dokumenta:
 - boja pozadine ili slika, boja teksta, boja posećenih i neposećenih linkova,
 - akcije koje se izvršavaju kada se dokument učitava, ili ako se ne učitava iz nekog razloga, itd.),ali je bolje da se ove karakteristike podešavaju CSS-om.

Osnovni tagovi

- `<h1>` – `<h6>`
 - naslovi
- `<p>`
 - paragraf
- `
`
 - line break
- `<hr />`
 - horizontal ruler – horizontalna crta
- `<!-- komentar -->`

Primer

```
<html>
<head>
  <title></title>
</head>

<body>
  <h1>Naslov</h1>
  <p>Ovo je paragraf.</p>

  <h2>Podnaslov</h2>
  <p>Ovo je paragraf.</p>
  <p>Ovo je paragraf.</p>

  <h1>Drugi naslov</h1>
  <p>Ovo je paragraf.</p>
</body>
</html>
```

Primer

Naslov

Ovo je paragraf.

Podnaslov

Ovo je paragraf.

Ovo je paragraf.

Drugi naslov

Ovo je paragraf.

Atributi

- Atributi obezbeđuju dodatnu informaciju za tagove
- Tagovi mogu da imaju attribute
- Atributi su oblika **ime="vrednost"**
- Atributi se uvek stavljaju u početni tag
- Primer:

```

```


Tagovi za formatiranje teksta

- ``
 - bold, **podebljano**
- `<i>`
 - italic, *iskošeno*
- `<u>`
 - underline, podvučeno
- `<sub>`
 - subscript, _{dole}
- `<sup>`
 - superscript, ^{gore}

Primer

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
  />
<title>Naslov</title>
</head>

<body>

<h2><u>Formatiranje teksta</u></h2>
<p><b>podebljano</b></p>
<p><i>iskoseno</i></p>
<p><u>podvuceno</u></p>
<p>X<sub>indeks</sub></p>
<p>X<sup>eksponent</sup></p>
</body>
</html>
```

Primer

Formatiranje teksta

podebljano

iskoseno

podvuceno

X_{indeks}

$x^{\text{eksponent}}$

Tagovi za formatiranje teksta

- `<big>`
 - veća slova
- `<small>`
 - manja slova
- ``
 - emphasized, naglašeno
- ``
 - naglašeno

Primer

```
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>

<h2><u>Tagovi za formatiranje teksta</u></h2>
<p><strong>naglašeno</strong></p>
<p><em>naglašeno</em></p>
<p><big>veća slova</big></p>
<p><small>manja slova</small></p>
</body>
</html>
```

Primer

Tagovi za formatiranje teksta

naglaseno

naglaseno

veća slova

manja slova

Tagovi za formatiranje teksta

- `<ins>`
 - inserted, podvučen tekst
- ``
 - deleted, obrisani, precrtani tekst

Primer

```
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>

<h2><u>Tagovi za formatiranje teksta</u></h2>
<p><ins>inserted, podvučen tekst</ins></p>
<p><del>precrtan tekst</del></p>
</body>
</html>
```


Primer

Tagovi za formatiranje teksta

inserted, podvučen tekst

~~prebrisan tekst~~

“Kompjuterski” tagovi

- `<code>`
 - za kratke delove koda, unutar rečenice
- `<pre>`
 - preformatted, sav sadržaj ovog taga se prikazuje u originalnom obliku, sa svim whitespace-ovima
 - pogodan za listinge
- `<var>`
 - za varijable u programu
- `<tt>`
 - teletype text, teleprinter
- `<samp>`
 - sample, uzorak
- `<kbd>`
 - keyboard, za prikaz unosa sa tastature

Primer

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
  />
<title>Naslov</title>
</head>

<body>

<h2><u>Kompjuterski tagovi </u></h2>
<p>Ovo je običan tekst dok je <code>ovo neki kod</code></p>
<pre>preformatted text</pre>
<p><var>varijable u programu</var></p>
<p><tt>teletype text, teleprinter</tt></p>
<p><samp>sample, uzorak</samp></p>
<p><kbd>keyboard, za prikaz unosa sa tastature</kbd></p>
</body>
</html>
```

Primer

Kompjuterski tagovi

Ovo je običan tekst dok je ovo neki kod

preformatted text

varijable u programu

teletype text, teleprinter

sample, uzorak

keyboard, za prikaz unosa sa tastature

Skraćenice, citati, definicije

- `<abbr title="objašnjenje">tekst</abbr>`
 - abbreviation, skraćenica
- `<acronym title="objašnjenje">tekst<acronym>`
 - akronim (WWW-**W**orld **W**ide**W**eb)
- `<address>`
 - za definisanje adrese
- `<bdo dir="rtl">`
 - smer ispisa teksta (sa leva na desno i obrnuto)
 - atribut **dir** (direction – smer): rtl, ltr

Primer

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>

<h2><u>Skraćenice, citati, definicije</u></h2>
<abbr title="objašnjenje pojma">abbreviation, skraćenica</abbr>
<p><acronym title="World Wide Web">WWW</acronym></p>
<address>adresa</address>
<p><bdo dir="ltr">smer ispisa teksta (sa leva na desno i obrnuto)</bdo></p>
<p><bdo dir="rtl">smer ispisa teksta (sa leva na desno i obrnuto)</bdo></p>
</p>
</body>
</html>
```

Primer

Skraćenice, citati, definicije

abbreviation, skraćenica

WWW

adresa

smer ispisa teksta (sa leva na desno i obrnuto)

(otunrbo i onsed an avel as) atsket asipsi remis

Skraćenice, citati, definicije

- `<blockquote>`
 - duži citat
- `<q>`
 - kraći citat
- `<cite>`
 - citat
- `<dfn>`
 - za definicije u tekstu

Primer

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>

<h2><u>Skraćenice, citati, definicije</u></h2>
<p>Ovo je običan tekst dok
<blockquote>ovde nesto citiramo - duzi citat</blockquote>
</p>
<p>Ovo je običan tekst dok
<q>ovde nesto citiramo - kraći citat</q>
</p>
<p>Ovo je običan tekst dok
<cite>ovde nesto citiramo.</cite>
</p>
<p>Ovo je običan tekst dok
<dfn>ovde nesto definisemo - definicija.</dfn>
</p>
</body>
</html>
```

Primer

Skraćenice, citati, definicije

Ovo je običan tekst dok

ovde nesto citiramo - duzi citat

Ovo je običan tekst dok "ovde nesto citiramo - kraći citat"

Ovo je običan tekst dok *ovde nesto citiramo.*

Ovo je običan tekst dok *ovde nesto definisemo - definicija.*

Specijalni karakteri

- Character entity
- Karakteri koji su rezervisani za tagove (<, >, &, itd.)
- Non-breaking Space
 - razmaci koji se ne sažimaju u jedan
 - više običnih razmaka se prikazuju kao jedan
 -
- Karakteri
 - specijalni karakteri (£, €, itd.)
 - naša slova (š, đ, itd.)
 - ćirilica (ш, ђ, itd.)
 - ostalo, po **Unicode** standardu

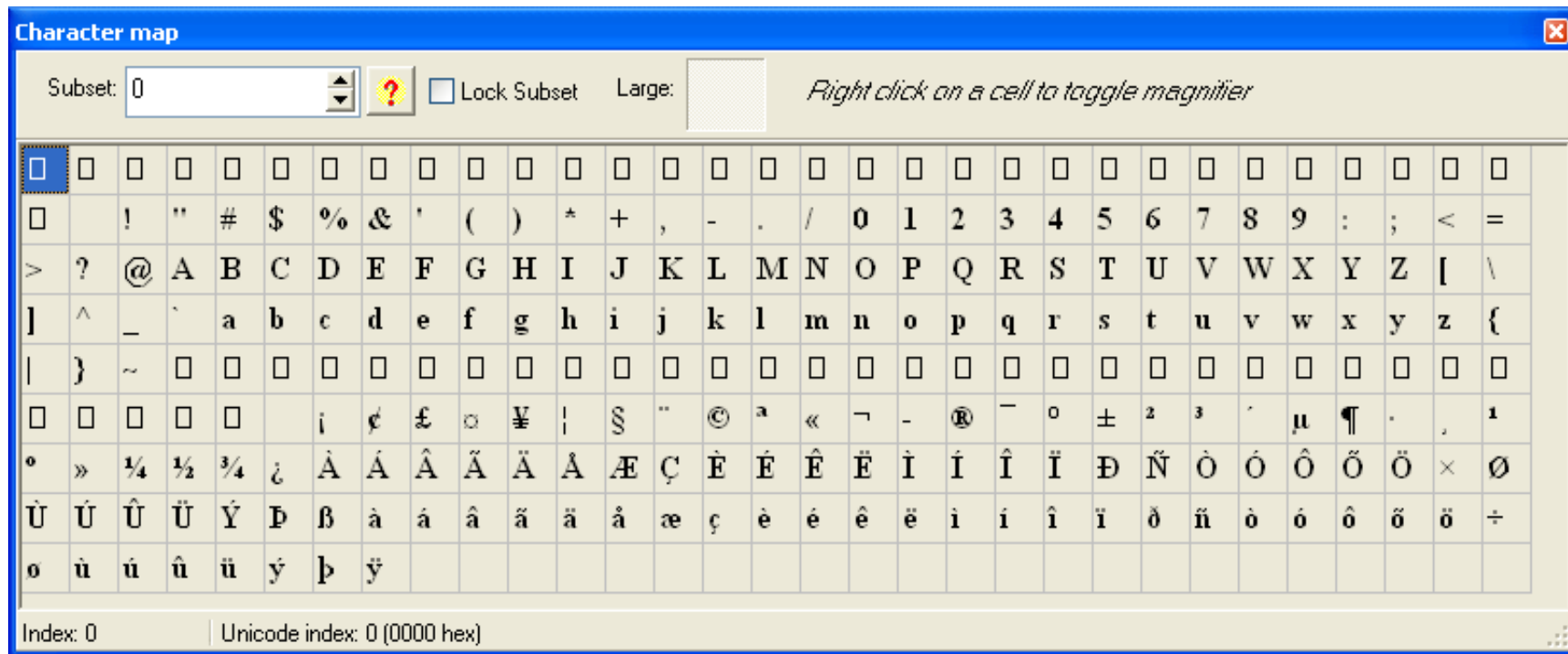
Specijalni karakteri

- Specijalni karakteri navode se u sledećem formatu:
 &oznaka_karaktera;
- Primeri:
 - &
 - <
 - >
 -
- Proizvoljan **Unicode** karakter može se navesti kao:
 &#xhex_kod;

Unicode

- Organizacija koja definiše standard po pitanju kodiranja slova u računarima
- Jedno slovo/karakter je u Unicode standardu trenutno 20-bitno
 - zove se code point
- Tri reprezentacije:
 - UTF-8 (<http://tools.ietf.org/html/rfc3629>)
 - jedan bajt za karakter sa vrednošću od U+0000 do U+007F
 - dva bajta za karakter sa vrednošću od U+0080 do U+07FF
 - tri bajta za karaktere u opsegu od U+0800 do U+FFFF
 - četiri bajta za karaktere u opsegu od U+10000 do U+10FFFF
 - UTF-16
 - svaki code point se prikazuje code unit-ima, koji su 16-bitni
 - ako je *code point* u rasponu od U+0000 do U+FFFF, onda je on smešten u *basic multilingual plane* i prikazuje se jednim *code unit*-om (samo 16 bita)
 - preostali *Unicode* karakteri su smešteni u opseg U+10000 do U+10FFFF, a reprezentovani su sa dva *code unit*-a
 - ovakvi karakteri se zovu *supplementary characters*
 - algoritam za konverziju *supplementary code point*-a u dva *code unit*-a se može naći na:
<http://en.wikipedia.org/wiki/UTF-16>
 - UTF-32
 - četiri bajta po slovu

Unicode



Unicode

Character map

Subset: 1 ☐ Lock Subset Large: *Right click on a cell to toggle magnifier*

	ā	Ă	ă	Ą	ą	Ć	ć	Ĉ	ĉ	Č	č	Ď	d'	Đ	đ	Ē	ē	Ĕ	ĕ	É	é	Ê	ê	Ë	ë	Ĝ	ĝ	Ğ		
ğ	Ġ	ġ	Ģ	ģ	Ĥ	ĥ	Ħ	ħ	Ĩ	ĩ	Ī	ī	Ĭ	ĭ	Į	į	ı	Ĳ	ĳ	Ĵ	ĵ	ķ	ķ	κ	Ł	ł	Ł	ł	Ł	
ł	Ł	ł	Ł	ł	Ń	ń	Ņ	ņ	Ň	ň	'n	Đ	đ	Ō	ō	Ŏ	ö	Ŏ	ö	Œ	œ	Ř	ř	Ŕ	ŕ	Ř	ř	Š	s	Ŝ
ŝ	Ş	ş	Š	š	Ţ	ţ	Ť	ť	Ŧ	ŧ	Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů	Ű	ű	Ų	ų	Ŵ	ŵ	Ŷ	ŷ	Ÿ	Ž	ž	Ž
ž	Ž	ž	ƒ	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
□	□	□	□	□	Œ	œ	□	□	□	□	□	□	□	□	□	□	□	□	□	Ů	ů	□	□	□	□	□	□	□	□	□
□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
Ů	ů	Ů	ů	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
□	□	Å	å	Æ	æ	Ø	ó																							

Index: 0 Unicode index: 256 (0100 hex)

Unicode

Character map

Subset: 4 ☐ Lock Subset Large: ☐ *Right click on a cell to toggle magnifier*

	Ё	Ђ	Ѓ	Є	Ѕ	І	Ї	Ј	Љ	Њ	Ћ	Ќ	Ў	Џ	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	
П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	а	б	в	г	д	е	ж	з	и	й	к	л	м	н
о	п	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	ѐ	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	
џ	љ	њ																												
																				Ѓ	г	Ѓ	ѓ			Ж	ж			Қ
қ	Қ	к					Ң	ң										Ү	ү	Ұ	ұ	Х	х					Ч	ч	
Һ	һ																												Ә	
ә														Ө	ө															

Index: 0 Unicode index: 1024 (0400 hex)

Primer

```
<html>
  <head>
    <title>Specijalni
    karakteri</title>
  </head>
  <body>
    <h1>Specijalni karakteri</h1>
    <p>
      Specijalni karakteri navode
      se u sledećem formatu:<br>
      <pre>&oznaka_karaktera;</pre>
    </p>
    <p>
      Znak ampersand (&amp;) ima
      specijalno značenje i
      predstavlja početak navođenja
      specijalnog karaktera. Kada
      treba ugraditi baš ovaj znak u
      tekst to se čini sekvencom
      &amp; amp;
    </p>
    <p>
      Znaci manje (&lt;) i veće
      (&gt;) navode se kao &amp;lt;
      odnosno &amp;gt;
    </p>
```

```
<p>
  Iako se višestruki white space
  karakteri ignorišu u HTML
  dokumentu, može se naglasiti da se
  više blank karaktera ponovi oznakom
  za <i>non-breaking
      space</i>, &amp;nbsp;
</p>

<p>
  Proizvoljan Unicode karakter može
  se navesti kao:
      <pre>&#xhex_kod;</pre>
  Na primer, ćirilичno slovo &#x0416;
  navodi se kao &amp;#x0416;
</p>
  </body>
</html>
```

Primer

Specijalni karakteri

Specijalni karakteri navode se u sledećem formatu:

`<oznaka_karaktera;`

Znak ampersand (&) ima specijalno značenje i predstavlja početak navođenja specijalnog karaktera. Kada treba ugraditi baš ovaj znak u tekst to se čini sekvencom `&`;

Znaci manje (<) i veće (>) navode se kao `<`, odnosno `>`;

Iako se višestruki white space karakteri ignorišu u HTML dokumentu, može se naglasiti da se više blank karaktera ponovi oznakom za *non-breaking space*, ` `;

Proizvoljan Unicode karakter može se navesti kao:

`&#x_kod;`

Na primer, ćirilčno slovo Ж navodi se kao `Ж`;

Hiperlinkovi (linkovi)

- Za prelaz na drugi dokument ili deo dokumenta
- `<a>` tag (anchor) se koristi za linkove
- Atribut **href** se koristi za definiciju odredišta
- Primer:

```
<a href="http://www.google.com">Google</a>
```

Hiperlinkovi (linkovi)

- Atribut **href** sadrži URL (Uniformed Resource Locator) do odredišta.
- Format URL-a:
protokol://računar:port/putanja/datoteka
- Oblici URL-a:
 - <http://www.ns.ac.yu/stara/index.html>
 - <http://www.ns.ac.yu/stara/>
 - <http://www.ns.ac.yu/>
 - <file:///D:/Prj/Aca/index.html>
 - <ftp://ftp1.freebsd.org/pub/FreeBSD/>
 - <mailto:someone@microsoft.com?cc=someoneelse@microsoft.com&bcc=andsomeoneelse2@microsoft.com&subject=Summer%20Party&body=You%20are%20invited%20to%20a%20big%20summer%20party!>

Link na deo stranice

- Link može da ukazuje na deo stranice
- Deo stranice se imenuje `<a>` tagom, ali uz upotrebu **name** atributa:

```
<a name="prvi">Ovaj tekst je obeležen imenom  
<b>prvi</b>.</a>
```

- Link na deo stranice:

Ovo je `link` na prvi deo.

ili

Ovo je `link
` na prvi deo.

Liste

- Tri vrste lista:
 - nabrojive (ordered)
 - neuređene (unordered) i
 - definicione (definition)
- Stavke nabrojive liste počinju rednim brojem (slovom, i sl.)
- Stavke neuređene liste počinju bullet-om.
- Stavke definicione liste su zapravo pojmovi sa objašnjenjem.

Nabrojive liste

- Počinje tagom ``
- Stavke se navode između `` i `` tagova
- Završava se tagom ``
- Atribut **start** definiše (početnu) vrednost broja ispred prve stavke.
- Atribut **type** definiše tip ("A", "a", "i", itd.)
- Primer:

```
<ol start="3">  
  <li>Voće</li>  
  <li>Povrće</li>  
</ol>
```

Primer

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>
<h2><u>Nabrojive liste </u></h2>

<h4>Nabrajanje u ovoj listi počinje od broja 3</h4>
<ol start="3">
  <li>Voće</li>
  <li>Povrće</li>
</ol>
<h4>dok u ovoj počinje od broja 1 - kod ove liste nije zadat atribut "start" pa nabrajanje počinje brojem 1</h4>
<ol>
  <li>Prva stavka</li>
  <li>Druga stavka</li>
</ol>
<h4>kod ove liste definisan je atribut "type" sa vrednošću "A" - u listi se za nabrajanje neće koristiti brojevi nego
velika slova</h4>
<ol type="A">
  <li>Prva stavka</li>
  <li>Druga stavka</li>
</ol>
<h4>type="i"</h4>
<ol type="i">
  <li>Prva stavka</li>
  <li>Druga stavka</li>
</ol>
<h4>type="I" i start="6"</h4>
<ol type="I" start=6>
  <li>Prva stavka</li>
  <li>Druga stavka</li>
</ol>
<h4>type="a" i start="5"</h4>
<ol type="a" start=5>
  <li>Prva stavka</li>
  <li>Druga stavka</li>
</ol>
</body>
</html>
```


Primer

Nabrojive liste

Nabrajanje u ovoj listi počinje od broja 3

3. Voće
4. Povrće

dok u ovoj počinje od broja 1 - kod ove liste nije zadat atribut "start" pa nabranje počinje brojem 1

1. Prva stavka
2. Druga stavka

kod ove liste definisan je atribut "type" sa vrednošću "A" - u listi se za nabranje neće koristiti brojevi nego velika slova

- A. Prva stavka
- B. Druga stavka

type="i"

- i. Prva stavka
- ii. Druga stavka

type="I" i start="6"

- VI. Prva stavka
- VII. Druga stavka

type="a" i start="5"

- e. Prva stavka
- f. Druga stavka

Neuređene liste

- Počinje tagom ``
- Stavke se navode između `` i `` tagova
- Završava se tagom ``
- Atribut **type** definiše tip bullet-a (disc, square, circle, itd.).
- Primer:

```
<ul type="square">  
  <li>Voce</li>  
  <li>Povrće</li>  
</ul>
```

Primer

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>
<h2><u>Neuredene liste</u></h2>

<h4>Atribut "type" ima vrednost "square"</h4>
<ul type="square">
  <li>Voće</li>
  <li>Povrće</li>
</ul>
<h4>Atribut "type" ima vrednost "circle"</h4>
<ul type="circle">
  <li>Voće</li>
  <li>Povrće</li>
</ul>
<h4>Atribut "type" ima vrednost "disc"</h4>
<ul type="disc">
  <li>Voće</li>
  <li>Povrće</li>
</ul>
</body>
</html>
```

Primer

Neuređene liste

Atribut "type" ima vrednost "square"

- Voće
- Povrće

Atribut "type" ima vrednost "circle"

- Voće
- Povrće

Atribut "type" ima vrednost "disc"

- Voće
- Povrće

Definizione liste

- Počinje tagom <dl>
- Stavke su parovi <dt> (*definition-list term* – pojam) i <dd> (*definition-list description* – opis) tagova sa vrednostima i odgovarajućim zatvarajućim tagovima
- Završava se tagom </dl>
- Primer:

```
<dl>
```

```
  <dt>Crna kafa</dt>
```

```
  <dd>Vruće piće</dd>
```

```
  <dt>Koka kola</dt>
```

```
  <dd>Hladno piće</dd>
```

```
</dl>
```

Primer

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8"
  />
<title>Naslov</title>
</head>

<body>
<h2><u>Definicione liste</u></h2>

<h4>Primer definicione liste sa dve stavke i dve definicije</h4>
<dl>
  <dt>Crna kafa</dt>
  <dd>Vruće piće</dd>
  <dt>Koka kola</dt>
  <dd>Hladno piće</dd>
</dl>
</body>
</html>
```

Primer

Definicione liste

Primer definicione liste sa dve stavke i dve definicije

Crna kafa

Vruće piće

Koka kola

Hladno piće

Slike

- Tag `` služi za prikaz slike
- Atribut **src** je obavezan i ukazuje na sliku.
- Opcioni atributi:
 - **alt** – alternativni tekst, koji se prikazuje ako se slika ne učita
 - **title** – tekst koji se prikazuje kada se pređe mišem preko slike
 - **width, height** – širina i visina slike
 - korisno je uneti informaciju o visini i širini slike u img tag
- Primer:

```

```

```

```


Primer

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>
<h2><u>Slike</u></h2>

<h4>Primer slike u okviru stranice</h4>

<h4>Ukoliko je lokacija slike netačna, čitač će prikazati "alt" tekst (u ovom slučaju "World Wide Web").</h4>

<h4>Uvođenjem atributa "height" i "width" možemo menjati dimenzije slike</h4>



</body>
</html>
```

Primer

Slike

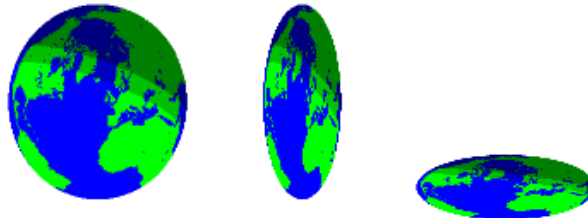
Primer slike u okviru stranice



Ukoliko je lokacija slike netačna, čitač će prikazati "alt" tekst (u ovom slučaju "World Wide Web").

World Wide Web

Uvođenjem atributa "height" i "width" možemo menjati dimenzije slike



Slika kao mapa linkova

- Atribut **usemap** u img tagu određuje putanju do opisa zona (do **<map>** taga).
 - mapa ne mora da bude u istoj HTML datoteci
- Tag **<map>** definiše geometrijske zone na slici koje će predstavljati linkove.
- Podtag **<area>** definiše konkretnu geometrijsku zonu
 - Atributi su:
 - **shape** – vrsta geometrijskog oblika (circle, rect, poly)
 - **coords** – lista koordinata (x, pa y vrednost) za zadatu vrstu geometrijskog oblika
 - ako je krug (circle), daju se x i y koordinata i poluprečnik
 - ako je pravougaonik (rect), daju se x i y koordinata gornjeg levog i donjeg desnog ugla
 - ako je mnogougao (poly), daje se niz x i y koordinata, koji čini niz tačaka koje čine konturu (zatvorenu)
 - **href** – URL do stranice na koju se ide klikom po zoni
 - **alt** – alternativni prikaz

Primer

```
<html>
  <head>
</head>
  <body>
    <p>
      Delovi slike predstavljaju linkove. Pronadite ih.
    </p>
    <p>
      
    </p>
    <map name="mapa">
      <area shape="poly" alt="poligon" coords="395,145, 322,91,
393,32" href="primer01.html">
      <area shape="rect" alt="pravougaonik" coords="51,119, 135,191"
href="primer02.html">
      <area shape="circle" alt="krug" coords="277,165,76"
href="primer03.html">
    </map>
  </body>
</html>
```

Tabele

- Tabele se sastoje od vrsta i kolona.
- U HTML-u tabela se sastoji od redova koji su podeljeni na polja, a prva polja svih redova čine prvu kolonu, druga polja drugu kolonu, itd.
- Za kreiranje tabele potrebna su tri taga:
 - `<table>` - za tabelu,
 - `<tr>` - za red,
 - `<td>` - za polje.
- Pomoću taga `<th>` definišu se zaglavlja vrsta ili kolona.

Tabele

- U okviru početnog taga `<table>`, moguće je definisati poravnanje tabele u dokumentu (atribut **align**, koji može imati vrednosti **left** ili **right**), debljinu ivice tabele (atribut **border**) i boju pozadine (atribut **bgcolor**)
 - svaka ćelija tabele može imati i svoju boju
 - za ovo je bolje koristiti CSS

Tabele

- Atribut **cellspacing** definiše rastojanje između ivice tabele i ivice ćelije u tabeli.
- Atribut **cellpadding** definiše rastojanje od ivice ćelije do sadržaja ćelije.
- Za svaku ćeliju se može definisati koliko će obuhvatiti kolona ili vrsta, pomoću atributa **colspan** i **rowspan**.

Tabele

- Prostiranje preko dve kolone:

```
<table border="1">
<tr>
  <th>Name</th>
  <th colspan="2">Telephone</th>
</tr>
<tr>
  <td>Bill Gates</td>
  <td>555 77 854</td>
  <td>555 77 855</td>
</tr>
</table>
```

Name	Telephone	
Bill Gates	555 77 854	555 77 855

Tabele

- Prostiranje preko dve vrste:

```
<table border="1">
```

```
<tr>
```

```
  <th>First Name:</th>
```

```
  <td>Bill Gates</td>
```

```
</tr>
```

```
<tr>
```

```
  <th rowspan="2">Telephone:</th>
```

```
  <td>555 77 854</td>
```

```
</tr>
```

```
<tr>
```

```
  <td>555 77 855</td>
```

```
</tr>
```

```
</table>
```

First Name:	Bill Gates
Telephone:	555 77 854
	555 77 855

Tabele

- Tag `<tr>` (Table Row) definiše vrste u tabeli.
- Tag `<tr>` sadrži tagove `<th>` (Table Heading), koji određuju zaglavlje tabele, i tagove `<td>` (Table Data) koji predstavljaju ćelije u tabeli.
- U tagu `<tr>` mogu se nalaziti sledeći atributi:
 - **align** - poravnanje tabele (**center**, **left**, **right**),
 - **valign** - poravnanje sadržaja u ćeliji (**baseline**, **bottom**, **middle**, **top**),
 - **bgcolor** - boja pozadine
 - za ovo je bolje koristiti CSS

Tabele

- Ako je potrebno ostaviti praznu ćeliju, trebalo bi ostaviti ` `; da bi bilo ispravno prikazano
 - `<td> </td>`
- Element **caption** definiše naslov tabele.
- Tag `<caption>` se koristi unutar taga `<table>`, a ne unutar tagova `<td>` ili `<tr>`.
- Atribut **align** specificira mesto naslova u odnosu na tabelu, a moguće vrednosti su mu **bottom** (naslov će biti ispod tabele), **top** (naslov će biti iznad tabele), **center**, **right** i **left**.

Primer1

```
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>
<h2><u>Tabele</u></h2>

<h4>Primer jednostavne tabele</h4>
<table width="200" border="1">
  <tr>
    <td>1</td>
    <td>A</td>
    <td>X</td>
  </tr>
  <tr>
    <td>2</td>
    <td>B</td>
    <td>Y</td>
  </tr>
  <tr>
    <td>3</td>
    <td>C</td>
    <td>Z</td>
  </tr>
</table>
```

```
<h4>Primer jednostavne tabele sa
      zaglavljima kolona</h4>
<table width="200" border="1">
  <tr>
    <th scope="col">Zaglavlje</th>
    <th scope="col">Zaglavlje</th>
    <th scope="col">Zaglavlje</th>
  </tr>
  <tr>
    <td>Ćelija 1 </td>
    <td>Ćelija 2 </td>
    <td>Ćelija 3 </td>
  </tr>
</table>

</body>
</html>
```

Primer1

Tabele

Primer jednostavne tabele

1	A	X
2	B	Y
3	C	Z

Primer jednostavne tabele sa zaglavlju na kolona

Zaglavlje	Zaglavlje	Zaglavlje
Ćelija 1	Ćelija 2	Ćelija 3

Primer2

```
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>
<h2><u>Tabele</u></h2>
<h4>Primer jednostavne tabele - atribut "border"
      ima vrednost "2", "bgcolor" ima vrednost
      #CCCCCC</h4>
<table bgcolor="#CCCCCC" width="200" border="2">
  <tr>
    <td>1</td>
    <td>A</td>
    <td>X</td>
  </tr>
  <tr>
    <td>2</td>
    <td>B</td>
    <td>Y</td>
  </tr>
  <tr>
    <td>3</td>
    <td>C</td>
    <td>Z</td>
  </tr>
</table>
```

```
<h4>Primer jednostavne tabele - atribut "align" ima
      vrednost "center"</h4>
<table align="center" width="200" border="1">
  <tr>
    <td>1</td>
    <td>A</td>
    <td>X</td>
  </tr>
  <tr>
    <td>2</td>
    <td>B</td>
    <td>Y</td>
  </tr>
  <tr>
    <td>3</td>
    <td>C</td>
    <td>Z</td>
  </tr>
</table>
```

Primer2

`<h4>Primer jednostavne tabele sa zaglavljima kolona - atribut "align" ima vrednost "right";</h4>`

`<table align="right" width="200" border="1">`

`<tr>`

`<th scope="col">Zaglavlje</th>`

`<th scope="col">Zaglavlje</th>`

`<th scope="col">Zaglavlje</th>`

`</tr>`

`<tr>`

`<td>Ćelija 1 </td>`

`<td>Ćelija 2 </td>`

`<td>Ćelija 3 </td>`

`</tr>`

`</table>`

`</body>`

`</html>`

Primer2

Tabele

Primer jednostavne tabele - atribut "border" ima vrednost "2", "bgcolor" ima vrednost #CCCCCC

1	A	X
2	B	Y
3	C	Z

Primer jednostavne tabele - atribut "align" ima vrednost "center"

1	A	X
2	B	Y
3	C	Z

Primer jednostavne tabele sa zaglavljinom kolona - atribut "align" ima vrednost "right"

Zaglavlje	Zaglavlje	Zaglavlje
Ćelija 1	Ćelija 2	Ćelija 3

Primer3

```
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>
<h2><u>Tabele</u></h2>
<h4>Primer jednostavne tabele sa povećanim
      razmakom između ćelija - "cellspacing"
      je "10"</h4>
<table width="200" border="1"
      cellspacing="10">
  <tr>
    <td>1</td>
    <td>A</td>
    <td>X</td>
  </tr>
  <tr>
    <td>2</td>
    <td>B</td>
    <td>Y</td>
  </tr>
  <tr>
    <td>3</td>
    <td>C</td>
    <td>Z</td>
  </tr>
</table>
```

```
<h4>Primer jednostavne tabele sa povećanim
      razmakom između ivice ćelija i
      sadržaja ćelija - "cellpadding" je
      "10"</h4>
<table width="200" border="1"
      cellpadding="10">
  <tr>
    <td>1</td>
    <td>A</td>
    <td>X</td>
  </tr>
  <tr>
    <td>2</td>
    <td>B</td>
    <td>Y</td>
  </tr>
  <tr>
    <td>3</td>
    <td>C</td>
    <td>Z</td>
  </tr>
</table>
</body>
</html>
```

Primer3

Primer jednostavne tabele sa povećanim razmakom između ćelija - "cellspacing" je "10"

1	A	X
2	B	Y
3	C	Z

Primer jednostavne tabele sa povećanim razmakom između ivice ćelija i sadržaja ćelija - "cellpadding" je "10"

1	A	X
2	B	Y
3	C	Z

Primer4

```
<html>
<head>
<meta http-equiv="Content-Type"
      content="text/html; charset=utf-8" />
<title>Naslov</title>
</head>

<body>
<h2><u>Tabele</u></h2>
<h4>Primer jednostavne tabele sa naslovom
      iznad tabele</h4>
<table width="200" border="1">
  <caption>Naslov koji je iznad
    tabele</caption>
  <tr>
    <td>1</td>
    <td>A</td>
    <td>X</td>
  </tr>
  <tr>
    <td>2</td>
    <td>B</td>
    <td>Y</td>
  </tr>
  <tr>
    <td>3</td>
    <td>C</td>
    <td>Z</td>
  </tr>
</table>
```

```
<h4>Primer jednostavne tabele sa naslovom
      ispod tabele</h4>
<table width="200" border="1">
  <caption align="bottom">Naslov koji je
    ispod tabele</caption>
  <tr>
    <td>1</td>
    <td>A</td>
    <td>X</td>
  </tr>
  <tr>
    <td>2</td>
    <td>B</td>
    <td>Y</td>
  </tr>
  <tr>
    <td>3</td>
    <td>C</td>
    <td>Z</td>
  </tr>
</table>
</body>
</html>
```

Primer4

Tabele

Primer jednostavne tabele sa naslovom iznad tabele

Naslov koji je iznad tabele

1	A	X
2	B	Y
3	C	Z

Primer jednostavne tabele sa naslovom ispod tabele

1	A	X
2	B	Y
3	C	Z

Naslov koji je ispod tabele

Forme

- Forma služi za unos podataka od strane korisnika
- Forma je prostor koji sadrži elemente forme
- Elementi forme omogućavaju korisniku da unese neke informacije (tekstualna polja, polja za unos teksta, padajući meniji, check-boksovi, itd)
- Forma je definisana tagom `<form>`
- U okviru njega je najčešće korišćen tag `<input>`

Forme

- Tag `<form>` definiše formu.
 - atribut **method** definiše način prenosa parametara unetih u formi,
 - atribut **action** definiše stranicu na koju će se preći po završetku unosa.
 - atribut **accept-charset** definiše kodnu stranu po kojoj će biti kodiran uneti tekst
- Tag `<input>` definiše elemente forme.
 - Tip unosa je definisan atributom **type**.
- Vrednosti atributa `type` mogu biti:
 - **text** – tekstualno polje,
 - **password** – polje za unos šifre,
 - **radio** – radio buttons (atribut **name** mora da se poklapa da bi pripadali istoj grupi),
 - **checkbox**,
 - **submit** – akcija koja treba da se izvrši,
 - **reset** – resetuje sve vrednosti polja na početne,
 - **image** – slika koja funkcioniše kao dugme.
 - **file** – za upload fajla,
 - **button** – obično dugme; nema funkciju bez skript jezika.
 - **hidden** – skriveno polje.
- Tag `<select>` – combo box/list box u kom se nalaze vrednosti definisane u tagu **option**; razlika je u atributu **size**.
- Tag `<textarea>` definiše višelinijnsko tekstualno polje.
- Tag `<fieldset>` definiše okvir oko elemenata koji su u okviru njega
 - podtag `<legend>` sadrži tekst koji se ispisuje na okviru

Primer

```
<html>
  <head>
    <title>Forme</title>
  </head>
  <body >
    <h1>Forme</h1>
    <p>
      Forma predstavlja deo dokumenta
      koji sadrži polja za unos podataka.
      Postoji više tipova ovakvih polja.
    </p>
    <form action="primer06-2.html"
      method="get">
    <p>
```

```
<fieldset>
<legend>Jednolinijska tekstualna
    polja</legend>
Jednolinijsko tekstualno polje:
<input type="text" name="polje1">
    <br><br>

Polje za unos lozinki:
<input type="password" name="polje2">
    <br><br>
</fieldset>

Checkbox:
<input type="checkbox" name="polje3">
    remind me <br><br>
```

```
Radio:  
<input type="radio" name="polje4"  
      checked> Male       
<input type="radio" name="polje4">  
Female  
<br><br>
```

Primer

Višelinijnsko polje:

<textarea name="polje5" rows="10"
cols="30">Sadržaj polja
</textarea>

Combo box:

```
<select name="polje6">  
  <option value="1">Prva  
    stavka</option>  
  <option value="2">Druga  
    stavka</option>  
  <option value="3">Treća  
    stavka</option>  
</select> <br><br>
```

List box:


```
<select name="polje7" size="3">  
  <option value="1">Prva  
    stavka</option>  
  <option value="2" selected>Druga  
    stavka</option>  
  <option value="3">Treća  
    stavka</option>  
</select> <br><br>
```

Skriveno polje:

```
<input type="hidden" name="polje8"  
value="vrednost"> <br><br>
```

Polje za upload fajla:


```
<input type="file" name="polje9">  
<br><br>
```

Dugme:

```
<input type="button"  
name="polje10" value=" Dugme ">  
<br><br>
```

Reset dugme:

```
<input type="reset" name="polje11"  
value=" Reset "> <br><br>
```

Submit dugme:

```
<input type="submit"  
name="polje12" value=" Submit ">  
<br><br>
```

Slika-dugme:

```
<input type="image" name="polje13"  
src="submit.gif"> <br><br>  
</p>  
</form>
```

</body>

</html>

Primer

Forme - Mozilla Firefox

File Edit View History Bookmarks Tools Help

file:///D:/Nastava/Web dizajn/1_HT ☆ language ε

Jednoslinijska tekstualna polja

Jednolinijsko tekstualno polje:

Polje za unos lozinki:

Checkbox: ☐ remind me

Radio: ☒ Male ☐ Female

Višelinijsko polje:

Sadržaj polja

Combo box: Prva stavka ▼

List box:

Prva stavka
Druqa stavka
Treća stavka

Skriveno polje:

Polje za upload fajla:

Browse...

Dugme:

Reset dugme:

Submit dugme:

Slika-dugme:

Done

Frejmovi

- Uz pomoć frejmova, moguće je prikazati više od jedne web stranice u istom prozoru browser-a.
- Svaka stranica je frejm za sebe i one su međusobno nezavisne.
- Tag **<frameset>** definiše kako da se podeli prozor uz pomoć atributa **cols** (za vertikalnu podelu) ili **rows** (za horizontalnu podelu), u kojima se navode procenti ili vrednost u pikselima.
- Tag **<frame>** definiše koji će HTML dokument biti u kom frejmu.
 - Atribut **src** definiše URL do html stranice koja će biti prikazana u frejmu.
 - Atribut **name** definiše naziv frejma. Koristi se za definisanje odredišnog frejma za linkove (atribut **target**).

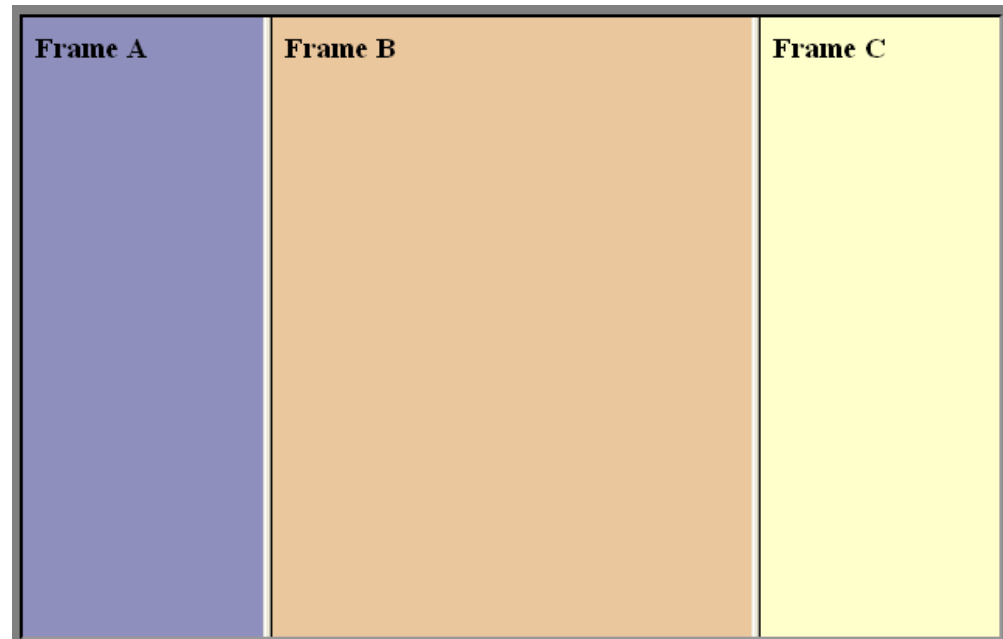
Atribut **target**

- Hiperlink može da sadrži atribut target
 - definiše kako će se otvoriti zadati link
- Vrednosti:
 - naziv frejma u kojem će se otovriti
 - `_blank` – stranica će se otvoriti u novom prozoru
 - `_self` – stranica će se otvoriti u istom frejmu gde je i link
 - `_parent` – stranica će se otvoriti u roditeljskom frejmu
 - `_top` – stranica će se otvoriti u punom prozoru (izaći će iz frejma)

Frejmovi

- Vertikalna podela

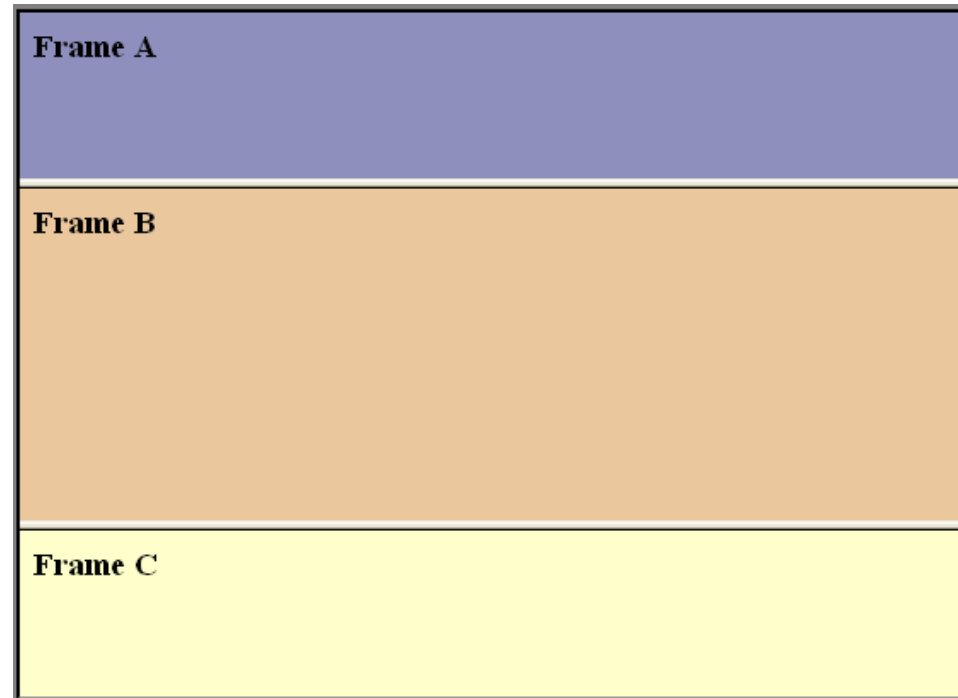
```
<html>  
<frameset cols="25%,50%,25%">  
<frame src="frame_a.htm">  
<frame src="frame_b.htm">  
<frame src="frame_c.htm">  
</frameset>  
</html>
```



Frejmovi

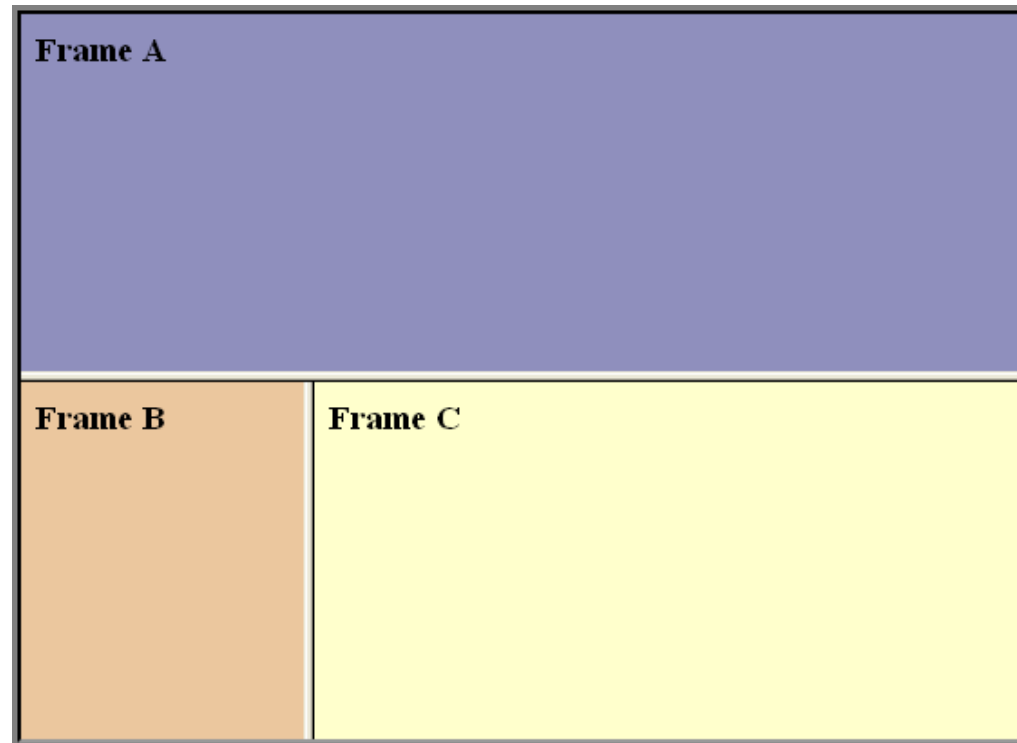
- Horizontalna podela

```
<html>  
<frameset rows="25%,50%,25%">  
<frame src="frame_a.htm">  
<frame src="frame_b.htm">  
<frame src="frame_c.htm">  
</frameset>  
</html>
```



Primer

```
<html>
<frameset rows="50%,50%">
<frame src="frame_a.htm">
<frameset cols="25%,75%">
  <frame src="frame_b.htm">
  <frame src="frame_c.htm">
</frameset>
</frameset>
</html>
```



Primer

```
<html>
  <head>
    <title>Frejmovi</title>
  </head>

  <frameset rows="20%,80%">
    <frame name="zaglavlje" src="primer04-1.html">
    <frame name="glavni" src="primer04-2.html">
  </frameset>
</html>
```

Primer

primer04-1.html

[illegible]

primer04-2.html

```
<html>

  <head>
    <title>Glavni frejm</title>
  </head>

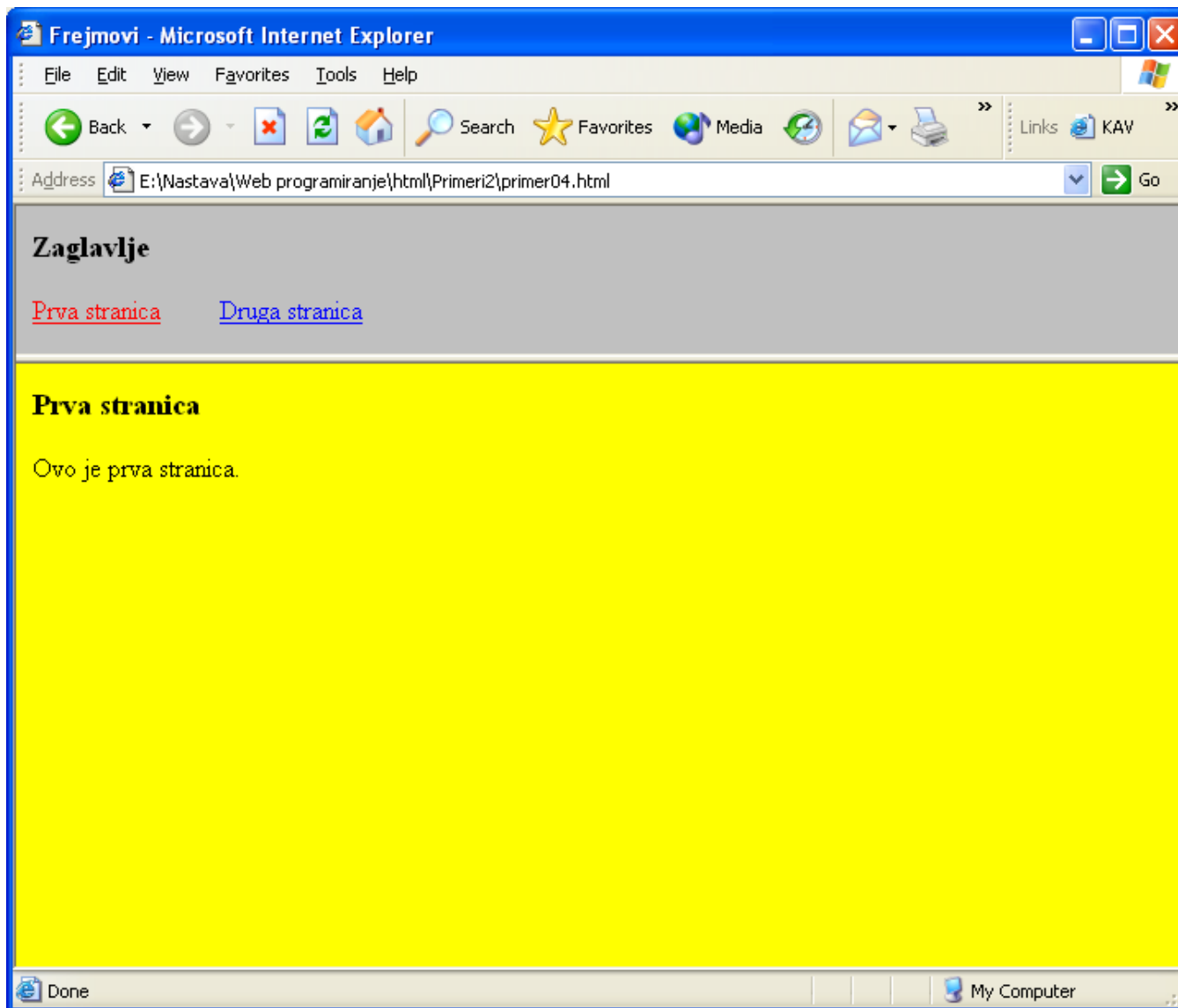
  <body bgcolor="yellow"
    link="blue" alink="red"
    vlink="blue">
    <h3>Prva stranica</h3>

    <p>
      Ovo je prva stranica.
    </p>

  </body>

</html>
```


Primer



iframe

- Inline frame – frejm na proizvoljnoj poziciji unutar stranice
- Atributi:
 - src – naziv HTML datoteke koja će biti prikazana u ovom prozoru
 - width, height – širina i visina
 - scrolling – da li da prikaže skrolere (yes, no, auto)
 - align – poravnanje (left, right, top, middle, bottom)

Primer

```
<html>
```

```
<body>
```

```
<iframe src="http://www.ftn.ns.ac.yu"></iframe>
```

```
<p>Some older browsers don't support iframes.</p>
```

```
<p>If they don't, the iframe will not be  
    visible.</p>
```

```
</body>
```

```
</html>
```



Some older browsers don't support iframes.

If they don't, the iframe will not be visible.

Dodatni atributi **body** taga

- Atribut **bgcolor** boji pozadinu stranice.
- Atribut **background** definiše putanju do pozadinske slike.
- Primer:

```
<body bgcolor="#d0d0d0">
```

```
<body background="background.jpg">
```

Boje

- Boje se zadaju imenom ili RGB vrednostima

Color	Color HEX	Color RGB
	#000000	rgb(0,0,0)
	#FF0000	rgb(255,0,0)
	#00FF00	rgb(0,255,0)
	#0000FF	rgb(0,0,255)
	#FFFF00	rgb(255,255,0)
	#00FFFF	rgb(0,255,255)
	#FF00FF	rgb(255,0,255)
	#C0C0C0	rgb(192,192,192)
	#FFFFFF	rgb(255,255,255)

<u>AliceBlue</u>	#F0F8FF	
<u>AntiqueWhite</u>	#FAEBD7	
<u>Aqua</u>	#00FFFF	
<u>Aquamarine</u>	#7FFFD4	
<u>Azure</u>	#F0FFFF	
<u>Beige</u>	#F5F5DC	
<u>Bisque</u>	#FFE4C4	
<u>Black</u>	#000000	
<u>BlanchedAlmond</u>	#FFEBCD	
<u>Blue</u>	#0000FF	
<u>BlueViolet</u>	#8A2BE2	
<u>Brown</u>	#A52A2A	
<u>BurlyWood</u>	#DEB887	
<u>CadetBlue</u>	#5F9EA0	
<u>Chartreuse</u>	#7FFF00	
<u>Chocolate</u>	#D2691E	
<u>Coral</u>	#FF7F50	
<u>CornflowerBlue</u>	#6495ED	
<u>Cornsilk</u>	#FFF8DC	
<u>Crimson</u>	#DC143C	
<u>Cyan</u>	#00FFFF	
<u>DarkBlue</u>	#00008B	
<u>DarkCyan</u>	#008B8B	

Meta tagovi

- Definišu dodatne informacije koje se ne prikazuju.
- Dodatne informacije se definišu kao parovi (naziv, vrednost), tj. atributi **name** i **content** <meta> taga
 - koriste se za obezbeđivanje informacija pretraživačima interneta (yahoo, google, itd.): author, description, keywords;
 - informacije za druge namene (proizvoljne vrednosti).
- Atribut **http-equiv** definiše podatke koji opisuju stranicu (upravljaju čitačem)
 - http-equiv može biti: Content-Type, Refresh, Cache-Control, Pragma, itd.
 - a, vrednosti (content): text/html, 5, no-cache, itd.

Primer

```
<html>
```

```
  <head>
```

```
    <title>META tagovi</title>
```

```
    <meta name="author" content="John Smith">
```

```
    <meta name="description" content="This is a page that  
demonstrates various META tags.">
```

```
    <meta name="keywords" content="HTML, META tags, description">  
  </head>
```

```
  <body>
```

```
    <h1>META tagovi (1/2)</h1>
```

```
    <p>(pogledajte izvorni kod dokumenta)</p>
```

```
  </body>
```

```
</html>
```

Primer

META tagovi (1/2)

(pogledajte izvorni kod dokumenta)

Primer

```
<html>
```

```
<head>
```

```
<title>META tagovi</title>
```

```
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">
```

```
<meta http-equiv="Refresh" content="5">
```

```
<meta http-equiv="Cache-Control" content="no-cache">
```

```
</head>
```

```
<body>
```

```
<h1>META tagovi (2/2)</h1>
```

```
<p>Malo unicode teksta:</p>
```

```
<p>Latinica ÄÄ†</p>
```

```
<p>Đ<Đ, Ñ€Đ, Đ»Đ, Ñ†Đ°</p>
```

```
</body>
```

```
</html>
```

Primer

```
<html>
```

```
<head>
```

```
<title>META tagovi</title>
```

```
<meta http-equiv="Content-Type" content="text/html;  
charset=UTF-8">
```

```
<meta http-equiv="Refresh" content="5">
```

```
<meta http-equiv="Cache-Control" content="no-cache">
```

```
</head>
```

```
<body>
```

```
<h1>META tagovi (2/2)</h1>
```

```
<p>Malo unicode teksta:</p>
```

```
<p>Latinica čć</p>
```

```
<p>Кирилица</p>
```

```
</body>
```

```
</html>
```

Primer

META tagovi (2/2)

Malo unicode teksta:

Latinica čć

Кирилица

Web programiranje

CSS

Problemi sa izgledom

- Osnovni skup tagova i njihovih atributa daje strukturu HTML dokumenata
 - čitači prikazuju sadržaj na predefinisani način
- Proizvođači čitača su počeli da dodaju tagove i attribute za podešavanje izgleda stranica
 - tipičan primer je `` tag, sa atributima **face**, **color**, **size**, itd.
- HTML dokumenti se zatrpavaju dodatnim tagovima i atributima za formatiranje izgleda
- CSS rešava ovaj problem
 - izgled se podešava definicijom stila
 - definicije stila se ne moraju nalaziti u istom dokumentu

CSS

- **Cascading Style Sheets**
 - može više definicija za isti element da se preklapa (cascade)
- HTML bi trebalo da se koristi za opis strukture dokumenta
 - HTML ima default način prikaza elemenata
- Vizuelna definicija HTML stranica se prepušta stilovima (CSS):
 - stilovi se definišu za elemente HTML-a (tagove)
 - stilovi definišu izgled elemenata (boja, font, pozadinska boja, itd.)
 - jedan stil može da se koristi i za više HTML datoteka

Gde se stilovi ugrađuju?

- Unutar samih HTML elemenata (atribut **style**)
 - inline style
- Upotrebom taga <style> unutar dokumenta (unutar <head> taga)
 - internal style sheet
- Kreiranjem spoljašnje datoteke stilova (.css datoteka)
 - external style sheet
- Ako dođe do preklapanja, prioritet je:
 1. unutar HTML elementa
 2. <style> tag
 3. spoljašnja datoteka stilova
 4. kako čitač prikazuje

Sintaksa

- Opšta sintaksa (sem za inline):

selektor {svojstvo: vrednost}

- Selektor definiše na koga se odnosi definicija stila
 - selektuje elemente u dokumentu
- Svojstvo je vizualna karakteristika koju želimo da promenimo
- Vrednost je nova vrednost svojstva
- Primer:

body {color: black}

- Komentar:

/* komentar */

Sintaksa

- Ako vrednost ima više od jedno reči, stavlja se unutar navodnika:

p {font-family: "sans serif"}

- Ako želimo da promenimo više svojstava za isti tag, stavljamo ';' između parova
svojstvo: vrednost:

p {text-align: center; color: red}

Sintaksa

- Ne moraju sva svojstva da se stave u jedan red:

```
p {  
    text-align: center;  
    color: black;  
    font-family: serif  
}
```

Selektor

- Možemo da grupišemo tagove u selektoru:

```
h1,h2,h3,h4,h5,h6 {  
  color: green  
}
```

Selektor

- Univerzalni selektor '*':

*** {color: green}**

Klase stilova

- Klasa stila se može primeniti na više HTML elemenata.
 - ime klase ne sme da počne brojem

- Sintaksa:

```
tag.naziv {definicija}
```

```
*.naziv {definicija}
```

```
.naziv {definicija}
```

- Primer:

```
.menu {color: blue}
```

```
...
```

```
<p class="menu">...</p>
```

- Klasa definisana za konkretan element:

```
p.menu {color : blue}
```

Stilovi identifikovani po ID-u

- Umesto klase, moguća je upotreba ID-a za odabir stila
- Sintaksa:

```
tag#naziv {definicija}
```

```
*#naziv {definicija}
```

```
#naziv {definicija}
```

- Primer:

```
#menu {color:blue}
```

```
...
```

```
<p id="menu">...</p>
```

- Stil dodeljen konkretnom elementu sa konkretnim ID-om:

```
p#menu {color : blue}
```

Stil dodeljen tagu sa specifičnim atributom

- Stil se može dodeliti određenom tagu sa specifičnim sadržajem atributa.
- Sintaksa:

tag[ime] {definicija}

- tag koji ima atribut po imenu 'ime'

tag[ime="vrednost"] {definicija}

- tag koji ima atribut po imenu 'ime' i čija vrednost je 'vrednost'

tag[ime~="vrednost"] {definicija}

- tag koji ima atribut po imenu 'ime', čija vrednost je lista reči razdvojenih razmakom i gde je jedna od reči 'vrednost'

tag[ime|="vrednost"] {definicija}

- tag koji ima atribut po imenu 'ime', čija vrednost je lista reči razdvojenih crticom, i gde jedna od reči počinje sa 'vrednost'

- Primer:

input[type="text"] {background-color: blue}

Stil dodeljen tagu sa specifičnim atributom

- Primer:

```
a[hello="Pera"][bye="Mika"] { color: blue; }
```

- važi za tag <a> koji ima dva atributa, prvi se zove 'hello' i ima vrednost 'Pera', a drugi se zove 'bye' i ima vrednost 'Mika'

```
*[lang|="sr"] { color : red }
```

- sve tagove sa atributom lang i vrednošću koja počinje sa 'sr' (sr, sr_SR, itd.) boji u crveno

Selektor potomaka

- Descendant selector
- Odnosi se na tag koji se nalazi unutar drugog taga (nedefinisane dubine)

- Primer:

a b { ... }

- važi za b koji je bilo koji potomak od a

h1 em {color: blue}

...

<h1>Naslov saizraženimdelom</h1>

- plavom bojom se označava tag, samo ako je unutar <h1> taga

Selektor potomaka

- Može i više od dva elementa u ovakvom selektoru
- Primer:

```
div * p { ... }
```

- stil se odnosi na `<p>` tag koji za pretka ima bilo koji tag, koji za svog pretka ima `<div>` tag

- Može i:

```
div p *[href] { ... }
```

- stil je definisan za bilo koji tag koji ima atribut `href` i ima za pretka `<p>` tag, koji ima za pretka `<div>` tag

Selektor direktnog potomka

- Child selector
- Odnosi se na tag čiji je roditelj naveden levo od '>' znaka

- Primer:

body > p {color: blue}

– važi za <p> tag koji je dete od <body> taga

- Primer:

div ol>li p

– važi za <p> tag koji se nalazi unutar taga, čiji je neposredni roditelj tag, a koji se nalazi unutar <div> taga

Selektor tagova sa istog nivoa

- Adjacent sibling selectors
- Odnosi se na tagove koji imaju istog roditelja i levi element je pre desnog

- Primer:

a + b { . . . }

- važi za element b koji ima istog pretka kao i element a, i element a se nalazi pre elementa b

Gde se stilovi ugrađuju?

- Unutar samih HTML elemenata (atribut **style**)
 - inline style
- Upotrebom taga <style> unutar dokumenta (unutar <head> taga)
 - internal style sheet
- Kreiranjem spoljašnje datoteke stilova (.css datoteka)
 - external style sheet
- Ako dođe do preklapanja, prioritet je:
 1. unutar HTML elementa
 2. <style> tag
 3. spoljašnja datoteka stilova
 4. kako čitač prikazuje

Stil unutar HTML elementa

- Koristi se atribut **style** unutar taga.

svojstvo: vrednost; ...

- Primer:

```
<h1 style="color: blue">Tekst</h1>
```

Stilovi definisani unutar dokumenta

- Koristi se tag `<style>` unutar `<head>` sekcije.
- Tako definisan stil se odnosi na sve elemente koji su navedeni u stilu.

- Format specifikacije stila:

selektor {svojstvo: vrednost; ...}

- Primer:

```
<style type="text/css">
```

```
h1, h2 {color: blue; text-align: center}
```

```
p {color: red}
```

```
</style>
```

```
...
```

```
<h1>Naslov</h1>
```

```
<p>paragraf</p>
```

Stilovi definisani u eksternoj stranici stilova

- Kreira se datoteka sa definicijom stilova
 - ekstenzija je uobičajeno .css
- Referenca na eksternu datoteku je upotrebom <link> taga unutar <head> sekcije.
- Primer:

```
<link rel="stylesheet" href="stilovi.css">
```


Preklapanje stilova (kaskadni stilovi)

- Svaki dodatno definisan stil se preklapa/kombinuje sa postojećim
- Atribut `!important` obezbeđuje da se osobina stavi na vrh kaskadnog procesa.
- Primer:

```
p {color: black !important}
```

Jedinice mere

- Svaka uneta numerička vrednost može se preciznije odrediti jedinicom mere
- Dužina:
 - relativne vrednosti
 - % – procentualna vrednost,
 - px – pikseli (pixel), tipično 1/96 inča, odn. 0,26 mm
 - ex – u odnosu na visinu slova 'x' u tekućem fontu,
 - em – u odnosu na širinu slova 'm' u tekućem fontu
 - Apsolutne vrednosti
 - cm – centimetar
 - in – inč
 - mm – milimetar
 - pt – tačke (point) – za veličinu fonta, veličine 1/72 inča
 - pc – pika (*pica*) je 12 tačaka (12 points)
- Boja:
 - tekstualno (red, black, blue,...)
 - numerički, RGB vrednosti (#0F0, #FF00FF, rgb(0, 255, 0), rgb(2%, 10%, 2%))
- Ne sme da postoji razmak između vrednosti i jedinice mere!
 - **margin-left: 20px**

Pikseli

- Piksel je najmanja tačka na ekranu
- Zavisi od rezolucije ekrana
 - nije fiksna jedinica mere
 - od 96 piksela po inču, do 120 piksela po inču (Windows Large Fonts), na ekranu
- Različiti operativni sistemi različito tretiraju rezoluciju ekrana:
 - MAC (do OS-X): 72 piksela po inču
 - Windows: 96 piksela po inču

Svojstva teksta

- *word-spacing* – razmak između reči (normal ili konkretna vrednost),
- *letter-spacing* – razmak između slova (normal ili konkretna vrednost),
- *white-space* – kako da prikazuje više uzastopnih razmaka (normal, pre),
- *text-transform* – transformacija teksta (none, capitalize (prvo slovo veliko), uppercase, lowercase),
- *text-decoration* – dekoracija teksta (none, underline, overline, line-through, blink),
- *color* – boja teksta,
- *text-shadow* – senka iza teksta

text_*.html

Poravnanje teksta

- *vertical-align* (top, bottom, middle, ...),
- *text-align* – poravnanje teksta (left, right, center, justify),
- *text-indent* – koliko je prvi red paragrafa uvučen,
- *line-height* – vertikalna udaljenost između dve linije

Fontovi

font_*.html

- *font-family* – vrsta fonta;
 - parametar je ime fonta ili lista imena odvojenih zarezom (alternative);
 - ime fonta može biti konkretno ili ime familije;
 - ime familije može da bude generičko (serif, sans-serif, cursive, fantasy, monotype),
- *font-style* – stil (normal, italic, oblique),
- *font-size* – veličina fonta (apsolutna vrednost (xx-small, x-small, small, large, x-large, xx-large), relativna vrednost (smaller, larger), vrednost, procenat),
- *font-size-adjust* – odnos visine malog slova 'x' i visine fonta
 - što je veći broj, font je pogodniji za male veličine slova
- *font-weight* – podebljanje fonta (normal, bold, bolder, lighter ili vrednost od 100 do 900),
- *font-variant* – način ispisa malih slova (normal, small-caps (mala slova su slična velikim, samo je veličina manja 😊),
- *font-stretch* – skuplja ili razvlači font (normal, wider, narrower, ultra-condensed, extra-condensed, condensed, semi-condensed, semi-expanded, expanded, extra-expanded, ultra-expanded),
- *font* – objedinjeni unos osobina fonta (stil variant weight size family)

Okvir

- *border-color* – boja ivice,
- *border-style* – stil linije ivice (none, dotted, dashed, solid double, groove, ridge, inset, outset),
- *border-width* – definiše debljinu okvira oko elementa,
- *border-left-color* – boja levog dela okvira,
- *border-left-style* – stil levog dela okvira,
- *border-left-width* – širina levog dela okvira,
- *border-left* – objedinjeni unos osobina leve ivice (boja stil širina),
- *border-right-color* – boja desnog dela okvira,
- *border-right-style* – stil desnog dela okvira,
- *border-right-width* – širina desnog dela okvira,
- *border-right* – objedinjeni unos osobina desne ivice (boja stil širina)

Okvir

- *border-bottom-color* – boja donjeg dela okvira,
- *border-bottom-style* – stil donjeg dela okvira,
- *border-bottom-width* – širina donjeg dela okvira,
- *border-bottom* – objedinjeni unos osobina donje ivice (boja stil širina),
- *border-top-color* – boja gornjeg dela okvira,
- *border-top-style* – stil gornjeg dela okvira,
- *border-top-width* – širina gornjeg dela okvira,
- *border-top* – objedinjeni unos osobina gornje ivice (boja stil širina),
- *border* – objedinjeni unos osobina za ceo okvir (boja stil širina)

Padding

- Veličina prostora između ivice i sadržaja elementa
- *padding-left*, *padding-right*, *padding-top*, *padding-bottom* – podešava udaljenost sadržaja od pojedinačne ivice (broj ili procenat),
- *padding* – objedinjeni unos osobina za ceo okvir (gore dole levo desno)

Outline

- Linija oko elementa, izvan okvira (bordera),
- *outline-color* – boja outline linije,
- *outline-style* – stil outline linije (none, dotted, dashed, solid, double, groove, ridge, inset, outset),
- *outline-width* – širina outline linije (thin, medium, thick, broj),
- *outline* – objedinjeni unos osobina (boja stil širina)

Margine

- *margin-top*, *margin-bottom*, *margin-left*, *margin-right* – podešavanja margina elementa (auto, konkretna vrednost ili procenat),
- *margin* – objedinjeni unos osobina (gornja donja leva desna)

Podešavanje stilova liste

- *list-style-type* – definiše stil oznake za nabranje (disc, circle, square za neuređene liste, decimal, decimal-leading-zero, lower-roman, upper-roman, lower-alpha, upper-alpha, hebrew, armenian, georgian, cjk-ideographic, hiragana, katakana, hiragana-iroha, katakana-iroha, za uređene liste),
- *list-style-image* – url do lokacije slike koja se koristi za stavke liste (isključuje list-style-type atribut),
- *list-style-position* – pozicija stavke u odnosu na tekst (inside – oznaka za nabranje i donji redovi teksta su jednako poravnati, outside – svi redovi stavke su desno od oznake za nabranje),
- *list-style* – objedinjuje sva prethodna podešavanja (tip pozicija slika),
- *marker-offset* – udaljenost markera od teksta

Tabele

- *border-collapse* – da li postoji samo jedan okvir oko tabele (collapse) ili svaka ćelija ima svoj okvir (detached),
- *border-spacing* – udaljenost između ćelija; može da bude jedna vrednost ili dve (horizontalna i vertikalna),
- *caption-side* – pozicija naslova tabele (top, bottom, left, right),
- *empty-cells* – prikazuje (show) ili skriva (hide) prazne ćelije,
- *table-layout* – širina ćelija; automatski se raširi da prikaže ceo sadržaj (auto) ili fiksne širine (fixed)

Definisanje boja i slika u pozadini

- *background-color* – boja pozadine elementa
- *background-image* – slika koja će biti u pozadini elementa (url(url-do-slike))
- *background-repeat* – da li se pozadinska slika ponavlja ili ne (repeat-x, repeat-y, no-repeat)
- *background-attachment* – da li da se pozadinska slika pomera sa sadržajem elementa (scroll, fixed),
- *background-position* – podešava inicijalnu poziciju pozadinske slike (procentualne, fiksne vrednosti, top, bottom, middle); navodi se prvo x, pa y pozicija,
- *background* – sva svojstva odjednom (color image repeat attachment position)

Pozicija elementa

- *float* – određuje sa koje strane će se tekst prelamati oko elementa (none, left, right)
- *clear* – navodi se sa koje strane u odnosu na element su zabranjeni floating elementi (none, left, right, both)

Prikaz elementa

- *visibility* – da li je element vidljiv ili ne (visible, hidden, collapse (za tabele, sakriva element))
- *display* – način prikaza elementa
 - none
 - block – novi blok
 - inline – novi blok unutar tekuće linije
 - list-item – za liste
 - table – tabela sa novim redom
 - inline-table – tabela unutar paragrafa
 - table-row – kao vrsta tabele
 - table-column – kao kolona tabele
 - table-cell – kao ćelija tabele
- **block** prikaz napravi nov red i prikaže element kao novi paragraf
- **inline** prikaz prikaže element unutar postojećeg paragrafa

Pozicioniranje elemenata

- *position* – određuje poziciju elementa (static, absolute, relative, fixed)
 - *static* – element se iscrtava zajedno sa ostatkom HTML stranice i ne može da se pomera, **default**,
 - *absolute* – pozicionira se na fiksnu poziciju određenu atributima *top* i *left*,
 - *relative* – relativna pozicija u odnosu na normalno sračunatu poziciju u odnosu na ostatak HTML stranice
 - *fixed* – kao apsolutno pozicioniranje, samo što se sadržaj ne skroluje sa stranicom, zato što se pozicionira u odnosu na ivice prozora čitača.
- *left* – horizontalna pozicija elementa
- *top* – vertikalna pozicija elementa
- *right*, *bottom* – alternativno pozicioniranje u odnosu na left/top
- *width*, *height* – širina i visina elementa
- *z-index* – redosled iscrtavanja elementa

Odsecanje dela elementa ili viška

- *overflow* – definiše šta sa viškom
 - *visible* – višak se prikazuje izvan elementa, **default**;
 - *hidden* – višak se ne vidi;
 - *scroll* – prikazuje se linija za skrolovanje
 - *auto* – neka navigator odluči
- *clip* – definiše region vidljivosti komponente
 - *auto* – vidljiva oblast je veličine objekta, **default**;
 - *rect(top, right, bottom, left)* – region clipping-a
 - ne koristi se ako je **overflow** podešen na **visible**

pozicije_clip.html

pozicije_overflow.html

Pseudoklase

- Koriste se da podese dodatna svojstva za neke selektore:
- Linkovi
 - a:link – podešava svojstva linka
 - a:hover – podešava svojstva za link kada je miš iznad
 - a:active – podešava svojstva linka kada se klikne na link
 - a:visited – podešava svojstva već posećenih linkova
- Ostali:
 - :focus – podešava svojstva za element koji je dobio fokus
 - :first-child – podešava svojstva za element koji je prvi podelement nekome

Pseudoelementi

- Koriste se da podese dodatna svojstva za neke selektore:
- :first-letter – podešava svojstva prvog slova u tekstu
- :first-line – podešava svojstva prvog reda u tekstu
- :before – postavlja zadati sadržaj pre pojave elementa
- :after – postavlja zadati sadržaj posle pojave elementa

Podešavanje svojstva kurzora

- Atribut *cursor* podešava svojstva kurzora za zadati element:
 - *auto* – podrazumevana vrednost
 - *none* – sakriva miša
 - *default* – osnovni oblik pokazivača (najčešće strelica)
 - *pointer* – pokazivač na linkove
 - *move* – pokazivač na pokretne objekte
 - *text* – kurzor za tekst
 - *wait* – kurzor za čekanje
 - *help* – kurzor za pomoć
 - *e-resize*, *w-resize*, *n-resize*, *s-resize*, *sw-resize*, *ne-resize*, *nw-resize* – promena orijentacije pokazivača
 - *url* – zadati url do resursa koji opisuje kurzor

Tipovi medija

- Može se podesiti različit prikaz u zavisnosti od tipa medijuma
- Primer:

```
@media screen {  
    p.test {font-family:verdana,sans-serif;  
            font-size:14px}  
}  
  
@media print {  
    p.test {font-family:times,serif; font-  
            size:10px}  
}  
  
@media screen,print {  
    p.test {font-weight:bold}  
}
```

Tipovi medija

- all – svi tipovi medija
- aural – čitač teksta
- braille – brajov čitač
- embossed – brajov štampač
- handheld – PDA, smart phones, itd.
- print – štampač
- projection – za projekcije poput slajdova
- screen – monitor
- tty – uređaji sa fiksnom širinom slova, poput terminala
- tv – televizor

Štampanje

- CSS pruža mogućnost upotrebe odvojenih stilova za različite medije

- Primer:

```
<link rel="stylesheet" type="text/css"  
      href="printstyles.css" media="print">
```

- Neki delovi prezentacije ne moraju da budu odštampani:

```
img.ad {display: none}
```

- Margine možemo da podesimo u mernim jedinicama koje više odgovaraju štampi:

```
div#content { margin-left: 1in; margin-right: 1.5in}
```


Štampanje

- Prelazak na novu stranicu se podešava stilom:
 - page-break-before (vrednosti: auto, always, avoid, left, right)
 - page-break-after (vrednosti: auto, always, avoid, left, right)
 - page-break-inside (vrednosti: auto, avoid)

Web programiranje

JavaScript

Skript jezici

- Obezbeđuju interaktivnost na web stranicama
- "Jednostavni" programski jezici
- Izvršavaju se u čitaču
- Ugrađuju se u HTML stranice
- Interpretirani jezik
 - nema kompajliranja
 - izvršava se momentalno

Skript

- Tag `<script>` specificira Script kod koji se pokreće direktno u browser-u
- Browser sve između tagova `<script>` i `</script>` smatra elementima skripta
- Tag `<script>` se može javiti bilo gde u HTML dokumentu
 - postoji razlika između `<head>` i `<body>` sekcije
 - kod definisan u tagu `<script>` u `<body>` sekciji se izvršava prilikom crtanja stranice
 - kod definisan u tagu `<script>` u `<head>` sekciji se ne izvršava automatski već se poziva iz skripta u `<body>` sekciji
- Ne mora kod da se nalazi u HTML datoteci
 - može i u drugoj datoteci, a da se pozove iz HTML datoteke
- Ako atribut **type** ima vrednost “**text/javascript**”, tada se radi o JavaScript programskom jeziku

Primer

```
<html>
<head>
<script type="text/javascript">
...
</script>
</head>
<body>
<script type="text/javascript">
...
</script>
</body>
```

Primer skripta u datoteci

```
<html>
```

```
<head>
```

```
<script src="skript.js"></script>
```

```
</head>
```

```
<body>
```

```
</body>
```

```
</html>
```

JavaScript

- Sintaksa slična programskom jeziku Java
 - nije programski jezik Java
- Nema tipove podataka
 - kod deklaracije promenljivih se ne stavlja tip (interpreter).
- Nema kreiranja novih klasa
 - ugrađene funkcije,
 - ugrađeni objekti
- Sistem događaja

Pozivanje JavaScript-a

- Kao reakciju na neki događaj.
- Unutar `<script>` taga bilo gde unutar HTML dokumenta
 - Ako koristimo JavaScript funkciju, nju moramo da definišemo unutar `<head>` taga da bismo mogli da je pozivamo iz bilo kog JavaScript koda.
- Kao adresu unutar `<a>` taga:
`
klikni`

Promenljive

- Promenljive sadrže informacije
- Deklaracija promenljivih upotrebom ključne reči var
- Primer:

```
var a;
```

```
var b = 5;
```

```
var c = "Pera";
```

Promenljive

- Nakon deklaracije, varijabla se može inicijalizovati:

```
var x;
```

```
x = 5;
```

- Inicijalizacija može i uz deklaraciju:

```
var x = 5;
```

- Varijabla može i da promeni tip:

```
var x = 5;
```

```
x = "Mika";
```

Aritmetički i operatori dodele

- Aritmetički: + - * / % ++ --

```
x = 5;
```

```
y = x * 4;
```

```
z = y % 5;
```

- Dodele: = += -= *= /= %=

```
y += 5;          y=y+5;
```

- Operator + ima posebno značenje kada su operandi stringovi:

```
a = "Pera";
```

```
b = "Car";
```

```
c = a + b;
```

- Kada sabiramo stringove i brojeve, rezultat je string

Aritmetički operatori

$y = 5;$

Operator	Rezultat
$x=y+2$	$x=7$
$x=y-2$	$x=3$
$x=y\%2$	$x=1$
$x=++y$	$x=6, y=6$
$x=y++$	$x=5, y=6$
$x=--y$	$x=4$

Operatori dodele

`x = 10;`

`y = 5;`

Operator	Isto kao	Rezultat
<code>x=y</code>		<code>x=5</code>
<code>x+=y</code>	<code>x=x+y</code>	<code>x=15</code>
<code>x-=y</code>	<code>x=x-y</code>	<code>x=5</code>
<code>x*=y</code>	<code>x=x*y</code>	<code>x=50</code>
<code>x/=y</code>	<code>x=x/y</code>	<code>x=2</code>
<code>x%=y</code>	<code>x=x%y</code>	<code>x=0</code>

Relacioni operatori

- Relacioni: `==` `===` `!=` `<` `<=` `>` `>=`

```
x = 5;
```

```
if (x == 5)
```

```
    document.write("x je jednako 5");
```

- Operator `===` će porediti i vrednost i tip:

```
if (x === "5")
```

```
    document.write("x je string sa  
sadržajem 5");
```

- Rezultat relacionih operatora je logička vrednost tačno (true) ili netačno (false)

Relacioni operatori

x = 5;

Operator	Rezultat
==	x == 8 je netačno (false)
===	x == 5 je tačno (true) x == "5" je netačno (false)
!=	x != 8 je tačno (true)
>	x > 8 je netačno (false)
<	x < 8 je tačno (true)
>=	x >= 8 je netačno (false)
<=	x <= 8 je tačno (true)

Logički operatori

- Logički: `&&` `||` `!`
- Rezultat logičkih operatora je tačno (true) ili netačno (false)
- Operandi logičkih operatora su logički izrazi

<code>&&</code>	0	1
0	0	0
1	0	1

<code> </code>	0	1
0	0	1
1	1	1

<code>!</code>	
0	1
1	0

Logički operatori

x = 6;

y = 3;

Operator	Objašnjenje	Primer
&&	konjukcija (and, i)	(x < 10 && y > 1) tačno (true)
	disjunkcija (or, ili)	(x==5 y==5) netačno (false)
!	negacija (not, ne)	!(x==y) tačno (true)

Uslovni operator

- Sintaksa

`promenljiva=(uslov)?vrednost1:vrednost2`

- To je kao:

```
if (uslov)
```

```
    promenljiva = vrednost1;
```

```
else
```

```
    promenljiva = vrednost2;
```

- Primer:

```
x = (y>3)?5:6;
```

Kontrola toka

- `if else`
- `switch`
- `for`
- `while`
- `do while`
- `break`
- `continue`

if else

- Opšta sintaksa:

```
if (uslov_1)
    telo_1
else if (uslov_2)
    telo_2
else
    telo_3
```

Js_if.html

Js_if_else.html

Primer

```
if (poeni > 94)
    ocena = 10;
else if (poeni > 84)
    ocena = 9;
else if (poeni > 74)
    ocena = 8;
else if (poeni > 64)
    ocena = 7;
else if (poeni > 54)
    ocena = 6;
else ocena = 5;
```

Primer

```
<script type="text/javascript">
var d = new Date();
var time = d.getHours();

if (time < 10)
{
    document.write("Dobro jutro!");
}
else
{
    document.write("Dobar dan!");
}
</script>
```

switch

- Izraz u `switch()` izrazu mora da proizvede celobrojnu vrednost.
- Ako ne proizvodi celobrojnu vrednost, ne može da se koristi `switch()`, već `if()`!
- Ako se izostavi `break`; propašće u sledeći `case`.
- Kod `default` izraza ne mora `break` - to se podrazumeva.

Primer

```
switch (a)
{
    case 1:
    case 2: i = j + 6;
            break;
    case 3: i = j + 14;
            break;
    default: i = j + 8;
}
```


Primer

```
<script type="text/javascript">
//Nedelja=0, Ponedeljak=1, Utorak=2, itd.

var d=new Date();
theDay=d.getDay();
switch (theDay)
{
case 5:
    document.write("Petak");
    break;
case 6:
    document.write("Subota");
    break;
case 0:
    document.write("Nedelja");
    break;
default:
    document.write("Jos nije vikend!");
}
</script>
```

while

- Za cikličnu strukturu kod koje se samo zna uslov za prekid.
- Telo ciklusa ne mora ni jednom da se izvrši
- Opšta sintaksa:
`while (uslov)`
`telo`
- Važno: izlaz iz petlje na false!

Primer

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=10)
{
    document.write("Trenutno je " + i);
    document.write("<br />");
    i=i+1;
}
</script>
</body>
</html>
```

Primer

```
<html>
<body>
<script type="text/javascript">
//racunanje a na n
i = 1; a = 2; n = 3;
stepen = 1;
while (i++ <= n)
    stepen *= a;
document.write("a na n je " + stepen);
</script>
</body>
</html>
```

do while

- Za cikličnu strukturu kod koje se samo zna uslov za prekid
- Razlika u odnosu na while petlju je u tome što se telo ciklusa izvršava makar jednom.
- Opšta sintaksa:

`do`

`telo`

`while (uslov) ;`

- Važno: izlaz iz petlje na false!

Primer

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
{
    document.write("The number is " + i);
    document.write("<br />");
    i=i+1;
}
while (i<0);
</script>
</body>
</html>
```

for

- Za organizaciju petlji kod kojih se unapred zna koliko puta će se izvršiti telo ciklusa.
- Petlja sa početnom vrednošću, uslovom za kraj i blokom za korekciju.

- Opšta sintaksa:

```
for (inicijalizacija; uslov; korekcija)  
    telo
```

Js_for.html

Js_for_loop_headers.html

for

```
for (i = 0; i < 10; i++)
```

```
    document.write(i + "<br/>");
```

- može i višestruka inicijalizacija i step-statement:

```
for(i = 0, j = 1; i < 10 && j != 11; i++, j++)
```

- oprez (može da se ne završi):

```
var x;
```

```
for (x = 0; x != 10; x+=0.1) ...
```


Primer

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0; i <= 10; i++)
{
    document.write("The number is " + i);
    document.write("<br />");
}
</script>
</body>
</html>
```

break i continue

- **break** – prekida telo tekuće ciklične strukture (ili `case` dela) i izlazi iz nje.
- **continue** – prekida telo tekuće ciklične strukture i otpočinje sledeću iteraciju petlje.

break i continue

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0; i <= 10; i++)
{
    if (i==3)
    {
        break;
    }
    document.write("The number is " + i);
    document.write("<br />");
}
</script>
</body>
</html>
```

Primer – izlaz iz ugnježdene petlje

```
for (...)
{
    for (...)
    {
        ...
        if (uslov)
            break;
    }
}
```

for ... in petlja

- Za iteriranje kroz nizove
- Opšta sintaksa:

```
for (promenljiva in niz) {  
    ...  
}
```

Primer

```
<html>
<body>
<script type="text/javascript">
var x;
var vozila = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";
for (x in vozila)
{
    document.write(vozila[x] + "<br />");
}
</script>
</body>
</html>
```

Funkcije

- Definicija funkcija unutar <head> taga:

```
function f(arg1, arg2) {  
    ...  
    return vrednost;  
}
```
- Poziv funkcije iz tela HTML dokumenta
(unutar <body> taga)

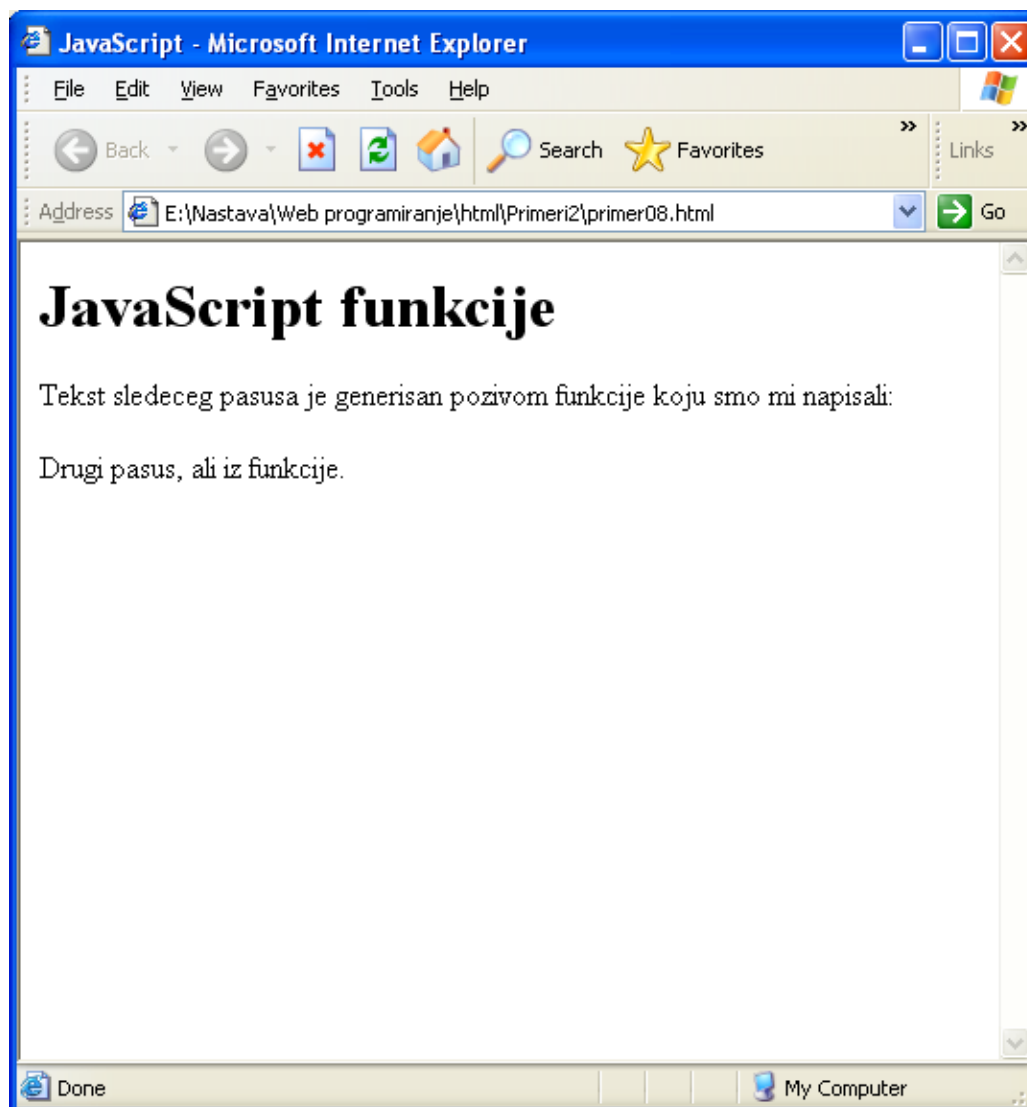
Funkcije

```
<html>
  <head>
    <title>JavaScript</title>
    <script type="text/javascript">
      function ispis() {
        document.write("Drugi pasus, ali iz funkcije.");
      }
    </script>
  </head>
  <body>
    <h1>JavaScript funkcije</h1>

    <p>
      Tekst sledeceg pasusa je generisan pozivom funkcije koju smo mi
      napisali:
    </p>

    <p>
      <script language="JavaScript">
        ispis();
      </script>
    </p>
  </body>
</html>
```


Funkcije



Događaji

- Događaji se registruju i odrađuju *event handler*-ima
- U skoro svaki element se može staviti atribut tipa događaja koji ima kao vrednost ime funkcije koja će se aktivirati (*event handler*)
- Primer:

```
<body onload="ucitavanje()">
```

Događaji

Atribut	Događa se kada ...
onabort	se prekine učitavanje slike
onblur	element izgubi fokus
onchange	korisnik pomeni sadržaj polja
onclick	se klikne mišem na objekat
ondblclick	se dva puta klikne po objektu
onerror	se dogodi greška prilikom učitavanja dokumenta ili slike
onfocus	element dobije fokus
onkeydown	se pritisne taster
onkeypress	se pritisne, pa otpusti taster, ili se drži pritisnut
onkeyup	se otpusti taster
onload	se stranica ili slika učitava
onmousedown	se pritisne dugme miša
onmousemove	se miš pomera
onmouseout	miš izađe izvan zone elementa
onmouseover	miš pređe preko elementa
onmouseup	se otpusti dugme miša
onreset	se klikne na reset dugme
onresize	se prozoru ili frejmu promeni veličina
onselect	je tekst selektovan
onsubmit	se klikne na dugme submit u formi
onunload	korisnik napusti stranicu

Događaji

```
<html>
  <head>
    <title>JavaScript</title>
    <script type="text/javascript">
      function mis() {
        confirm("Da li ste sigurni?");
      }

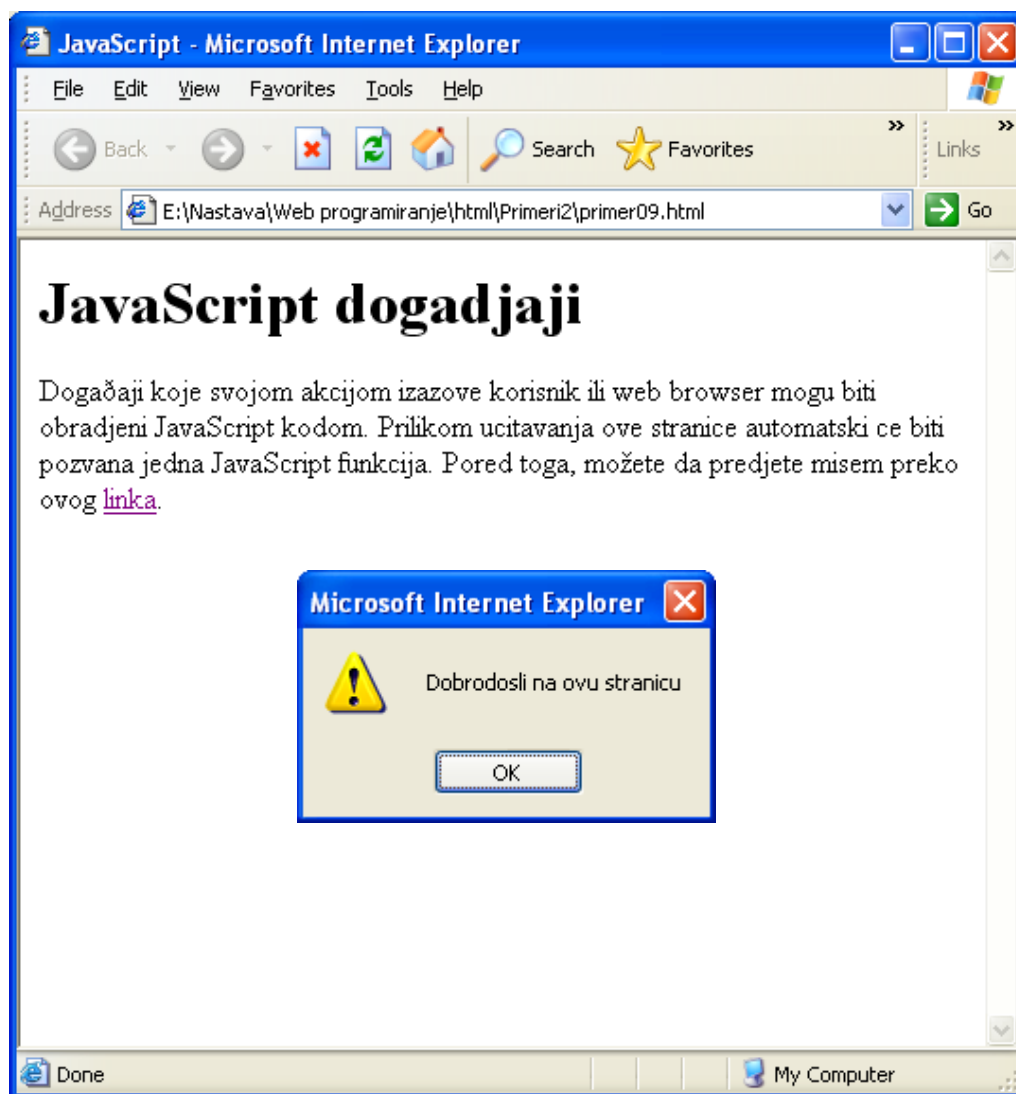
      function greeting() {
        alert("Dobrodošli na ovu stranicu");
      }
    </script>
  </head>
  <body onload="greeting()">
    <h1>JavaScript događaji</h1>
    <p>
      Događaji koje svojom akcijom izazove korisnik ili web browser mogu biti
      obrađeni JavaScript kodom. Prilikom učitavanja ove stranice automatski će
      biti pozvana jedna JavaScript funkcija. Pored toga, možete da predjete
      misem preko ovog <a href="primer09.html" onmouseover="mis()">linka</a>.
    </p>
  </body>
</html>
```

Js_dogadjaji.html

Js_head_alert.html

Js_call_function.html

Događaji



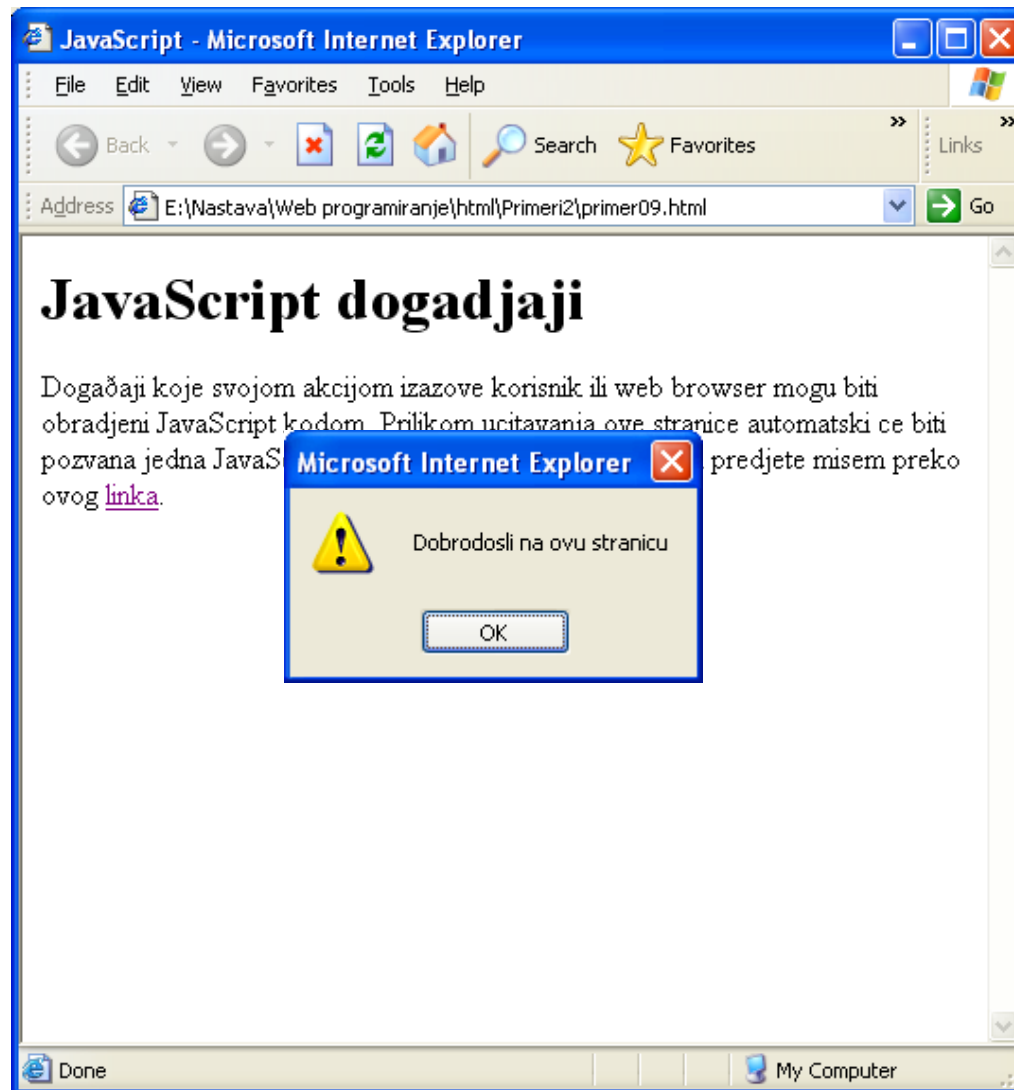
Pozivanje JavaScript-a

- Kao reakciju na neki događaj.
- Unutar `<script>` taga bilo gde unutar HTML dokumenta
 - ako koristimo JavaScript funkciju, nju moramo da definišemo unutar `<head>` taga da bismo mogli da je pozivamo iz bilo kog JavaScript koda.
- Kao adresu unutar `<a>` taga:
`
klikni`

Reakcija na neki događaj

```
<html>
  <head>
    <title>JavaScript</title>
    <script type="text/javascript">
      function greeting() {
        alert("Dobro dosli na ovu stranicu");
      }
    </script>
  </head>
  <body onLoad="greeting()">
    <h1>JavaScript dogadjaji</h1>
    <p>
      Dogadjaji koje svojom akcijom izazove korisnik ili web
      browser mogu biti obradjeni JavaScript kodom. Prilikom
      učitavanja ove stranice automatski će biti pozvana jedna
      JavaScript funkcija.
    </p>
  </body>
</html>
```

Reakcija na neki događaj



Preko <script> taga unutar <body> sekcije

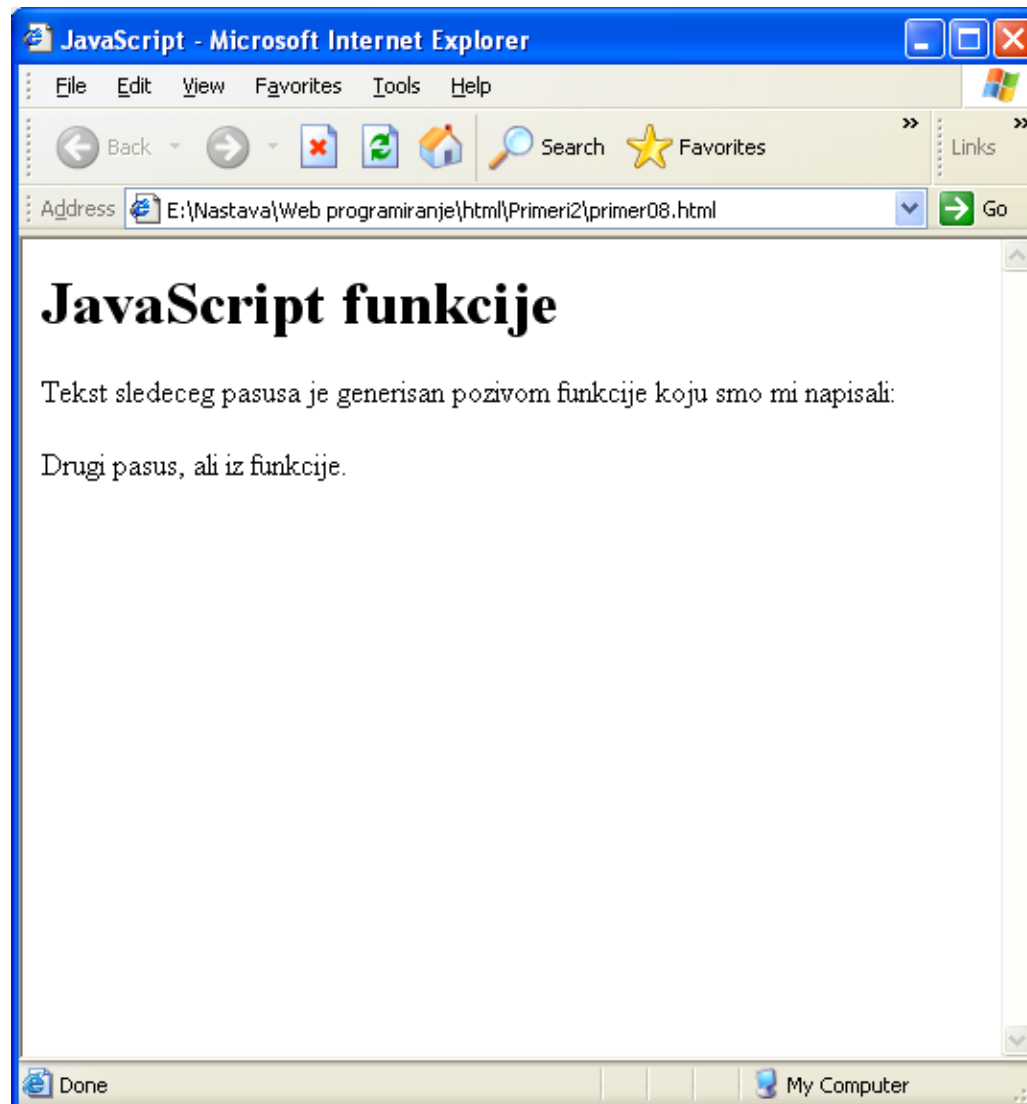
```
<html>
  <head>
    <title>JavaScript</title>
    <script type="text/javascript">
      function ispis() {
        document.write("Drugi pasus, ali iz funkcije.");
      }
    </script>
  </head>
  <body>
    <h1>JavaScript funkcije</h1>

    <p>
      Tekst sledeceg pasusa je generisan pozivom funkcije koju smo mi napisali:
    </p>

    <p>
      <script language="JavaScript">
        ispis();
      </script>
    </p>
  </body>
</html>
```

Js_dobar_dan.html

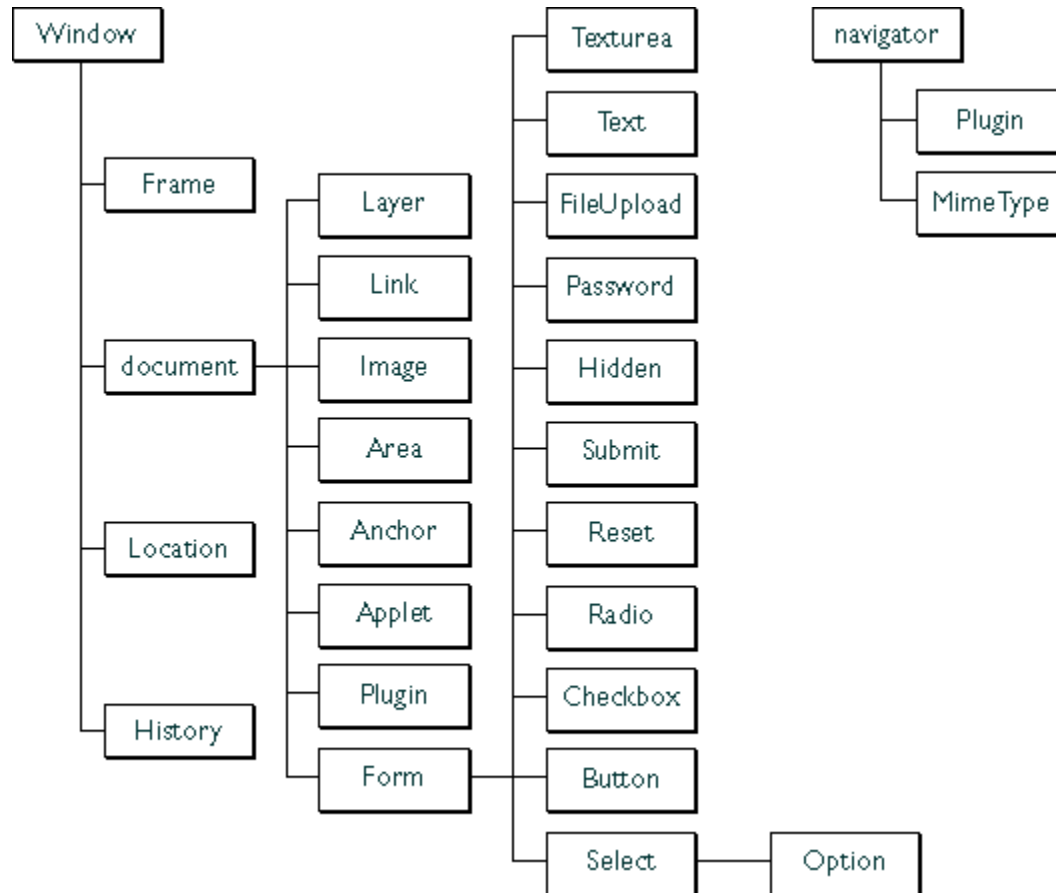
Preko `<script>` taga unutar `<body>` sekcije



Ugrađene funkcije

- **Sistemske funkcije:**
 - *isNaN()* – vraća true ako prosleđeni string nije broj,
 - *eval()* – interpretira prosleđeni string kao JavaScript kod,
 - *parseInt()* – parsira string u intidžer,
 - *parseFloat()* – parsira string u float promenljivu,
 - *alert()* – ispis poruke u MessageBox-u
 - *escape()*, *unescape()* – kodira/dekodira URL-ove (npr. zamenjuje razmak simbolom '+' i sl.)

Hijerarhija objekata



Window objekt

- Omogućuje manipulaciju prozorima
- Sadrži informacije o tekućem prozoru
- Metode:
 - *alert()*, *confirm()*, *prompt()* - poruka u prozoru (MessageBox)
 - *back()*, *forward()* - povratak na prethodnu stranicu/odlazak na sledeću (iz istorije)
 - *moveBy()*, *MoveTo()* - pomera prozor
 - *open()* - otvara nov prozor
 - *setTimeout("kod", timeout)/clearTimeout()* – podešava/isključuje kod koji će se izvršavati kada istekne timeout
 - *setInterval("kod", perioda)/clearInterval()* – zadaje funkciju koja će se periodično izvršavati
- Atributi:
 - *history* - istorija odlazaka na stranice,
 - *document* - tekući HTML dokument,
 - *frames* - niz svih frejmova u prozoru,
 - *location* – kompletan URL tekuće stranice,
 - *statusbar* - statusna linija na dnu ekrana

Location objekt

- Reprezentuje URL stranice koja je učitana u navigator:

`location = "http://www.google.com"`

- Sadrži informacije o tekućem dokumentu
- Metode:
 - *reload()* - ponovno učitavanje tekućeg prozora
 - *replace()* - učitava novi URL
- Atributi:
 - *href* – pun URL do stranice:

`location.href="http://www.google.com"`

- *protocol* – protokol iz URL-a
- *host* – adresa servera iz URL-a
- *port* – port iz URL-a
- *pathname* – putanja do resursa
- *search* – parametri forme

History objekt

- Omogućuje kontrolu pristupa već viđenim stranicama
- Sadrži listu adresa posećenih stranica
- Metode:
 - *back()* - učitava prethodnu stranicu iz liste
 - *forward()* – učitava sledeću stranicu iz liste
 - *go()* - učitava zadatu adresu iz liste
- Atributi:
 - *current* – trenutno učitana adresa
 - *length* – broj stavki u history listi
 - *next* – zadavanje sledećeg elementa
 - *previous* – zadavanje prethodnog elementa

Document objekt

- Omogućuje ispis HTML-a na ekran
- Sadrži informacije o tekućem dokumentu
- Metode:
 - *write()* - ispisuje na ekran tekst
- Atributi:
 - *forms* - niz svih formi u dokumentu
 - *links* - niz svih linkova u dokumentu
 - *applets* - niz svih apleta u dokumentu
 - *title* - sadržaj **title** taga

String objekt

- Reprezentuje string
 - string konstanta “tekst” reprezentuje string
- Metode:
 - *substring()* – vraća deo stringa
 - *split()* – vraća niz stringova kao rezultat “razbijanja” stringa
 - *indexOf()*, *lastIndexOf()* – vraća poziciju nekog podstringa
 - *charAt()* – vraća karakter sa zadate pozicije
- Atributi:
 - *length* – dužina stringa

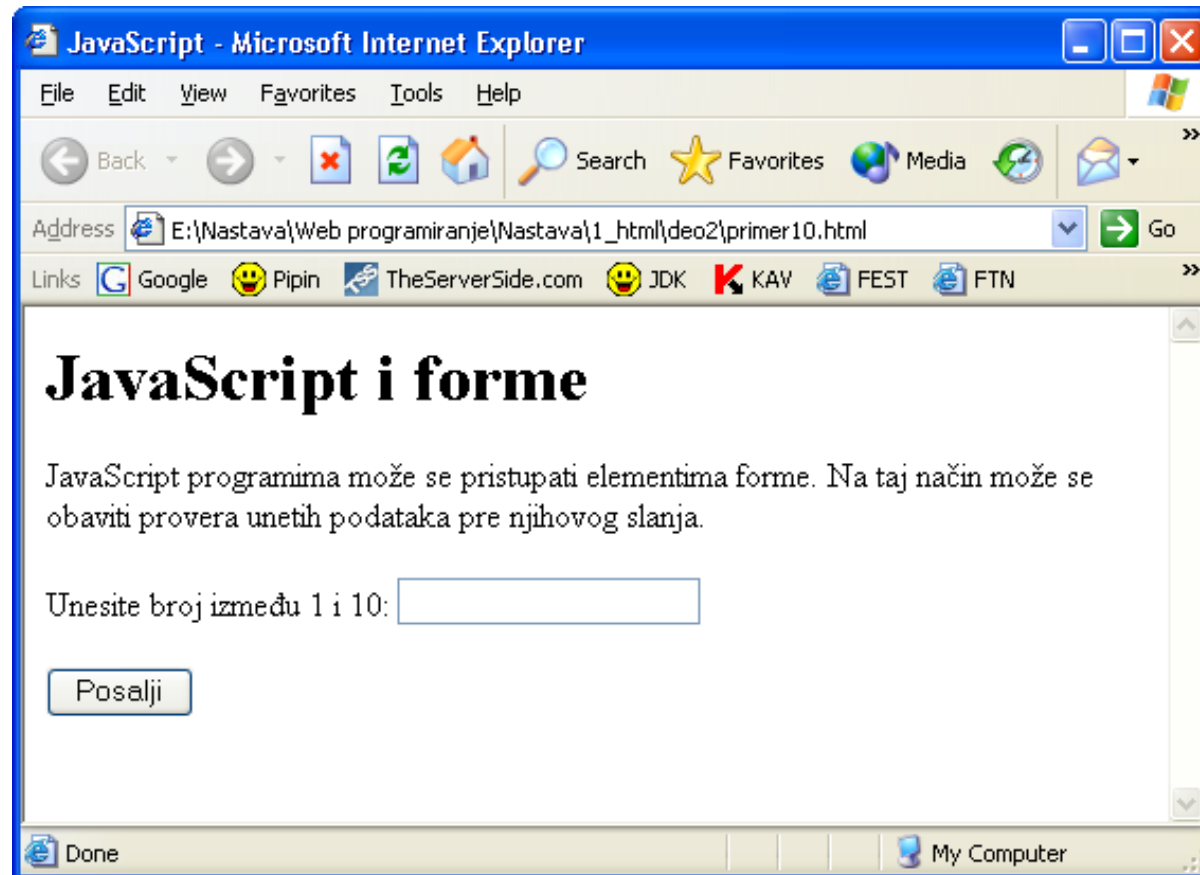
Forme

- Reprezentovane **form** objektom.
- Metode:
 - *submit()* - šalje podatke iz forme na odredište definisano **action** atributom **form** taga.
 - *reset()* - simulira pritisak na Reset dugme forme.
- Atributi:
 - *elements* - niz elemenata forme. Svaki element ima **value** atribut za pristup sadržaju,
 - *length* - broj elemenata na formi.
 - *action* - sadržaj action atributa.

Forme

```
<html>
  <head>
    <title>JavaScript</title>
    <script type="text/javascript">
      function provera() {
        vrednost = document.forms['forma'].polje1.value;
        if (isNaN(vrednost)) {
          alert("Niste uneli broj");
          return false;
        } else if (vrednost >= 1 && vrednost <= 10) {
          return true;
        } else {
          alert("Niste uneli broj u opsegu od 1 do 10");
          return false;
        }
      }
    </script>
  </head>
  <body>
    <h1>JavaScript i forme</h1>
    <p>
      JavaScript programima može se pristupati elementima forme. Na taj nacin moze se
      obaviti provera unetih podataka pre njihovog slanja.
    </p>
    <form name="forma" action="primer10-2.html" onSubmit="return provera()">
      <p>
        Unesite broj između 1 i 10:
        <input type="text" name="polje1"> <br><br>
        <input type="submit" name="polje2" value=" Posalji ">
      </p>
    </form>
  </body>
</html>
```

Forme



Forme

```
<html>

  <head>
    <title>JavaScript</title>
  </head>

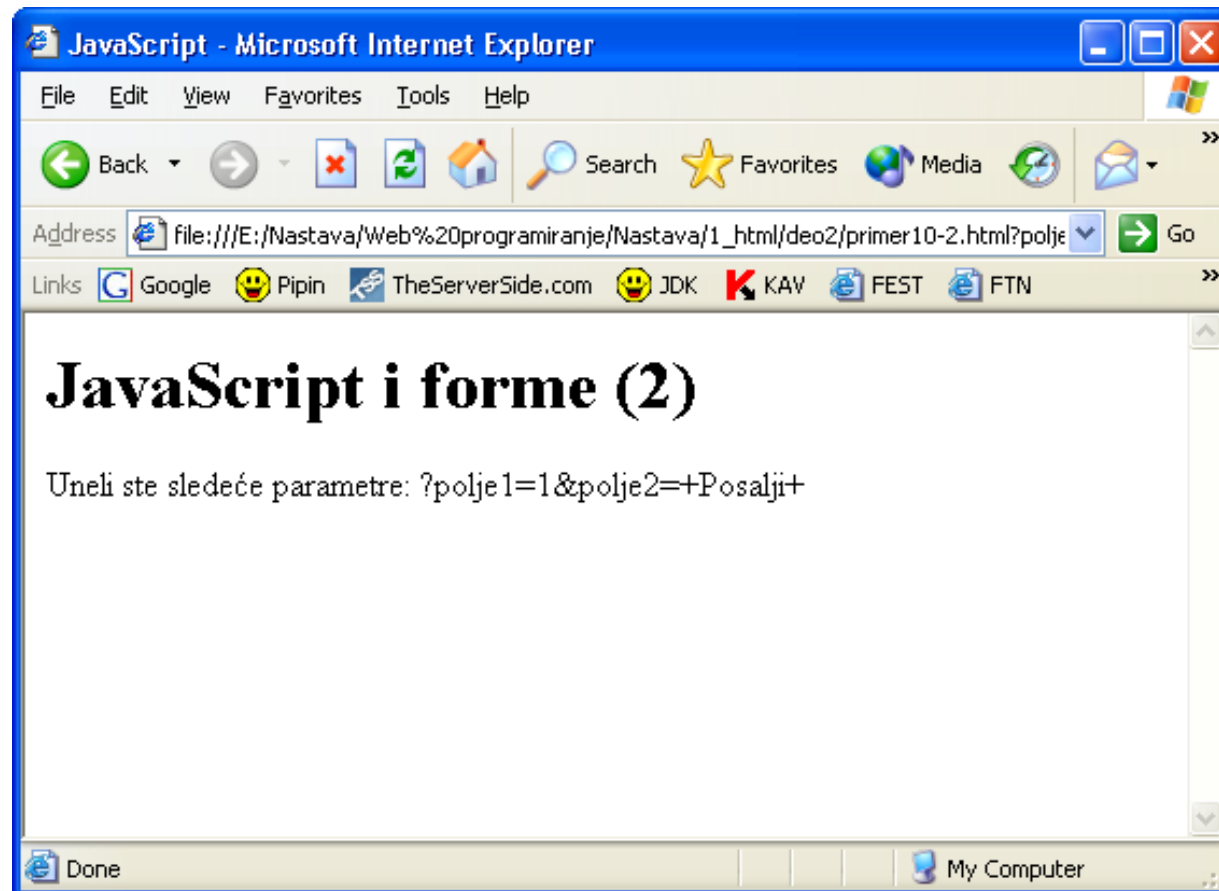
  <body>
    <h1>JavaScript i forme (2)</h1>

    <p>
      Uneli ste sledece parametre:
      <script type="text/javascript">
        document.write(window.location.search);
      </script>
    </p>

  </body>

</html>
```

Forme



Web programiranje

DHTML

DHTML

- Kombinacija tehnologija pomoću kojih se može kontrolisati izgled stranice na konzistentan način:
 - HTML,
 - CSS,
 - JavaScript,
 - DOM
- Unutar HTML-a se uglavnom koriste slojevi (layers)

Slojevi

- Tag za sloj je `<div>` ili ``
 - razlika je u načinu prikaza: `div` je block element, a `span` je inline element
- Uglavnom se “obogaćuje” CSS atributima
- Primer:

```
<div id="sloj1"  
  style="position:absolute;left:100;top:50">
```

Tekst unutar sloja

```
</div>
```

Stilovi za slojeve

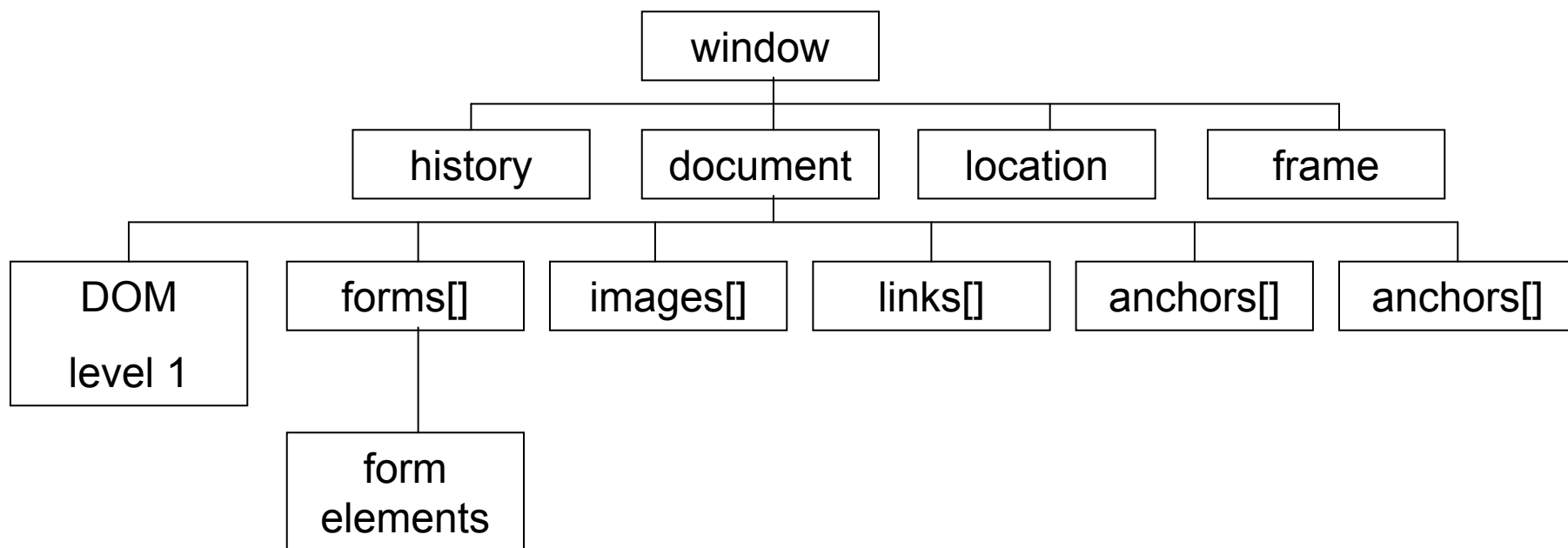
- *position* – određuje poziciju elementa (static, absolute, relative, fixed)
 - *static* – element se iscrtava zajedno sa ostatkom HTML stranice i ne može da se pomera, **default**,
 - *absolute* – pozicionira se na fiksnu poziciju određenu atributima *top* i *left*,
 - *relative* – relativna pozicija u odnosu na normalno sračunatu poziciju u odnosu na ostatak HTML stranice
 - *fixed* – kao apsolutno pozicioniranje, samo što se sadržaj ne skroluje sa stranicom, zato što se pozicionira u odnosu na ivice prozora čitača.
- *left* – horizontalna pozicija elementa
- *top* – vertikalna pozicija elementa
- *right*, *bottom* – alternativno pozicioniranje u odnosu na left/top
- *width*, *height* – širina i visina elementa
- *z-index* – redosled iscrtavanja elementa

Document Object Model (DOM)

- DOM predstavlja objektnu reprezentaciju XML dokumenta.
- JavaScript poseduje skup funkcija za rad sa DOM objektima.
- Postoji više nivoa reprezentacije:
 - DOM Level 0 i
 - DOM Level 1,
 - DOM Level 2,
 - DOM Level 3.

DOM Level 0

- DOM Level 0 omogućuje pristup elementima stranice preko predefinisanih objekata.

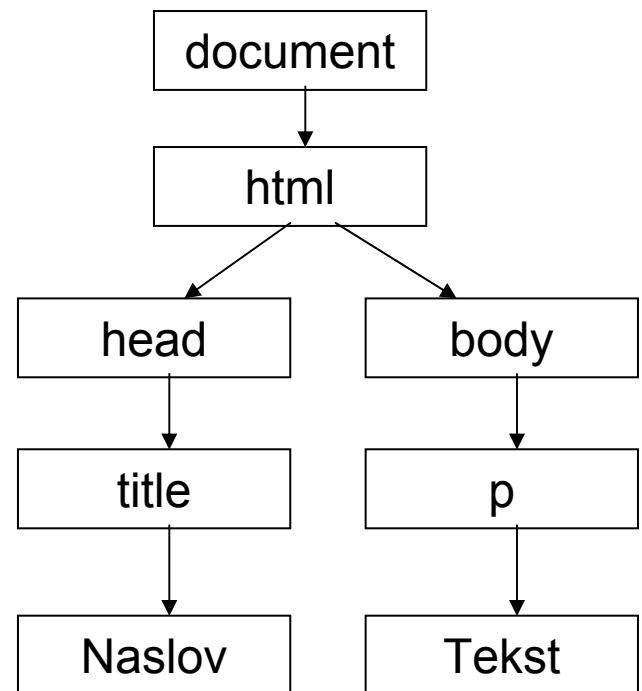


DOM Level 1-3

- DOM nivoi 1-3 predstavljaju objektnu reprezentaciju sadržaja HTML dokumenta

- Primer:

```
<html>  
  <head>  
    <title>Naslov</title>  
  </head>  
  <body>  
    <p>Tekst</p>  
  </body>  
</html>
```



DOM reprezentacija

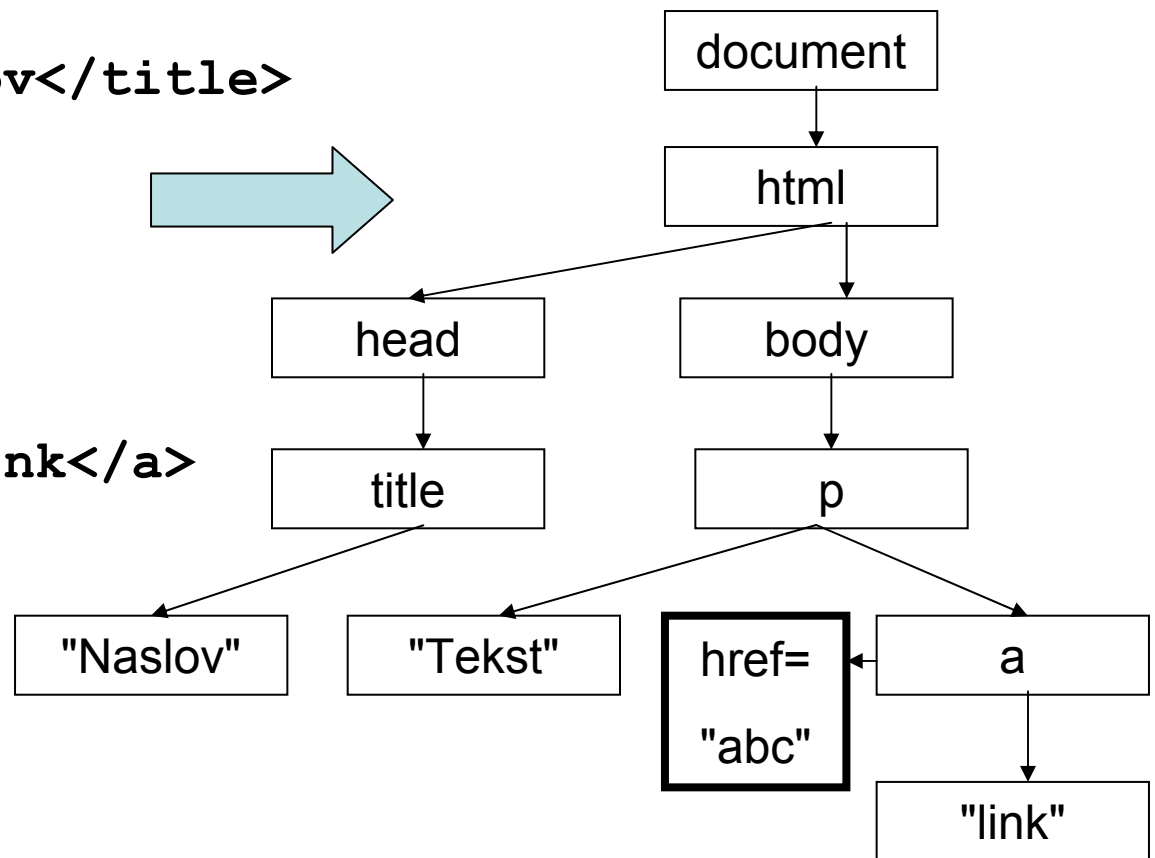
- HTML dokument se posmatra kao stablo koje se sastoji iz elemenata
- Koren stabla je `<html>` tag
- Svaki HTML tag je čvor tipa element u stablu
- Svaki atribut je čvor tipa atribut u stablu
- Svaki tekst je čvor tipa tekst (tekstualni čvor) u stablu
- Svaki komentar je čvor tipa komentar u stablu

DOM Stablo

- Svaki HTML dokument se posmatra kao DOM stablo
- Čvor na vrhu se zove korenski čvor
- Svaki čvor osim korenskog ima jednog roditelja (čvor iznad)
- Svaki čvor može da ima potomke (decu – čvorovi ispod)
- List je čvor bez dece
- Čvorovi istog nivoa (sibling) su čvorovi sa istim roditeljem

DOM Stablo

```
<html>
  <head>
    <title>Naslov</title>
  </head>
  <body>
    <p>
      Tekst
      <a href="abc">link</a>
    </p>
  </body>
</html>
```



DOM i JavaScript

- DOM objektima se može pristupiti jedino iz skripta.
- JavaScript poseduje attribute i metode za pristup DOM elementima.
- Osnovni element je document objekat.
 - On sadrži sve čvorove DOM stabla koji reprezentuju HTML stranicu
- Obično se elementima HTML stranice doda atribut ***id*** da bi se lakše pronašli u DOM stablu:
<p id="paragraf">Tekst</p>

Objekat *document*

- Atributi:
 - *document.documentElement* – dokument čvor
- Metode:
 - *document.getElementById(id)* – vraća element sa zadatim id-om
 - *document.getElementsByTagName(name)* – vraća elemente koji se zovu kao zadati parametar
 - *document.createElement(tag)* – kreira novi element sa zadatim imenom
 - *document.createTextNode(tekst)* – kreira tekstualni čvor sa zadatim tekstom

Objekat tipa čvor (node)

- Atributi:
 - *nodeName* – ime čvora
 - *nodeType* – tip čvora (1 za HTML tagove, 2 za attribute, 3 za tekstualne čvorove, 8 za komentar, 9 za dokument)
 - *nodeValue* – sadržaj tekstualnog čvora
 - *innerHTML* – sadržaj čvora kao HTML
 - *id* – ID čvora
 - *firstChild*, *lastChild* – prvi/poslednji čvor ispod u hijerarhiji
 - *childNodes* – niz čvorova koji su u prvom nivou ispod, u hijerarhiji
 - *parentNode* – objekat koji sadrži tekući čvor
- Atributi stila – svaki čvor ima atribut stila – *style*:
 - *cvor.style.top=10* – stil {top:10}
 - *cvor.style.visibility="visible"* – stil {visibility:visible}
- Ako je naziv stila sa crticom, u JavaScriptu se spaja i koristi veliko slovo:
 - *cvor.style.borderWidth = 0* – stil {border-width:0}

Objekat tipa čvor (node)

- Metode:
 - *appendChild(čvor)* – dodaje tekućem čvoru novi čvor, na kraj prvog nivoa ispod u hijerarhiji
 - *insertBefore(čvor, drugi)* – ubacuje zadati čvor ispred drugog čvora
 - *removeChild(čvor)* – uklanja zadati čvor iz stabla
 - *getAttribute(ime)* – vraća vrednost zadatog atributa
 - *setAttribute(ime, vrednost)* – postavlja vrednost atributa
 - *removeAttribute(ime)* – uklanja zadati atribut
 - *hasAttributes()* – vraća true ako tekući čvor ima attribute

Veza između JavaScript-a i stilova

- Sloju se dodeli ID:

```
<div id="aName" style="position...">...</div>
```

- Registruje se JavaScript funkcija za neki tip događaja:

```
<div id="file" onmouseover="showmain(this)" >
```

- Ako je potrebno da se *event handler* definiše na nivou dokumenta, radi se ovako:

```
function init() {  
document.onmousedown=engage;  
document.onmousemove=dragLayer;  
document.onmouseup=disengage;  
}
```

Veza između JavaScript-a i stilova

- Iz JavaScript-a se elementima HTML stranice pristupa preko DOM modela:

```
target = document.getElementById(neki_id);
```

- Atributi stila proizvoljnog elementa se menjaju preko atributa style elementa:

```
target.style.display = "none";
```

- Ako atribut ima '-' u imenu, izbací se '-' i stavi veliko slovo:

```
target.style.borderWidth = 0;
```

Primer

- Listanje svih <p> tagova u dokumentu:

```
<html>
<body>
<p>Jedan paragraf</p>
<div>
<p>Drugi paragraf</p>
<p>Treći paragraf</p>
</div>
<script type="text/javascript">
x=document.getElementsByTagName("p");
document.write("<ul>");
for (i=0;i<x.length;i++)
{
    document.write("<li>" + x[i].innerHTML + "</li>");
}
document.write("</ul>");
</script>
</body>
</html>
```

Primer

Jedan paragraf

Drugi paragraf

Treći paragraf

- Jedan paragraf
- Drugi paragraf
- Treći paragraf

Primer

- Ispis prvog potomka zadatog <p> taga:

```
<html>
```

```
<body>
```

```
<p id="intro">Jedan paragraf</p>
```

```
<div>
```

```
<p>Drugi paragraf</p>
```

```
<p>Treći paragraf</p>
```

```
</div>
```

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

```
x=document.getElementById("intro");
```

```
document.write("Sadržaj prvog paragrafa: " +  
    x.firstChild.nodeValue);
```

```
</script>
```

```
</body>
```

```
</html>
```

Primer

Jedan paragraf

Drugi paragraf

Treći paragraf

Sadržaj prvog paragrafa: Jedan paragraf

Primer

- Drag and Drop

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Primer drag & drop tehnike</title>

<script type="text/javascript">
var engaged=null;
function engage(e) {
    // x i y koordinate klika na ekranu
    var Xin = (navigator.appName.indexOf("Internet Explorer")==-1)?
        e.clientX : event.clientX;
    var Yin = (navigator.appName.indexOf("Internet Explorer")==-1)?
        e.clientY : event.clientY;

    var stil = document.getElementById("aName").style;

    // left i top od lejera
    var L = parseFloat(stil.left);
    var T = parseFloat(stil.top);

    // width i height od lejera
    var W = parseFloat(stil.width);
    var H = parseFloat(stil.height);

    // samo ako smo kliknuli unutar pravougaonika
    if(Xin > L && Xin < (L + W) && Yin > T && Yin < (T + H)) {
        engaged=true;
        moveX = Xin - L;
        moveY = Yin - T;
        return;
    }

    engaged=null;
}

function disengage(){
    if(engaged==null){
        return false;
    }
    moveX=null;
    moveY=null;
    engaged=null;
}
```

```
function dragLayer(e){
    if(engaged==null){
        return false;
    }
    var Xin = (navigator.appName.indexOf("Internet Explorer")==-1)? e.clientX
        : event.clientX;
    var Yin = (navigator.appName.indexOf("Internet Explorer")==-1)? e.clientY
        : event.clientY;

    var stil=document.getElementById("aName").style;
    stil.left = Xin - moveX;
    stil.top = Yin - moveY;
}

function inicijalizacija() {
    document.onmousedown=engage;
    document.onmousemove=dragLayer;
    document.onmouseup=disengage;
}

</script>
</head>

<body onload="inicijalizacija()" >
<div id="aName" style="position:absolute; left: 40px;top: 40px; z-index: 3;
    visibility: visible;width: 300px;height: 50px;background-color:
    #ffffff;color:#000000;padding: 2px;border-style: solid;border-width:
    2px;border-color: #0000ff">
Ovo je tekst.
</div>

</body>
</html>
```

drag-and-drop.html

Primer

Ovo je tekst.

Primer

- Drop down navigacija

```
<html>
<head>

<script type="text/javascript">

function go()
{
location=document.forms[0].gowhere.value;
}

</script>

<style type="text/css">
.style1 {color: #FF0000}
</style>
</head>
<body>

<form>
<select id="gowhere" onchange="go()">
<option>-Izaberite lokaciju-
<option value="http://www.grid.ns.ac.yu">GRID</option>
<option value="http://www.ftn.ns.ac.yu">FTN </option>
<option value="http://www.kba.de">KBA </option>
</select>
</form>
<span class="style1">Napomena: Ovde ima bug. Kada se izabere "-Izaberi lokaciju-", program i dalje radi
redirekciju. </span>
</body>
</html>
```

drop_down.html

Primer

Napomena: Ovde ima bug. Kada se izabere "-Izaberi lokaciju-", program i dalje radi redirekciju.

Primer

- Horizontalni meni

```
<html>
<head>
<style>
body{font-family:arial;}
table{font-size:80%;background:black}
a{color:black;text-decoration:none;font:bold}
a:hover{color:#606060}
td.menu{background:lightblue}
table.menu
{
font-size:100%;
position:absolute;
visibility:hidden;
}
</style>
<script type="text/javascript">
function showmenu(elmnt)
{
document.getElementById(elmnt).style.visibility="visible";
}
function hidemenu(elmnt)
{
document.getElementById(elmnt).style.visibility="hidden";
}
</script>
</head>

<body>
<h3>Drop down menu</h3>
<table width="100%">
<tr bgcolor="#FF8080">
<td onmouseover="showmenu('tutorials')"
onmouseout="hidemenu('tutorials')">
<a href="/default.asp">Tutorials</a><br />
<table class="menu" id="tutorials" width="120">
<tr><td class="menu"><a href="/html/default.asp">HTML</a></td></tr>
<tr><td class="menu"><a href="/xhtml/default.asp">XHTML</a></td></tr>
<tr><td class="menu"><a href="/css/default.asp">CSS</a></td></tr>
<tr><td class="menu"><a href="/xml/default.asp">XML</a></td></tr>
<tr><td class="menu"><a href="/xsl/default.asp">XSL</a></td></tr>
</table>
</td>
```

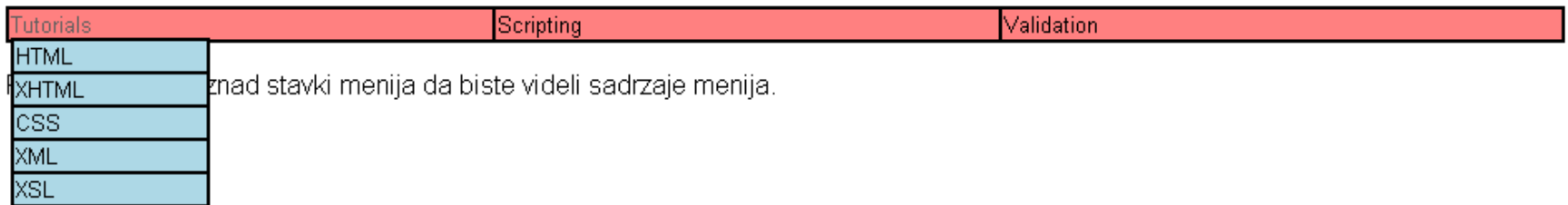
```
<td onmouseover="showmenu('scripting')"
onmouseout="hidemenu('scripting')">
<a href="/default.asp">Scripting</a><br />
<table class="menu" id="scripting" width="120">
<tr><td class="menu"><a href="/js/default.asp">JavaScript</a></td></tr>
<tr><td class="menu"><a href="/vbscript/default.asp">VBScript</a></td></tr>
<tr><td class="menu"><a href="/default.asp">DHTML</a></td></tr>
<tr><td class="menu"><a href="/asp/default.asp">ASP</a></td></tr>
<tr><td class="menu"><a href="/ado/default.asp">ADO</a></td></tr>
</table>
</td>
<td onmouseover="showmenu('validation')"
onmouseout="hidemenu('validation')">
<a href="/site/site_validate.asp">Validation</a><br />
<table class="menu" id="validation" width="120">
<tr><td class="menu"><a href="/site/site_validate.asp">Validate
HTML</a></td></tr>
<tr><td class="menu"><a href="/site/site_validate.asp">Validate
XHTML</a></td></tr>
<tr><td class="menu"><a href="/site/site_validate.asp">Validate
CSS</a></td></tr>
<tr><td class="menu"><a href="/site/site_validate.asp">Validate
XML</a></td></tr>
<tr><td class="menu"><a href="/site/site_validate.asp">Validate
WML</a></td></tr>
</table>
</td>
</tr>
</table>
<p>Postavite kursor iznad stavki menija da biste videli sadržaje
menija. </p>
</body>

</html>
```

horizontalni_meni.html

Primer

Drop down menu



Primer

- Meni koji je uvek pri vrhu stranice bez obzira na skrolovanje

```
<html>
<head>
<script type="text/javascript">
var timer;
function scrolltop()
{
document.getElementById('scrollmenu').style.top=document.body.scrollTop;
timer=setTimeout("scrolltop()",1);
}
function stoptimer()
{
clearTimeout(timer);
}
</script>
</head>

<body onLoad="scrolltop()" onUnload="stoptimer()">
<span id="scrollmenu" style="position:absolute">
<b>Meni</b><br />
<a href="http://www.w3schools.com">W3Schools</a><br />
<a href="http://www.microsoft.com">Microsoft</a><br />
<a href="http://www.altavista.com">Altavista</a><br />
</span>

<table border="0" width="100%">
<tr>
<td width="100"> </td>
<td>Skrolujte stranicu da vidite &quot;uvek-na-vrhu&quot; meni
<br /><br /><br /><br /><br />
Skrolujte stranicu da vidite &quot;uvek-na-vrhu&quot; meni
<br /><br /><br /><br /><br />
</td>
</tr>
</table>

</body>
</html>
```

meni_scroll_top.html

Primer

Meni

[W3Schools](#)

[Microsoft](#)

[Altavista](#)

Skrolujte stranicu da vidite "uvek-na-vrhu" meni

Skrolujte stranicu da vidite "uvek-na-vrhu" meni

Skrolujte stranicu da vidite "uvek-na-vrhu" meni

Skrolujte stranicu da vidite "uvek-na-vrhu" meni

Skrolujte stranicu da vidite "uvek-na-vrhu" meni

Primer

- Meni sa drugačijim borderom nad selektovanom stavkom

```
<html>
<head>
<script type="text/javascript">
function inset(elmnt)
{
elmnt.style.borderStyle="inset";
}
function outset(elmnt)
{
elmnt.style.borderStyle="outset";
}
</script>
<style>
td
{
background:C0C0C0;
border:2px outset;
}
</style>
</head>
<body>

<table width="80">
<tr><td onmouseover="inset(this)" onmouseout="outset(this)"><a href="/default.asp">HOME</a></td></tr>
<tr><td onmouseover="inset(this)" onmouseout="outset(this)"><a href="/js/default.asp">JavaScript</a></td></tr>
<tr><td onmouseover="inset(this)" onmouseout="outset(this)"><a href="http://www.microsoft.com">Explorer</a></td></tr>
<tr><td onmouseover="inset(this)" onmouseout="outset(this)"><a href="http://my.netscape.com">Navigator</a></td></tr>
<tr><td onmouseover="inset(this)" onmouseout="outset(this)"><a href="http://www.altavista.com">AltaVista</a></td></tr>
<tr><td onmouseover="inset(this)" onmouseout="outset(this)"><a href="http://www.yahoo.com">Yahoo!</a></td></tr>
</table>

</body>
</html>
```

menu_borders.html

Primer

[HOME](#)

[JavaScript](#)

[Explorer](#)

[Navigator](#)

[AltaVista](#)

[Yahoo!](#)

Primer

- Meni sa objašnjenjem kada se pređe mišem preko stavke

```
<html>
<head>
<style>
table
{
background:black;
}
a
{
text-decoration:none;
color:#000000;
}
th
{
width:150px;
background:#FF8080;
}
td
{
font:bold;
background:#ADD8E6;
}
</style>
<script type="text/javascript">
function gettip(txt)
{
document.getElementById('tip').innerHTML=txt;
}
function reset()
{
document.getElementById('tip').innerHTML=" ";
}
</script>
</head>
```

```
<body>
<b>Postavite kursor iznad linkova da vidite opise
pojmovia </b><br />
<table width="400px">
<tr>
<th>
<a href="http://www.w3schools.com"
onmouseover="gettip('W3Schools is the best Web
Developers resource on the Web') "
onmouseout="reset()">W3Schools.com</a>
</th>
<td rowspan="3" id="tip"> </td>
</tr>
<tr>
<th>
<a href="http://www.microsoft.com"
onMouseOver="gettip('Internet Explorer is winning
the browser war') "
onmouseout="reset()">Internet Explorer</a>
</th>
</tr>
<tr>
<th>
<a href="http://my.netscape.com"
onMouseOver="gettip('The Navigator is Netscapes
browser tribute to web surfers') "
onmouseout="reset()">Netscape Navigator</a>
</th>
</tr>
</table>

</body>
</html>
```

menu_mouseover_description.html

Primer

Postavite kursor iznad linkova da vidite opise pojmova

W3Schools.com	W3Schools is the best Web Developers resource on the Web
Internet Explorer	
Netscape Navigator	

Primer

- Meni sa slikom koja se prikaže kada se pređe mišem preko stavke

```
<html>
<head>
<style>
table
{
background:black;
}
a
{
text-decoration:none;
color:#000000;
}
th
{
width:150px;
background:#FF8080;
}
td
{
font:bold;
background:#ADD8E6;
}
</style>
<script type="text/javascript">
function gettip(image)
{
document.getElementById('tip').innerHTML="<img src='" +
image + "' />";
}
function reset()
{
document.getElementById('tip').innerHTML=" ";
}
</script>
</head>
```

```
<body>
<b>Postavite kursor iznad linkova da vidite logo
pojmovna</b><br />
<table width="400px">
<tr>
<th>
<a href="/default.asp"
onmouseover="gettip('w3schools.gif')"
onmouseout="reset()">W3Schools.com</a>
</th>
<td rowspan="3" id="tip" align="center" valign="center">
</td>
</tr>
<tr>
<th>
<a href="http://www.microsoft.com"
onmouseover="gettip('microsoft.gif')"
onmouseout="reset()">Internet Explorer</a>
</th>
</tr>
<tr>
<th>
<a href="http://my.netscape.com"
onmouseover="gettip('netscapelink.gif')"
onmouseout="reset()">Netscape Navigator</a>
</th>
</tr>
</table>
</body>

</html>
```

menu_mouseover_logo.html

Primer

Postavite kursor iznad linkova da vidite logo pojnova

W3Schools.com	
Internet Explorer	
Netscape Navigator	

Programski jezik Java

Uvod

Uvod

- Dr Milan Vidaković
- Preporučena literatura:

Java i Internet programiranje,

Branko Milosavljević, Milan Vidaković,

FTN izdavaštvo, Novi Sad, 2007.

ISBN 978-86-7892-047-9

Programski jezik Java

1. Java: platforma za izvršavanje programa
2. Java: programski jezik

Java kao platforma

- dizajniran da što manje zavisi od specifičnih karakteristika konkretnog računarskog sistema
- jednom napisan i *preveden* program se izvršava na bilo kojoj platformi koja podržava Javu

Java kao platforma

- interpretirani jezik
 - just in time compiler
- bajt-kod
 - specifikacija je dostupna – više implementacija kompajlera
- Java virtuelna mašina (JVM)
 - specifikacija je dostupna – više implementacija JVM

Java kao platforma

- više vrsta Java programa
 - aplikacije – u desktop računarima
 - apleti (applets) – u web čitačima
 - servleti (sevlets) – u web serverima
 - kardleti (cardlets) – u smart karticama (kreditne kartice, lične karte, itd.)
 - midleti (midlets) – u mobilnim telefonima (igrice, itd.)

Java kao platforma

- aplikacije
 - izvršavaju se kao regularne aplikacije
 - neograničene mogućnosti i pristup
- apleti
 - izvršavaju se u okviru WWW čitača
 - automatska distribucija i instalacija
 - ograničene mogućnosti apleta iz razloga bezbednosti

Java kao programski jezik

- jezik opšte namene
- konkurentno, objektno-orijentisano programiranje
- literatura
 - Referentna dokumentacija: JavaSoft homepage
<http://java.sun.com>
 - Preporučena knjiga:
Milosavljević, Vidaković: *Java i Internet programiranje*
Bruce Eckel: *Thinking in Java*,
<http://www.bruceeckel.com>

Izvršavanje programa

- metoda `main()`

Hello.java

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

Prevođenje i pokretanje

- prevođenje:

```
javac Hello.java
```

- pokretanje:

```
java Hello
```

[ovo važi sa standardni razvojni paket JDK (Java Development Kit)]0

Osnovni koncepti

- Objektno-orijentisan jezik:
 - atributi: promenljive unutar klase
 - metode: funkcije i procedure unutar klase
- Klasa – model objekta
 - tip podatka `int` predstavlja model celobrojne promenljive.
- Objekat – instanca klase.
 - promenljiva `a` je instanca **`int`** tipa podatka:
`int a;`
 - promenljiva `auto` je instanca **`Auto`** tipa podatka:
`Auto a = new Auto();`

Osnovni koncepti

- sintaksa: podseća na C++
- programski blok je ograden vitičastim zagradama:
{ ... }
- tipovi podataka
 - primitivni tipovi
 - kao lokalne promenljive i parametri metoda, čuvaju se na steku
 - kao parametri, uvek se prenose po vrednosti!
 - objekti
 - čuvaju se na heap-u
 - postoje samo reference na objekte, nikada se ne može pristupiti samom objektu
 - kao lokalne promenljive i parametri metoda, reference se čuvaju na steku
- metode: povratna_vrednost naziv(parametri) { }

Osnovni koncepti

- primitivni tipovi podataka

Primitivni tip	Veličina	Minimum	Maksimum
boolean	1-bit	–	–
char	16-bit	Unicode 0	Unicode $2^{16} - 1$
byte	8-bit	–128	+127
short	16-bit	-2^{15}	$+2^{15} - 1$
int	32-bit	-2^{31}	$+2^{31} - 1$
long	64-bit	-2^{63}	$+2^{63} - 1$
float	32-bit	IEEE754	IEEE754
double	64-bit	IEEE754	IEEE754
void	–	–	–

Deklaracija promenljive primitivnog tipa

- Promenljiva se može deklarirati u bilo kom bloku – ne mora na početku metode.
- `int a;`
- `int a = 0;`
- `int a, b;`
- `int a = 0, b = 3;`

Implicitna konverzija tipova

- Sa “užeg” ili “manjeg” tipa na “širi” ili “veći” tip.
- Nema gubitka informacije jer “uži” tip podatka staje u “širi” tip podatka.
- Primer:

```
long a;
```

```
int i = 5;
```

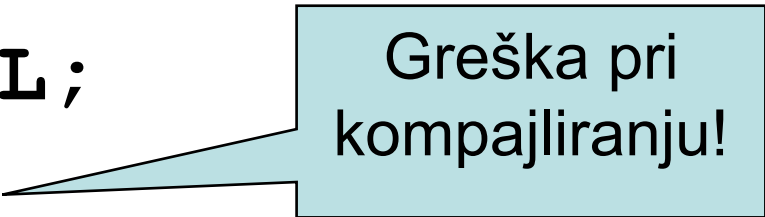
```
a = i;
```

EksPLICITNA konverzija tipova

- Sa “šireg” na “uži” tip podatka – posledica je gubljenje informacije.
- Primer:

```
long a = 5L;
```

```
int b = a;
```



Greška pri
kompajliranju!

Eksplicitna konverzija tipova

- Pravilna eksplicitna konverzija – upotreba **cast** operatora:
- Primer:

```
long a = 5L;
```

```
int b = (int)a;
```

Enumeracije

- Nabrojivi tipovi podataka (celobrojni)
- Primer:

```
enum Size {SMALL, MEDIUM, LARGE,  
          EXTRA_LARGE};
```

```
Size s = Size.MEDIUM;
```

Operatori

- aritmetički operatori
- relacioni i logički
- bit-operatori
- operator dodele

Aritmetički operatori

- Osnovne operacije:

$+$, $-$, $*$, $/$, $\%$

- Umesto $x = x + 1$

$x += 1$

- Automatski inkrement: $++x$ odn. $x++$

Aritmetički operatori

$y = 5;$

Operator	Rezultat
$x=y+2$	$x=7$
$x=y-2$	$x=3$
$x=y\%2$	$x=1$
$x=++y$	$x=6, y=6$
$x=y++$	$x=5, y=6$
$x=--y$	$x=4$

Aritmetički operatori

$x = 10;$

$y = 5;$

Operator	Isto kao	Rezultat
$x=y$		$x=5$
$x+=y$	$x=x+y$	$x=15$
$x-=y$	$x=x-y$	$x=5$
$x*=y$	$x=x*y$	$x=50$
$x/=y$	$x=x/y$	$x=2$
$x\%=y$	$x=x\%y$	$x=0$

Relacioni i logički operatori

- Relacioni: < > <= >= == !=

- Logički: && (I), || (ILI), ! (NE)

- Short-circuiting:

```
if(test1() && test2() && test3())
```

- Ako npr. `test2()` bude false, `test3()` se neće ni pozvati!
- Ako je operacija ILI (`||`) i prvi izraz je true, drugi se ni ne računa, a ukupan izraz je true.

Relacioni operatori

`x = 5;`

Operator	Rezultat
<code>==</code>	<code>x == 8</code> je netačno (false)
<code>!=</code>	<code>x != 8</code> je tačno (true)
<code>></code>	<code>x > 8</code> je netačno (false)
<code><</code>	<code>x < 8</code> je tačno (true)
<code>>=</code>	<code>x >= 8</code> je netačno (false)
<code><=</code>	<code>x <= 8</code> je tačno (true)

Logički operatori

- Logički: `&&` `||` `!`
- Rezultat logičkih operatora je tačno (true) ili netačno (false)
- Operandi logičkih operatora su logički izrazi

<code>&&</code>	false	true
false	false	false
true	false	true

<code> </code>	false	true
false	false	true
true	true	true

<code>!</code>	
false	true
true	false

Logički operatori

x = 6;

y = 3;

Operator	Objašnjenje	Primer
&&	konjukcija (and, i)	(x < 10 && y > 1) tačno (true)
	disjunkcija (or, ili)	(x==5 y==5) netačno (false)
!	negacija (not, ne)	!(x==y) tačno (true)

Bit operatori

- Logičko I nad bitovima: &
- Logičko ILI nad bitovima: |
- Ekskluzivno ILI (XOR) nad bitovima: ^
- Logička negacija nad bitovima -unarni operator: ~
- Kombinacija sa =:
&= |= ^=

Bit operatori

- `a = 3; // a=011 binarno`
- `b = 6; // b=110 binarno`
- `c = a & b;`

```
  011
& 110
-----
```

010

- `c ← 2;`

Bit operatori

- Shift-ovanje (pomeranje):

$a \gg b$ – pomera bitove u a za b mesta

- ako je a pozitivan, ubacuje 0
- ako je a negativan, ubacuje 1

$a \ll b$ – pomera bitove u levo i ubacuje 0

$a \ggg b$ – pomera bitove u a u desno za b mesta i ubacuje 0 bez obzira na znak a .

- Rezultat pomeranja je 32-bitan, osim ako promenljiva koja prihvata rezultat nije long (tada je 64-bitan)!

Pomeranje

- `a = 3; // a = 011 binarno`
- `b = a<<2; // b = 01100 binarno`

- `a = 7; // a = 111 binarno`
- `b = a>>2; // b = 001 binarno`

Operator dodele

- Ako su operandi primitivni tipovi, kopira se sadržaj:

```
int i = 3, j = 6;
```

```
i = j; // u i ubačeno 6
```

- Ako su operandi reference, kopira se sadržaj reference, a ne kompletni objekti na koje ukazuju!

Kontrola toka

- `if else`
- `switch`
- `for`
- `while`
- `do while`
- `break`
- `continue`

if else

```
int result = 0;  
if(testval > target)  
    result = -1;  
else if(testval < target)  
    result = +1;  
else  
    result = 0; // match
```

Uslovni operator

```
a = i < 10 ? i * 100 : i * 10;
```

- isto kao:

```
if (i < 10)
    a = i * 100;
else
    a = i * 10;
```

switch

- Izraz u switch() izrazu mora da proizvede celobrojnu vrednost.
- Ako ne proizvodi celobrojnu vrednost, ne može da se koristi switch,() već if()!
- Ako se izostavi break; propašće u sledeći case:
- Kod default: izraza ne mora break; - to se podrazumeva.

switch

```
switch(c) {  
    case 'a':  
    case 'e':  
    case 'i':  
    case 'o':  
    case 'u':  
        System.out.println("samoglasnik");  
        break;  
default:  
    System.out.println("suglasnik");  
}
```

for

- Za organizaciju petlji kod kojih se unapred zna koliko puta će se izvršiti telo ciklusa.
- Petlja sa početnom vrednošću, uslovom za kraj i blokom za korekciju.

- Opšta sintaksa:

```
for (inicijalizacija; uslov; korekcija)  
    telo
```

for

```
for (int i = 0; i < 10; i++)
```

```
    System.out.println(i) ;
```

- može i višestruka inicijalizacija i step-statement:

```
for(int i = 0, j = 1;
```

```
i < 10 && j != 11; i++, j++)
```

- oprez (može da se ne završi):

```
for (double x = 0; x != 10; x+=0.1) ...
```

while

- Za cikličnu strukturu kod koje se samo zna uslov za prekid.
- Telo ciklusa ne mora ni jednom da se izvrši
- Opšta sintaksa:
`while (uslov)`
 `telo`
- Važno: izlaz iz petlje na false!

while

```
int i = 0;
while (i <= 10)
{
    System.out.println("Trenutno
je " + i);
    i=i+1;
}
```

- Važno: izlaz iz petlje na false!

do while

- Za cikličnu strukturu kod koje se samo zna uslov za prekid
- Razlika u odnosu na while petlju je u tome što se telo ciklusa izvršava makar jednom.
- Opšta sintaksa:

`do`

`telo`

`while (uslov) ;`

- Važno: izlaz iz petlje na false!

do while

```
int i = 0;  
do {  
    System.out.println(i++);  
} while (i < 10);
```

- Važno: izlaz iz petlje na false!

break i continue

- break – prekida telo tekuće ciklične strukture (ili case: dela) i izlazi iz nje.
- continue – prekida telo tekuce ciklične strukture i otpočinje sledeću iteraciju petlje.

break i continue

```
for(int i = 0; i < 10; i++) {  
    if (i==7) {  
        break;  
    }  
    if (i == 2)  
        continue;  
  
    System.out.println("Broj je:" + i);  
}
```

Izlaz iz ugnježdene petlje

```
for (...)
{
    for (...)
    {
        ...
        if (uslov)
            break;
    }
}
```

Programski jezik Java

Objektno orijentisano programiranje

Osnovni koncepti objektnog programiranja – klase i objekti

- klasa: model objekta
 - uključuje:
 - attribute
 - metode
- objekat: instanca klase

Primer klase

Automobil
+ radi : boolean
+ upali () : void
+ ugasi () : void

```
class Automobil {  
    boolean radi;  
    void upali() {  
        radi = true;  
    }  
    void ugasi() {  
        radi = false;  
    }  
}
```


Kako se instancira i koristi?

Automobil
- radi : boolean
+ upali () : void
+ ugasi () : void

```
...  
Automobil a = new Automobil();  
Automobil b = new Automobil();  
...  
a.upali();  
b.ugasi();
```

Sve je objekat

- nije moguće definisati funkcije i promenljive izvan neke klase
- deklaracija klase ne postoji, već samo njena definicija

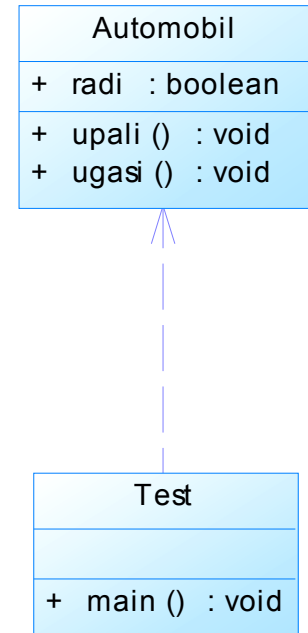
Program sa dve klase

Automobil.java

```
class Automobil {  
    boolean radi;  
    void upali() { radi = true; }  
    void ugasi() { radi = false; }  
}
```

Test.java

```
class Test {  
    public static void main(String args[]) {  
        Automobil a;  
        a = new Automobil();  
        a.upali();  
    }  
}
```



Veza dve klase tipa zavisnosti (UML i Java)

- Veza tipa zavisnosti ako je druga klasa:
 - parametar metode
 - povratna vrednost
 - lokalna promenljiva

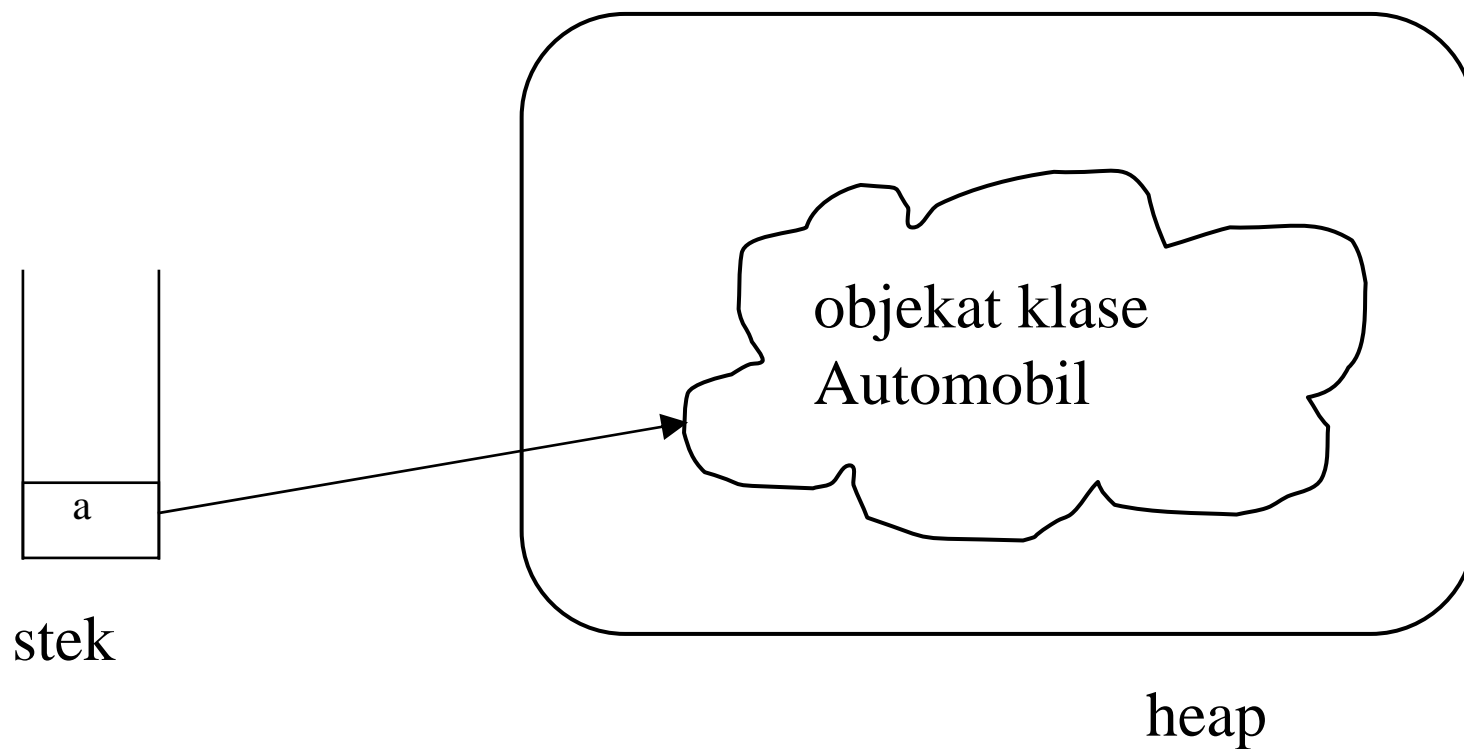
Reference na objekte

```
Automobil a;
```

```
a = new Automobil();
```

lokalna promenljiva **a** nije objekat, već
referenca na objekat

Reference na objekte



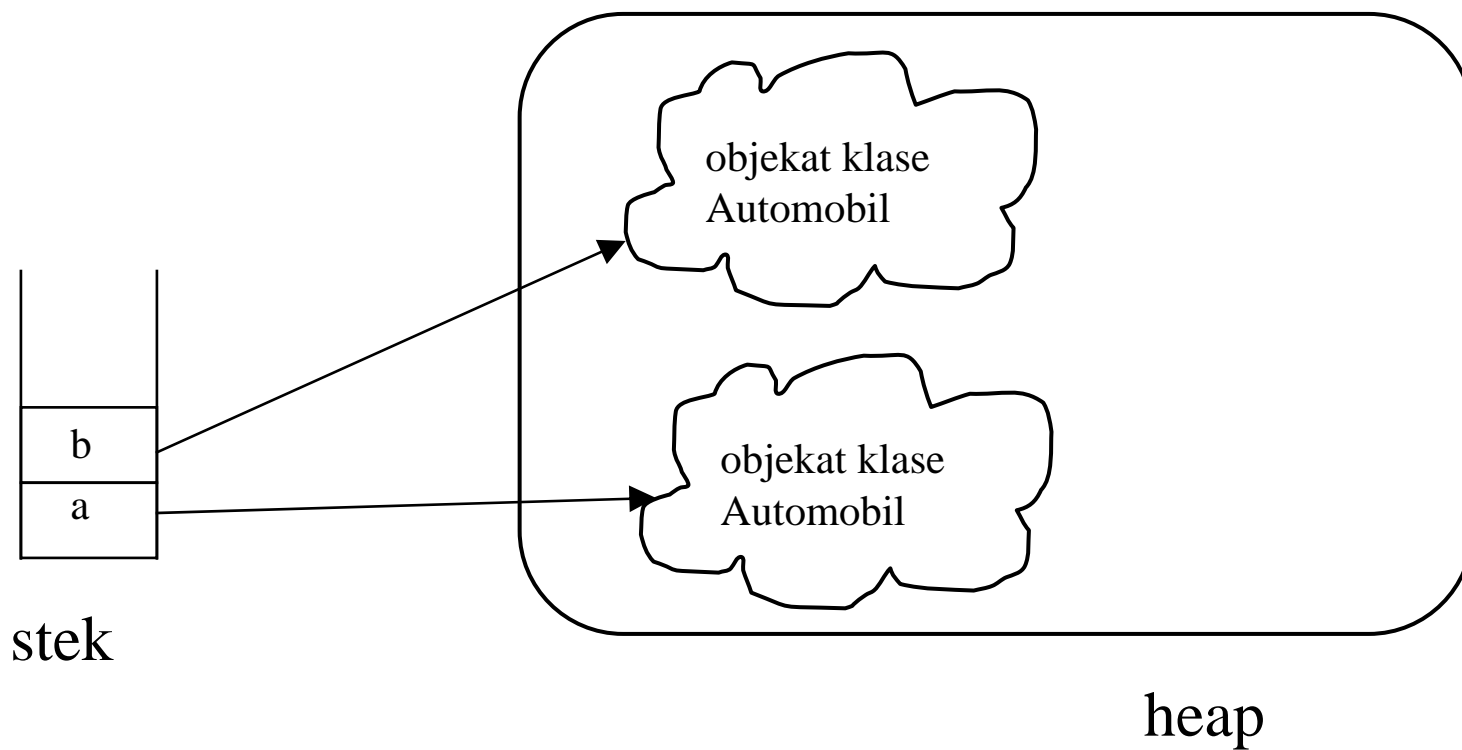
Operator dodele vrednosti

```
Automobil a = new Automobil();  
Automobil b = new Automobil();  
b = a;
```



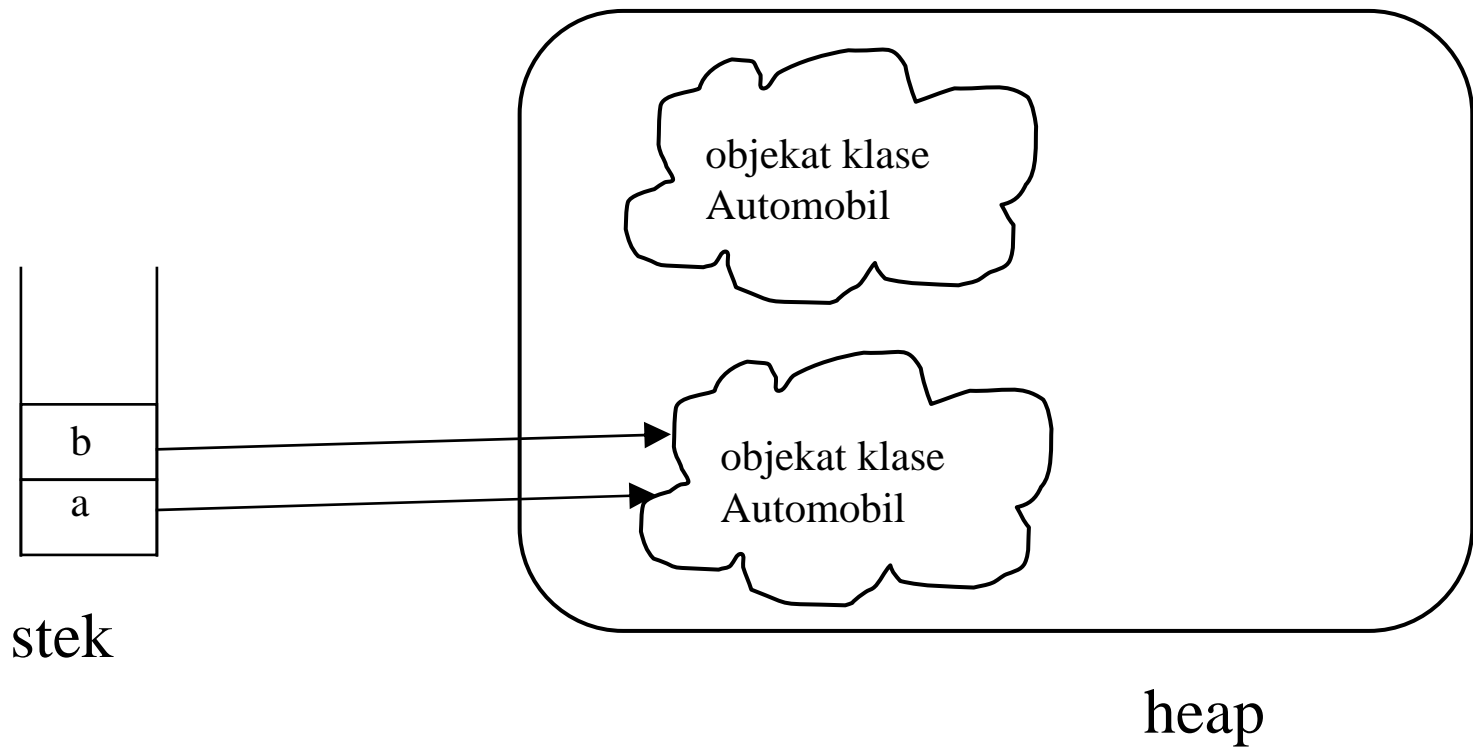
Vrši se kopiranje
vrednosti reference!

Reference na objekte



Reference na objekte

$b = a$



Default vrednosti promenljivih

- primitivni tipovi kao atributi klase

Primitivni tip	Default
boolean	false
char	'\u0000'
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

- reference kao atributi klase
 - null
- lokalne promenljive
 - nemaju default vrednost – upotreba pre inicijalizacije izaziva grešku kod kompajliranja!

Inicijalizacija objekata

- Ako želimo posebnu akciju prilikom kreiranja objekta neke klase, napravićemo konstruktor
- Konstruktor se automatski poziva prilikom kreiranja objekta

Automobil a = new Automobil();



- Ako ne napravimo konstruktor, kompajler će sam napraviti default konstruktor, koji ništa ne radi

Inicijalizacija objekata

- konstruktor

```
class A {  
    A() {  
        System.out.println("konstruktor");  
    }  
}  
...  
A varA = new A();
```

na konzoli će pisati:

konstruktor

Inicijalizacija objekata

- Može da se napravi i konstruktor sa parametrima

```
class A {  
    A(String s) {  
        System.out.println(s);  
    }  
}  
...  
A varA = new A("blabla");
```

na konzoli će pisati:

blabla

Uništavanje objekata – Garbage collector

- radi kao poseban proces u pozadini
- automatska dealokacija memorije
- automatska defragmentacija memorije

Garbage collector

- ne postoji destruktor
- posebna metoda `finalize()` se poziva neposredno pre oslobađanja memorije koju je objekat zauzimao

Parametri i rezultat metoda

- parametri mogu biti:
 - primitivni tipovi
 - reference na objekte
- rezultat može biti:
 - primitivni tip
 - referenca na objekat
- Metoda vraća vrednost naredbom:
return vrednost
ili
return (vrednost)

Prenos parametra po vrednosti

```
void test(int i) {  
    i = 0;  
}
```

...

```
int b = 5;  
test(b);
```

Referenca na objekat kao parametar metode

```
void test(Automobil a) {  
    a.radi = true;  
}
```

...

```
Automobil x = new Automobil();  
x.radi = false;  
test(x);
```

Varijabilan broj parametara

- Ako želimo da metoda ima varijabilan broj parametara, deklarišemo na sledeći način:

```
void f(int... params) {...}
```

- Parametrima se pristupa kao elementima niza:

```
int a = params[2];
```

Method overloading

- U klasi može da postoji više metoda sa istim imenom
- Razlikuju se po parametrima
- Metode se nikada ne razlikuju po povratnoj vrednosti

Method overloading

```
class A {  
    int metoda() { ... }  
    int metoda(int i) { ... }  
    int metoda(String s) { ... }  
}
```

- metode se nikada ne razlikuju po povratnoj vrednosti!

Ključna reč `final`

- `final` atributi: konstante

```
final int a = 5;
```

- `final` metode: ne mogu se redefinisati prilikom nasleđivanja

Ključna reč `static`

- Definiše statičke attribute i metode
- Statički atributi i metode postoje i bez kreiranje objekta
 - zato im se može pristupiti preko imena klase
 - `StaticTest.i++`;
- Statički atributi imaju istu vrednost u svim objektima
 - ako promenim statički atribut u jednom objektu, on će se promeniti i kod svih ostalih objekata
- Namena statičkih metoda:
 - pristup i rad sa statičkim atributima
 - opšte metode za koje nije potrebno da se kreira objekat
 - `Math.sin(x)`

Ključna reč `static`

```
class StaticTest {  
    static int i = 47;  
    static void metoda() { i++; }  
}
```

...

```
StaticTest st1 = new StaticTest();  
StaticTest st2 = new StaticTest();
```

...

```
st1.i++;                // isto što i st2.i++;  
StaticTest.i++;  
StaticTest.metoda();
```


Ključna reč `static`

- `System.out.println();`
- `Math.random();`

Nizovi

```
int a[]; // još uvek nije napravljen niz!
```

```
a = new int[5];
```

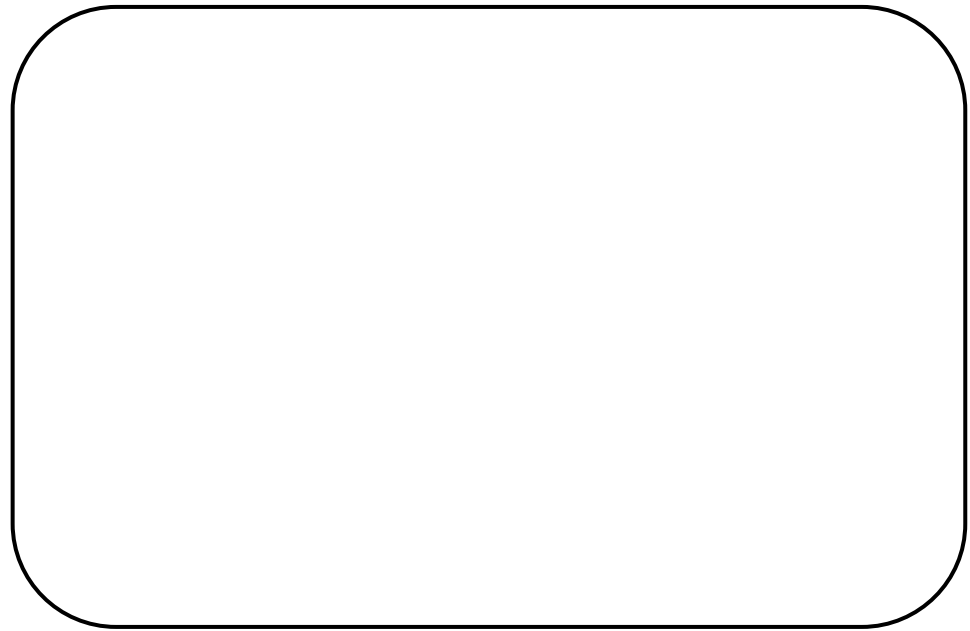
```
int a[] = { 1, 2, 3, 4, 5 };
```

```
Automobil[] parking = new Automobil[20];  
for(int i = 0; i < parking.length; i++)  
    parking[i] = new Automobil();
```

Nizovi primitivnih tipova ^{1/3}



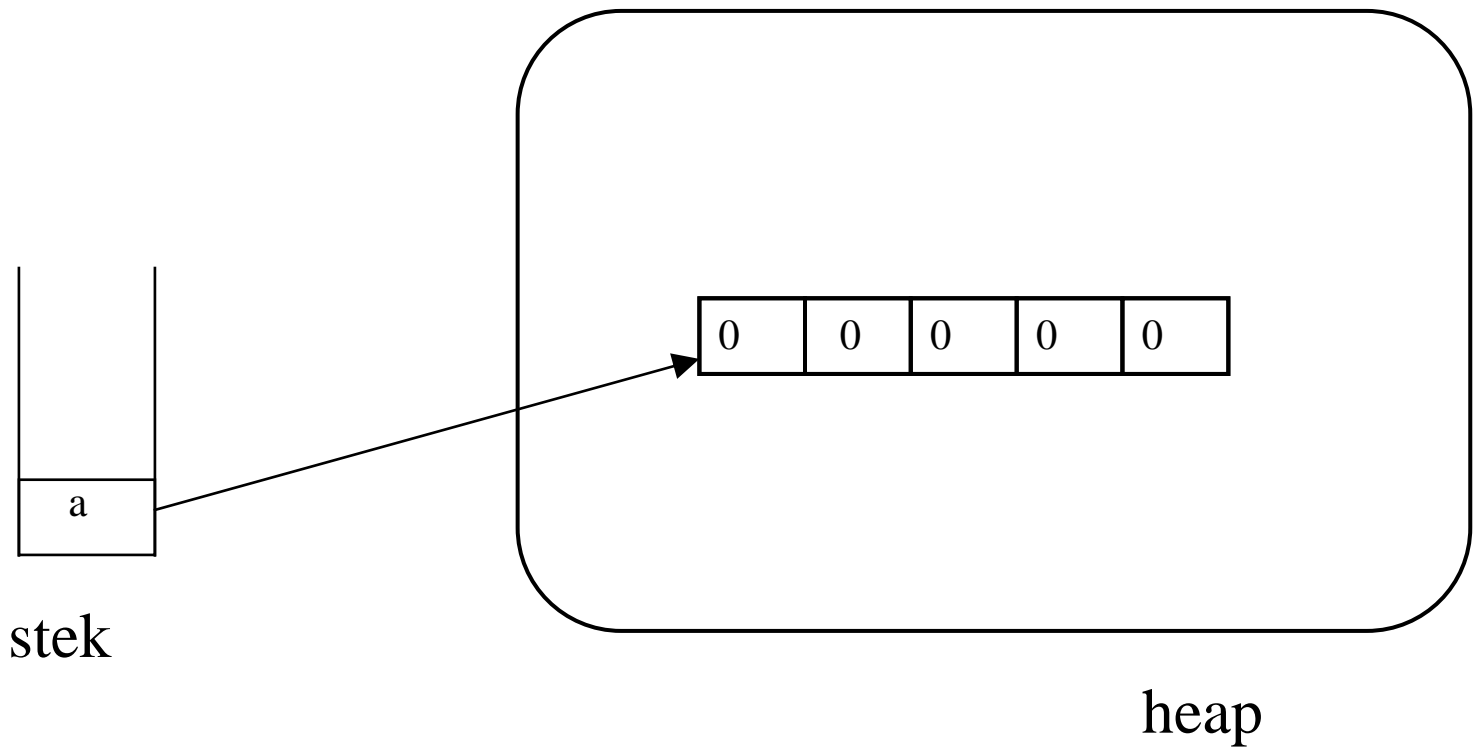
stek



heap

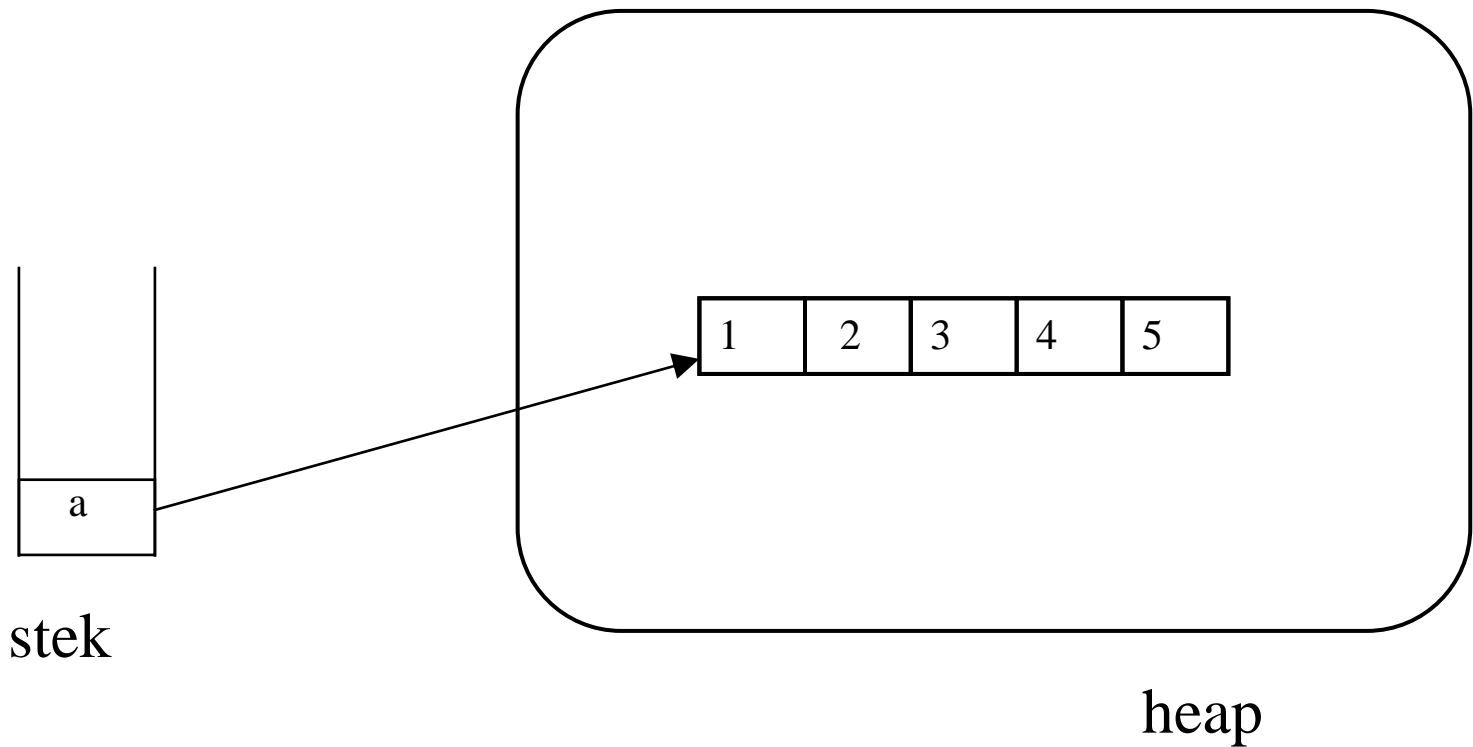
```
int a[];
```

Nizovi primitivnih tipova ^{2/3}



```
a = new int[5];
```

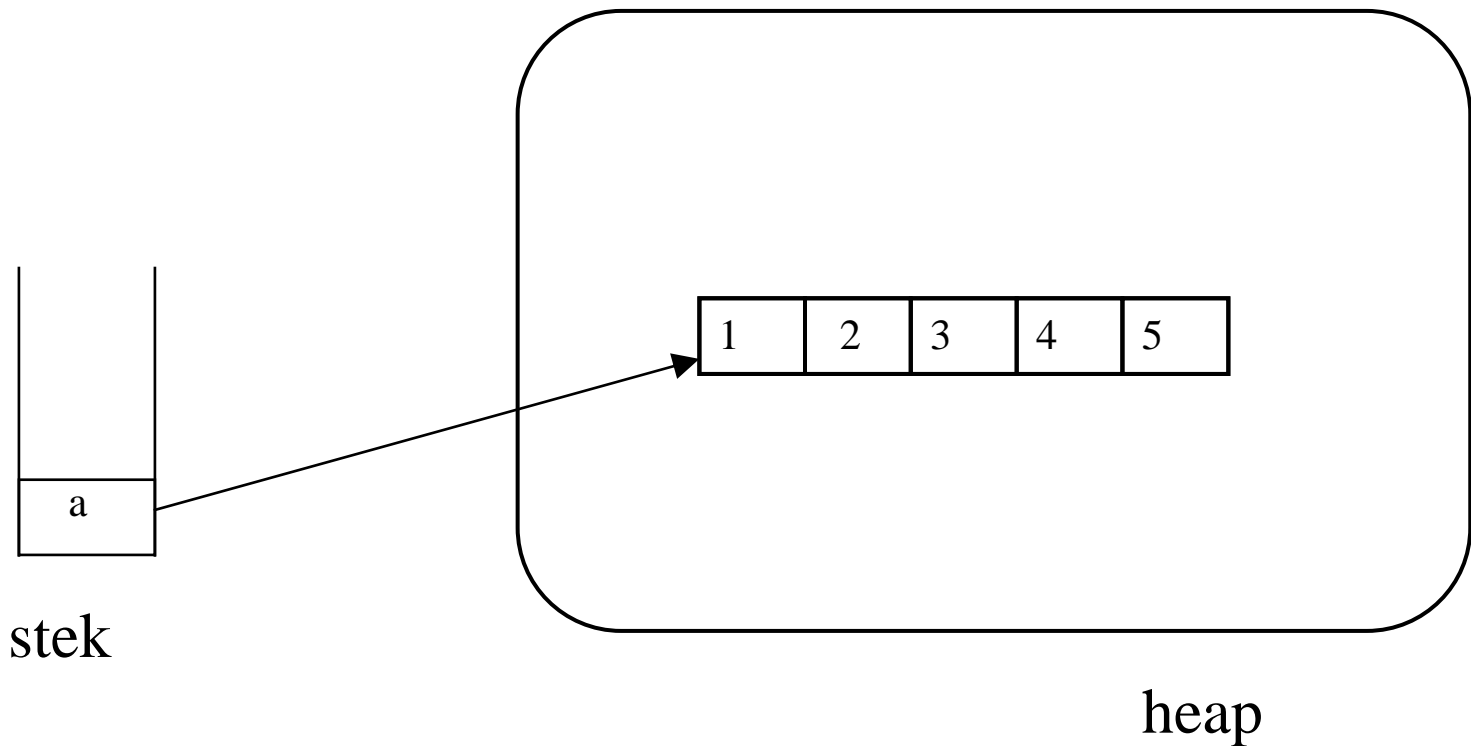
Nizovi primitivnih tipova ^{3/3}



```
a[0]=1; a[1]=2; a[2]=3; a[3]=4; a[4]=5;
```

Nizovi primitivnih tipova

jednim potezom

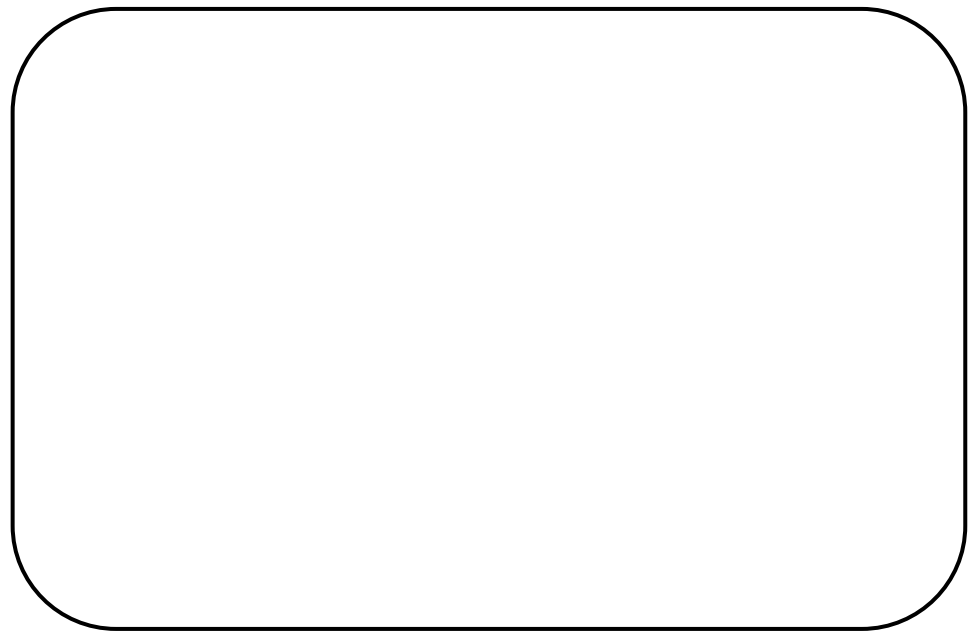


```
int a[] = { 1, 2, 3, 4, 5 };
```

Nizovi referenci na objekte ^{1/3}



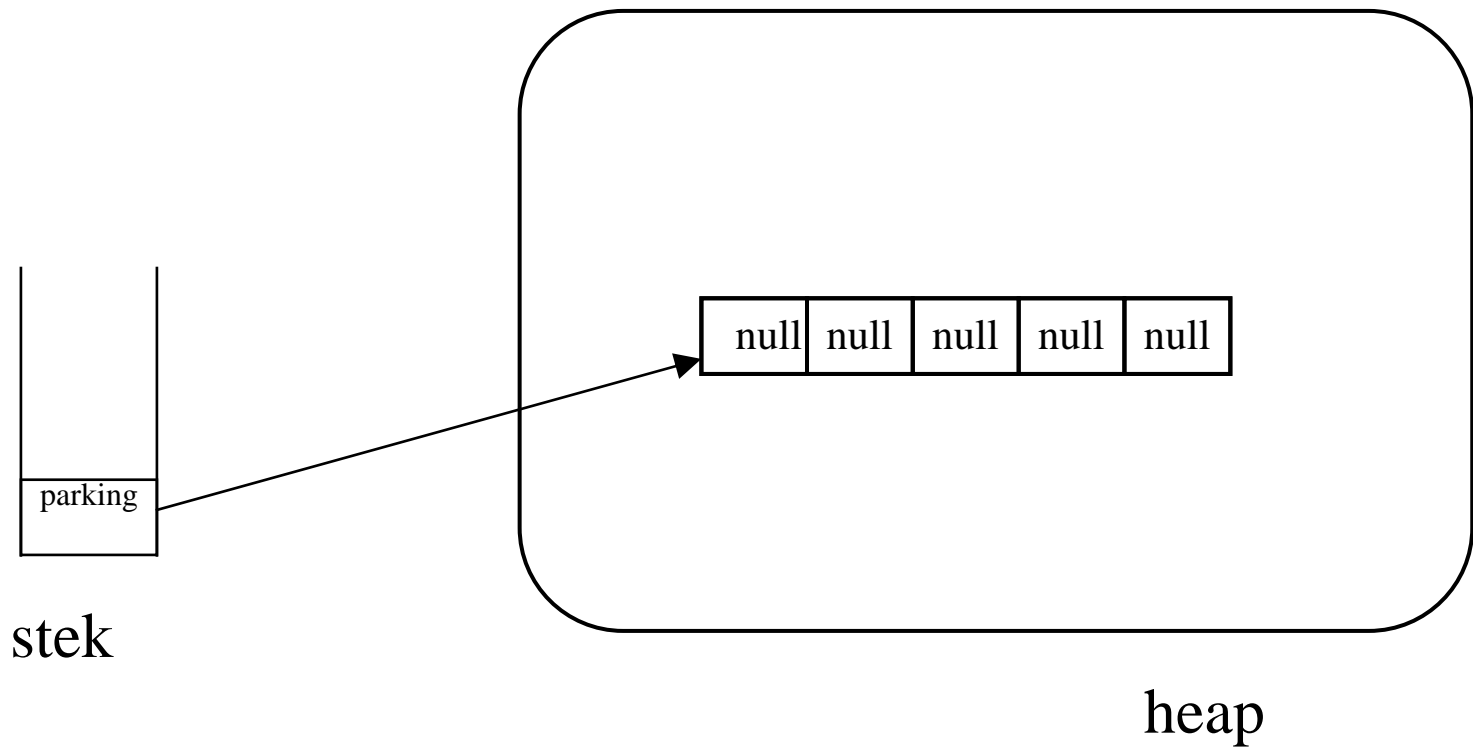
stek



heap

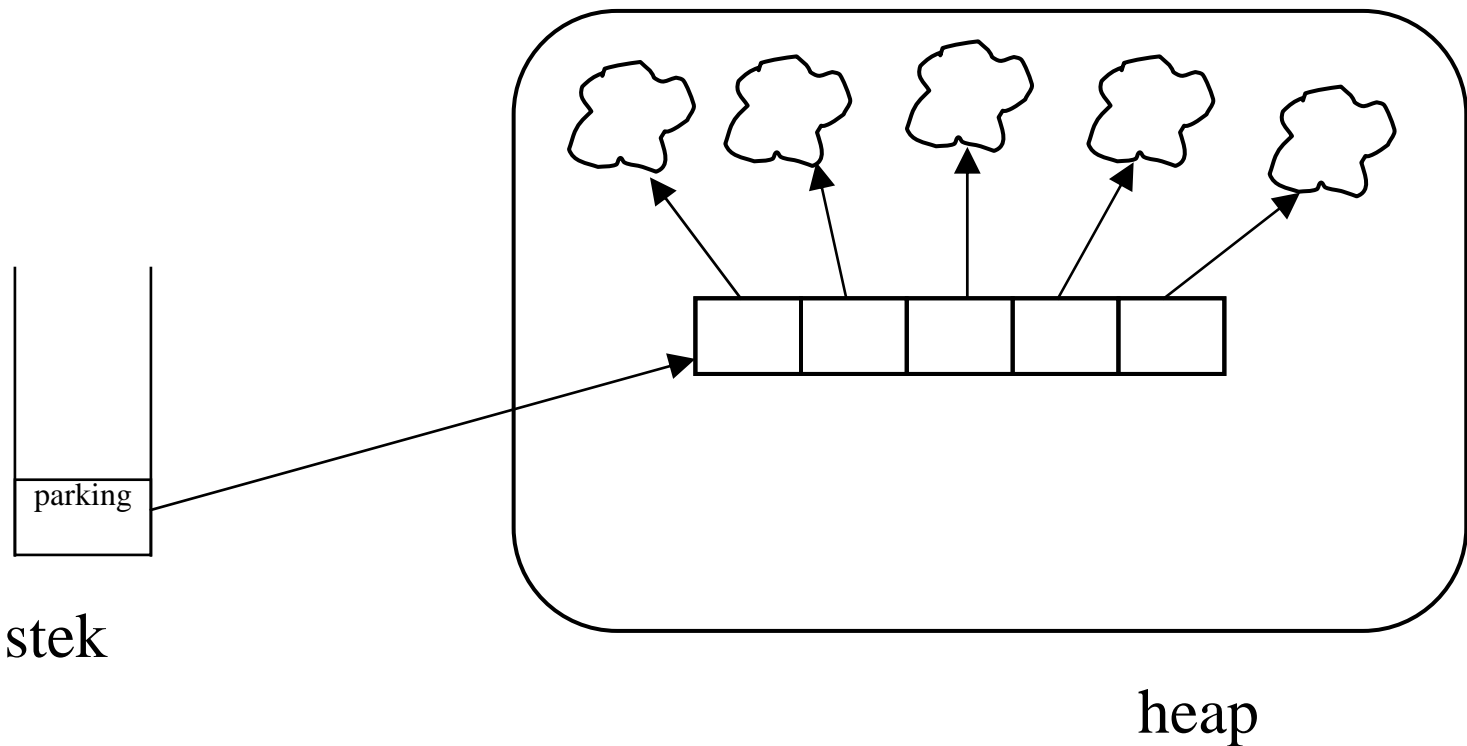
```
Automobil[] parking;
```

Nizovi referenci na objekte ^{2/3}



```
parking = new Automobil[5];
```


Nizovi referenci na objekte 3/3



```
for(int i = 0; i < parking.length; i++)  
    parking[i] = new Automobil();
```

Višedimenzijski nizovi

```
int a[][] = { {1, 2, 3 }, {4, 5, 6 } };
```

```
int a[][] = new int[2][3];
```

```
int a[][] = new int[2][];  
for(int i = 0; i < a.length; i++) {  
    a[i] = new int[3];  
}
```

```
Automobil[][] a = {  
    { new Automobil(), new Automobil() },  
    { new Automobil(), new Automobil() }  
};
```

Kolekcije

- Nizovi imaju jednu manu – kada se jednom naprave nije moguće promeniti veličinu.
- Kolekcije rešavaju taj problem.
- Zajedničke metode:
 - dodavanje elemenata,
 - uklanjanje elemenata,
 - iteriranje kroz kolekciju elemenata

Kolekcije

Implementacija Koncept	Hash table	Resizable Array	Balanced Tree	Linked List	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Tipizirane kolekcije - Generics

- U dosadašnjim kolekcijama mogli su da se smeste bilo koji objekti.
- Mana: prilikom pristupa elementu iz kolekcije, obavezno se kastuje u konkretan tip:

```
ArrayList kolekcija = new ArrayList();  
kolekcija.add("tekst");  
String s = (String)kolekcija.get(0);
```

Potencijalan problem
prilikom pogrešnog
kastovanja

- Tipizirane kolekcije omogućavaju smeštaj samo jednog tipa podatka u kolekciju.
- Primer:

```
ArrayList<String> kolekcija = new ArrayList<String>();  
kolekcija.add("tekst");  
String s = kolekcija.get(0);
```

for bez indeksiranja (poznat i kao *foreach*)

- Omogućuje prolazak kroz niz ili kolekciju.

- Opšta sintaksa:

```
for (varijabla : niz) {  
    ... // telo  
}
```

- Primer:

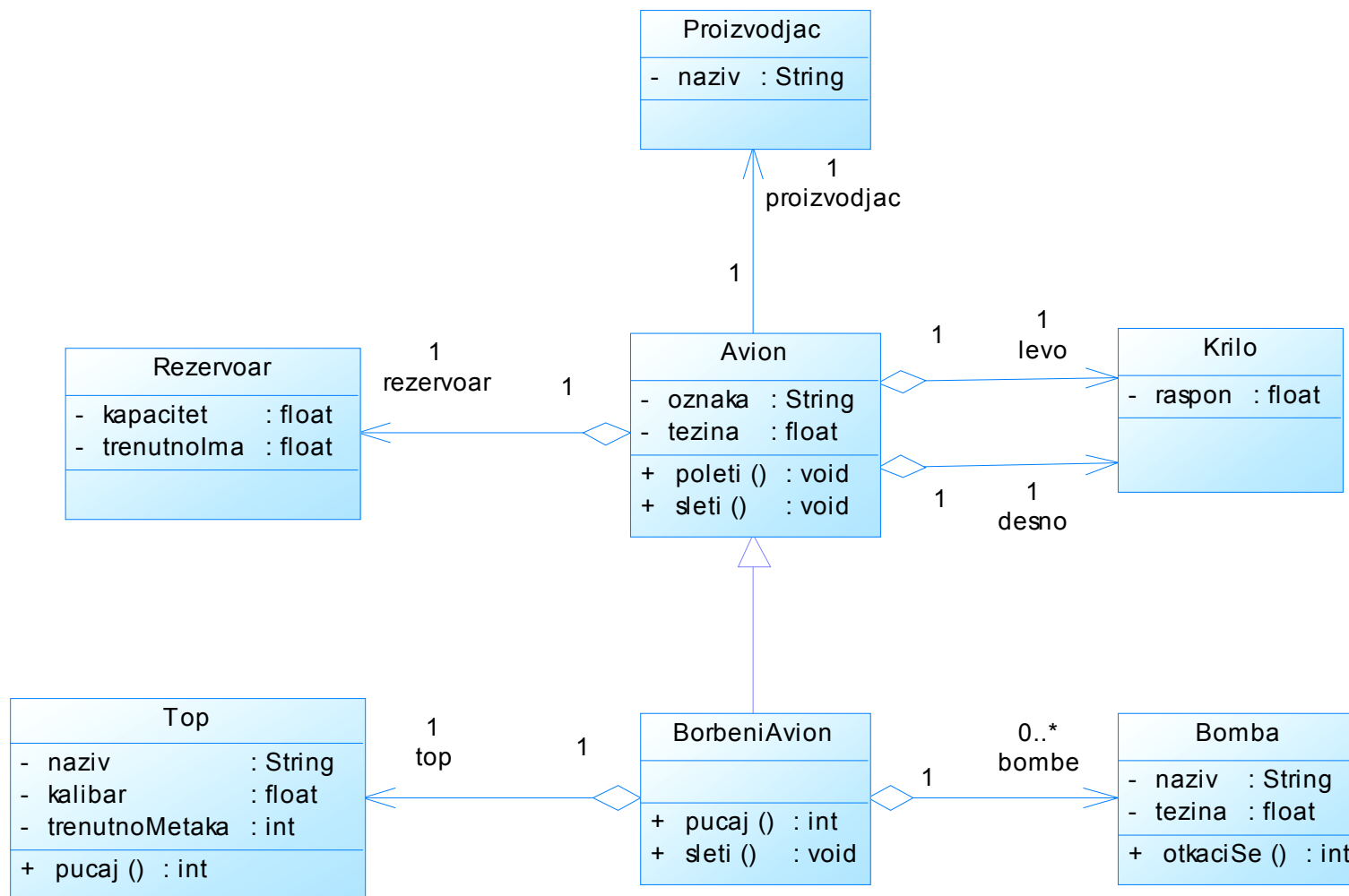
```
for (int i : niz) {  
    System.out.println(i);  
}
```

```
for (Auto a : kolekcija) {  
    System.out.println(a.radi);  
}
```

Programski jezik Java

Objektno orijentisano programiranje

Nasleđivanje



Nasleđivanje

```
class Avion {  
    String oznaka;  
    float tezina;  
    Rezervoar rezervoar;  
    Krilo levo, desno;  
    Proizvodjac proizvodjac;  
    void poleti() { ... }  
    void sleti() { ... }  
}
```

```
class BorbeniAvion extends Avion {  
    Top top;  
    Collection<Bomba> bombe;  
    void sleti() { ... }  
    int pucaj() { ... }  
}
```

- postoji samo jednostruko nasleđivanje

Veze tipa asocijacije i agregacije (UML i Java)

- Veza tipa asocijacije je za attribute koji nisu isključivi deo glavne klase (klasa Proizvođač je u vezi tipa asocijacije sa klasom Avion), već mogu da postoje i nezavisno od glavne klase
- Veza tipa agregacije je za attribute koji su deo celine (Rezervoar \leftarrow Avion, Krilo \leftarrow Avion) i nema smisla da postoje nezavisno od glavne klase
- Kardinalnost veze određuje da li će atribut biti promenljiva ili kolekcija (niz, ArrayLista i sl.)

Modifikatori pristupa

- **public** – vidljiv za sve klase
- **protected** – vidljiv samo za klase naslednice i klase iz istog paketa
- **private** – vidljiv samo unutar svoje klase
- nespecificiran (*friendly*) – vidljiv samo za klase iz istog paketa

Method overriding

- Pojava da u klasi naslednici postoji metoda istog imena i parametara kao i u baznoj klasi
- Primer:
 - klasa A ima metodu **metoda1()**
 - klasa B nasleđuje klasu A i takođe ima metodu **metoda1()**

Method overriding

```
class A {
    int metoda1() {
        System.out.println("metoda1 klase A");
    }
    int metoda2() {
        System.out.println("metoda2 klase A");
    }
}
class B extends A {
    int metoda1() {
        System.out.println("metoda1 klase B");
    }
}
...
A varA = new A();
B varB = new B();
varA.metoda1();
varB.metoda1();
varA.metoda2();
varB.metoda2();
```

Method overriding

- na konzoli će pisati

```
metoda1 klase A
```

```
metoda1 klase B
```

```
metoda2 klase A
```

```
metoda2 klase A
```

Apstraktne klase

- klase koje ne mogu imati svoje objekte, već samo njene klase naslednice mogu da imaju objekte (ako i one nisu apstraktne)

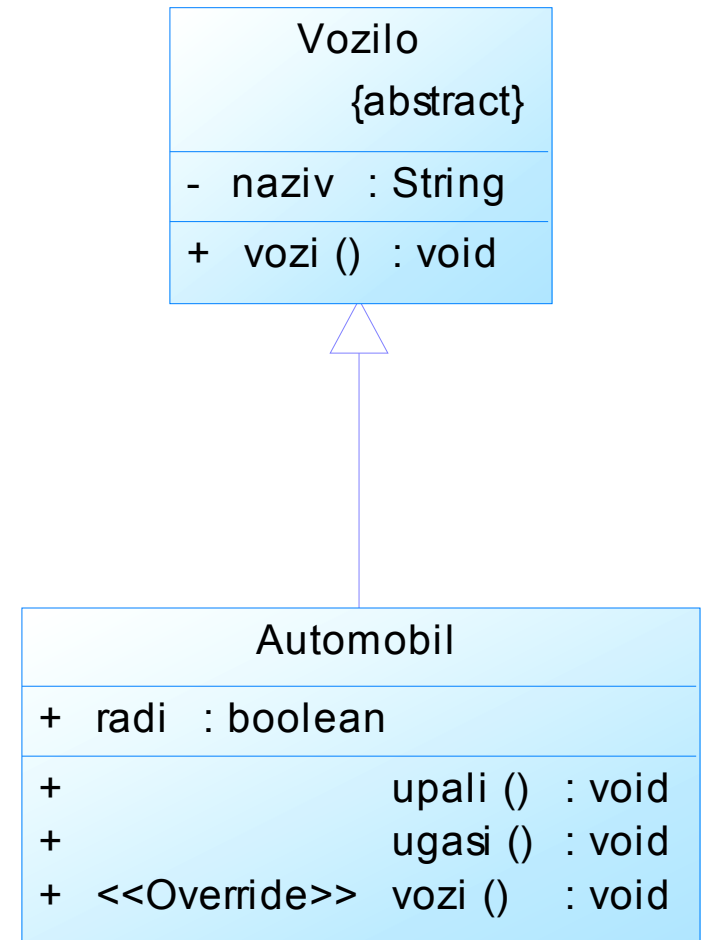
```
abstract class A {  
    int i;  
    public void metoda1() { ... }  
    public abstract void metoda2();  
    ...  
}
```

```
class B extends A {  
    public void metoda2() { ... }  
}
```

- Ako klasa ima makar jednu apstraktnu metodu, mora da se deklarise kao apstraktna.
- Apstraktna klasa ne mora da ima apstraktne metode!

Apstraktne klase

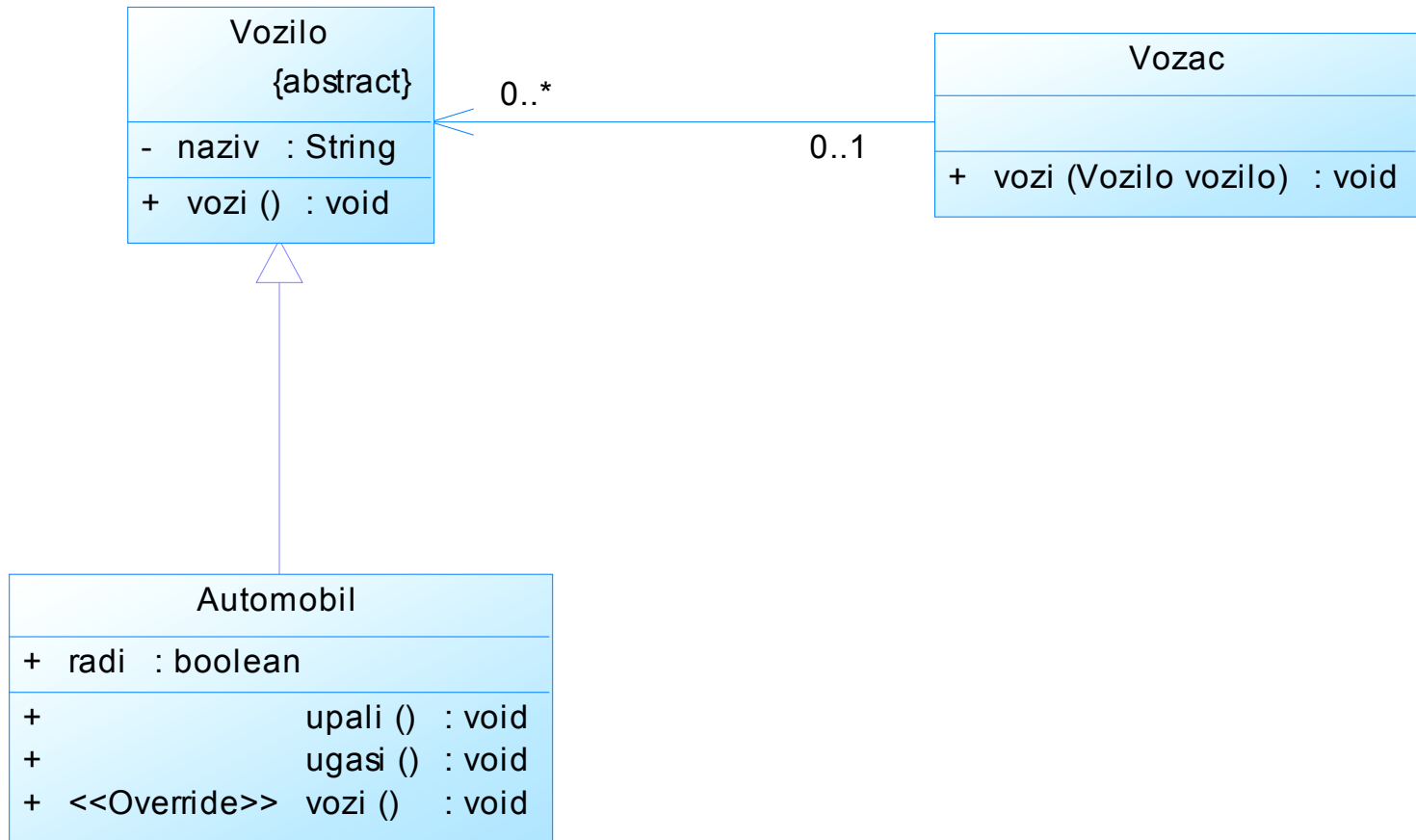
```
public abstract class Vozilo {  
    private String naziv;  
    public abstract void vozi();  
}  
  
public class Automobil extends Vozilo {  
    public boolean radi;  
    public void upali() {  
        radi = true;  
    }  
    public void ugasi() {  
        radi = false;  
    }  
    public void vozi() {  
        ...  
    }  
}
```



Polimorfizam

- Situacija kada se poziva metoda nekog objekta, a ne zna se unapred kakav je to konkretan objekat
 - ono što se zna je koja mu je bazna klasa
- Tada je moguće u programu pozivati metode bazne klase, a da se zapravo pozivaju metode konkretne klase koja nasleđuje baznu klasu

Polimorfizam



Polimorfizam

```
abstract class Vozilo {  
    abstract void vozi();  
}  
class Auto extends Vozilo {  
    void vozi() { ... }  
}  
class Vozac {  
    void vozi(Vozilo v) {  
        v.vozi();  
    }  
}  
...  
Vozac v = new Vozac();  
v.vozi(new Automobil());
```

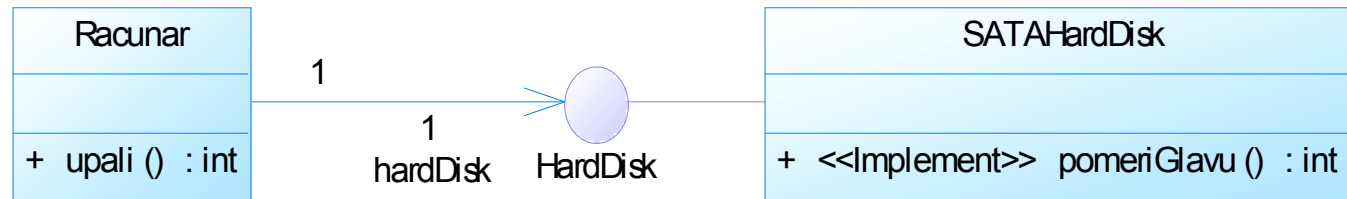


sve metode su
virtuelne!

Interfejsi

- Omogućavaju definisanje samo apstraktnih metoda, konstanti i statičkih atributa
- Interfejs nije klasa! On je spisak metoda i atributa koje klasa koja implementira interfejs mora da poseduje.
- Sve metode su implicitno public, a svi atributi su implicitno public static final.
- Interfejsi se ne nasleđuju, već implementiraju
- Da bi klasa implementirala interfejs, mora da redefiniše sve njegove metode
- Jedan interfejs može da nasledi drugog
- **Jedna klasa može da implementira jedan ili više interfejsa**

Interfejsi

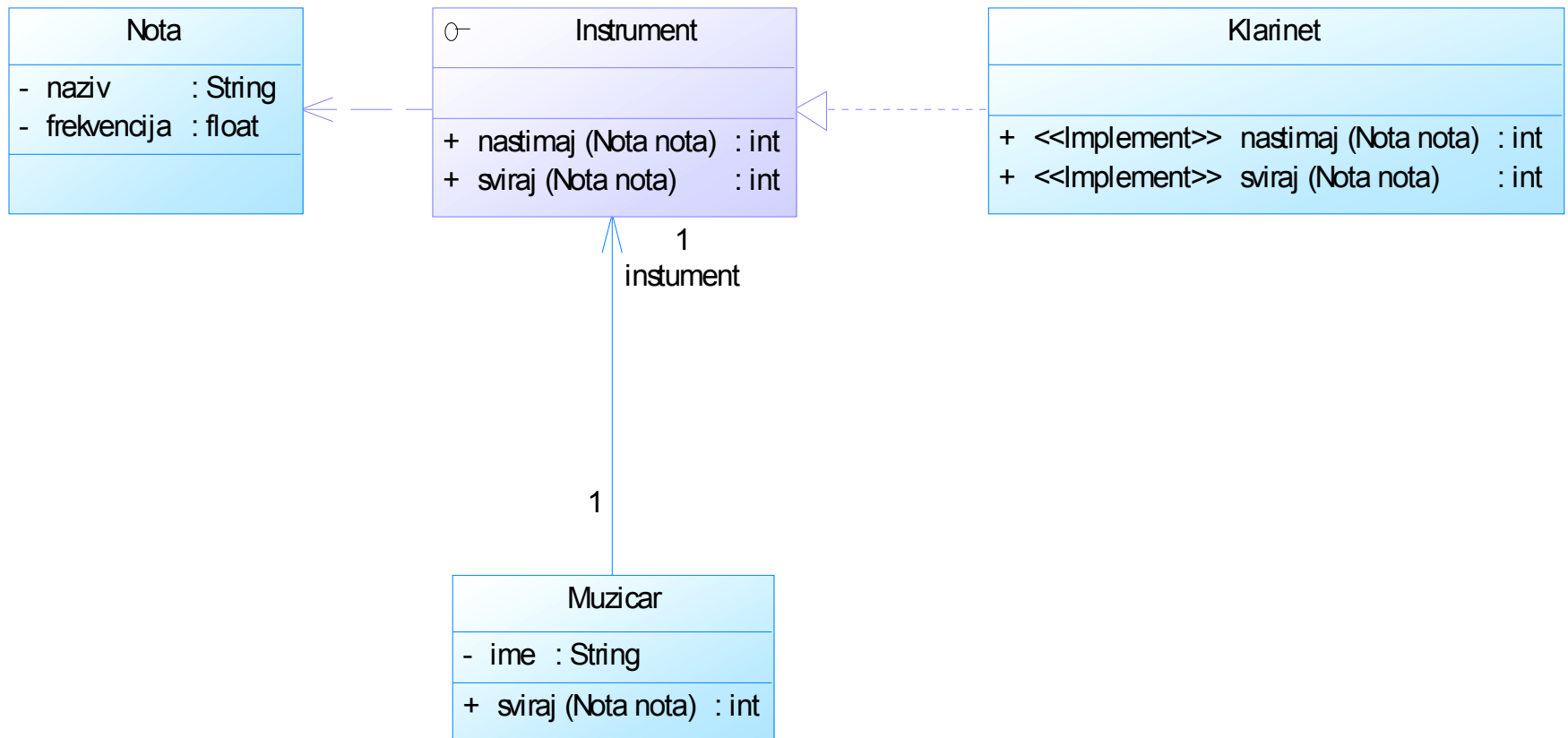


```
public class Racunar {
    public HardDisk hardDisk;
    public int upali() {
    }
}

public interface HardDisk {
    int pomeriGlavu();
}

public class SATAHardDisk implements HardDisk {
    public int pomeriGlavu() {
        ...
    }
}
```

Interfejsi



Interfejsi

```
interface Instrument {
    int sviraj(Nota nota);
    int nastimaj(Nota nota);
}

class Klarinet implements Instrument {
    public int sviraj(Nota nota) { ... }
    public int nastimaj(Nota nota) { ... }
}

class Muzicar {
    Instrument instrument;
    int sviraj(Nota nota) {
        return instrument.sviraj(nota);
    }
}

...

Muzicar m = new Muzicar();
m.instrument = new Klarinet();
m.sviraj(nota);
```

Izuzeci

- Mehanizam prijavljivanja greške
- Greška se signalizira "bacanjem" izuzetka
- Metoda koja poziva potencijalno "grešnu" metodu "hvata" izuzetak

Izuzeci

- dve vrste izuzetaka:
 - checked (Exception) – moraju da se uhvate
 - EOFException
 - SQLException
 - ...
 - unchecked (RuntimeException) – ne moraju da se uhvate
 - NullPointerException
 - ArrayIndexOutOfBoundsException
 - ArithmeticException
 - ...

Izuzeci

```
try {  
    // kod koji može da izazove  
    // izuzetak  
}  
catch (java.io.EOFException ex) {  
    System.out.println("Kraj datoteke pre vremena!");  
}  
catch (ArrayIndexOutOfBoundsException ex) {  
    System.out.println("Pristup van granica niza");  
}  
catch (Exception ex) {  
    System.out.println("Svi ostali izuzeci");  
}  
finally {  
    // kod koji se izvršava u svakom slučaju  
}
```

Izuzeci

- programsko izazivanje izuzetka

```
throw new Exception("Ovo je jedan izuzetak");
```

- korisnički definisani izuzeci

```
class MojException extends Exception {  
    MojException(String s) {  
        super(s);  
    }  
}
```

Izuzeci

- ključna reč **throws**

```
void f(int i) throws MojException { ... }
```

- propagacija izuzetaka

Wrapper klase i autoboxing

- Za sve primitivne tipove postoje odgovarajuće klase:
 - `int` → `Integer`
 - `long` → `Long`
 - `boolean` → `Boolean`
- Imaju korisnu statičku metodu `Xxxx.parseXxxx()`
 - `Integer.parseInt("10")`
 - `Long.parseLong("10")`
- autoboxing/unboxing:
 - ako metoda prima `Integer` kao parametar, može da se prosledi i `int`, odn. promenljivoj tipa `Integer` može da se dodeli vrednost promenljive tipa `int`
 - radi i u obrnutom pravcu – promenljivoj tipa `int` može da se dodeli vrednost promenljive tipa `Integer`

Paketi

- način za hijerarhijsko organizovanje programa u module
- implicitni paket
- upotreba

```
import java.io.*;
```

```
import java.util.ArrayList;
```

Paketi

- kreiranje paketa

```
package imePaketa;  
...  
public class MojaKlasa { ... }
```

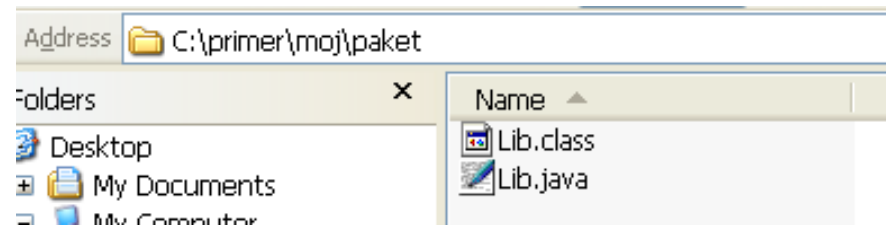
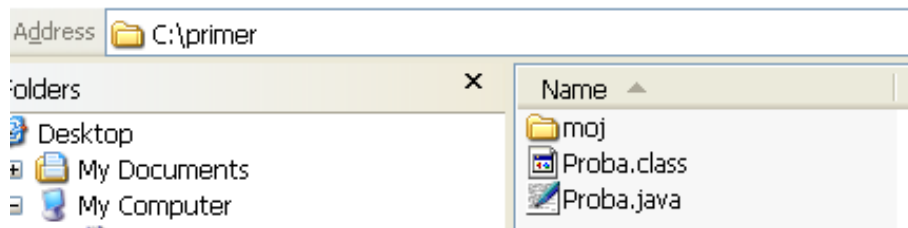
- korišćenje paketa

```
import imePaketa.MojaKlasa;  
...  
MojaKlasa m = new MojaKlasa();
```

```
imePaketa.MojaKlasa m = new imePaketa.MojaKlasa();
```

Paketi

- direktorijumi
 - hijerarhija paketa se poklapa sa hijerarhijom direktorijuma:
 - `moj.paket.Lib` -> `moj\paket\Lib.class`



Paketi

- **CLASSPATH** environment varijabla:
 - Predstavlja spisak foldera i JAR arhiva gde VM traži klasu koja se koristi.
 - Ako **CLASSPATH** ne postoji, podrazumevaju se tekući direktorijum i standardne Java biblioteke.
 - Ako **CLASSPATH** postoji, mora da sadrži i tekući direktorijum (standardne Java biblioteke se podrazumevaju). Primer:
CLASSPATH=. ;C:\Java\lib;C:\Java\bibl.jar

Paketi

- JAR archive
 - klasičan ZIP format
 - sadrži i folder `META-INF` u kojem je najbitnija datoteka `manifest.mf`
 - Sadržaj `manifest.mf` datoteke:
 - Manifest-Version: 1.0
 - Created-By: 1.4.2_02 (Sun Microsystems Inc.)
 - Main-Class: `moj.paket.Klasa`
 - Class-Path: `biblioteka.jar druga_biblioteka.jar`

Inner classes (unutrašnje klase)

```
class Spoljasnja {  
    Spoljasnja() { ... }  
    void metoda() { ... }  
  
    class Unutrasnja {  
        void metoda() { ... }  
    }  
}
```

Inner classes (unutrašnje klase)

- konstrukcija objekta unutrašnje klase izvan spoljašnje klase

```
Spoljasnja sp = new Spoljasnja();
```

```
Spoljasnja.Unutrasnja un = sp.new Unutrasnja();
```

javadoc

- specijalni komentari u izvornom kodu
- automatsko generisanje programske dokumentacije
- HTML format na izlazu
- Kompletna dokumentacija Jave je generisana javadoc alatom.
 - Lokacija: %JAVA_HOME%\doc

Odabrane klase







Klasa Object

- sve Java klase implicitno nasleđuju klasu Object
- Reprezentativne metode:
 - getClass()
 - equals(o)
 - hashCode()
 - toString()

Klasa String

- Niz karaktera je podržan klasom String. String **nije** samo niz karaktera – on je klasa!
- Objekti klase String se ne mogu menjati (*immutable*)!
- Reprezentativne metode:
 - str.length()
 - str.charAt(i)
 - str.indexOf(s)
 - str.substring(a,b), str.substring(a)
 - str.equals(s), str.equalsIgnoreCase(s) – **ne koristiti ==**
 - str.startsWith(s)

Klasa String

```
class StringTest {  
  
    public static void main(String args[]) {  
        String s1 = "Ovo je";  
        String s2 = "je string";  
        System.out.println(s1.substring(2));  o je  
        // karakter na zadatoj poziciji  
        System.out.println(s2.charAt(3));  s  
        // poređenje po jednakosti  
        System.out.println(s1.equals(s2));  false  
        // pozicija zadanog podstringa  
        System.out.println(s1.indexOf("je"));  4  
        // dužina stringa  
        System.out.println(s2.length());  9  
        // skidanje whitespace-ova sa poč. i kraja  
        System.out.println(s1.trim());  Ovo je  
        // provera da li string počinje podstringom  
        System.out.println(s2.startsWith("je"));  true  
  
    }  
}
```

Ispis na konzoli:

Redefinisan + operator sa stringovima

- Ako je jedan od operandata klase String, ceo izraz je string!

```
String a = "Vrednost i je: " + i;
```

- metoda `toString()`

Klasa `ArrayList`

- Predstavlja kolekciju, odn. dinamički niz
- Elementi se u `ArrayList` dodaju metodom `add()`
- Elementi se iz `ArrayList` uklanjaju metodom `remove()`
- Elementi se iz `ArrayList` dobijaju (ne uklanjaju se, već se samo čitaju) metodom `get()`

Klasa ArrayList

```
import java.util.ArrayList;
class ArrayListTest {
    public static void main(String args[]) {
        ArrayList v = new ArrayList();
        v.add("Ovo");
        v.add("je");
        v.add("probni");
        v.add("tekst");
        for (int i = 0; i < v.size(); i++)
            System.out.print((String)v.get(i) + " ");
    }
}
```

Tipizirana klasa `ArrayList`

```
import java.util.ArrayList;
class ArrayListTest {
    public static void main(String args[]) {
        ArrayList<String> v = new ArrayList<String>();
        v.add("Ovo");
        v.add("je");
        v.add("probni");
        v.add("tekst");
        for (int i = 0; i < v.size(); i++)
            System.out.print(v.get(i) + " ");
    }
}
```

Klasa HashMap

- Predstavlja asocijativnu mapu
- U HashMap se stavljaju dva podatka:
 - ključ po kojem će se pretraživati
 - vrednost koja se skladišti u HashMap i koja se pretražuje po ključu
- Metodom put() se ključ i vrednost smeštaju u HashMap
- Metodom get() se na osnovu ključa dobavlja (samo čita) vrednost iz HashMap

Klasa HashMap

```
import java.util.HashMap;
public class HashMapTest {
    public static void main(String args[]) {
        HashMap ht = new HashMap();
        ht.put("E10020", "Marko Markovic");
        ht.put("E10045", "Petar Petrovic");
        ht.put("E10093", "Jovan Jovanovic");
        String indeks = "E10045";
        System.out.println("Student sa brojem indeksa " +
                           indeks + " je " + ht.get(indeks));
        indeks = "E10093";
        System.out.println("Student sa brojem indeksa " +
                           indeks + " je " + ht.get(indeks));
    }
}
```

Tipizirana klasa HashMap

```
import java.util.HashMap;
public class HashMapTest {
    public static void main(String args[]) {
        HashMap<String, String> ht =
            new HashMap<String, String>();
        ht.put("E10020", "Marko Markovic");
        ht.put("E10045", "Petar Petrovic");
        ht.put("E10093", "Jovan Jovanovic");
        String indeks = "E10045";
        System.out.println("Student sa brojem indeksa " +
                           indeks + " je " + ht.get(indeks));
        indeks = "E10093";
        System.out.println("Student sa brojem indeksa " +
                           indeks + " je " + ht.get(indeks));
    }
}
```


Metoda split() klase **String**

- "cepa" osnovni string na niz stringova po zadatom šablonu
 - originalni string se ne menja
 - parametar je regularni izraz
- Poziv: `String[] rez = s.split("regex");`
- Primer:

```
String s = "ja sam svetski mega car";  
String[] rez = s.split(" ");
```

Metoda split() klase String

```
class SplitTest {  
    public static void main(String args[]) {  
        String text = "Ovo je probni tekst";  
        String[] tokens = text.split(" ");  
        for (int i = 0; i < tokens.length; i++)  
            System.out.println(tokens[i]);  
    }  
}
```

Metoda matches() klase `String`

- Vraća `true` ako je string u skladu sa regularnim izrazom
- Primer:

```
String s = "001-AB";
```

```
boolean rez = s.matches("\\d{3}-[A-Z]{2}");
```

Klasa `StringTokenizer`

- Radi sličan posao kao i metoda `split()` klase `String` – "cepa" zadati string po delimiteru (ili delimiterima)
- Ne radi sa regularnim izrazima
- Rezultat cepanja je kolekcija stringovva kroz koju se iterira metodama `hasMoreTokens()` i `nextToken()`

Klasa StringTokenizer

```
import java.util.*;
class TokenizerTest {
    public static void main(String args[]) {
        String text = "Ovo je probni tekst";
        StringTokenizer st = new StringTokenizer(text, " ");
        while (st.hasMoreTokens()) {
            System.out.println(st.nextToken());
        }
    }
}
```

Klase `BigInteger` i `BigDecimal`

- Koriste se za brojeve sa proizvoljnim brojem cifara.
- Primer:

```
BigInteger a = BigInteger.valueOf(10) ;
```

```
BigInteger b = BigInteger.valueOf(20) ;
```

```
BigInteger c = a.multiply(b) ;
```

Konvencije davanja imena

- nazivi klasa (`MojaKlasa`)
- nazivi metoda (`mojaMetoda`)
- nazivi atributa (`mojAtribut`)
- nazivi paketa (`mojpaket.drugipaket`)
- set/get metode (`setAtribut/getAtribut`)
- konstante (`MAX_INTEGER`)

Programski jezik Java

Ulazno izlazni podsistem

Ulazno izlazni podsistem

- standardna biblioteka za ulazno/izlazne operacije
- izvorišta/odredišta:
 - memorija
 - fajl sistem
 - mrežne konekcije
- oslanja se na tokove (streams) i čitače/pisače (reader/writer)
- nezavisno od tokova/čitača postoji i `RandomAccessFile` klasa i `File` klasa

File klasa

- za manipulaciju datotekama i direktorijumima:
 - kreiranje datoteka i direktorijuma
 - brisanje datoteka i direktorijuma
 - pristup atributima datoteka i direktorijuma
 - modifikacija naziva i atributa datoteka i direktorijuma

File klasa

- `File f = new File(".");`
- `File f = new File("C:\\Windows");`
- `f = new File(f, "pera");`
- `f = new File(f, "..");`
- `if(f.exists()) ...`
- `if(f.isDirectory()) ...`
- `f.createNewFile();`

Tokovi (streams) _{1/4}

- bazirani na bajtovima
 - prenos jednog bajta
 - prenos niza bajtova
- omogućuju prenos podataka:
 - datoteke (FileInputStream, FileOutputStream)
 - niza bajtova (ByteArrayInputStream, ByteArrayOutputStream)
 - sekvence drugih tokova (SequenceInputStream)
 - itd.

Tokovi (streams) _{2/4}

- osmišljeni kao mehanizam koji omogućuje unificiran pristup podacima
- isti kôd se koristi za čitanje/pisanje iz, na primer, datoteke ili mrežne konekcije
- koncept filtera - donose dodatnu funkcionalnost tokovima:
 - prenos primitivnih tipova na mašinski nezavisan način (DataInputStream, DataOutputStream)
 - baferizovan prenos podataka (BufferedInputStream, BufferedOutputStream)
 - prenos objekata (ObjectInputStream, ObjectOutputStream)
 - formatiranje podataka (PrintStream)
 - održavanje čeksuma nad podacima (CheckedOutputStream)
CRC32 , Adler32

Tokovi (streams) _{3/4}

- Metode u tokovima:
 - read() – čita jedan bajt iz toka
 - read(byte[]) – čita niz bajtova
 - skip(long n) – preskače zadati broj bajtova
 - available() – vraća broj raspoloživih bajtova iz toka koji se mogu pročitati pre blokiranja sledećeg čitanja
 - close() – zatvara tok

Tokovi (streams) 4/4

- Primer upotrebe – kopiranje sadržaja datoteke:

```
byte[] buffer = new byte[BUFFER_LENGTH];  
while((read=in.read(buffer, 0,BUFFER_LENGTH)) != -1) {  
    // obrada učitanoeg niza bajtova  
    out.write(buffer);  
}
```

Čitači/pisači (readers/writers) ^{1/3}

- ispravljaju problem sa tokovima – slabu podršku Unicode rasporedu:
 - tokovi ne prenose dobro Unicode stringove
 - poseban problem predstavljaju različite hardverske platforme (*little-endian*, *big-endian*)
- čitači/pisači ne zamenjuju tokove – oni ih dopunjuju
- čitači/pisači se koriste kada je potrebno preneti Unicode stringove ili karaktere – u ostalim situacijama koriste se tokovi

Čitači/pisači (readers/writers) _{2/3}

- omogućuju prenos karaktera iz/u:
 - datoteke (FileReader/FileWriter)
 - druge nizove karaktera (CharArrayReader/CharArrayWriter)
 - stringove (StringReader/StringWriter)
- Klase za spregu tokova i čitača/pisača –
InputStreamReader, OutputStreamWriter:

```
BufferedReader in = new BufferedReader(  
    new InputStreamReader(System.in));
```

Čitači/pisači (readers/writers) _{3/3}

- Metode u čitačima:
 - read() – čita jedan karakter iz toka
 - read(char[]) – čita niz karaktera
 - skip(long n) – preskače zadati broj karaktera
 - close() – zatvara čitač

Zaključak

- podaci se u čitaju iz ulaznih tokova, a pišu u izlazne tokove
- iz programa se retko radi direktno sa bajtovima
 - zato se tokovi ugrađuju u Filter klase koje imaju odgovarajuće metode za čitanje/pisanje
 - zato imamo tokove objekata, tokove primitivnih tipova itd.
- ako radimo sa karakterima/stringovima, koristimo čitače i pisače

Štampanje na ekran

- `System.out` je izlazni tok:

```
System.out.print("Poruka");
```

```
System.out.println("Poruka");
```

- Ispis se može i formatirati:

```
System.out.printf("format", argumenti);
```

```
System.out.printf("%.2f %d", (10000.0 / 3), 5);
```

Unos sa tastature

- `System.in` je ulazni tok:

```
BufferedReader in = new BufferedReader( new  
    InputStreamReader(System.in));
```

```
String s = in.readLine();
```

- Alternativa je klasa `Scanner` koja ne učitava samo stringove:

```
Scanner sc = new Scanner(System.in);
```

```
String s = sc.nextLine();
```

```
int i = sc.nextInt();
```

```
float f = sc.nextFloat();
```

Unos drugih primitivnih tipova sa tastature

- Koristi se wrapper klasa i njena metoda `parseXxx()`:

```
BufferedReader in = new BufferedReader( new  
    InputStreamReader(System.in) );
```

```
String s = in.readLine();
```

```
int i = Integer.parseInt(s);
```

- Kraće je klasom `Scanner`:

```
Scanner sc = new Scanner(System.in);
```

```
int i = sc.nextInt();
```

Serijalizacija objekata

- serijalizacija objekta – prevođenje objekta u niz bajtova i njegova rekonstrukcija iz niza u “živ” objekat
- serijalizovan niz bajtova se može snimiti u datoteku ili poslati preko mreže – i jedno i drugo upotrebom tokova
- prilikom serijalizacije, serijalizuju se osim samog objekta i njegovi atributi – stablo serijalizovanih objekata
- da bi se neki objekat serijalizovao:
 - potrebno je da implementira `java.io.Serializable` interfejs
 - da su atributi i parametri metoda takođe serijalizabilni

Rad sa arhivama

- podržan rad sa GZip i Zip formatima arhiva
- klase koje podržavaju rad sa arhivama:
 - GZipInputStream, GZipOutputStream
 - ZipInputStream, ZipOutputStream
 - ZipFile

Case Study – INI datoteke

- INI datoteke – Windows sistem za smeštanje parametara programa
- tekstualne datoteke
- struktura:
 - sekcije
 - parovi (ključ, vrednost)
 - komentari-redovi koji počinju znakom ‘;’

INI datoteke - primer

[General]

; naslov prozora

Title = Naslov

; x koordinata prozora

x = 100

Test aplikacija

```
public class TestIni {  
    public TestIni() {  
        INIFile ini = new INIFile("test.ini");  
        String title = ini.getString("General", "Title");  
        int x = ini.getInt("General", "x");  
        System.out.println("Title: " + title);  
        System.out.println("x: " + x);  
    }  
  
    public static void main(String[] args) {  
        new TestIni();  
    }  
}
```

INIFile klasa

```
import java.io.*;
import java.net.URL;
import java.util.*;

public class INIFile {

    /** Hash mapa koja sadrzi kategorije (sekcije). Hash kljuc
        je naziv kategorije (string), a vrednost je hash mapa koja
        sadrzi parove (parametar, vrednost). */
    private Hashtable categories = new Hashtable();

    public INIFile(String filename) {
        BufferedReader in = null;
        try {
            in = new BufferedReader(new FileReader(filename));
            readINI(in);
            in.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

private void readINI(BufferedReader in) {
    String line, key, value; StringTokenizer st; int pos;
    String currentCategory = "default";
    Hashtable currentMap = new Hashtable();
    categories.put(currentCategory, currentMap);
    try {
        while ((line = in.readLine()) != null) {
            line = line.trim();
            if (line.equals("") || line.indexOf(';') == 0)
                continue;
            if (line.charAt(0) == '[') {
                currentCategory = line.substring(1, line.length()-1);
                currentMap = new Hashtable();
                categories.put(currentCategory, currentMap);
            } else {
                st = new StringTokenizer(line, "=");
                if (st.hasMoreTokens()) {
                    key = st.nextToken().trim();
                    value = st.nextToken().trim();
                    currentMap.put(key, value);
                }
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

```

```

/** Vraca vrednost datog parametra u obliku stringa.
 * @param category Kategorija (sekcija) u kojoj se nalazi parametar
 * @param key Naziv parametra
 * @return String koji sadrzi vrednost parametra
 */
public String getString(String category, String key) {
    Hashtable hm = (Hashtable)categories.get(category);
    if (hm == null)
        return "";
    else
        return (String)hm.get(key);
}

/** Vraca vrednost datog parametra u obliku integera.
 * @param category Kategorija (sekcija) u kojoj se nalazi parametar
 * @param key Naziv parametra
 * @return Integer vrednost parametra
 */
public int getInt(String category, String key) {
    Hashtable hm = (Hashtable)categories.get(category);
    if (hm == null)
        return 0;
    else
        return (new Integer((String)hm.get(key))).intValue();
}

```

Konkurentno programiranje u Javi

Konkurentno programiranje u Javi

- Java sadrži koncepte potrebne za konkurentno programiranje
- svaki Java program je konkurentni program – garbage collector je posebna programska nit (thread)
- u Javi je moguće pisati konkurentne programe bez pozivanja sistemskih rutina za konkurentno programiranje – sve što je potrebno enkapsulirano je u odgovarajućim klasama i u VM

Kreiranje programskih niti

- Nasleđivanje klase **Thread**

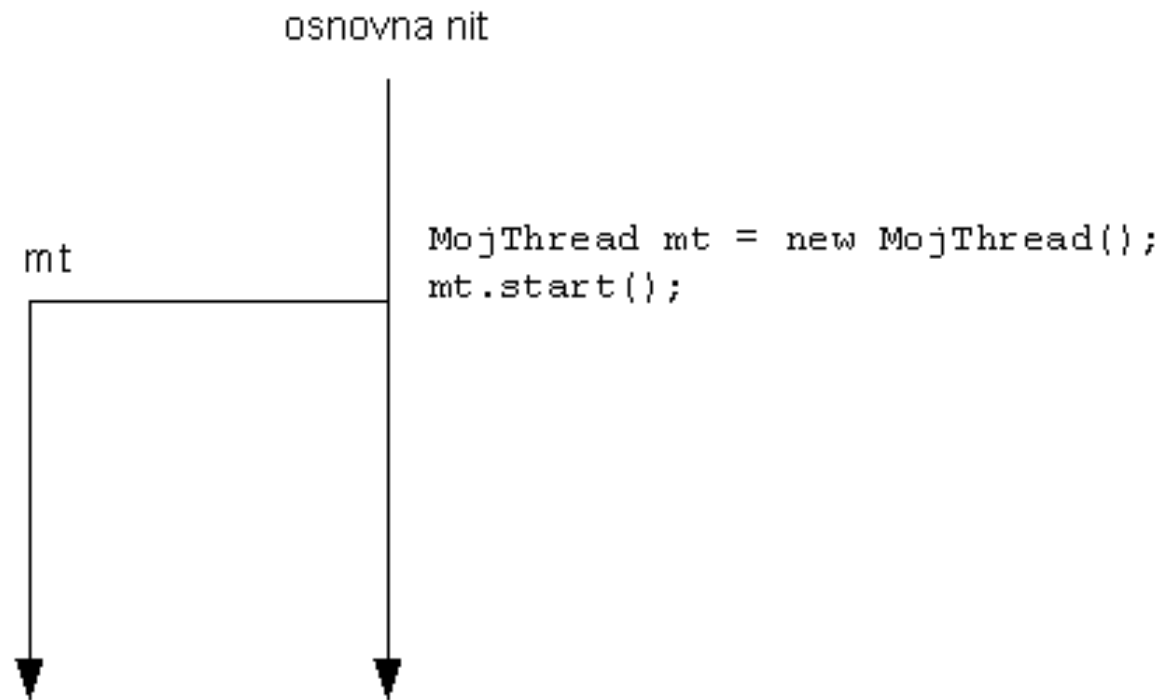
```
public class MojThread extends Thread {  
    public void run() {  
        // programski kod niti je ovde  
    }  
}  
  
...  
MojThread mt = new MojThread();  
mt.start();
```

Kreiranje programskih niti

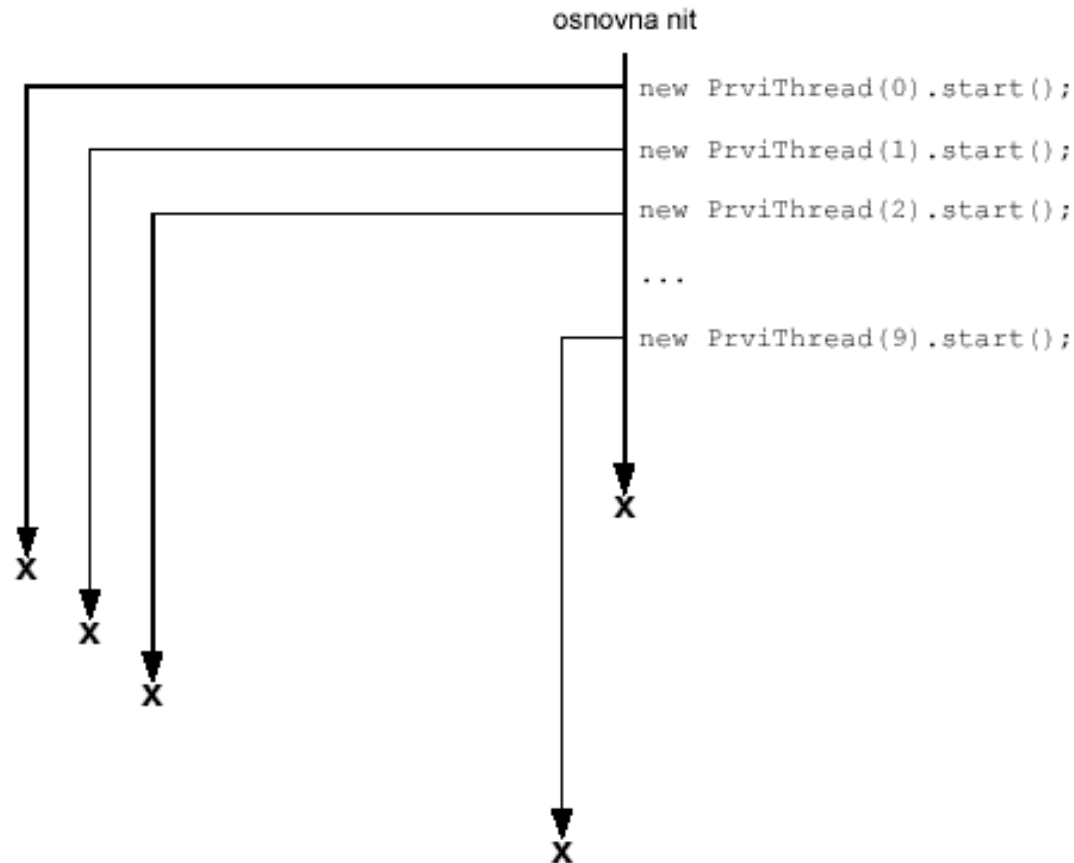
- Implementiranje interfejsa **Runnable**

```
public class MojThread implements Runnable {  
    public void run() {  
        // programski kod niti je ovde  
    }  
}  
  
...  
MojThread mt = new MojThread();  
Thread t = new Thread(mt);  
t.start();
```

Kreiranje programskih niti



Kreiranje više programskih niti



Daemon i non-daemon niti

- Java program završava svoje izvršavanje kada se završe sve njegove non-daemon niti
- Java program inicijalno kreće sa jednom non-daemon niti
- Daemon atribut: `setDaemon()` / `getDaemon`

Deljenje resursa između niti

- Mehanizam zaključavanja objekata: u jednom trenutku samo jedna nit može pristupati objektu

Deljenje resursa između niti

- **synchronized blok**

```
...
```

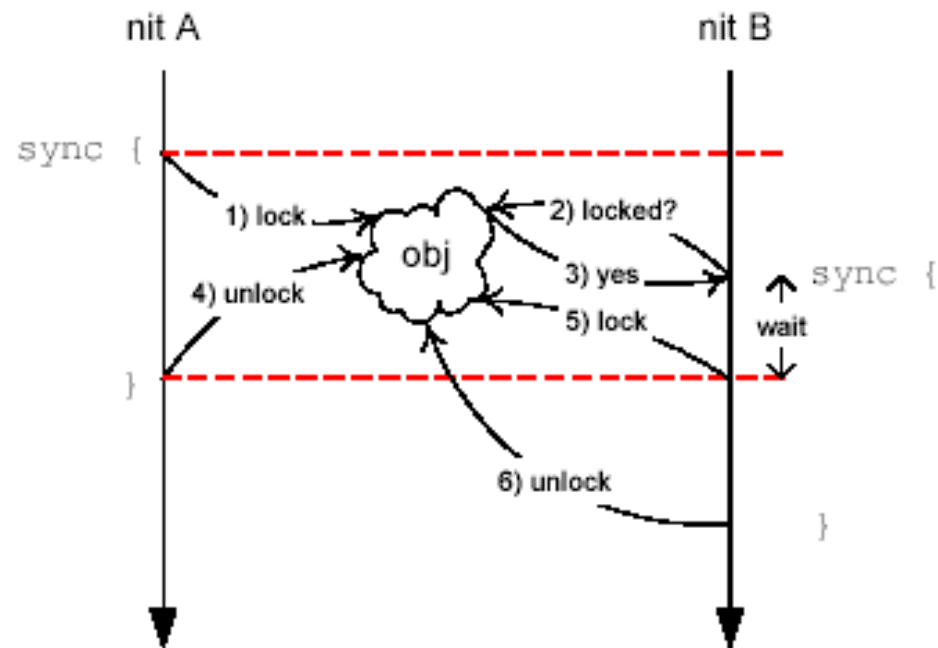
```
synchronized(this) {  
    // kod koji radi sa deljenim resursima  
}
```

```
...
```

- **synchronized metoda**

```
public synchronized void metoda() { ... }
```

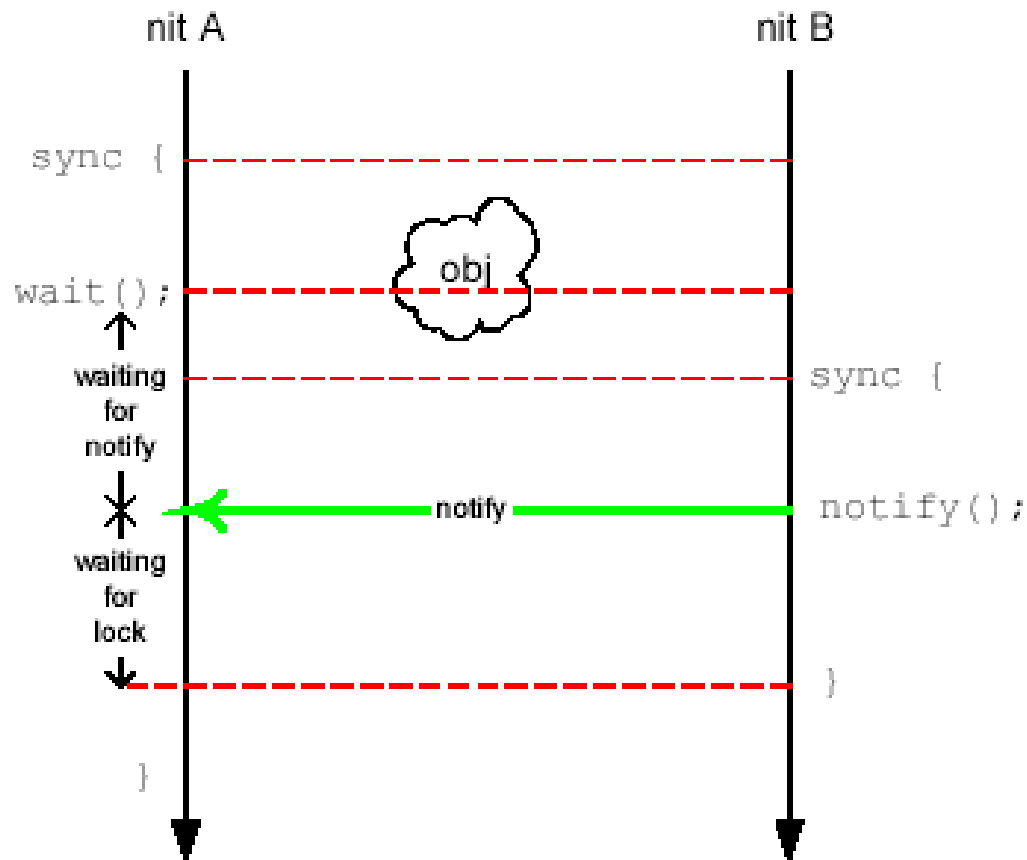
sinchronyzed blok



Deljenje resursa između niti

- `wait()`: oslobađa pristup objektu unutar `synchronized` bloka i blokira izvršavanje niti dok neka druga nit ne pošalje `notify()`
- `notify()`: obaveštava jednu od niti koje čekaju sa `wait()` da nastavi sa izvršavanjem
 - ako više niti čeka, ne zna se koja će nit nastaviti sa radom
- `notifyAll()`: obaveštava **sve** niti koje čekaju sa `wait()` da nastave sa radom

wait-notify



sleep(n) metoda

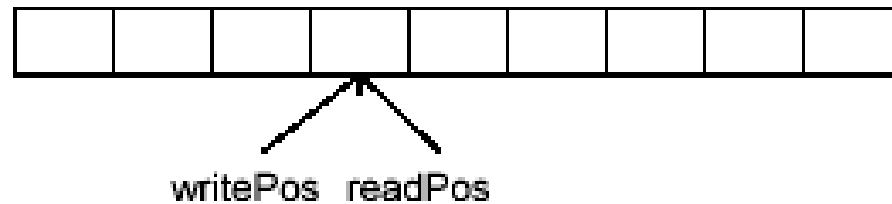
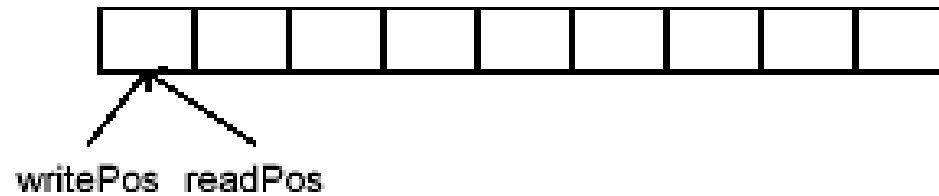
- **sleep(n)**: programska nit čija se metoda **sleep(n)** pozove "spava" zadati broj (n) milisekundi.
- statička metoda **Thread.currentThread()** vraća referencu na programsku nit iz koje je pozvana
 - **main()** programska nit

Deljenje resursa između niti

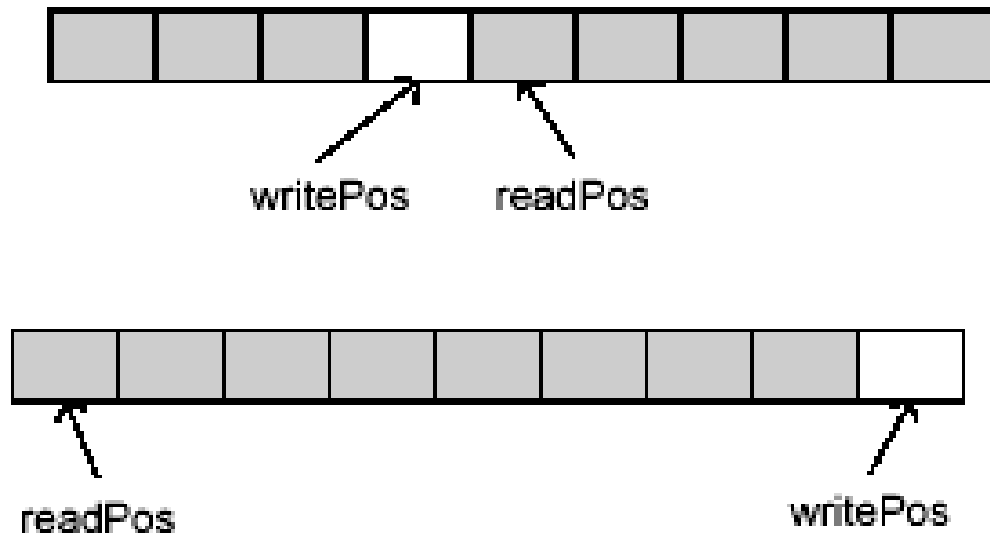
- primer: proizvođač→bafer→potrošač



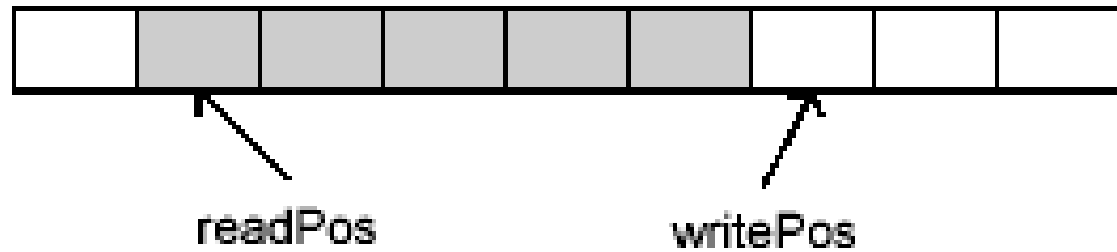
Prazan bafer



Pun bafer



Bafer ni pun ni prazan



Semafor

(od Jave v1.5)

- Semafori održavaju skup dozvola
 - svaki poziv metode *acquire* uzima dozvolu
 - svaki poziv metode *release* dodaje dozvolu
- Ne čuvaju se stvarne dozvole, već se broje dozvole.
 - ako više nema dozvola, nit koja je pozvala *acquire* se blokira
 - poziv metode *release* može da deblokira pozivajuću nit (ako je bila blokirana)

CyclicBarrier

(od Jave v1.5)

- Barijera zaustavlja sve niti dok se interni brojač ne dovede na nulu
 - tada se sve niti deblokiraju
- Barijera blokira pozivajuću nit (nit koja je pozvala metodu *await*) i smanjuje interni brojač
 - kada brojač dospe do nule, sve niti se deblokiraju,
 - opciono može posebna nit da se tada startuje
- Služi za sinhronizaciju većeg broja niti na jedan događaj
 - sve niti čekaju na istoj barijeri

CountDownLatch (od Jave v1.5)

- Blokira nit dok se interni brojač ne dovede na nulu
- Nit koja pozove *await* metodu se blokira
- Svaki poziv *countDown* metode smanjuje interni brojač
 - kada brojač dođe do nule, blokirana nit se deblokira
- Služi da jedna nit sačeka da ostale niti obave neki posao
 - tek kada sve niti obave posao, ova nit se deblokira

BlockingQueue interfejs (od Jave v1.5)

- Klase koje implementiraju ovaj interfejs (ArrayBlockingQueue, ListBlockingQueue, itd.) implementiraju red (*Queue*)
- Red ima početak i kraj (head i tail)
 - novi elementi se dodaju na kraj (tail)
 - elementi se vade sa početka (head)
- BlockingQueue obezbeđuje da:
 - ako je red prazan, pokušaj vađenja elementa dovodi do blokiranja
 - ako je red pun, pokušaj dodavanja elementa dovodi do blokiranja
- Ako se u konstruktoru doda drugi (*boolean*) parametar (*fairnes*) sa vrednošću *true*, garantuje se redosled niti koje su pristupale redu (uz degradiranje preformansi)

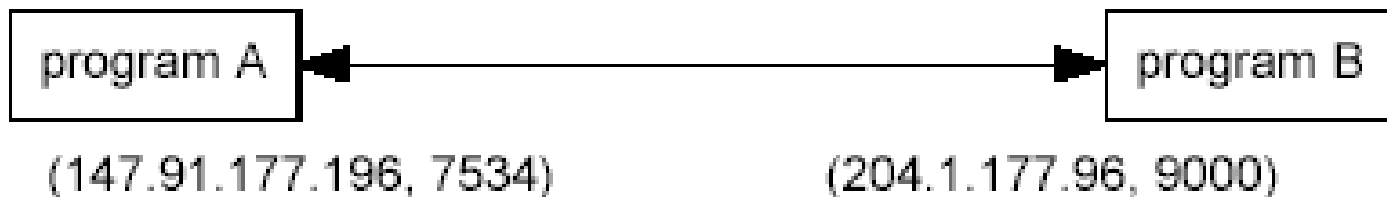
Mrežno programiranje u Javi

Osnovne odrednice

- Oslanja se na IP mrežni protokol
- Pojam *socket*-a: par (IP adresa, port)
- Mogućnost korišćenja TCP i UDP protokola
- Komunikacija se odvija kroz *stream*-ove; jednako kao i sa fajlovima u okviru fajl-sistema
- Paket `java.net`

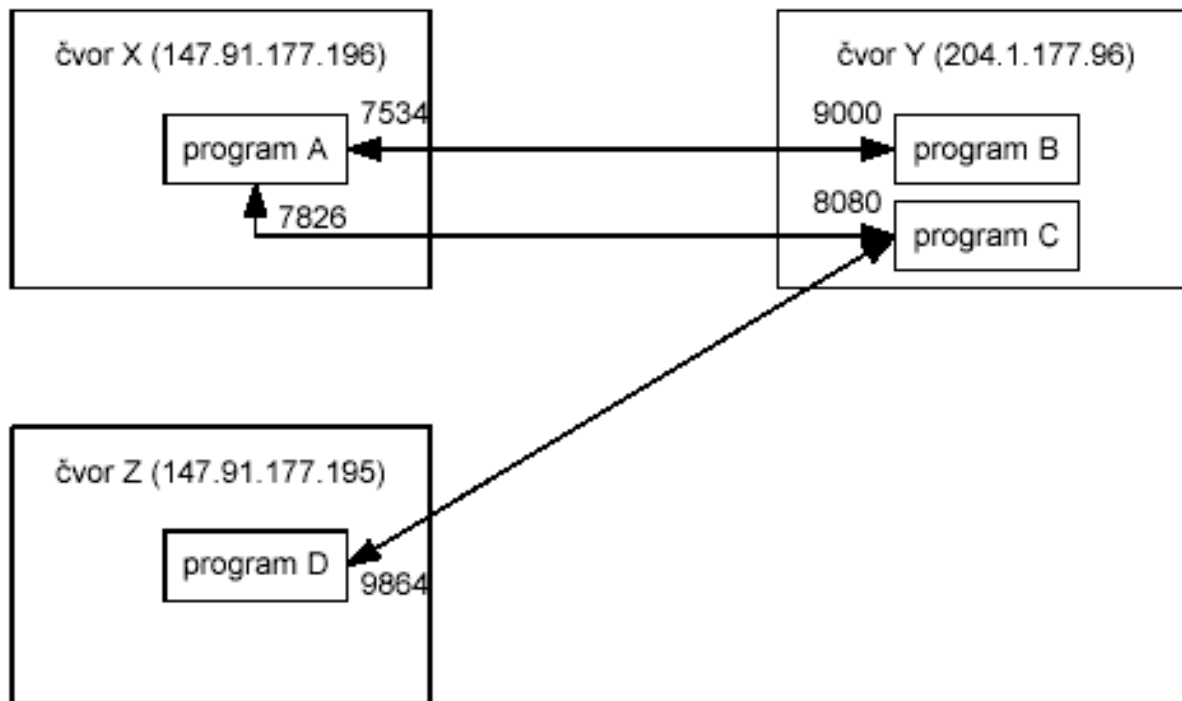
Socket _{1/2}

- Par (IP adresa, port)
- Uspostavljena veza između dva programa određena je parom socket-a



Socket 2/2

- Više programa u vezi



Identifikacija čvorova mreže

- Pomoću simboličke adrese, npr. `www.yahoo.com`
- Pomoću numeričke adrese, podeljene u oktete, npr. `147.91.177.196`
- Klasa **InetAddress** predstavlja adresu čvora u mreži:

```
InetAddress a = InetAddress.getByName("www.yahoo.com");  
InetAddress b = InetAddress.getByName("147.91.177.196");  
InetAddress c = InetAddress.getLocalHost();  
byte[] bytes = a.getAddress();  
InetAddress[] addresses = a.getAllByName(host);
```


Klasa Socket

- Predstavljá TCP konekciju između dva čvora na mreži
`Socket(InetAddress address, int port)`
`Socket(String host, int port)`
- Metoda `getInputStream()`
- Metoda `getOutputStream()`
- Timeout
 - `socket.setSoTimeout(10000);`
 - od Java v1.4 postoji i metoda:
`socket.connect(
 new InetSocketAddress(host, port),
 timeout);`

Klasa `ServerSocket`

- Predstavlja serverski socket
`ServerSocket(int port)`
- Metoda `accept()` blokira izvršenje sve dok neki klijent ne uspostavi vezu; tada vraća konstruisan `Socket` objekat preko koga se komunicira sa klijentom

Tipičan tok komunikacije – klijent strana

```
// inicijalizacija
Socket s = new Socket(addr, port);
BufferedReader in = new BufferedReader(...,s);
PrintWriter out = new PrintWriter(...,s);

// komunikacija
out.println("zahtev");           // šaljem zahtev
String response = in.readLine(); // čitam odgovor

// prekid veze
in.close();
out.close();
s.close();
```

Tipičan tok komunikacije – server strana

```
// čekam klijenta...
ServerSocket ss = new ServerSocket(port);
Socket s = ss.accept();

// inicijalizacija
BufferedReader in = new BufferedReader(...,s);
PrintWriter out = new PrintWriter(...,s);

// komunikacija
String request = in.readLine();    // čitam zahtev
out.println("odgovor");            // šaljem odgovor

// prekid veze
in.close();
out.close();
s.close();
```

Server koji opslužuje više klijenata

```
// Serverska petlja
ServerSocket ss = new ServerSocket(port);
while (true) {
    Socket s = ss.accept();
    ServerThread st = new ServerThread(s);
}

// poseban thread - obrada pojedinačnog zahteva
class ServerThread extends Thread {
    ...
    public void run() {
        // inicijalizacija
        // komunikacija
        // prekid veze
    }
}
```

Praćenje korisničke sesije

- Stateful protokoli: čuvaju stanje tokom komunikacije
 - primer: POP3, SMTP protokol
- Stateless protokoli: ne čuvaju stanje komunikacije između dva klijentska zahteva
 - primer: HTTP protokol (osnovni, bez proširenja za praćenje sesije).

Prenos podataka

- Metode `getInputStream()` i `getOutputStream()` daju ulazni, odn. izlazni tok podataka od/ka klijentu
- Ne pružaju više od čitanja jednog ili niza bajtova
- Moraju se ubaciti u filter klase za prenos složenijih tipova podataka

Prenos podataka

- Prenos veće količine podataka
- Problem: kako da server zna kada smo završili sa prenosom podataka
 - kod datoteka smo imali indikator kraja datoteke
 - kod mrežnih tokova to **nemamo**
- Rešenje:
 - unapred poslati broj elemenata, pa same elemente
 - poslati elemente, pa zatim marker kraja
 - specijalna oznaka
 - oznaka novog reda kod prenosa podataka

Primer – hello klijent/server

- Klijent
 - klijent uspostavlja vezu sa serverom
 - šalje tekst "HELLO" (zahtev)
 - čita odgovor servera
 - završava komunikaciju
- Server
 - čeka klijente u beskonačnoj petlji
 - za svakog klijenta pokreće poseban thread za obradu:
 - čita zahtev klijenta
 - šalje odgovor – redni broj obrađenog zahteva

Primer – listanje direktorijuma na serveru

- Klijent
 - klijent uspostavlja vezu sa serverom
 - šalje putanju na serveru koja se lista
 - čita odgovor servera
 - završava komunikaciju
- Server
 - čeka klijente u beskonačnoj petlji
 - za svakog klijenta pokreće poseban thread za obradu:
 - čita putanju koju je klijent poslao
 - lista zadati direktorijum
 - šalje odgovor – spisak stavki iz direktorijuma

Primer – registracija korisnika na serveru

- Klijent
 - klijent uspostavlja vezu sa serverom
 - šalje korisničko ime za registraciju
 - čita odgovor servera
 - završava komunikaciju
- Server
 - čeka klijente u beskonačnoj petlji
 - za svakog klijenta pokreće poseban thread za obradu:
 - čita korisničko ime koje je klijent poslao
 - doda to ime u listu
 - izlista spisak prijavljenih korisnika
 - šalje odgovor – spisak registrovanih korisnika

Demonstracija mrežnog programiranja u Javi – chat aplikacija

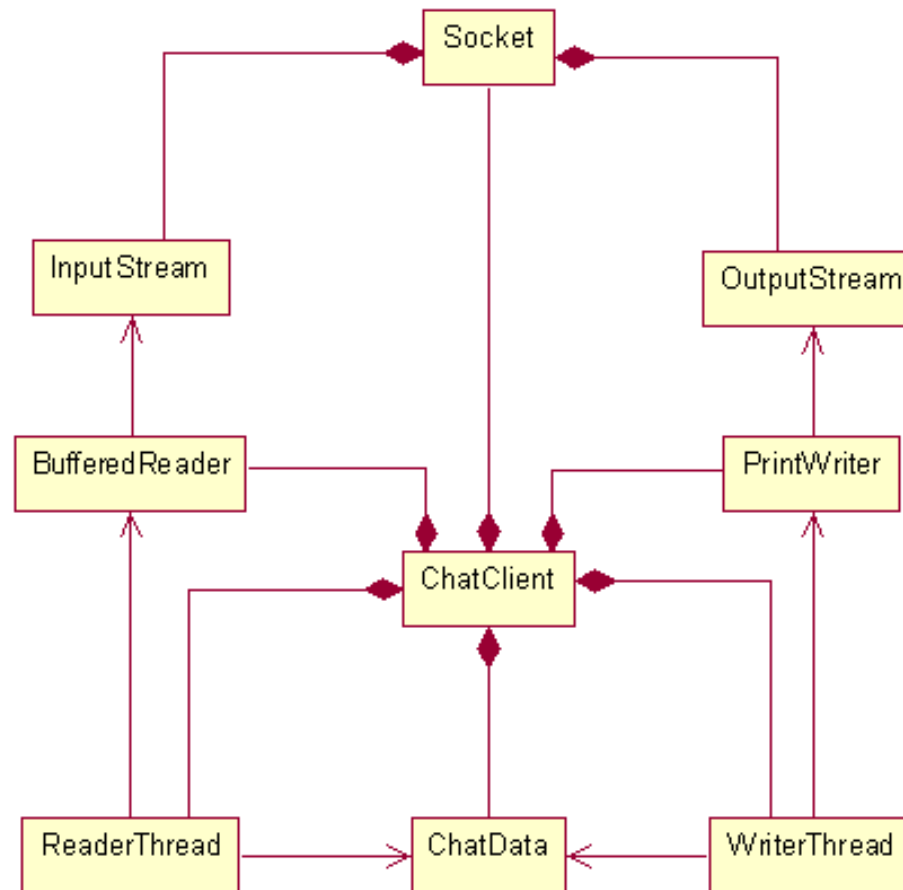
Zadatak

- Napisati klijent/server konzolnu aplikaciju koja omogućava chat za sve korisnike koji se prijave na isti server
- Zadatak klijenta: da omogući unos novih i pregled pristiglih poruka (sa podacima o pošiljaocu poruke)
- Zadatak servera: da poruke poslate od strane jednog klijenta prosledi do ostalih klijenata

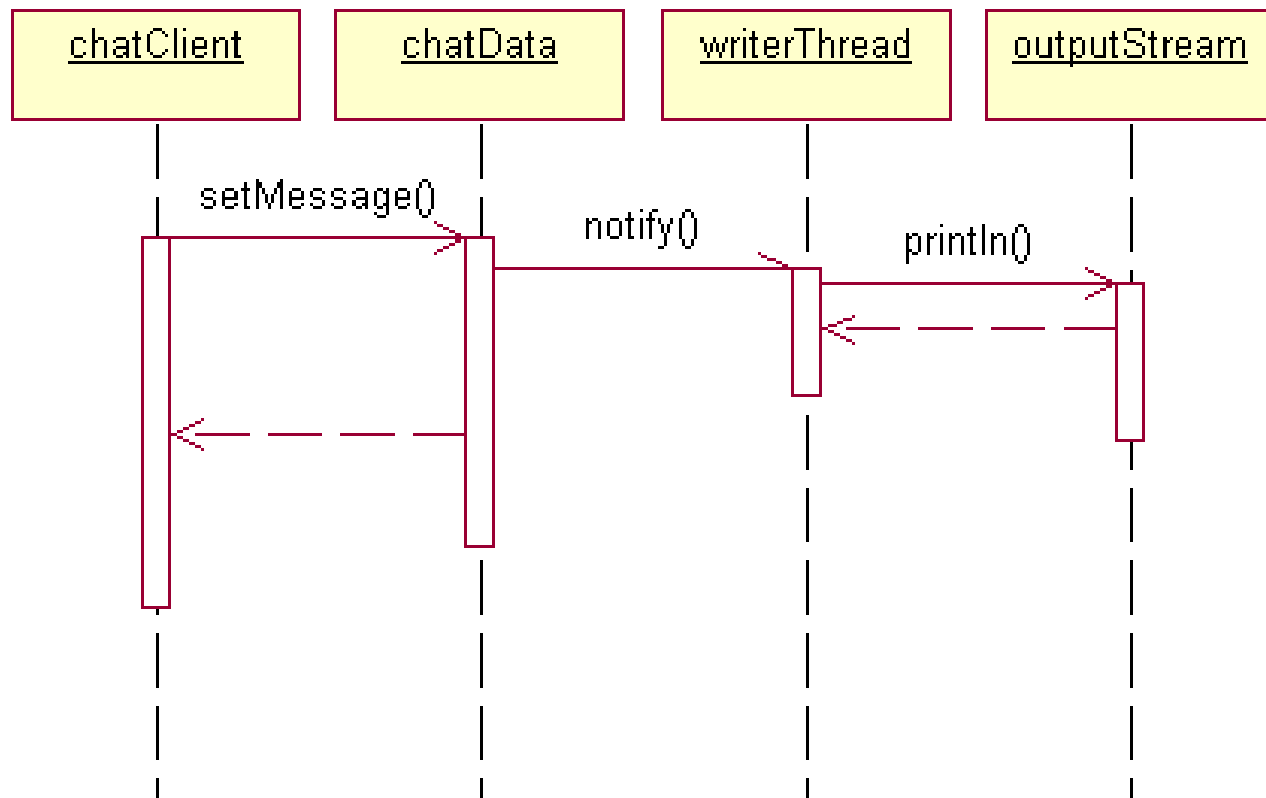
Klijent

- Programske niti:
 - glavna (main) – obavlja prijavljivanje korisnika na server, kreira preostale dve niti i poruku unetu tastaturom prepušta WriterThread-u
 - ReaderThread – čita poruke sa servera i prikazuje ih na ekranu
 - WriterThread – poruke unete sa tastature šalje na server

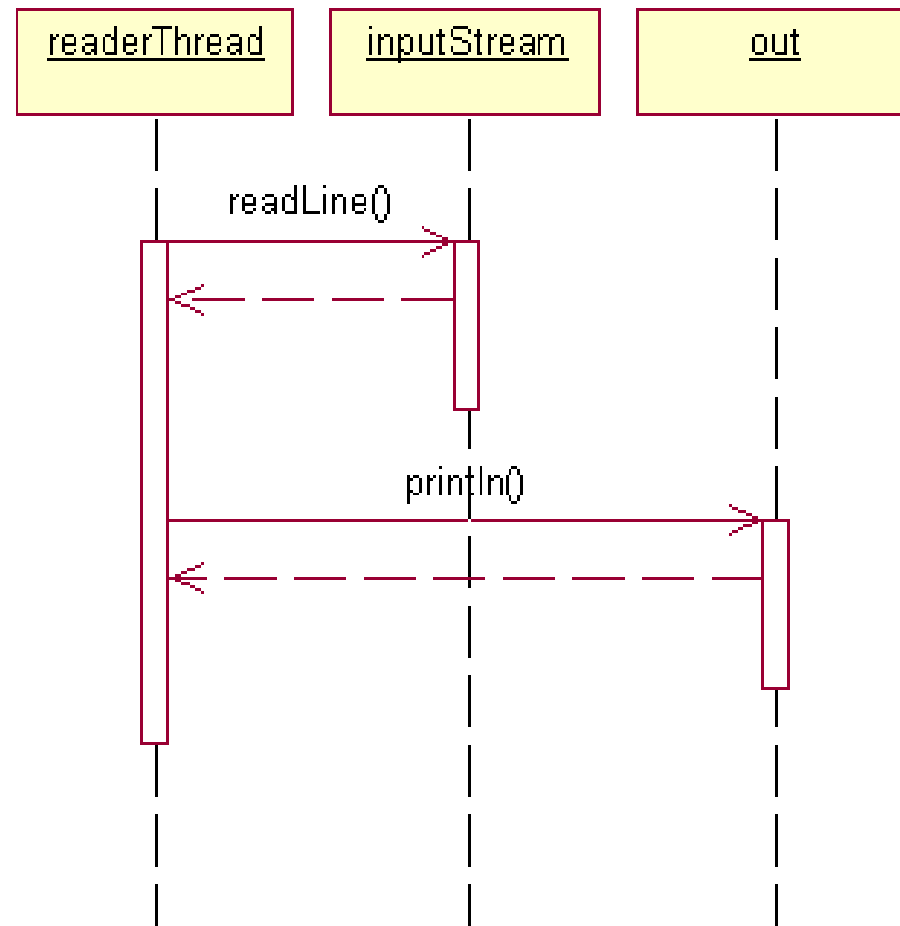
Dijagram klasa klijentske aplikacije



Klijent – slanje poruke



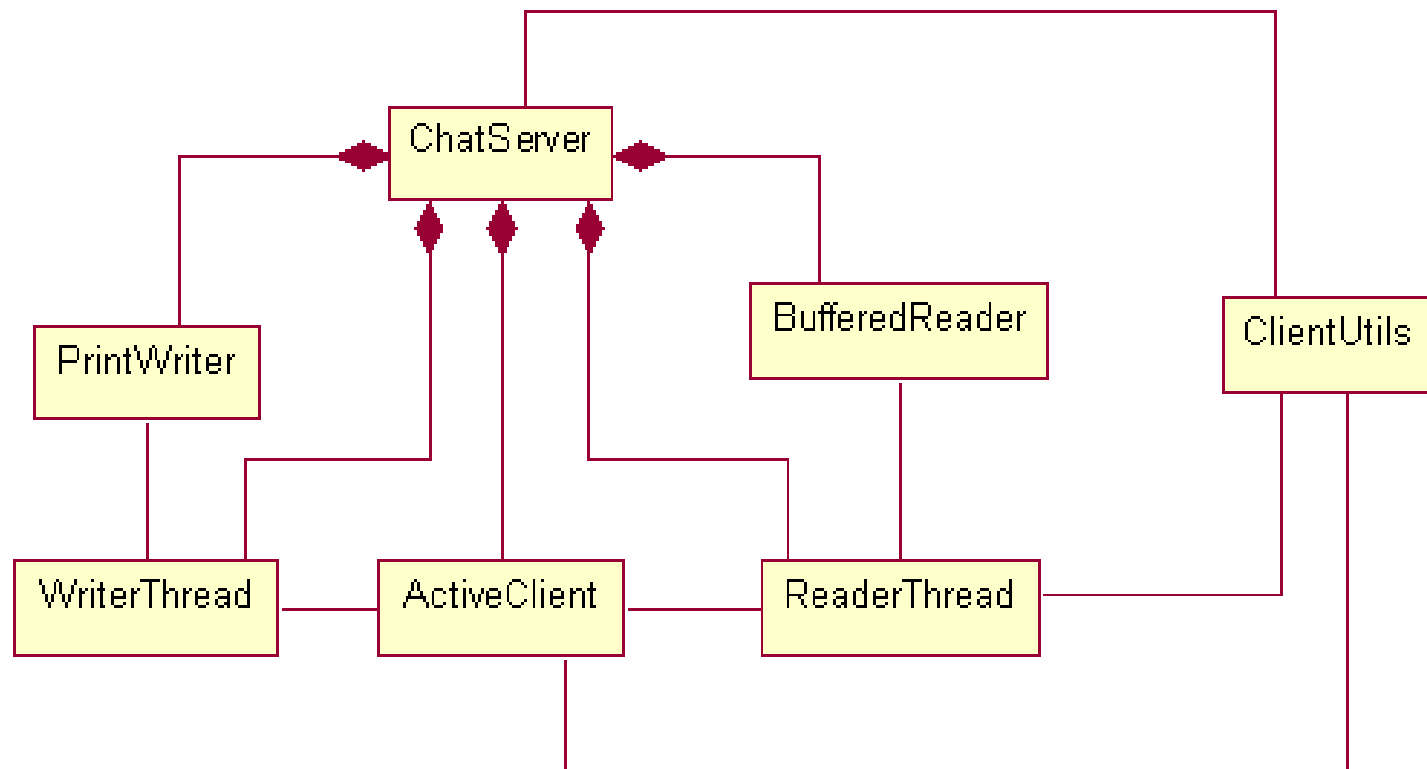
Klijent – prijem poruke



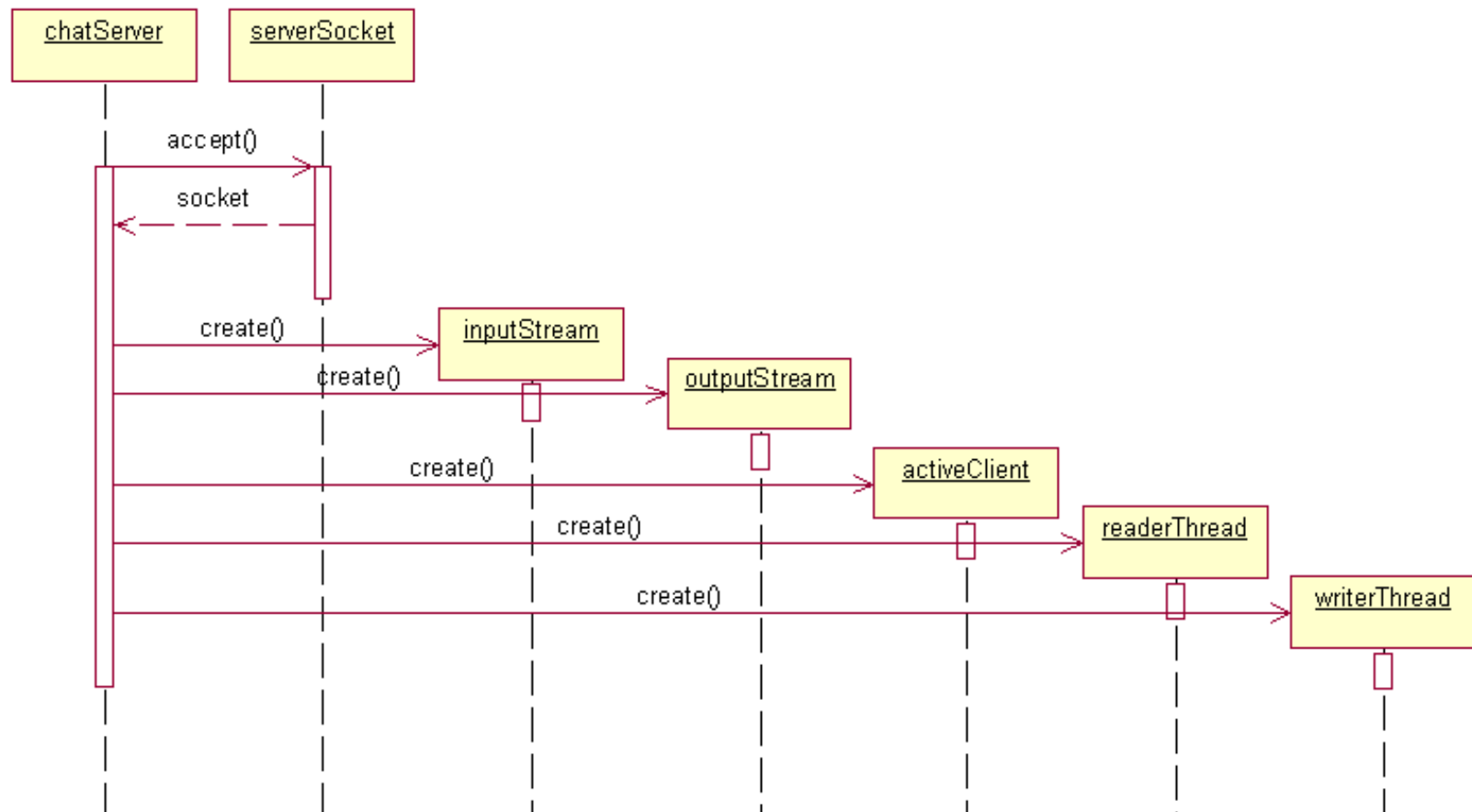
Server

- Programske niti:
 - glavna (main) –čeka na klijente da se spoje, registruje korisničko ime i za svakog ispravno prijavljenog klijenta kreira ReaderThread i WriterThread
 - ReaderThread – prima poruku od klijenta
 - WriterThread – šalje poruku klijentu

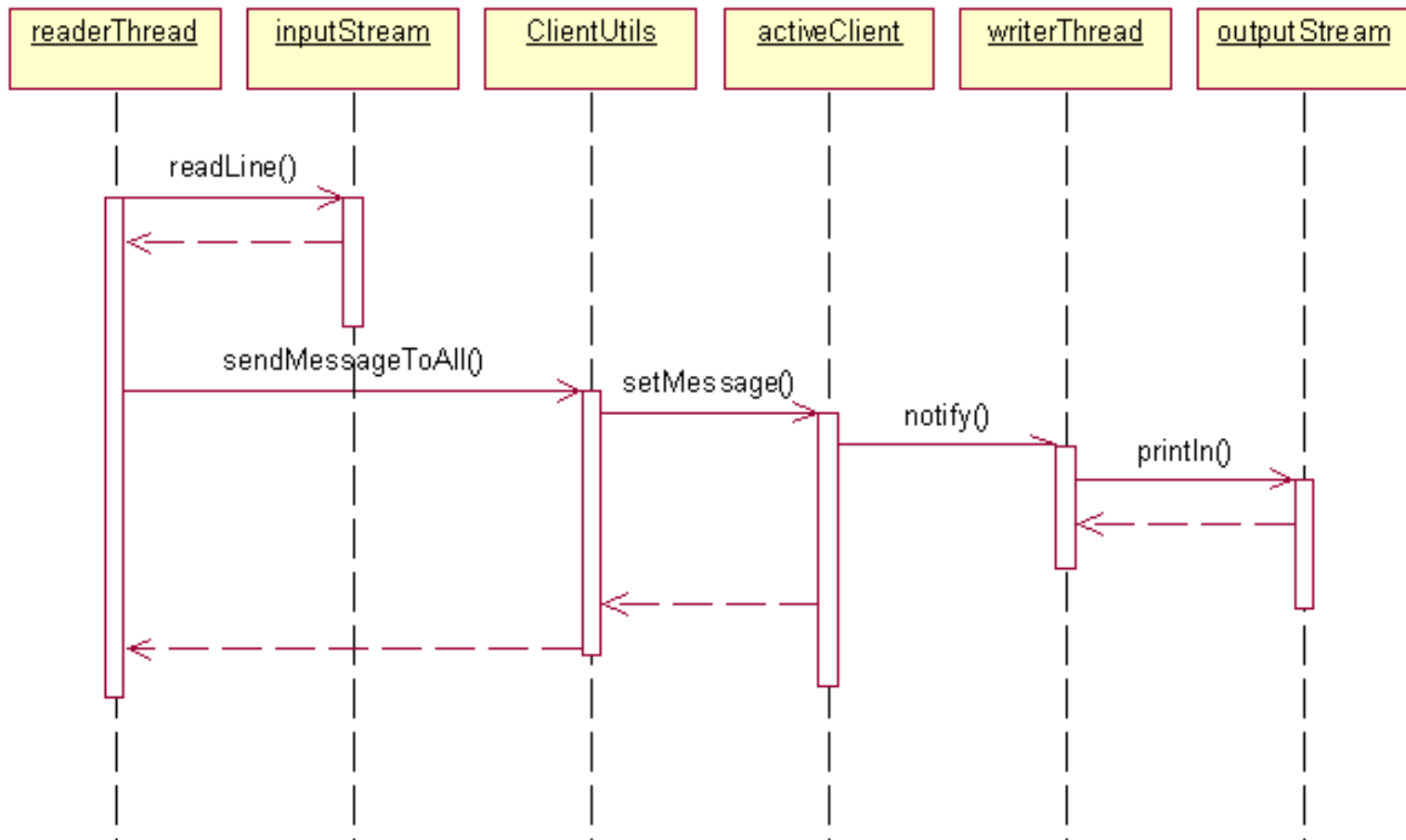
Server – diagram klasa



Server – prijava klijenta



Server – prijem poruke i slanje svim klijentima



Web programiranje

HTTP protokol

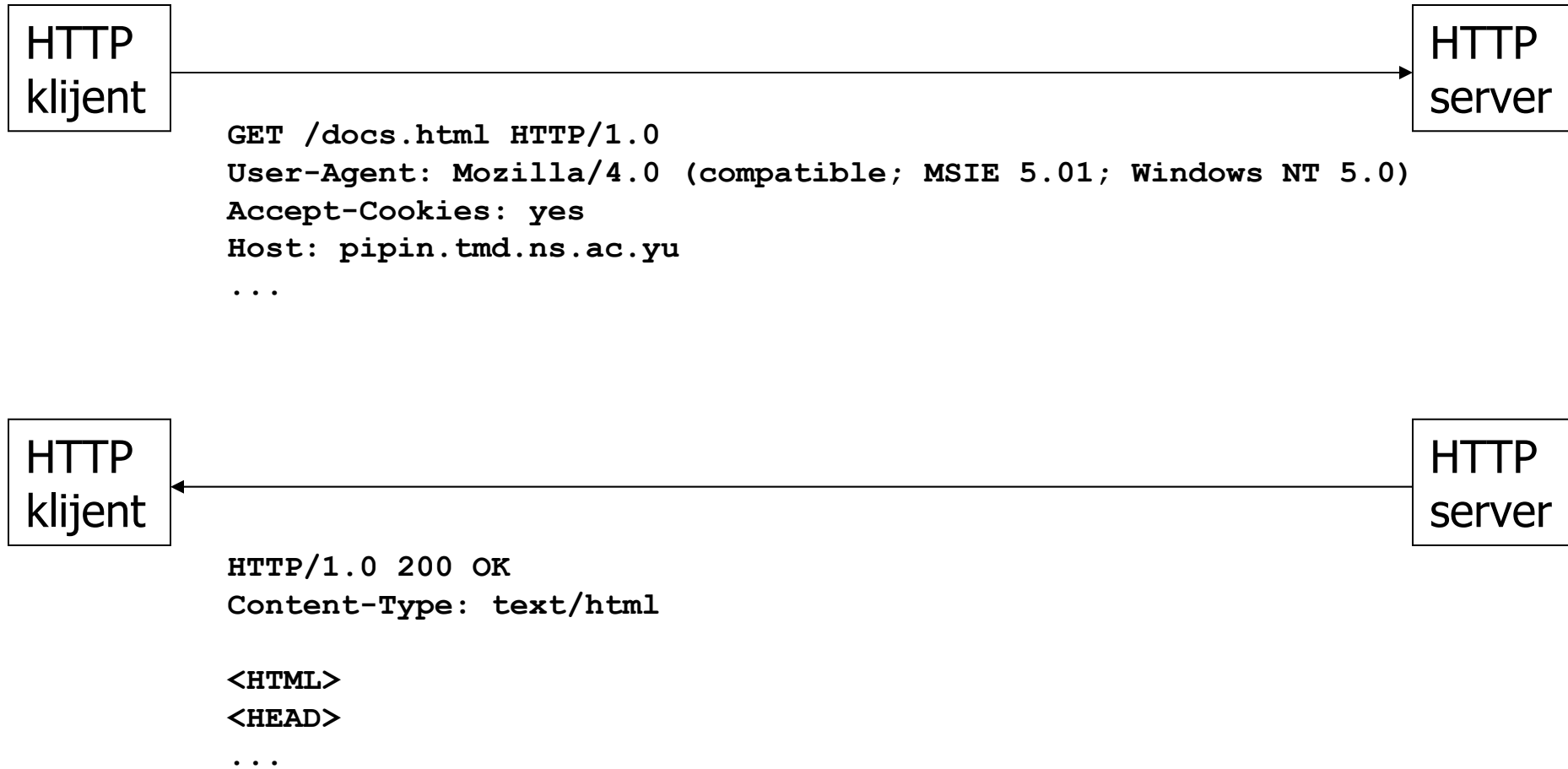
Sadržaj www sajta

- HTML stranice
- multimedijalni elementi (slike, animacije, itd)
- drugi tipovi datoteka
- www server i klijent komuniciraju preko HTTP protokola
- Verzije
 - HTTP/1.0
 - HTTP/1.1 (permanent connection)

HTTP komunikacija _{1/2}

- zasnovana na zahtev/odgovor principu
- svaki par zahtev/odgovor se smatra nezavisnim od ostalih
- ne omogućava praćenje korisničke sesije, tj. niza zahteva upućenih od strane istog klijenta

HTTP komunikacija 2/2



HTTP zahtev

- Počinje redom:
METHOD /putanja HTTP/verzija
- METHOD je:
 - GET,
 - POST, i dr.
- dodatni redovi sadrže attribute oblika:
Ime: vrednost
- prazan red na kraju

METHOD

- GET – zahteva resurs od web servera
- POST – šalje parametre forme i traži odgovor
- HEAD – zahteva samo HTTP odgovor (response), bez slanja samog resursa
- PUT – omogućava klijentu da pošalje datoteku na web server
- OPTIONS – od web servera se traži spisak metoda koje podržava
- DELETE – omogućava klijentu da obriše resurs sa web servera

Atributi u HTTP zahtevu

- User-Agent – identifikuje web browser

User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.2) Gecko/20070219 Firefox/2.0.0.2

- Accept – definiše koje tipove resursa navigator prihvata kao odgovor na ovaj zahtev

Accept:

text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5

- Accept-Language – definiše koji jezike očekuje kao odgovor

Accept-Language: en-us,en;q=0.5

- Accept-Encoding – definiše koje kodiranje očekuje kao odgovor

Accept-Encoding: gzip,deflate

Atributi u HTTP zahtevu

- Accept-Charset – definiše koju kodnu stranu očekuje

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

- Cookie – definiše mehanizam praćenja sesije

Cookie: id1172566682241_1=1172566682241_1

- Referer – definiše URL sa kojeg se došlo na ovu stranicu
 - koristi se za statistiku
 - hotlinking

Referer: http://localhost/

- Connection – HTTP1.1 "kaže" serveru da ne zatvara konekciju po isporuci resursa

Connection: Keep-Alive

- q=broj definiše *qvalue*, odn. floating point vrednost "težine" parametra

Primer HTTP zahteva

GET / HTTP/1.1

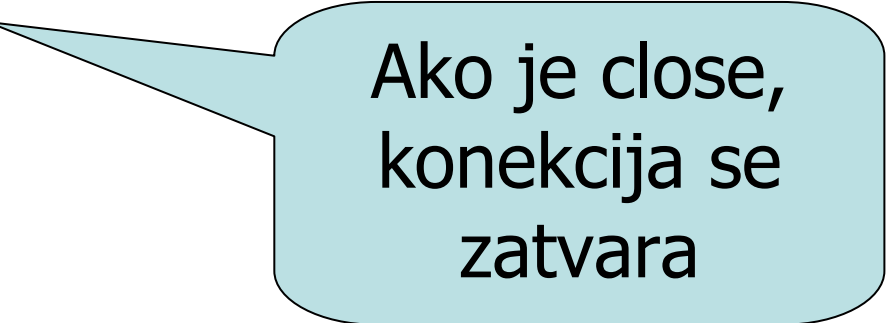
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/vnd.ms-powerpoint,
application/msword, application/x-shockwave-flash, */*

Accept-Language: sr

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1; .NET CLR 1.1.4322)

Host: localhost

Connection: Keep-Alive



Ako je close,
konekcija se
zatvara

HTTP odgovor

- Počinje redom:
HTTP/verzija kod tekstualni_opis
- dodatni redovi sadrže attribute:
Ime: vrednost
- prazan red
- sledi sadržaj datoteke

"200" ; OK

"201" ; Created

"202" ; Accepted

"204" ; No Content

"301" ; Moved
Permanently

"302" ; Moved
Temporarily

"304" ; Not Modified

"400" ; Bad Request

"401" ; Unauthorized

"403" ; Forbidden

"404" ; Not Found

"500" ; Internal Server
Error

"501" ; Not
Implemented

"502" ; Bad Gateway

"503" ; Service
Unavailable

Atributi u HTTP odgovoru

- Content-type – definiše tip odgovora

Content-Type: text/html

- Cache-Control – definiše kako se keš na klijentu ažurira
 - koristi se i Pragma: no-cache

Cache-Control: no-cache

- Location – definiše novu adresu kod redirekcije

Location: new.html

- Connection – HTTP1.1 potvrda klijentu da li da zatvori konekciju ili da je ostavi otvorenu

Connection: Keep-Alive

Primer HTTP odgovora

HTTP/1.0 200 OK

Date: Tue, 04 May 02004 08:55:09 GMT

Status: 200

Servlet-Engine: Tomcat Web Server/3.1 (JSP 1.1; Servlet 2.2; Java 1.4.2_02; Windows XP 5.1 x86; java.vendor=Sun Microsystems Inc.)

Content-Type: text/html

Last-Modified: Fri, 24 Oct 02003 16:07:24 GMT

Content-Length: 2524

Content-Language: en

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
```

```
<html>
```

```
<head>
```

```
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
```

```
  <meta name="GENERATOR" content="Mozilla/4.72 [en] (WinNT; U) [Netscape]">
```

```
  <meta name="Author" content="Anil K. Vijendran">
```

```
  <title>Tomcat v3.1</title>
```

```
</head>
```

```
<body></body>
```

```
</html>
```

Primer HTTP odgovora sa redirekcijom

HTTP/1.1 302 Object moved
Server: Microsoft-IIS/5.1
Date: Mon, 26 Apr 2004 17:50:55 GMT
X-Powered-By: ASP.NET
Location: localstart.asp
Connection: Keep-Alive
Content-Length: 135
Content-Type: text/html
Set-Cookie: ASPSESSIONIDGGQQAQAEK=JKKPPFKCMNDNMEEHOHAADJKPM; path=/
Cache-control: private

Ako je close,
konekcija se
zatvara

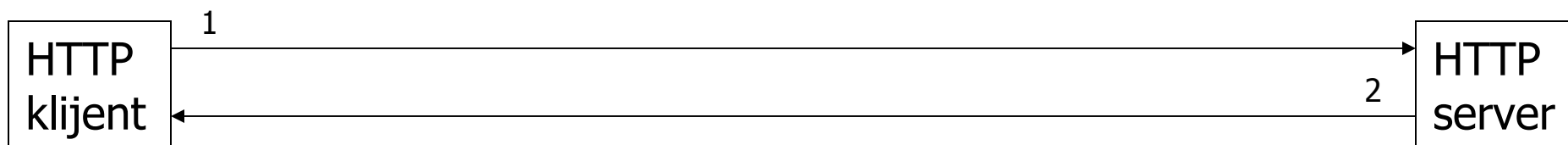
```
<html>
<head>
  <title>Object moved</title>
</head>
<body><h1>Object Moved</h1>This object may be found <a
  HREF="localstart.asp">here</a>.
</body>
<html>
```

Vrste WWW sadržaja

- statički (unapred uskladišteni)
- dinamički (generisani po zahtevu)

Isporuka statičkih sadržaja

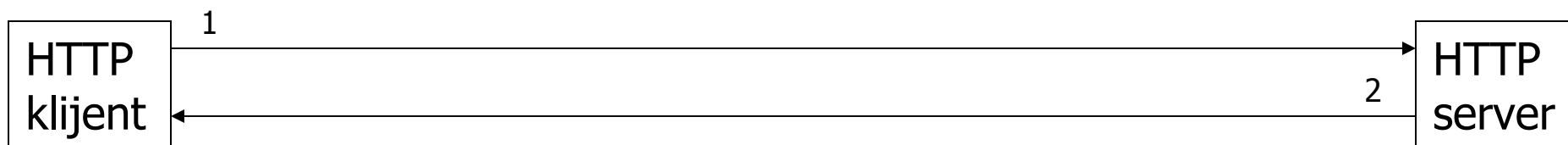
- statički sadržaji se nalaze u okviru datoteka WWW servera



1. klijent zahteva datoteku
2. server je učitava sa svog fajl-sistema i šalje je klijentu

Isporuka dinamičkih sadržaja

- traženi sadržaj se generiše po zahtevu i šalje klijentu



1. klijent zahteva "datoteku"
2. server je generiše i šalje klijentu; ne snima je u svoj fajl-sistem

Preuzimanje podataka sa formi

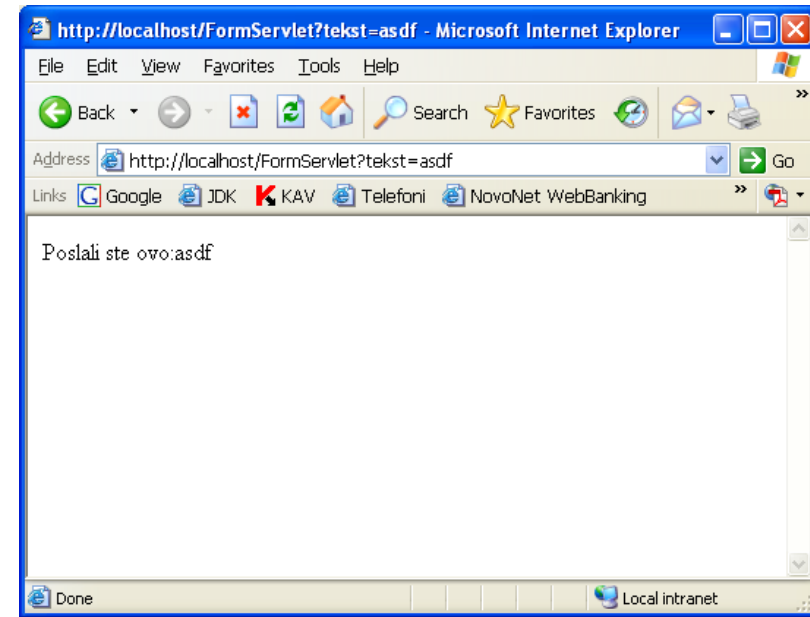
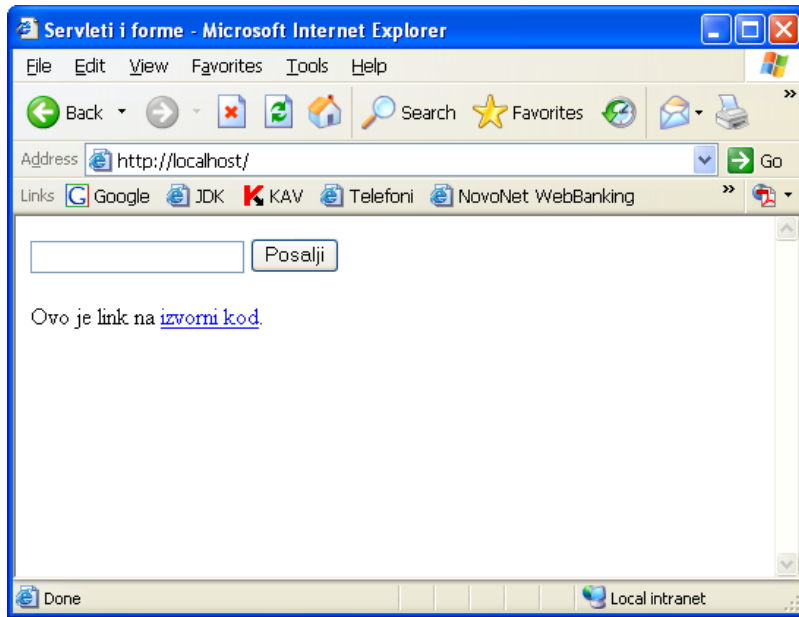
- HTML kod na klijentu:



accept-charset="utf-8"

```
<form method="get" action="FormServlet">  
  <input type="text" name="tekst">  
  <input type="submit" value="Posalji">  
</form>
```

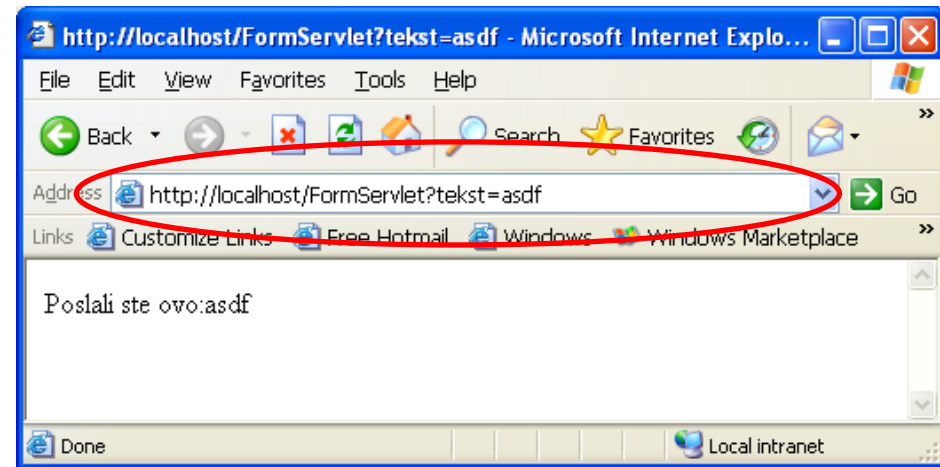
Preuzimanje podataka sa formi



GET i POST zahtevi

GET /FormServlet?tekst=asdf HTTP/1.1

...



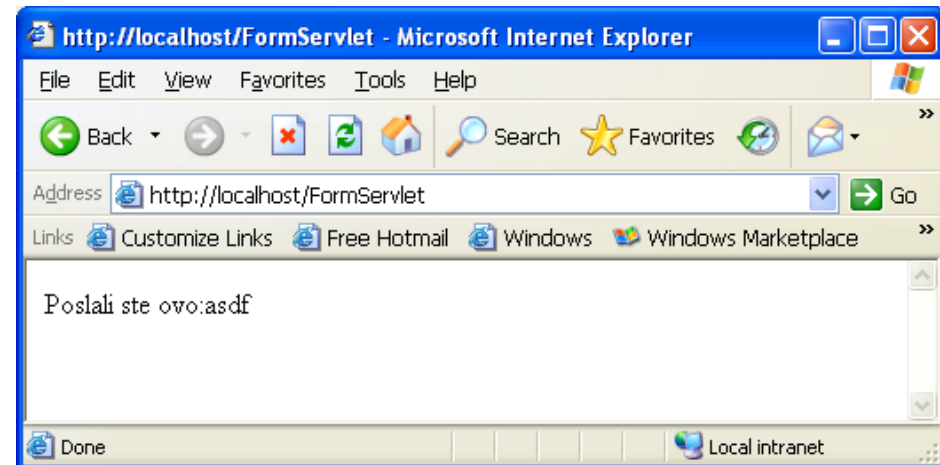
POST /FormServlet HTTP/1.1

...

Content-length: 10

...

tekst=asdf



Praćenje sesije korisnika _{1/3}

- HTTP protokol ne prati sesiju – veza klijenta i servera se zatvara po isporuci resursa.
- Koristi se *cookie* mehanizam:
 - Server šalje *cookie* klijentu u okviru http response
 - klijent čuva primljeni *cookie* i šalje ga uz svaki http request.
- Ako navigator ne prihvata *cookie-je*, koristi se URL Rewriting mehanizam:

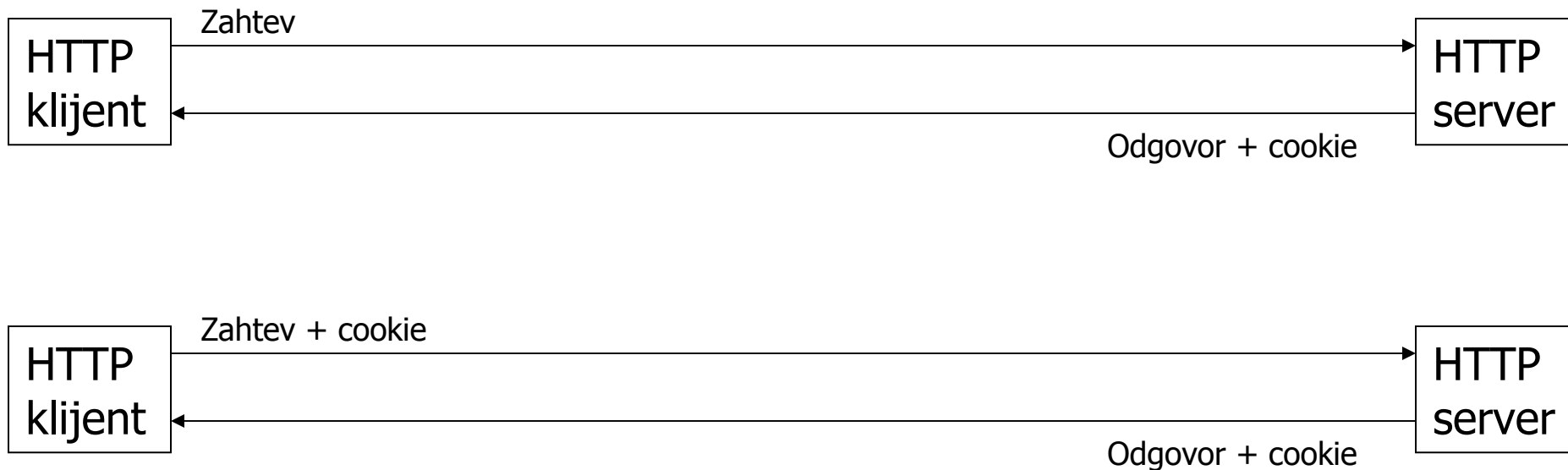
```
<a href="http://www.myserver.com/catalog/index.html;jsessionid=1234">
```

```
link
```

```
</a>
```

Praćenje sesije korisnika _{2/3}

- *cookie* mehanizam



Praćenje sesije korisnika 3/3

GET / HTTP/1.1

Accept-Language: sr

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)

Host: localhost

Connection: Keep-Alive

HTTP/1.0 200 OK

Date: Tue, 04 May 02004 08:55:09 GMT

Status: 200

Content-Type: text/html

Content-Length: 2524

Content-Language: en

Set-Cookie: ASPSESSIONIDGGQQAQAEK=JKKPPFKCMNDNMEEHOHAADJKPM

<html><head></head>

<body></body>

</html>

GET /Test HTTP/1.1

Accept-Language: sr

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)

Host: localhost

Connection: Keep-Alive

Cookie: ASPSESSIONIDGGQQAQAEK=JKKPPFKCMNDNMEEHOHAADJKPM

Jednostavan web server

web server

- sluša na portu 80
- po uspostavi veze, prima jedan red
- ako je to komanda GET, pokupi adresu resursa
- pošalje zaglavlje HTTP protokola
- pošalje resurs
- zatvori vezu sa klijentom

Glavna petlja web servera

```
ServerSocket srvr = new ServerSocket(port);
Socket skt = null;
while(true) {
    try {
        // čekamo klijenta
        skt = srvr.accept();
        // kupimo adresu resursa
        String resource = getResource(skt.getInputStream());
        // obrada "default" stranice
        if (resource.equals(""))
            resource = "index.html";
        // pošaljemo resurs
        sendResponse(resource, skt.getOutputStream());
        // zatvorimo konekciju
        skt.close();
        skt = null;
    } catch(IOException ex) {
        ex.printStackTrace();
    }
}
```

HTTP zahtev (request)

- svaki HTTP zahtev je oblika:
METODA /resurs HTTP/verzija
DODATNI_REDOVI
- METODA može da bude:
 - GET,
 - POST i dr.
- resurs je putanja do datoteke koja će se preneti na klijentsku stranu
- verzija je:
 - 1.0
 - 1.1
- Dodatni redovi su oblika:
 - Naziv: vrednost

Obrada HTTP zahteva

```
BufferedReader dis
    = new BufferedReader(new InputStreamReader(is));
// kupimo prvi red
String s = dis.readLine();
// tokenizujemo po razmacima
StringTokenizer hdr = new StringTokenizer(s);
// kupimo metod (prvi token)
String method = hdr.nextToken();
if (!method.equals("GET")) {
    return null;
}
// kupimo drugi token, a to je putanja do resursa
String rsrc = hdr.nextToken();
//sklonimo karakter '/' sa početka
rsrc = rsrc.substring(1);
// ignorišemo ostatak zahteva
String s1;
while (!(s1 = dis.readLine()).equals(""))
    System.out.println(s1);
return rsrc;
```

HTTP odgovor (response)

- Počinje redom:
HTTP/verzija kod tekstualni_opis
- dodatni redovi sadrže attribute:
Ime: vrednost
- prazan red
- sledi sadržaj resursa (datoteke)

Slanje resursa (datoteke)

```
PrintStream ps = new PrintStream(os);
// zamenimo web separator ('/') sistemskim separatorom
resource = resource.replace('/', File.separatorChar);
// pristupimo resursu
File file = new File(resource);
// ako ne postoji
if (!file.exists()) {
    // greška za nepostojeći resurs je 404
    ps.print("HTTP/1.0 404 File not Found\r\n\r\n");
    ps.flush();
    return;
}
// ispišemo zaglavlje HTTP odgovora (prvi red i prazan red)
ps.print("HTTP/1.0 200 OK\r\n\r\n");
// pošaljemo sadržaj resursa (datoteke)
FileInputStream fis = new FileInputStream(file);
byte[] data = new byte[8192];
int len;
while((len = fis.read(data)) != -1) {
    ps.write(data, 0, len);
}
ps.flush();
fis.close();
```

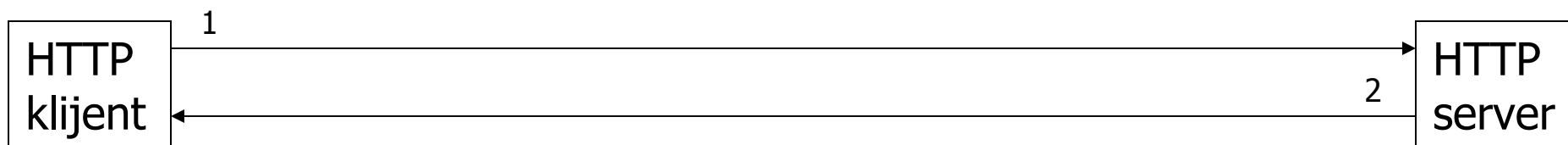
Dinamičko generisanje HTML-a i servleti

Vrste WWW sadržaja

- statički (unapred uskladišteni)
- dinamički (generisani po zahtevu)

Isporuka statičkih sadržaja

- statički sadržaji se nalaze u okviru datoteka WWW servera



1. klijent zahteva datoteku
2. server je učitava sa svog fajl-sistema i šalje je klijentu

Isporuka dinamičkih sadržaja

- traženi sadržaj se generiše po zahtevu i šalje klijentu



1. klijent zahteva "datoteku"
2. server je generiše i šalje klijentu; ne snima je u svoj fajl-sistem

Servleti _{1/2}

- Tehnologija za generisanje dinamičkih sadržaja
- WWW server se proširuje podrškom za servlete
- Rezultat izvršenja servleta je dinamički kreiran sadržaj

Servleti 2/2

redefinisati
metodu:

doGet(...)
doPost(...)

- klasa koja nasleđuje klasu `HttpServlet`:

```
public abstract class HttpServlet {
    protected void init(ServletConfig cnf) {}
    protected void doGet(HttpServletRequest request, HttpServletResponse response) {}
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        {}
    protected void doPut(HttpServletRequest request, HttpServletResponse response)
        {}
    protected void doHead(HttpServletRequest request, HttpServletResponse response)
        {}
    protected void delete(HttpServletRequest request, HttpServletResponse response)
        {}
    protected void doOptions(HttpServletRequest request, HttpServletResponse
        response) {}
    protected void doTrace(HttpServletRequest request, HttpServletResponse response)
        {}
    protected void destroy() {}
    protected void service(HttpServletRequest request, HttpServletResponse response)
    {
        if (request.getMethod().equals("GET"))
            doGet(request, response);
        else if (request.getMethod().equals("POST"))
            doPost(request, response);
        else if ...
    }
}
```

HttpServlet.init()

- namenjena za inicijalizaciju prilikom pokretanja servleta

```
public void init() {  
    Connection conn = DriverManager.getConnection(...);  
    ...  
}
```

```
public void init(ServletConfig cnf) {  
    super.init(cnf);  
    Connection conn = DriverManager.getConnection(...);  
    ...  
}
```

HttpServlet.destroy()

- namenjena za clean-up zadatke neposredno pre uništenja servleta

```
public void destroy() {  
    conn.close();  
}
```

HttpServlet.doGet()

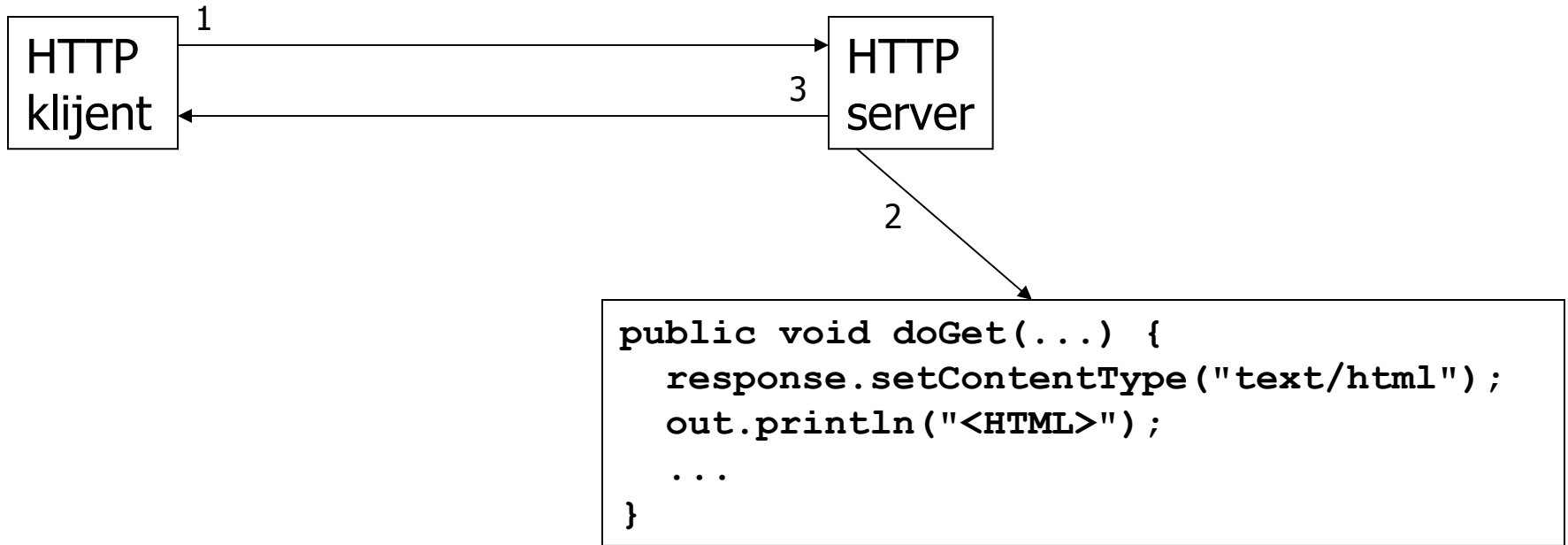
- Svaki poziv servleta se svodi na poziv ove metode
- Namenjena za obradu GET zahteva
- Tipičan scenario poziva:
 - postavi Content-type HTTP odgovora
 - uzmi PrintWriter ka klijentu
 - kroz PrintWriter šalji dinamički kreiran sadržaj

```
public void doGet(HttpServletRequest req, HttpServletResponse res) {  
    res.setContentType("text/html");  
    PrintWriter out = res.getWriter();  
    out.println("<HTML>");  
    out.println("<HEAD><TITLE>Test</TITLE></HEAD>");  
    out.println("<BODY>");  
    ...  
}
```

Konkurentni pristup servletu

- za svaku servlet klasu instancira se tačno jedan objekat koji opslužuje sve klijente
- njegove `doGet()` i `doPost()` metode mogu biti istovremeno pozvane iz više programskih niti Web servera

Generisanje dinamičkih sadržaja



Modifikacija web servera

- Uvode se servleti
- Naziv resursa iz HTTP zaglavlja se koristi za pretragu postojećih servleta
- Ako je pronađen, poziva se njegova **service()** metoda, koja poziva **doGet()**, **doPost()** ili neku drugu metodu
- Ako nije pronađen, u pitanju je resurs, pa se on vraća (ako postoji)

Glavna petlja web servera

```
// iz inicijalizacije datoteke pokupimo spisak svih servleta
collectServlets();
while(true) {
    // cekamo na klijenta
    skt = srvr.accept();
    // pripremimo objekat koji reprezentuje zahtev od klijenta
    request = new HttpServletRequest(skt.getInputStream());
    // preuzmemo uri do resursa
    String resource = request.getResource();
    // pripremimo objekat koji reprezentuje odgovor servera
    response = new HttpServletResponse(skt.getOutputStream());
    // pripremimo pracenje sesije
    handleCookies(request, response);
    // potrazimo servlet na osnovu imena
    HttpServlet s = findServlet(resource);
    if (s != null) // ako smo ga nasli, startujemo ga
        s.service(request, response);
    else // ako ne, onda je to staticki web sadrzaj
        sendResponse(resource, response);
    skt.close();
    skt = null;
}
```


Prikupljanje servleta _{1/2}

- Konfiguraciona datoteka (httpd.conf):
- Format:
Alias=NazivJavaKlase
- Alias se navodi u okviru URI do servleta:
Ovo je link na servlet.
- Primer sadržaja httpd.conf datoteke:
Test=TestServlet

Prikupljanje servleta 2/2

```
private void collectServlets() {
    BufferedReader bin = new BufferedReader(
        new FileReader("httpd.conf"));

    String s, sName, sClass;
    int idx;
    while ((s = bin.readLine()) != null) {
        if (s.trim().equals(""))
            continue;
        idx = s.indexOf("=");
        sName = s.substring(0, idx);
        sClass = s.substring(idx+1);
        HttpServlet srv =
            (HttpServlet)Class.forName(sClass).newInstance();
        servletMap.put(sName, srv);
    }
}
```

HTTP zahtev

- Počinje redom:
METHOD /putanja HTTP/verzija
- METHOD je:
 - GET,
 - POST,
 - HEAD,
 - PUT,
 - DELETE,
 - OPTIONS,
 - TRACE.
- dodatni redovi sadrže attribute oblika:
Ime: vrednost
- prazan red na kraju

HTTP zahtev (klasa HttpServletRequest) _{1/3}

- Reprezentuje HTTP zahtev
- Izdvaja parametre forme prenete GET ili POST metodom i smešta ih u asocijativnu listu (naziv_polja_iz_forme, vrednost)
 - metode `getParameter(ime)`, `getParameterNames()`, `getParameterMap()`
- Prikuplja sve parametre zaglavlja HTTP zahteva i smešta ih u asocijativnu listu (naziv, vrednost)
 - metode `getHeader(ime)`, `getHeaderNames()` i `getHeaders(ime)`

HTTP zahtev (klasa HttpServletRequest) 2/3

```
public HttpServletRequest(InputStream is) {  
    ...  
    BufferedReader rdr  
        = new BufferedReader(new InputStreamReader(is));  
    // kupimo prvi red iz http zahteva  
    String s = rdr.readLine();  
    StringTokenizer hdr = new StringTokenizer(s);  
    // kupimo METHOD  
    method = hdr.nextToken();  
    // preskočili smo METHOD, pa je sledeći string putanja do resursa  
    String rsrc = hdr.nextToken();  
    //izbacimo vodeći '/' znak  
    rsrc = rsrc.substring(1);  
    // izdvojimo parametre GET metode forme (ako ih ima),  
    // a ostatak je putanja do resursa  
    resource = extractGetParameters(rsrc);  
    // iščitamo zaglavlje http zahteva i popunimo asocijativnu listu  
    // parametara iz zaglavlja  
    readHeader(rdr);  
    ...  
}
```

HTTP zahtev (klasa HttpServletRequest) 3/3

```
/** Čita parametre http zaglavlja i smešta ih u asocijativnu mapu. */
private void readHeader(BufferedReader rdr) {
    try {
        String s1, name, value;
        while (!(s1 = rdr.readLine()).equals("")) {
            System.out.println(s1);
            int idx = s1.indexOf(":");
            if (idx != -1) {
                name = s1.substring(0, idx); // levo od ':' je ime
                value = s1.substring(idx+1); // desno od ':' je vrednost
                headerMap.put(name.toUpperCase(), value.trim());
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

/** Vraća vrednost parametra iz http zaglavlja (ako ga ima). */
public String getHeader(String name) {
    return (String)headerMap.get(name.toUpperCase());
}
```

HTTP odgovor

- Počinje redom:
HTTP/verzija kod tekstualni_opis
- dodatni redovi sadrže attribute:
Ime: vrednost
- prazan red
- sledi sadržaj datoteke

"200" ; OK

"201" ; Created

"202" ; Accepted

"204" ; No Content

"301" ; Moved
Permanently

"302" ; Moved
Temporarily

"304" ; Not Modified

"400" ; Bad Request

"401" ; Unauthorized

"403" ; Forbidden

"404" ; Not Found

"500" ; Internal Server
Error

"501" ; Not
Implemented

"502" ; Bad Gateway

"503" ; Service
Unavailable

HTTP odgovor (klasa `HttpServletResponse`) ^{1/4}

- Reprezentuje HTTP odgovor
- Čuva tip odgovora (atribut `Content-Type`)
 - metoda `setContentType(vrednost)`
- Čuva *cookie* (atribut `SetCookie`)
 - metoda `addCookie(cookie)`
- Omogućuje redirekciju (`Location`)
 - metoda `sendRedirect(nova_lokacija)`
- Podešava proizvoljan atribut zaglavlja
 - metoda `setHeader(naziv, vrednost)`
- Ugrađuje ID sesije ako cookies nisu uključeni
 - metode `encodeURL(url)` i `encodeRedirectURL(url)`
- Čuva izlazni tok podataka

HTTP odgovor (klasa HttpServletResponse) 2/4

```
public HttpServletResponse(OutputStream out) {  
    outputStream = out;  
    writer = new PrintWriter(new OutputStreamWriter(out), true);  
}  
  
private PrintWriter writer = null;  
public PrintWriter getWriter() {  
    return writer;  
}  
  
private OutputStream outputStream = null;  
public OutputStream getOutputStream() {  
    return outputStream;  
}  
  
private String location;  
public void sendRedirect(String url) {  
    location = url;  
}
```

HTTP odgovor (klasa HttpServletResponse) 3/4

```
private String contentType = null;
private String getEncoding(String s) {
    String retVal = null;
    String[] tokens = s.split(";");
    if (tokens.length == 2) {
        String token = tokens[1].trim();
        int idx = token.indexOf("=");
        if (idx != -1 && token.substring(0,idx).equals("charset")) {
            retVal = token.substring(idx+1);
        }
    }
    return retVal;
}

public void setContentType(String c) {
    // podesi tip povratne datoteke i...
    contentType = c;
    if (c != null) {
        String encoding = getEncoding(c);
        if (encoding != null) {
            try {
                writer = new PrintWriter(new OutputStreamWriter(outputStream,
                                                                    encoding), true);
            } catch (Exception ex) {}
        }
    }
    // posalji zaglavlje HTTP protokola ka klijentu
    sendHeader();
}
```

HTTP odgovor (klasa HttpServletResponse) 4/4

```
private void sendHeader() {  
    // pošaljemo HTTP zaglavlje  
    if (location == null)  
        writer.print("HTTP/1.0 200 OK\r\n");  
    else {  
        writer.print("HTTP/1.0 302 Object moved\r\n");  
        writer.print("Location: " + location + "\r\n");  
    }  
    if (contentType != null)  
        writer.print("Content-type: " + contentType + "\r\n");  
    if (cookie != null)  
        writer.print("Set-Cookie: " + cookie + "\r\n");  
    writer.print("\r\n");  
    writer.flush();  
}
```

Primer: elementarni servlet

```
public class TestServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) {
        PrintWriter pout = response.getWriter();
        response.setContentType("text/html");
        pout.println("<html>");
        pout.println("<head>");
        pout.println("</head>");
        pout.println("<body>");
        pout.println("Hello World!");
        pout.println("<br>Klijent koji je pozvao ovaj
servlet je: " + request.getHeader("User-Agent"));
        pout.println("</body>");
        pout.println("</html>");
    }
}
```

Preuzimanje podataka iz formi

- Parametri iz forme se za GET metodu smeštaju u zaglavlje GET zahteva.

```
<form method="get" action="FormServlet">  
  <input type="text" name="tekst_polje">  
  <input type="submit" value="Posalji">  
</form>
```

- GET HTTP zahtev:

GET /FormServlet?tekst_polje=asdf HTTP/1.1

Primer: servlet koji ispisuje parametar unet u formi

```
public void doGet(HttpServletRequest request,
    HttpServletResponse response) {
    response.setContentType("text/html");
    PrintWriter pout = response.getWriter();
    pout.println("<html>");
    pout.println("<head>");
    pout.println("</head>");
    pout.println("<body>");
    pout.println("Poslali ste ovo:" +
        request.getParameter("tekst_polje"));
    pout.println("</body>");
    pout.println("</html>");
    pout.flush();
}
```

Izdvajanje parametara iz formi (klasa HttpServletRequest)

```
private String extractGetParameters(String rsrc) {
    StringTokenizer hdr = new StringTokenizer(rsrc, "?");
    if (hdr.countTokens()>1) { // ako imamo parametre forme
        // zapamtimo prvi deo, tj. "putanju", jer cemo to vratiti
        String retVal = hdr.nextToken();
        String s = hdr.nextToken(); // uzmemo parametre
        // izdelimo ih na pojedinačne parove "ime=vrednost"
        hdr = new StringTokenizer(s, "&");
        paramMap.clear();
        while (hdr.hasMoreTokens()) {
            s = hdr.nextToken();
            int idx = s.indexOf("=");
            // levo od '=' je ime
            String pName = s.substring(0, idx);
            // desno od '=' je vrednost
            String pValue = s.substring(idx+1);
            paramMap.put(pName, pValue);
        }
        return retVal;
    } else
        return rsrc;
}
```

Pristup parametrima forme (klasa HttpServletRequest)

```
/** Svi parametri iz forme se smeštaju u  
 * asocijativnu mapu.  
 */  
private Hashtable paramMap = new Hashtable();  
public String getParameter(String name) {  
    return (String)paramMap.get(name);  
}
```


Redirekcija

- Redirekcija se svodi na slanje poruke: 302 Object moved i postavljanje parametra HTTP odgovora: Location: nova_adresa

Primer: servlet vrši redirekciju

```
public void doGet(HttpServletRequest request, HttpServletResponse response) {  
    if (request.getParameter("proba") == null) {  
        response.sendRedirect("index.html");  
    } else {  
        response.setContentType("text/html");  
        PrintWriter pout = response.getWriter();  
        pout.println("<html>");  
        pout.println("<head>");  
        pout.println("</head>");  
        pout.println("<body>");  
        pout.println("Ovo je stranica koja se dobija ako je postavljen parametar" +  
            "<b>proba</b> na vrednost: " +  
            request.getParameter("proba") + "<br>");  
        pout.println("Ovo je <a href=\"RedirectServlet\">link na ovaj isti" +  
            "servlet</a>, bez parametra, da bismo izazvali redirekciju.<br>");  
        pout.println("</body>");  
        pout.println("</html>");  
        pout.flush();  
    }  
}
```

HTTP odgovor (klasa HttpServletResponse)

```
private String location;
public void sendRedirect(String url) {
    location = url;
    sendHeader();
}

private void sendHeader() {
    // pošaljemo HTTP zaglavlje
    if (location == null)
        writer.print("HTTP/1.0 200 OK\r\n");
    else {
        writer.print("HTTP/1.0 302 Object moved\r\n");
        writer.print("Location: " + location + "\r\n");
    }
    if (contentType != null)
        writer.print("Content-type: " + contentType + "\r\n");
    if (cookie != null)
        writer.print("Set-Cookie: " + cookie + "\r\n");
    writer.print("\r\n");
    writer.flush();
}
```

Character Encoding

- Metodom `setContentType` se podešava i *character encoding*:

```
response.setContentType("text/html; charset=UTF-8");
```

- Parametar *charset* definiše kodnu stranu kojom će biti kodirani svi stringovi ka klijentu.

Primer: servlet sa UTF-8 encoding-om

```
public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws java.io.IOException {
    response.setContentType("text/html; charset=UTF-8");
    PrintWriter pout = response.getWriter();
    pout.println("<html>");
    pout.println("<head>");
    pout.println("<meta http-equiv=\"Content-Type\"
        content=\"text/html; charset=UTF-8\">");
    pout.println("</head>");
    pout.println("<body>");
    try {
        pout.println("Ovo je stranica sa UTF-8 karakterima: \u0428
            \u0429<br>");
    } catch (Exception ex) {
        pout.println(ex.getMessage());
    }
    pout.println("</body>");
    pout.println("</html>");
    pout.flush();
}
```

HTTP odgovor (klasa HttpServletResponse)

```
private String contentType = null;
private String getEncoding(String s) {
    String retVal = null;
    String[] tokens = s.split(";");
    if (tokens.length == 2) {
        String token = tokens[1].trim();
        int idx = token.indexOf("=");
        if (idx != -1 && token.substring(0,idx).equals("charset")) {
            retVal = token.substring(idx+1);
        }
    }
    return retVal;
}

public void setContentType(String c) {
    // podesi tip povratne datoteke i...
    contentType = c;
    if (c != null) {
        String encoding = getEncoding(c);
        if (encoding != null) {
            try {
                writer = new PrintWriter(new OutputStreamWriter(outputStream,
                                                                    encoding), true);
            } catch (Exception ex) {}
        }
    }
    // posalji zaglavlje HTTP protokola ka klijentu
    sendHeader();
}
```

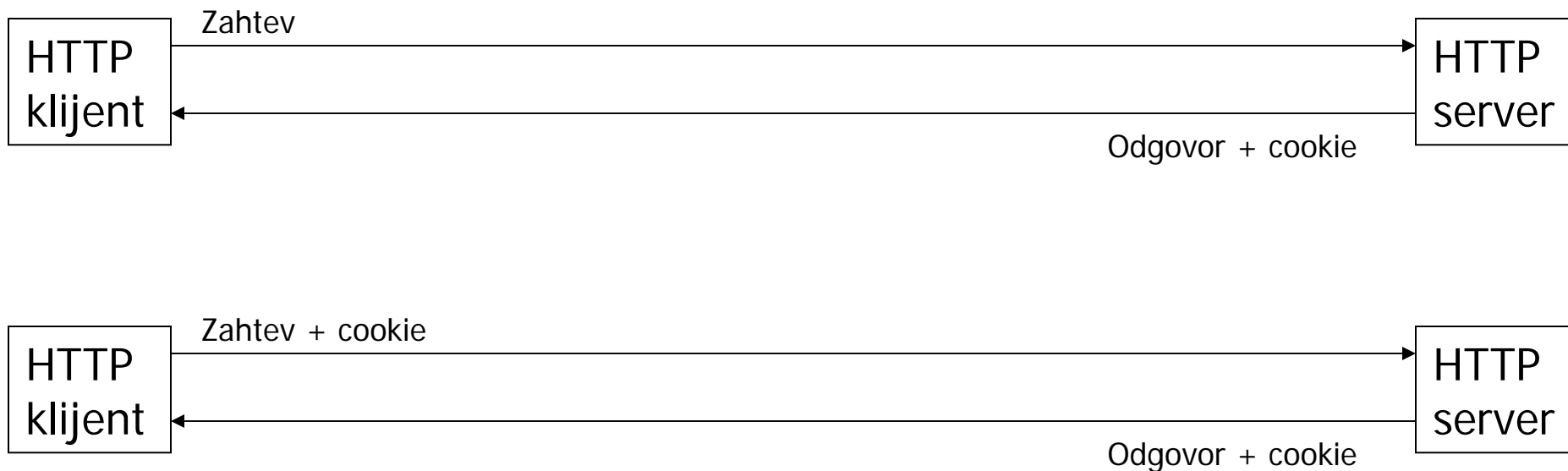
Praćenje sesije

Praćenje sesije korisnika ^{1/3}

- HTTP protokol ne prati sesiju – veza klijenta i servera se zatvara po isporuci resursa.
- Koristi se *cookie* mehanizam:
 - Server šalje *cookie* klijentu u okviru http response
 - klijent čuva primljeni *cookie* i šalje ga uz svaki http request.

Praćenje sesije korisnika ^{2/3}

- kada se koristi cookie mehanizam



Praćenje sesije korisnika 3/3

GET / HTTP/1.1

Accept-Language: sr

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)

Host: localhost

Connection: Keep-Alive

HTTP/1.0 200 OK

Date: Tue, 04 May 02004 08:55:09 GMT

Status: 200

Content-Type: text/html

Content-Length: 2524

Content-Language: en

Set-Cookie: ASPSESSIONIDGGQQAQAEK=JKKPPFKCMNDNMEEHOHAADJKPM

<html><head></head>

<body></body>

</html>

GET /Test HTTP/1.1

Accept-Language: sr

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; .NET CLR 1.1.4322)

Host: localhost

Connection: Keep-Alive

Cookie: ASPSESSIONIDGGQQAQAEK=JKKPPFKCMNDNMEEHOHAADJKPM

Detaljnije o Cookie-ima i HTTP odgovoru

- Vraća se u atributu Set-Cookie
- Opcioni atributi
 - domain – domen u kome važi cookie
 - path – za koje URL-ove na sajtu važi
 - expires – datum isticanja
- Kada ističe cookie?
 - eksplicitan momenat (expires)
 - nedefinisano – kada se ugasi browser

Primer

HTTP/1.1 200 OK

Cache-Control: private

Content-Type: text/html

Set-Cookie:

PREF=ID=3ff1557ca378ed16:TM=1151585607:LM=1151585607:S=rVfoQD15sUOZajEt; expires=Sun, 17-Jan-2038 19:14:07 GMT; path=/; domain=.google.com

Content-Encoding: gzip

Server: GWS/2.1

Content-Length: 1349

Date: Thu, 29 Jun 2006 12:53:27 GMT

Alternativan način

- Ako navigator ne prihvata *cookie-je*, koristi se URL Rewriting mehanizam
- u hiperlink () koji "gađa" naš server ugradimo id sesije:

- `HTTPServletResponse.encodeURL()` metoda
- `HTTPServletResponse.encodeRedirectURL()` metoda
- Da li uvek možemo da koristimo ovu tehniku?
- Gde se stavlja id sesije u formi?

Implementacija *cookie* –a u klasi httpd

```
// pripremimo objekat koji reprezentuje zahtev od
// klijenta
request = new Request(skt.getInputStream());
// izvucemo url do resursa
String resource = request.getResource();

// pripremimo objekat koji reprezentuje odgovor servera
response = new Response(skt.getOutputStream());

// pripremimo praćenje sesije
handleCookies(request, response);

// potrazimo servlet na osnovu imena
Servlet s = findServlet(resource);
if (s != null) // ako smo ga našli, startujemo ga
    s.service(request, response);
else // ako ne, onda je to statički web sadržaj
    sendResponse(resource, response);
```

Implementacija *cookie* –a metodom `handleCookies()`

- Proveri se da li se u HTTP zahtevu nalazi *cookie* (`Cookie: ime=vrednost`)
 - ako se nalazi, to je iz sesije, pa se ta vrednost smešta u http odgovor;
 - ako se ne nalazi, klijent se spaja prvi put, pa se generiše jedinstven *cookie* i smešta se u HTTP odgovor.

Implementacija *cookie* -a

```
/** Odraduje praćenje sesije upotrebom cookie-ja. Ako u http
 * zahtevu nema cookie-ja, generiše nov i spremi ga u
 * response objektu (da se pošalje klijentu prilikom slanja
 * odgovora). Ako cookie postoji, zapamti se za dalje
 * praćenje sesije.
 */
private void handleCookies(Request request, Response response) {
    // pogledamo da li u http zaglavlju postoji parametar Cookie
    String cookieFromRequest = request.getHeader("Cookie");
    if (cookieFromRequest != null) {
        // ako cookie postoji u request-u, onda je došao
        // od našeg servera, pa ga treba zapamtiti
        response.setCookie(cookieFromRequest);
        request.getSession().setId(cookieFromRequest);
    } else { // nema cookie-a u request-u, pa ga treba izgenerisati
        count++;
        String cookie = "id" + System.currentTimeMillis() + "_" + count + "=" +
            System.currentTimeMillis() + "_" + count;;
        response.setCookie(cookie);
        request.getSession().setId(cookie);
    }
}
```


Klasa HttpServletRequest i sesija

```
/** Konstruktor. */
public HttpServletRequest(InputStream is) {
    try {
        // kreiramo objekat koji reprezentuje sesiju.
        session = new HttpSession();
        ...
    }

    /** Objekat klase HttpSession koji čuva podatke
     *  o trenutnoj sesiji.
     */
    private HttpSession session;
    public HttpSession getSession() {
        return session;
    }
}
```

Klasa HttpSession

- Reprezentuje sesiju
- Čuva cookie ili ID sesije za URL redirection
 - metoda getId()
- Čuva objekte vezane za sesiju
 - metode getAttribute(ime), setAttribute(ime, objekat), removeAttribute(ime)
- Invalidira sesiju i razvezuje sve objekte vezane za nju
 - metoda invalidate()
- podešava period neaktivnosti
 - metoda setMaxInactiveInterval(sekunde)

Klasa HttpSession

```
/** Čuva id sesije. */
private String sessionId;
public void setId(String c) {
    sessionId = c;
}
public String getId() {
    return sessionId;
}

/** Asocijativna mapa objekata dodeljenih sesiji.
 * Ako je potrebno da neki objekat prati sesiju, dodaće se u ovu mapu.
 * Ključ je id sesije.
 * Mapa je statička da bi se iz svih zahteva moglo pristupiti
 * dodeljenim objektima.
 */
private static Hashtable sessionMap = new Hashtable();
public Object getAttribute(String key) {
    return sessionMap.get(sessionId+key);
}
public void setAttribute(String key, Object attr) {
    sessionMap.put(sessionId+key, attr);
}
```

Klasa HttpServletResponse i *cookie*

```
private String cookie;
public void setCookie(String c) {
    cookie = c;
}

private void sendHeader() {
    if (location == null)
        writer.print("HTTP/1.0 200 OK\r\n");
    else {
        writer.print("HTTP/1.0 302 Object moved\r\n");
        writer.print("Location: " + location + "\r\n");
    }
    if (contentType != null)
        writer.print("Content-type: " + contentType + "\r\n");

    if (cookie != null)
        writer.print("Set-Cookie: " + cookie + "\r\n");

    writer.print("\r\n");
    writer.flush();
}
```

Praćenje sesije korisnika iz servleta_{1/3}

- Praćenje sesije se svodi na kreiranje objekata koji se vezuju na sesiju
 - objekat bi trebalo da je serijalizabilan
- Kada korisnik prozove neki servlet, u okviru njega se koristi objekat vezan za sesiju:
 - ako do tada nije postojao objekat, on se kreira i veže za sesiju;
 - ako postoji, koristi se.
- Session inactivity – obično 30 minuta
 - podešava se u web.xml
 - `session.setMaxInactiveInterval`

Praćenje sesije korisnika iz servleta 2/3

```
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws java.io.IOException {
    ...
    HttpSession session = request.getSession();
    SessionCounter sc = (SessionCounter)session.getAttribute(
        "brojac");
    if (sc != null) {
        sc.inc();
        out.println(", ukupno pristupa:" + sc.getCount() + "<br>");
    } else {
        out.println(", prvi pristup.<br>");
        sc = new SessionCounter();
        sc.inc();
        session.setAttribute("brojac", sc);
    }
    ...
}
```

Klasa SessionCounter _{3/3}

```
class SessionCounter {  
  
    private int count = 0;  
  
    public int getCount() {  
        return count;  
    }  
  
    public void setCount(int c) {  
        count = c;  
    }  
  
    public void inc() {  
        count++;  
    }  
}
```

Na šta se objekat može vezati?

- Na request:

```
request.setAttribute("ime", referenca);
```

- Na sesiju:

```
request.getSession().setAttribute("ime",  
referenca);
```

- Na aplikaciju:

```
getServletContext().setAttribute("ime",  
referenca);
```


Kako saznati gde se na serveru nalazi folder naše web aplikacije?

- `getServletContext().getRealPath("");`

Case study – web shop

- Dve stranice web shop-a:
 - pregled svih raspoloživih proizvoda
 - dodavanje na spisak ili pregled spiska proizvoda odabranih za kupovinu

Pregled proizvoda

http://localhost/Index - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media

Address http://localhost/Index Go

Links javni deo admin deo CSI administracija Odsek za računarstvo i automatiku

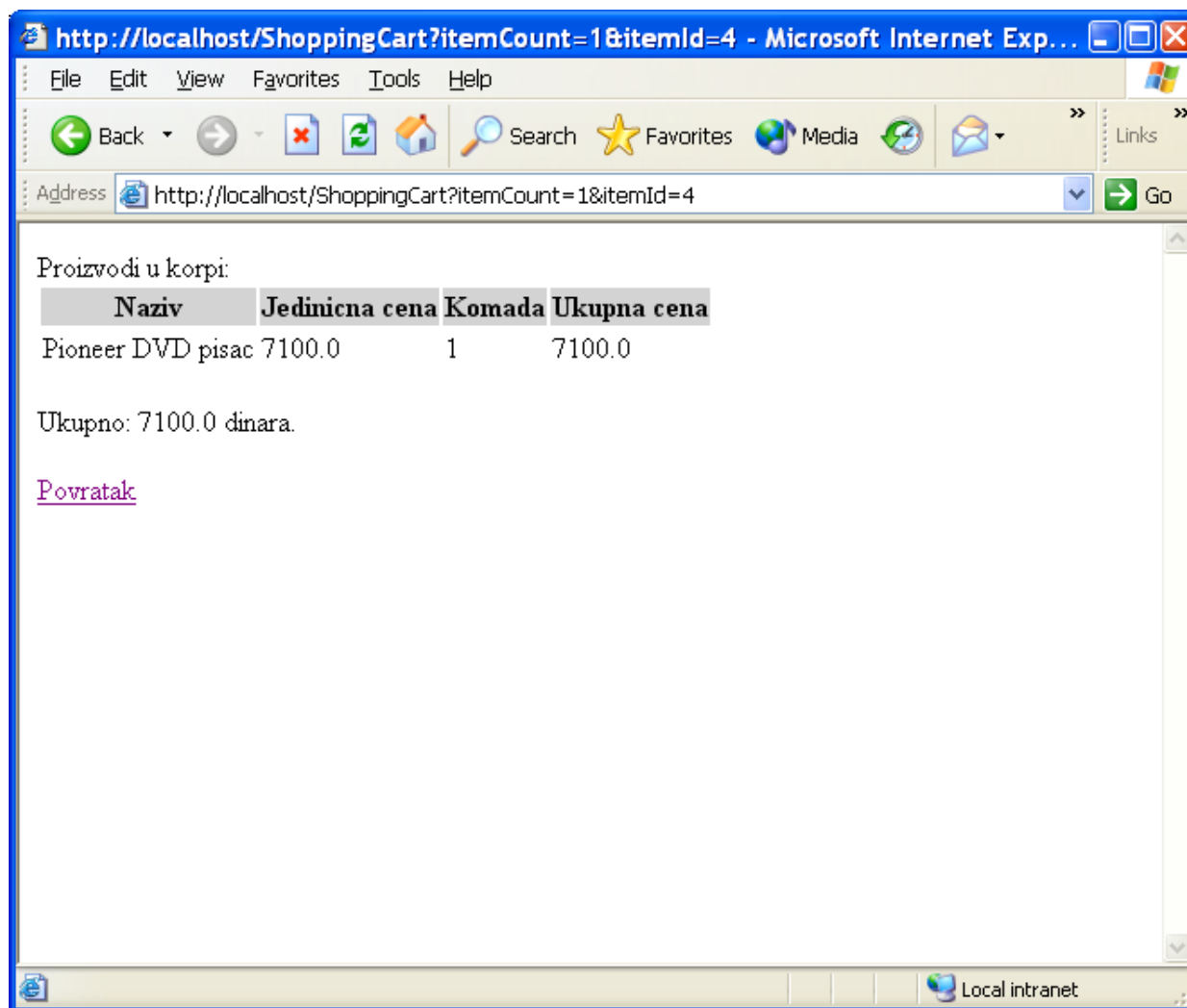
Raspoloživi proizvodi:

Naziv	Cena	
Pioneer DVD pisac	7100.0	<input type="text"/> Dodaj
Samsung monitor 17"	35000.0	<input type="text"/> Dodaj
Sony digitalna kamera	32000.0	<input type="text"/> Dodaj
Televizor marke Sony, 51 cm dijagonala	22000.0	<input type="text"/> Dodaj

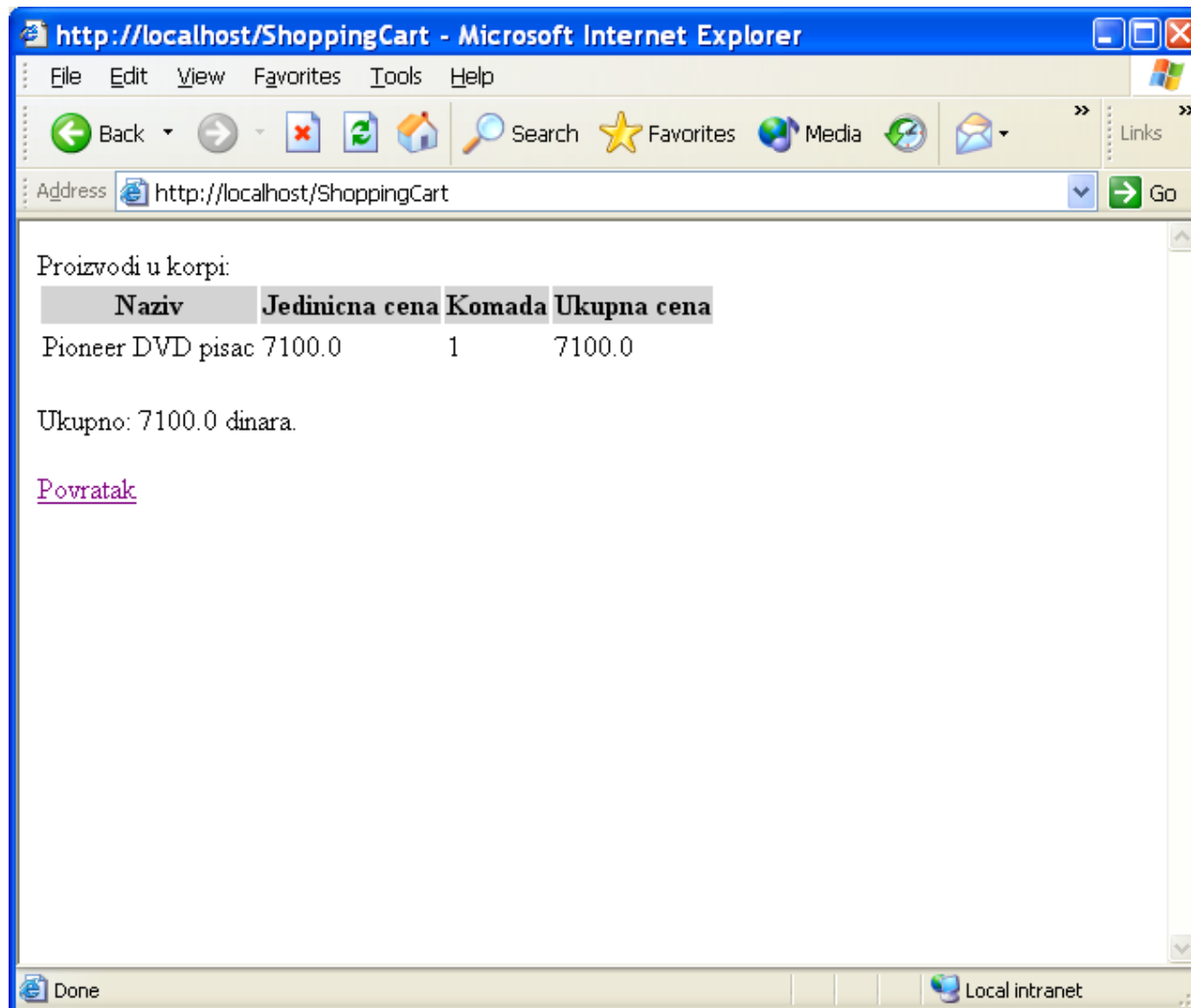
[Pregled sadržaja korpe](#)

Done Local intranet

Dodavanje proizvoda na listu za kupovinu (shopping cart)



Pregled liste za kupovinu



Lista raspoloživih proizvoda (WebShopServlet)

```
ServletContext ctx = getServletContext();
Products products = new Products(ctx.getRealPath(""));
ctx.setAttribute("products", products);
...
pout.println("Raspoloživi proizvodi:");
pout.println("<table border=\"1\"><tr
    bgcolor=\"lightgrey\"><th>Naziv</th><th>Cena</th><th>&nbsp;</th></tr>");
for (Product p : products.values()) {
    pout.println("<tr>");
    pout.println("<form method=\"get\" action=\"ShoppingCartServlet\">");
    pout.println("<td>" + p.getName() + "</td>");
    pout.println("<td>" + p.getPrice() + "</td>");
    pout.println("<td>");
    pout.println("<input type=\"text\" size=\"3\" name=\"itemCount\">");
    pout.println("<input type=\"hidden\" name=\"itemId\" value=\""
        + p.getId() + "\">");
    pout.println("<input type=\"submit\" value=\"Dodaj\">");
    pout.println("</form>");
    pout.println("</td>");
    pout.println("</tr>");
}
pout.println("</table>");
```

Dodavanje/pregled proizvoda u listu za kupovinu (ShoppingCartServlet)

```
HttpSession session = request.getSession();
ShoppingCart sc = (ShoppingCart)session.getAttribute("ShoppingCart");
if (sc == null) {
    // ako ne postoji, kreiramo ga i dodelimo tekucioj sesiji.
    // Na ovaj nacin, objekat klase ShoppingCart ce pratiti sesiju.
    sc = new ShoppingCart();
    session.setAttribute("ShoppingCart", sc);
}
PrintWriter pout = response.getWriter();
response.setContentType("text/html");
if (request.getParameter("itemId") != null) {
    // ako smo pozvali ovaj servlet sa parametrima za dodavanje proizvoda u korpu
    try {
        Products products = (Products)getServletContext().getAttribute("products");
        // probamo da ga dodamo
        sc.addItem(
            products.getProduct(request.getParameter("itemId")),
            Integer.parseInt(request.getParameter("itemCount")));
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
pout.println("Proizvodi u korpi:");
pout.println("<table><tr bgcolor=\"lightgrey\"><th>Naziv</th><th>Jedinicna  
cena</th><th>Komada</th><th>Ukupna cena</th></tr>");
double total = 0;
for (ShoppingCartItem i : sc.getItems()) {
    pout.println("<tr>");
    pout.println("<td>" + i.getProduct().getName() + "</td>");
    ...
}
```

Datoteka products.txt

1;Televizor marke Sony, 51 cm dijagonala;22000
2;Sony digitalna kamera;32000
3;Samsung monitor 17";35000
4;Pioneer DVD pisac;7100

Paket webshop – klasa Products ^{1/4}

- Reprezentuje spisak proizvoda
- Iz datoteke products.txt učitava spisak proizvoda i smešta u asocijativnu listu (ključ je id proizvoda, a objekat koji se smešta u listu je klase *Product*)

Paket webshop – klasa Products ^{2/4}

```
public Products(String path) {  
    BufferedReader in = null;  
    try {  
        in = new BufferedReader(new FileReader(  
                                path + "/products.txt"));  
        readProducts(in);  
        in.close();  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

Paket webshop – klasa Products 3/4

```
private void readProducts(BufferedReader in) {
    String line, id = "", name = "", price = "";
    StringTokenizer st;
    int pos;
    try {
        while ((line = in.readLine()) != null) {
            line = line.trim();
            if (line.equals("") || line.indexOf('#') == 0)
                continue;
            st = new StringTokenizer(line, ";");
            while (st.hasMoreTokens()) {
                id = st.nextToken().trim();
                name = st.nextToken().trim();
                price = st.nextToken().trim();
            }
            products.put(id, new Product(id, name,
                Double.parseDouble(price)));
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

Paket webshop – klasa Products ^{4/4}

```
private HashMap<String, Product> products =  
new HashMap<String, Product>();
```

```
/** Vraca kolekciju proizvoda. */  
public Collection<Product> values() {  
    return products.values();  
}
```

```
/** Vraca proizvod na osnovu njegovog id-a.  
*/  
public Product getProduct(String id) {  
    return (Product) products.get(id);  
}
```

Paket webshop – klasa Product _{1/2}

- Reprezentuje pojedinačan proizvod
- Karakterišu je:
 - id,
 - naziv i
 - cena.

Paket webshop – klasa Product_{2/2}

```
public class Product {
    private String id;
    public void setId(String i) {
        id = i;
    }
    public String getId() {
        return id;
    }
    private String name;
    public void setName(String n) {
        name = n;
    }
    public String getName() {
        return name;
    }
    private double price;
    public void setPrice(double p) {
        price = p;
    }
    public double getPrice() {
        return price;
    }
    public Product(String id, String name, double price) {
        this.id = id;
        this.name = name;
        this.price = price;
    }
}
```

Paket webshop – klasa ShoppingCart ^{1/2}

- Reprezentuje listu proizvoda za kupovinu (korpu, shopping cart)
- Sadrži listu odabranih proizvoda za kupovinu (svaki odabrani proizvod je reprezentovan objektom klase *ShoppingCartItem*)

Paket webshop – klasa ShoppingCart_{2/2}

```
public class ShoppingCart {
    ArrayList <ShoppingCartItem> items;

    public ShoppingCart() {
        items = new ArrayList <ShoppingCartItem>();
    }

    public void addItem(Product product, int count) {
        items.add(new ShoppingCartItem(product, count));
    }

    public ArrayList <ShoppingCartItem> getItems() {
        return items;
    }
}
```


Paket webshop – klasa ShoppingCartItem _{1/2}

- Reprezentuje jednu stavku iz spiska proizvoda koje je mušterija odabrala za kupovinu
- Čuva:
 - referencu na proizvod (referencu na objekat klase *Product*),
 - količinu.

Paket webshop – klasa ShoppingCartItem _{2/2}

```
public class ShoppingCartItem {  
  
    private Product product;  
    public void setProduct(Product p) {  
        product = p;  
    }  
    public Product getProduct() {  
        return product;  
    }  
  
    private int count;  
    public void setCount(int c) {  
        count = c;  
    }  
    public int getCount() {  
        return count;  
    }  
  
    public ShoppingCartItem(Product p, int count) {  
        this.product = p;  
        this.count = count;  
    }  
}
```

Prijava na sistem

- Klasa User:
 - reprezentuje korisnika
 - vezuje se na sesiju
 - prilikom prijavljivanja na sistem
 - sadrži Shopping Cart kao atribut
 - sadrži informaciju o tome da li je korisnik prijavljen

Odjava sa sistema

- Korisnik inicira odjavu
- Klasa User sadrži informaciju o prijavi/odjavi
- Šta ako se korisnik odjavi, pa pritisne *Back* u navigatoru?
 - mora se zapamtiti da je odjavljen
 - sesija se poništava – *session.invalidate()*
 - podesiti *Cache-Control* na *no-cache*
 - `response.setHeader("Cache-Control", "no-cache")`
 - META tag sa istim sadržajem

Cross-site scripting

- Ubacuje se JavaScript u polja forme, a čiji sadržaj će se prikazati na nekoj stranici
 - prilikom prikaza takvih podataka, izvršava se JavaScript
- Rešenje: očistiti podatke koji se unose u formu

POST metoda i slanje
datoteka

GET i POST zahtevi

GET /FormServlet?tekst=asdf HTTP/1.1

...

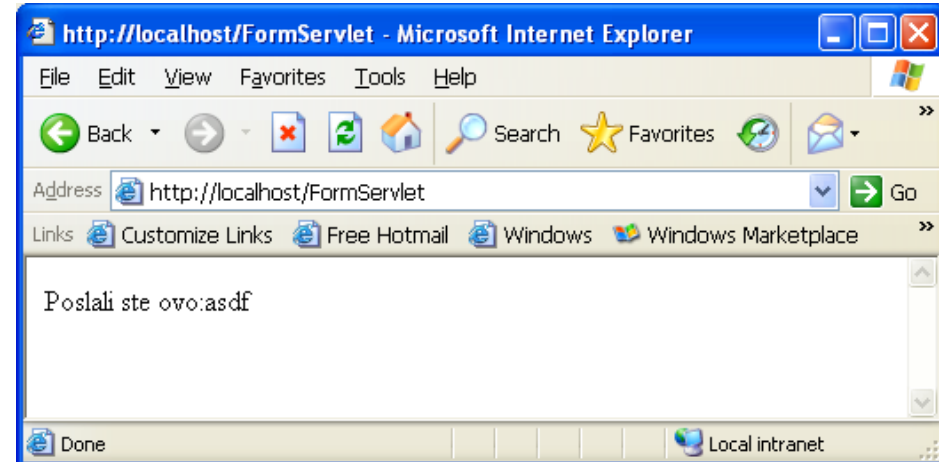
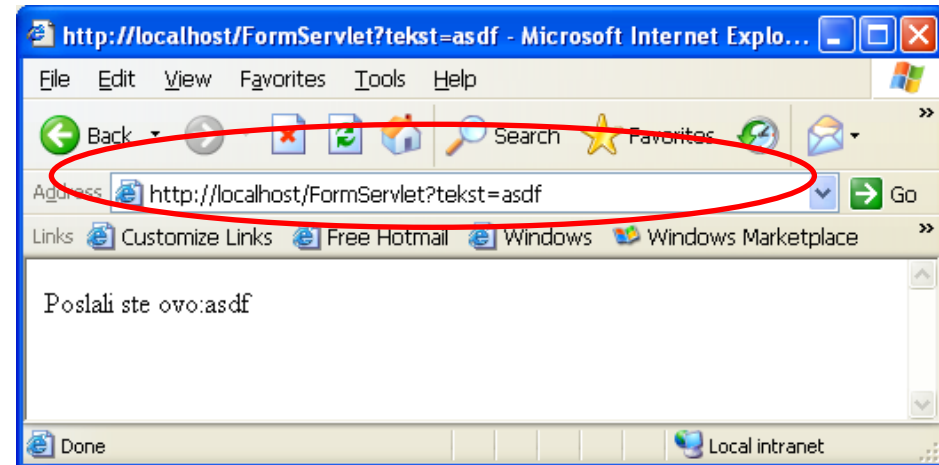
POST /FormServlet HTTP/1.1

...

Content-length: 10

...

tekst=asdf



1. Parametri forme sa POST metodom (bez slanja datoteka)

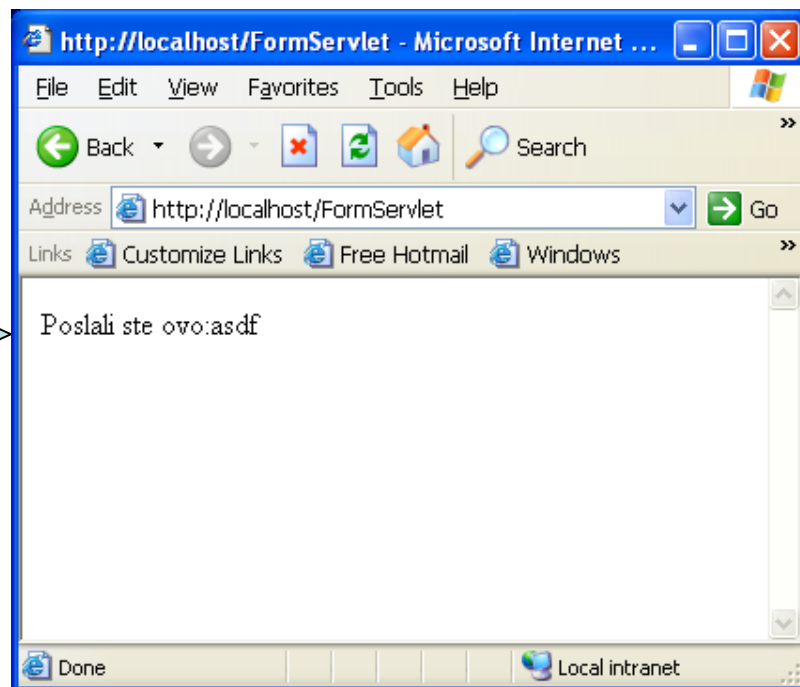
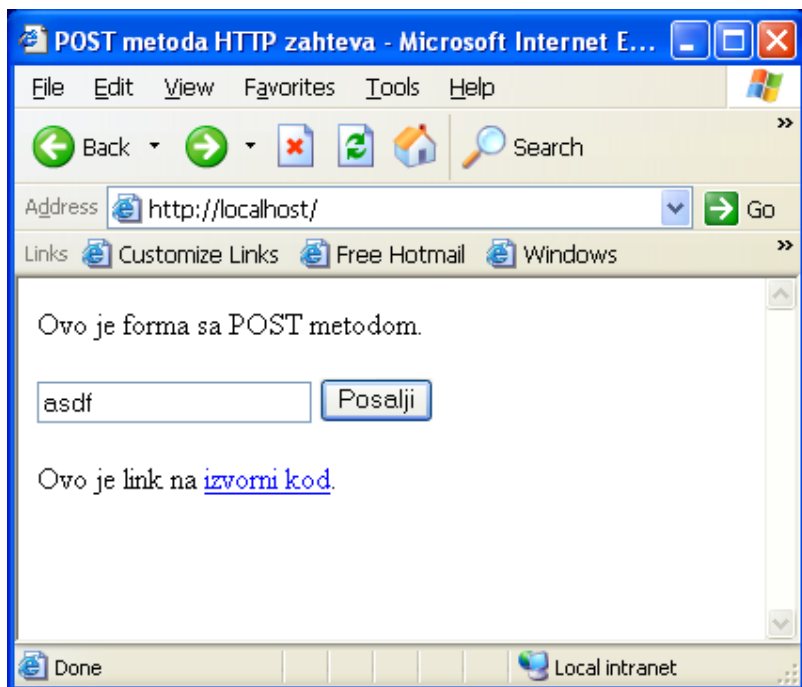
- HTML kod na klijentu:



accept-charset="UTF-8"

```
<form method="post"  
  action="http://localhost/FormServlet">  
  <input type="text" name="tekst">  
  <input type="submit" value="Posalji">  
</form>
```


Preuzimanje podataka sa formi



Parametri forme u POST zahtevu

- Parametri forme se smeštaju u poslednjem redu HTTP zahteva (i nema “\r\n” karaktera na kraju reda – ne može se učitati `readLine()` metodom!).
- Ako je u pitanju samo unos parametara bez slanja datoteke, default tip podataka koji se šalju na server je smešten u parametru zahteva
`Content-Type: application/x-www-form-urlencoded`
- Ukupna dužina parametara je smeštena u parametar HTTP zahteva
`Content-Length: 10`

HTTP zahtev

- Na server stiže sledeći zahtev:

POST /FormServlet HTTP/1.1

Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg,application/x-shockwave-flash, application/vnd.ms-excel,
application/vnd.ms-powerpoint, application/msword, */*

Accept-Language: sr

Content-Type: application/x-www-form-urlencoded

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)

Host: localhost

Content-Length: 10

Connection: Keep-Alive

Cache-Control: no-cache

Cookie: id=1104441583390

tekst=asdf

Modifikacija HttpServletRequest klase za podršku POST metodi

```
// izdvojimo parametre Get metode forme
// (ako ih ima), a ostatak je uri do resursa
resource = extractGetParameters(rsrc);

// iscitamo zaglavlje http zahteva i popunimo
// asocijativnu listu svih parametara iz zaglavlja
readHeader(rdr);

// ako je post metoda, izdvoj parametre
// u istu asoc. listu paramMap
if (method.equals("POST"))
    extractPostParameters(rdr);
```

Modifikacija HttpServletRequest klase za podršku POST metodi

```
private void extractPostParameters(BufferedReader rdr) {
    try {
        String lengthStr = getHeader("Content-Length");
        String contentType = getHeader("Content-Type");
        if ((lengthStr != null) && (contentType != null)) {
            int len = 0;
            try { len = Integer.parseInt(lengthStr); } catch (Exception ex) {}
            if (len > 0 && contentType.equalsIgnoreCase(
                "application/x-www-form-urlencoded")) {
                String name, value, s1;
                char buff[] = new char[len];
                rdr.read(buff, 0, len);
                s1 = new String(buff);
                System.out.println(s1);
                putInParamMap(s1);
            }
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

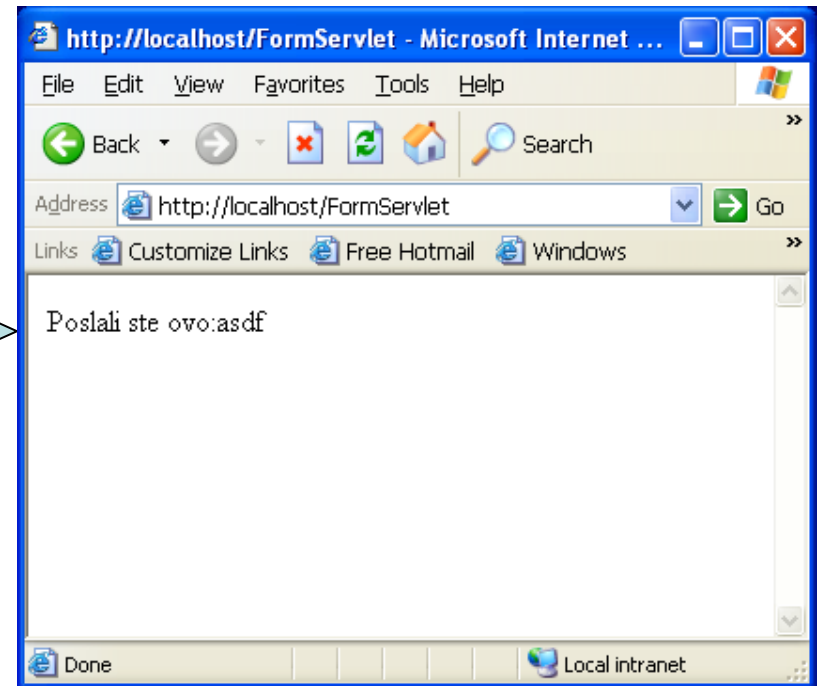
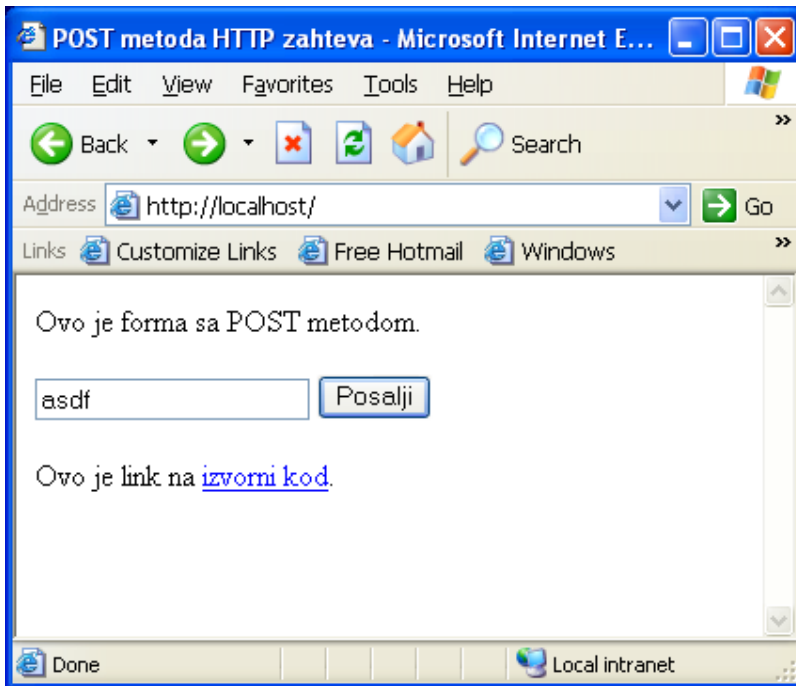
Modifikacija HttpServletRequest klase za podršku POST metodi

```
private void putInParamMap(String s) {  
    // izdelimo ih na pojedinačne parove "ime=vrednost"  
    StringTokenizer hdr = new StringTokenizer(s, "&");  
    while (hdr.hasMoreTokens()) {  
        s = hdr.nextToken();  
        int idx = s.indexOf("=");  
        // levo od '=' je ime  
        String pName = s.substring(0, idx).trim();  
        // desno od '=' je vrednost  
        String pValue = s.substring(idx+1).trim();  
        paramMap.put(pName, pValue);  
    }  
}
```

Servlet za prikaz poslatih parametara

```
public void doPost(HttpServletRequest request,
    HttpServletResponse response) {
    response.setContentType("text/html");
    PrintWriter pout = response.getWriter();
    pout.println("<html>");
    pout.println("<head>");
    pout.println("</head>");
    pout.println("<body>");
    pout.println("Poslali ste ovo:" +
        request.getParameter("tekst"));
    pout.println("</body>");
    pout.println("</html>");
}
```

Rezultat rada servleta



2. Slanje datoteka

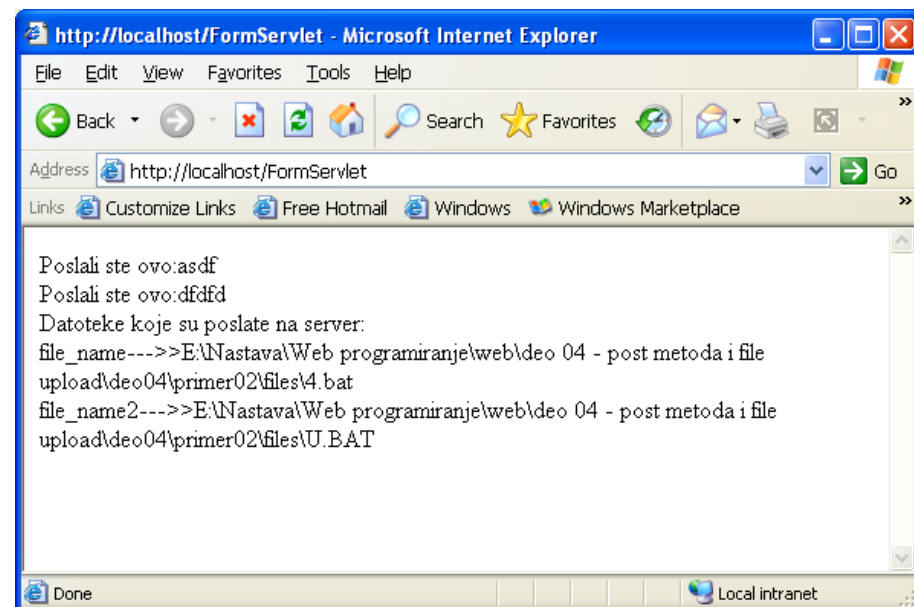
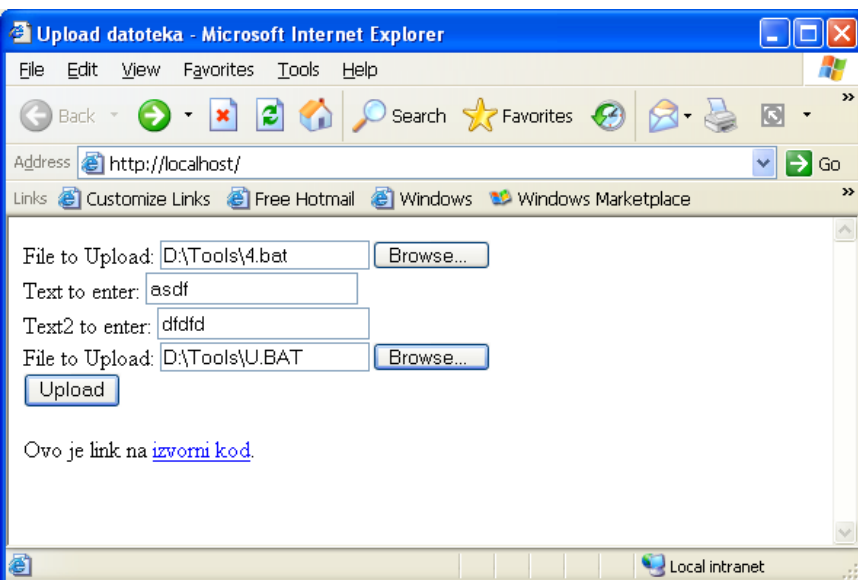
- POST metod, `multipart/form-data` format
- Servletski kontejner nema podršku za ovaj format POST metode

Slanje datoteka

- HTML kod na klijentu:

```
<form action="http://localhost/FormServlet"
  enctype="multipart/form-data" method="post">
  File to Upload: <input type="file" name="file_name">
  <br>
  Text to enter: <input type="text" name="text_field">
  <br>
  Text2 to enter: <input type="text" name="text_field2">
  <br>
  File to Upload: <input type="file" name="file_name2">
  <br>
  <input type="submit" value="Upload">
</form>
```

Slanje datoteka



Parametri forme u POST zahtevu koji je ima i slanje datoteka

- Parametri forme se smeštaju na kraj HTTP zahteva.
- Ako je u pitanju unos parametara sa upload-om datoteka, tip podataka koji se šalju na server je smešten u parametru zahteva

```
Content-Type: multipart/form-data; boundary=-----  
-----7d43993b1002b4
```

- Deo parametra `content-Type` koji se zove `boundary` služi za definisanje granice između pojedinih elemenata forme.
- Ukupna dužina parametara je smeštena u parametar HTTP zahteva

```
Content-Length: 668
```

HTTP zahtev

```
POST /FormServlet HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash,
       application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://localhost/
Accept-Language: sr
Content-Type: multipart/form-data; boundary=-----7d43993b1002b4
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: localhost
Content-Length: 668
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: id=1104441583390
```

```
-----7d43993b1002b4
Content-Disposition: form-data; name="file_name"; filename="D:\Tools\4.bat"
Content-Type: application/octet-stream
```

```
ping %1 smtp.eunet.yu
```

```
-----7d43993b1002b4
Content-Disposition: form-data; name="text_field"
```

```
asdf
```

```
-----7d43993b1002b4
Content-Disposition: form-data; name="text_field2"
```

```
dfdfd
```

```
-----7d43993b1002b4
Content-Disposition: form-data; name="file_name2"; filename="D:\Tools\U.BAT"
Content-Type: application/octet-stream
```

```
start D:\Progra~1\ultraedit\uedit32 %1 %2 %3 %4 %5 %6
```

```
-----7d43993b1002b4--
```



Podrška multipart/form-data POST metodi

```
private Vector uploadedFiles = new Vector();  
public int getUploadedFilesCount() {  
    return uploadedFiles.size();  
}  
public UploadedFile getUploadedFile(int i) {  
    return  
        (UploadedFile)uploadedFiles.elementAt(i);  
}
```

Podrška multipart/form-data POST metodi

```
public class UploadedFile {
    private String fieldName;
    public String getFieldName() {
        return fieldName;
    }
    public void setFieldName(String s) {
        fieldName = s;
    }

    private String filePath;
    public String getFilePath() {
        return filePath;
    }
    public void setFilePath(String s) {
        filePath = s;
    }

    public UploadedFile(String fn, String fp) {
        fieldName = fn;
        filePath = fp;
    }
}
```

Podrška multipart/form-data POST metodi

```
private void extractPostParameters(InputStream dis) {
    if (len > 0 && contentType.equalsIgnoreCase(
        "application/x-www-form-urlencoded")) {
        ...
    } else if (len > 0 && contentType.startsWith("multipart/form-data")) {
        // ako imamo file upload
        bytes = new byte[len];
        // učitamo ostatak zaglavlja u niz bajtova
        int total = 0;
        int l, razlika;
        do {
            razlika = len-total;
            l=dis.read(bytes, total, (razlika >= 1024?1024:razlika));
            total += l;
        } while (total < len);
        // prevedemo celo zaglavlje u jedan veliki string, iz koga necemo snimati
        // datoteku, ali cemo koristiti za analizu i izdvajanje parametara
        s1 = new String(bytes, "US-ASCII");
        // string "boundary" oznacava granicu izmedju pojedinih elemenata (multipart)
        int i = contentType.indexOf("boundary");
        if (i != -1) {
            String boundary = contentType.substring(i).trim();
            i = boundary.lastIndexOf("-");
            if (i != -1) {
                String boundaryText = boundary.substring(i+1).trim();
                //System.out.println(s1);
                // imamo izdvojen granicnik i zaglavlje; sada to treba da procesiramo
                processMultiPart(boundaryText, s1.trim());
            }
        }
    }
}
```


Podrška multipart/form-data POST metodi

```
/** Razbije ostatak HTTP zaglavlja na pojedinačne elemente i
 * procesira ih.
 */
private void processMultiPart(String boundary, String s) {
    boolean flag = true;
    String parts[];
    /**
     * - razbijemo ostatak HTTP zaglavlja upotrebom granicnika
     * - granicnik razdvaja pojedinačne elemente zaglavlja
     * - granicnik je oblika: "---...---boundary"
     * - na kraju granicnika ide CRLF (ili samo LF),
     *   a kod poslednjeg granicnika na kraju ide "--"
     * - HTTP zaglavlje delimo regularnim izrazom kao
     *   parametrom metode split()
     */
    parts = s.split("-+" + boundary + "\\r?\\n?-*");
    for (int i = 0; i < parts.length; i++)
        processPart(parts[i], s.indexOf(parts[i]));
}
```

HTTP zahtev

```
POST /FormServlet HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash,
       application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://localhost/
Accept-Language: sr
Content-Type: multipart/form-data; boundary=-----7d43993b1002b4
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: localhost
Content-Length: 668
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: id=1104441583390
```

```
-----7d43993b1002b4
Content-Disposition: form-data; name="file_name"; filename="D:\Tools\4.bat"
Content-Type: application/octet-stream
```

```
ping %1 smtp.eunet.yu
```

```
-----7d43993b1002b4
Content-Disposition: form-data; name="text_field"
```

```
asdf
-----7d43993b1002b4
Content-Disposition: form-data; name="text_field2"
```

```
dfdfd
-----7d43993b1002b4
Content-Disposition: form-data; name="file_name2"; filename="D:\Tools\U.BAT"
Content-Type: application/octet-stream
```

```
start D:\Progra~1\ultraedit\uedit32 %1 %2 %3 %4 %5 %6
```

```
-----7d43993b1002b4--
```

Podrška multipart/form-data POST metodi

```
private void processPart(String s, int pos) {
    // ako ne pocinje stringom "Content-Disposition"
    // onda je to ostatak koji ne predstavlja nista
    int idx = s.indexOf("Content-Disposition");
    if (idx == -1)
        return;
    /* U ovom trenutku, string "s" izgleda, na primer, ovako:
     * Content-Disposition: form-data; name="filename"; filename="E:\Temp\proba.txt"
     * Content-Type: text/plain
     *
     * asdfasf
     */
    // pokusamo da izdvojimo naziv parametra "name", sto je ime polja u formi
    String fieldName = extractField("name", s);
    if (fieldName == null) // ako ne postoji, nesto ne valja
        return;
    // Zatim prelazimo na parametar ContentType.
    // Ako ga nema, u pitanju je polje sa tekstualnim sadrzajem;
    // ako ga ima, u pitanju je datoteka za upload.
    idx = s.indexOf("Content-Type");
```

Podrška multipart/form-data POST metodi

```
if (idx == -1) {  
    // ako nema Content-Type, u pitanju je polje koje nije za  
    // upload, vec element forme, pa ga razbijemo na redove  
    String[] lines = s.split("\n");  
    if (lines.length == 3) {  
        // uzmemo treci red, koji predstavlja parametar,  
        // oblika: ime=vrednost  
        idx = s.indexOf(lines[2].trim());  
        int len = lines[2].trim().length();  
        byte[] buff = new byte[len];  
        System.arraycopy(bytes, pos+idx, buff, 0, len);  
        System.out.println(s);  
        // posto se prilikom multipart POST zahteva tekstualna polja  
        // kodiraju po zadatom enkodingu, a ne poput %XX, moramo da  
        // ih smestimo u obliku bajtova koje cemo dekodirati na  
        // osnovu encoding atributa  
        paramMap.put(fieldName, buff);  
    }  
}
```

HTTP zahtev

```
POST /FormServlet HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-shockwave-flash,
       application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://localhost/
Accept-Language: sr
Content-Type: multipart/form-data; boundary=-----7d43993b1002b4
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
Host: localhost
Content-Length: 668
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: id=1104441583390
```

```
-----7d43993b1002b4
Content-Disposition: form-data; name="file_name"; filename="D:\Tools\4.bat"
Content-Type: application/octet-stream
```

```
ping %1 smtp.eunet.yu
```

```
-----7d43993b1002b4
Content-Disposition: form-data; name="text_field"
```

```
asdf
-----7d43993b1002b4
Content-Disposition: form-data; name="text_field2"
```

```
dfdfd
-----7d43993b1002b4
Content-Disposition: form-data; name="file_name2"; filename="D:\Tools\U.BAT"
Content-Type: application/octet-stream
```

```
start D:\Progra~1\ultraedit\uedit32 %1 %2 %3 %4 %5 %6
```

```
-----7d43993b1002b4--
```

Podrška multipart/form-data POST metodi

```
} else { // polje za upload
    // uzmemo naziv polja za upload fajla
    String fileName = extractField("filename", s);
    if (fileName != null) {
        // ako ga nema, nije uneto nista u polju za datoteku, u formi
        if (fileName.equals(""))
            return;
        // sada treba preskociti zaglavlje i otici direktno na fajl
        String s1 = stripHeader(s.substring(idx));
        pos += s.indexOf(s1);
        // izvucemo samo naziv fajla
        fileName = extractFileName(fileName);
        try {
            // pripremimo putanju za snimanje (u files poddirektorijum)
            File file = new File("files" + File.separatorChar + fileName);
            FileOutputStream fs = new FileOutputStream(file);
            // pozicioniramo se u nizu bajtova u koji je ucitan ostatak zaglavlja
            fs.write(bytes, pos, s1.length());
            fs.close();
            // dodamo u spisak ucitanih fajlova par
            // (naziv_polja_iz_forme, putanja_na_serveru)
            uploadedFiles.addElement(new UploadedFile(fieldName,
                                                         file.getAbsolutePath()));
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

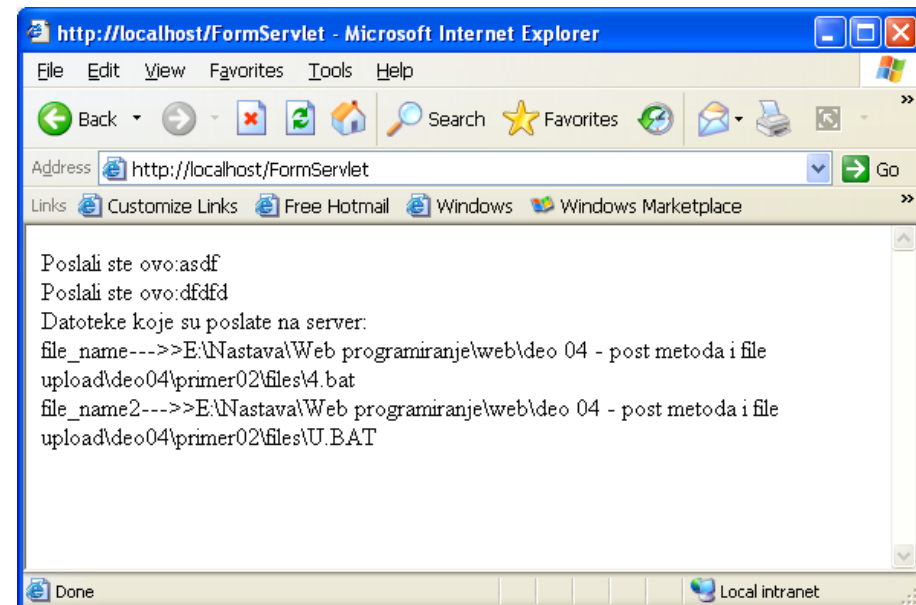
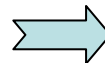
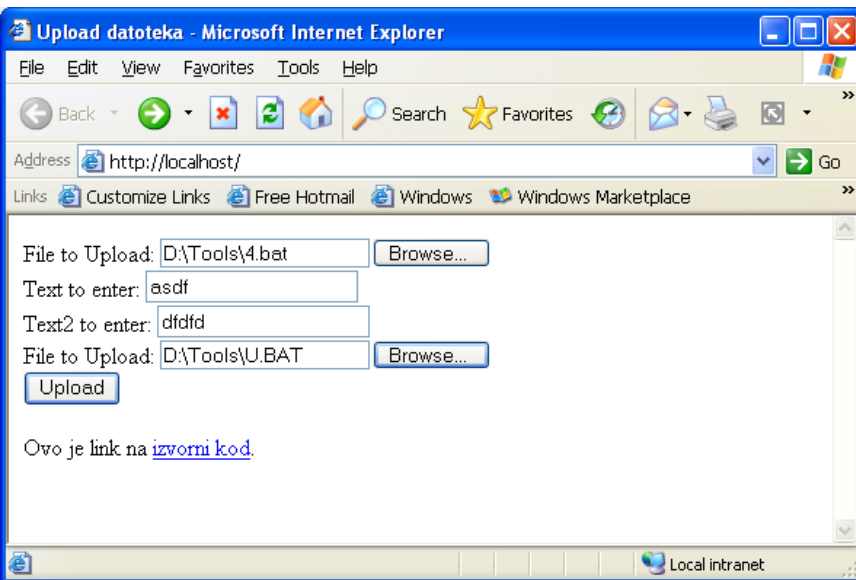
Podrška multipart/form-data POST metodi

```
/** Uklanja zaglavlje ispred datoteke. */  
private String stripHeader(String s) {  
    // na kraju dela se nalazi prelazak u novi red,  
    // pa to treba preskociti  
    int bytesToSkipAtTheEnd;  
    int nl = s.indexOf("\r\n");  
    nl +=4;  
    bytesToSkipAtTheEnd = 2;  
    String s1 = s.substring(nl,  
                             s.length()-bytesToSkipAtTheEnd);  
    return s1;  
}
```

Servlet za prikaz poslatih parametara

```
public void doPost(HttpServletRequest request, HttpServletResponse response) {
    response.setContentType("text/html");
    PrintWriter pout = response.getWriter();
    pout.println("<html>");
    pout.println("<head>");
    pout.println("</head>");
    pout.println("<body>");
    pout.println("Poslali ste ovo:" + request.getParameter("text_field") + "<br>");
    pout.println("Poslali ste ovo:" + request.getParameter("text_field2") + "<br>");
    pout.println("Datoteke koje su poslate na server:<br>");
    for (int i = 0; i < request.getUploadedFilesCount(); i++) {
        pout.print(request.getUploadedFile(i).getFieldName() + "--->>");
        pout.println(request.getUploadedFile(i).getFilePath() + "<br>");
    }
    pout.println("</body>");
    pout.println("</html>");
}
```


Rezultat rada servleta



Rad sa "regularnim" servletskim kontejnerima

- Ne postoji podrška za multipart/form-data format POST metode
- U primerima koristimo *Apache Commons* biblioteku
 - dva jar-a stavljamo u WEB-INF/lib folder
 - commons-fileupload-1.1.jar
 - commons-io-1.1.jar

Servlet za file upload

- Statička metoda
`FileUpload.isMultipartContent(request)`
- Ako jeste multipart/file-data, onda se koristi sledeći kod:

```
DiskFileItemFactory factory = new DiskFileItemFactory();  
factory.setSizeThreshold(2000000);  
ServletFileUpload upload = new ServletFileUpload(factory);  
upload.setSizeMax(3000000);  
  
List<FileItem> items = upload.parseRequest(request);
```

Servlet za file upload

- Ako je "obično" polje forme
(`item.isFormField()`), koristi se sledeći kod:

```
String name = item.getFieldName();  
String value = item.getString("UTF-8");
```

- Ako je datoteka, koristi se sledeći kod:

```
String fieldName = item.getFieldName();  
String fileName = item.getName();  
File uploadedFile = new File(  
    getServletContext().getRealPath("") + "/files/" +  
    fileName);  
item.write(uploadedFile);
```

Diskusija – Unicode karakteri u formi i na stranici

- U formi staviti atribut `accept-charset="UTF-8"`
- U servletu:
 - ako je GET ili "običan" POST, staviti `request.setCharacterEncoding("UTF-8");` na početak servleta
 - ako je multipart/form-data POST metod i koristi se *Apache Commons*, koristiti: `item.getString("UTF-8");`

Diskusija – Unicode karakteri u formi i na stranici

```
response.setContentType(  
    "text/html; charset=UTF-8" );  
request.setCharacterEncoding(  
    response.getCharacterEncoding( )  
    );
```

Java Server Pages

Nedostaci servlet tehnologije

- dizajn stranica (HTML) i programska obrada (Java) su pomešani u istim datotekama
- teško je razdvojiti funkcije dizajnera i programera
- svaka promena u izgledu stranice zahteva kompajliranje servleta

JSP ideja

- HTML + dinamički elementi

```
<html>
```

```
...
```

```
<h4>Dobrodošli, <%= username %></h4>
```

```
Danas je <%= new java.util.Date() %>.
```

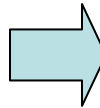
```
...
```

```
</html>
```

JSP realizacija (1)

- JSP stranice se konvertuju u servlete koji generišu upravo onakav izlaz kakav je specificiran u JSP fajlu

```
<html>
...
<h4>Dobrodošli, <%= username %></h4>
Danas je <%= new java.util.Date() %>.
...
</html>
```



```
public class SomeServlet extends ... {

    public void _jspService(...) {
        ...
        out.println("<html>");
        ...
    }
}
```

JSP realizacija (2)

- Dobijeni servlet se kompajlira i poziva
- Rezultat njegovog izvršavanja je tražena JSP stranica

JSP realizacija (3)

- Kod sledećih poziva iste stranice, web server poziva odgovarajući servlet (ne kompajira iznova jsp stranicu)
- Generisanje servleta, njegovo kompajliranje i pozivanje je zadatak servletskog kontejnera

Vrste dinamičkih elemenata

- izrazi (expressions):
`<%= java_izraz %>`
`<%= new java.util.Date() %>`
- skriptleti (scriptlets):
`<% java_kod %>`
`<% for (int i = 0; i < 10; i++) ... %>`
- deklaracije (declarations):
`<%! java_deklaracija %>`
`<%! int a; %>`
- direktive (directives)
`<%@ direktiva attr="..." %>`
`<%@ page contentType="text/plain" %>`

JSP izrazi

```
<html>
```

```
...
```

```
<h4>Dobrodošli, <%= username %></h4>
```

```
Danas je <%= new java.util.Date() %>.
```

```
...
```

```
</html>
```

u pitanju je izraz, dakle
ne završava se sa ;

za izraze koji nisu tipa String
automatski se poziva toString()

JSP skriptleti ^{1/2}

```
<html>
```

```
...
```

```
<% if (Math.random() < 0.5) { %>
```

```
Dobar dan!
```

```
<% } else { %>
```

```
Dobro veče!
```

```
<% } %>
```

```
...
```

```
</html>
```

JSP skriptleti 2/2

```
<html>
...

<table border=1>
<tr>
    <td>R.br.</td>
    <td>Ime</td>
</tr>
<%
String names[] = { "Bata", "Pera", "Mika", "Laza", "Sima" };
for (int i = 0; i < names.length; i++) {
%>
<tr>
    <td><%= i %></td>
    <td><%= names[i] %></td>
</tr>
<% } %>
</table>

...
</html>
```

skriptlet se ugrađuje direktno u kod generisanog servleta; tako je brojač petlje i vidljiv i u okviru drugog skriptleta (on se nalazi "unutar" for petlje)

JSP deklaracije

- Kod se ubacuje u definiciju klase, izvan bilo koje metode:
 - deklaracija atributa
 - deklaracija metoda.

```
<%! int hitCount = 0; %>  
<%! private int getRandom() {  
    return (int) (Math.random() * 100) ;  
}  
%>
```

JSP direktive ^{1/2}

- Omogućavaju kontrolu strukture generisanog servleta.
- Dva osnovna tipa direktiva:
 - page direktive
 - import – za import paketa
 - contentType – podešava ContentType odgovora
 - include direktive
 - uključuje zadatu stranicu u postojeću

JSP direktive 2/2



text/html; charset=utf-8

– page direktive

```
<%@ page contentType="text/html" %>
```

```
<%@ page import="java.util.Vector" %>
```

– include direktive

```
<jsp:include page="asd.html"/>
```

– uključuje stranicu u momentu zahtevanja strane

```
<%@ include file="asd.jsp" %>
```

– uključuje stranicu u momentu kada se stranica prevodi u servlet

Razlika između `<jsp:include>` i `<%@ include>` direktiva

Sintaksa	<code><jsp:include page="..."></code>	<code><%@ include file="..."></code>
Kada se stranica uključuje	U momentu zahtevanja strane	U momentu prevođenja u servlet
Šta se uključuje	Izlaz stranice	Sadržaj datoteke
Broj rezultujućih servleta	2	1
Može li uključena strana da podešava zaglavlje HTTP odgovora?	Ne	Da
Može li uključena strana da definiše metode i attribute koje će glavna strana koristiti?	Ne	Da
Da li se glavna strana ažurira kada se uključena strana promeni	Ne	Da

Predefinisane promenljive

ime	tip
request	HttpServletRequest
response	HttpServletResponse
out	JspWriter
session	HttpSession
application	ServletContext
page	(this)

request predefinisana promenljiva

- Objekat tipa **HttpServletRequest** povezan sa zahtevom
- Omogućava pristup parametrima prosleđenim sa zahtevom (putem metode **getParameter**), utvrđivanje tipa zahteva (GET, POST, HEAD, itd.), i HTTP zaglavlju (cookies, User-Agent, itd.)

response predefinisana promenljiva

- **HttpServletResponse** povezan sa odgovorom klijentu.
- Dozvoljeno je postavljanje HTTP statusnih kodova i zaglavlja odgovora (response headers) .

out predefinisana promenljiva

- **PrintWriter** koji se koristi za slanje odgovora klijentu.
- **out** je baferovana verzija **PrintWriter** pod nazivom **JspWriter**.
- Moguće je podešavanje veličine bafera, kao i njegovo potpuno isključivanje pomoću **buffer** atributa **page** direktive.
- **out** se koristi skoro isključivo u skriptletima, jer se JSP izrazi automatski smeštaju u izlazni tok

session predefinisana promenljiva

- **HttpSession** objekt povezan sa zahtevom.
- Sesije se kreiraju automatski, tako da je ova varijabla već povezana čak iako nema ulazne reference na sesiju.
 - izuzetak je ako se koristi **session** atribut **page** direktive kako bi se isključilo praćenje sesija. U tom slučaju pokušaj pristupanja **session** varijabli rezultuje generisanjem poruke o grešci od strane servera u momentu prevođenja JSP strane u servlet

application predefinisana promenljiva

- Objekat klase **ServletContext** koji služi za komunikaciju servleta i aplikacionog servera.
- Uobičajena upotreba je za smeštanje globalnih promenljivih uz pomoć metoda **setAttribute()** / **getAttribute()**.

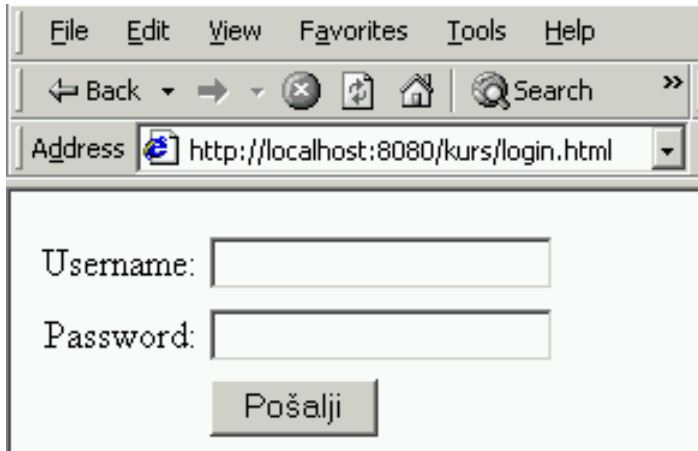
page predefinisana promenljiva

- sinonim za ključnu reč **this**

Podaci koje unosi korisnik

- Podaci se unose preko HTML formi
- Koristi se ista klasa/metoda kao kod servleta – `request.getParameter()`
- Mogu se automatski smeštati i zadati objekat – uvod u JavaBeans

Skladištenje podataka: JavaBeans



A screenshot of a web browser window. The address bar shows the URL `http://localhost:8080/kurs/login.html`. The main content area contains a login form with two text input fields: "Username:" and "Password:". Below the "Password:" field is a button labeled "Pošalji" (Send).

```
public class User implements Serializable{  
    public User() {}  
  
    public void setUsername(String x) {  
        username = x;  
    }  
  
    public void setPassword(String x) {  
        password = x;  
    }  
  
    public String getUsername() {  
        return username;  
    }  
  
    public String getPassword() {  
        return password;  
    }  
  
    private String username;  
    private String password;  
}
```

Prenošenje podataka u JavaBeans

login.jsp

```
<form action="result.jsp">
Username:
  <input type="text" name="username">
Password:
  <input type="password" name="password">
</form>
```

GET /result.jsp?username=asdf&password=***

result.jsp

```
<jsp:useBean id="user" class="somepackage.User" scope="session"/>
<jsp:setProperty name="user" property="username" param="username"/>
<jsp:setProperty name="user" property="password" param="password"/>
<html>
...
</html>
```

Prenošenje podataka u JavaBeans

Ako nema parametara?
- ne poziva se setter

```
<jsp:useBean id="user" class="somepackage.User" scope="session"/>
<jsp:setProperty name="user" property="username" param="username"/>
<jsp:setProperty name="user" property="password" param="password"/>

<html>

...

<% if (user.login()) { %>
    Uspešno ste se prijavili!
<% } else { %>
    Niste se uspešno prijavili!
<% } %>

...

</html>
```

koristimo user kao da je u
pitanju JSP deklaracija

Java kod u JavaBeans

- Ograničiti Java kod u jsp stranicama
 - razdvajanje vizuelnog dela koda od programskog dela
- Efikasniji Java kod u klasi nego u jsp stranici:
 - sintaksne greške se vide prilikom kompajliranja klase, a ne prilikom ponovnog učitavanja stranice
 - testiranje se lakše izvodi iz komandne linije/integrisanog okruženja, nego iz jsp stranice
 - ponovna upotreba u drugim stranicama/projektima

Opseg vidljivosti komponenti ^{1/2}

- application
istu instancu beana dele svi korisnici sajta
- session
svaki korisnik sajta ima svoju instancu
- request
svaki zahtev za stranicom ima svoju instancu
- page (default)
svaka stranica ima svoju instancu

Opseg vidljivosti komponenti 2/2

- specificira se u `<jsp:useBean>` elementu

```
<jsp:useBean id="user" class="somepackage.User" scope="session"/>
```

U prevedenom servletu

- session scope:

```
beans.User user = null;
synchronized (session) {
    user = (beans.User) _jspx_page_context.getAttribute("user",
        PageContext.SESSION_SCOPE);
    if (user == null){
        user = new beans.User();
        _jspx_page_context.setAttribute("user", user,
            PageContext.SESSION_SCOPE);
    }
}
```

U prevedenom servletu

- application scope:

```
beans.User user2 = null;
synchronized (application) {
    user2 = (beans.User) _jspx_page_context.getAttribute("user2",
        PageContext.APPLICATION_SCOPE);
    if (user2 == null){
        user2 = new beans.User();
        _jspx_page_context.setAttribute("user2", user2,
            PageContext.APPLICATION_SCOPE);
    }
}
```

U prevedenom servletu

- request scope:

```
beans.User user3 = null;
synchronized (request) {
    user3 = (beans.User) _jspx_page_context.getAttribute("user3",
        PageContext.REQUEST_SCOPE);
    if (user3 == null){
        user3 = new beans.User();
        _jspx_page_context.setAttribute("user3", user3,
            PageContext.REQUEST_SCOPE);
    }
}
```

U prevedenom servletu

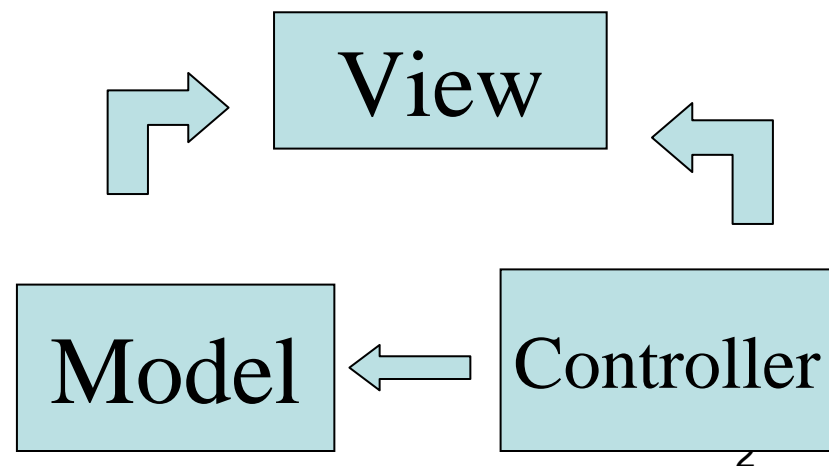
- page scope:

```
beans.User user4 = null;
synchronized (_jspx_page_context) {
    user4 = (beans.User) _jspx_page_context.getAttribute("user4",
        PageContext.PAGE_SCOPE);
    if (user4 == null){
        user4 = new beans.User();
        _jspx_page_context.setAttribute("user4", user4,
            PageContext.PAGE_SCOPE);
    }
}
```

Java Server Pages MVC

Model Controller View (MVC)

- Design pattern koji se zasniva na sinhronom radu tri komponente:
 - Model
 - View
 - Controller



Model

- Modelira podatke i procese
- Obavlja:
 - interakciju sa DB
 - sva potrebna izračunavanja
- Podaci i procesi su nezavisni od prezentacije

View

- Predstavlja prezentacioni sloj
- Vizualizuje podatke iz modela
- Nije svestan porekla podataka (za to je zaslužan model)

Controller

- Obezbeđuje vezu između korisnika i podataka
 - upravlja prezentacionim slojem i slojem podataka
 - upravlja tokom izvršenja aplikacije
 - svi zahtevi (akcije korisnika) idu preko kontrolera, a on donosi odluku koji segment aplikacije će biti prikazan

Razvoj web aplikacija

1. HTML stranice
2. JSP stranice i servleti
3. MVC Model 1
4. MVC Model 2
5. Web application frameworks
 - Struts
 - Tapestry
 - JSF
 - Ajax

Razvoj web aplikacija

kompleksnost



HTML
stranice

Osnovne
JSP stranice
i servleti

JSP stranice
sa modularnim
komponentama

JSP stranice
sa modularnim
komponentama
i enterprise beans

HTML
stranice

HTML
stranice

JSP stanice
servleti

HTML
stranice

JSP stanice
servleti

JavaBeans
komponente

dodatni tagovi

HTML stranice

JSP stanice

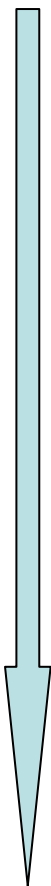
servleti

JavaBeans
komponente

dodatni tagovi

enterprise beans

robustnost



Bez MVC modela

Model 1

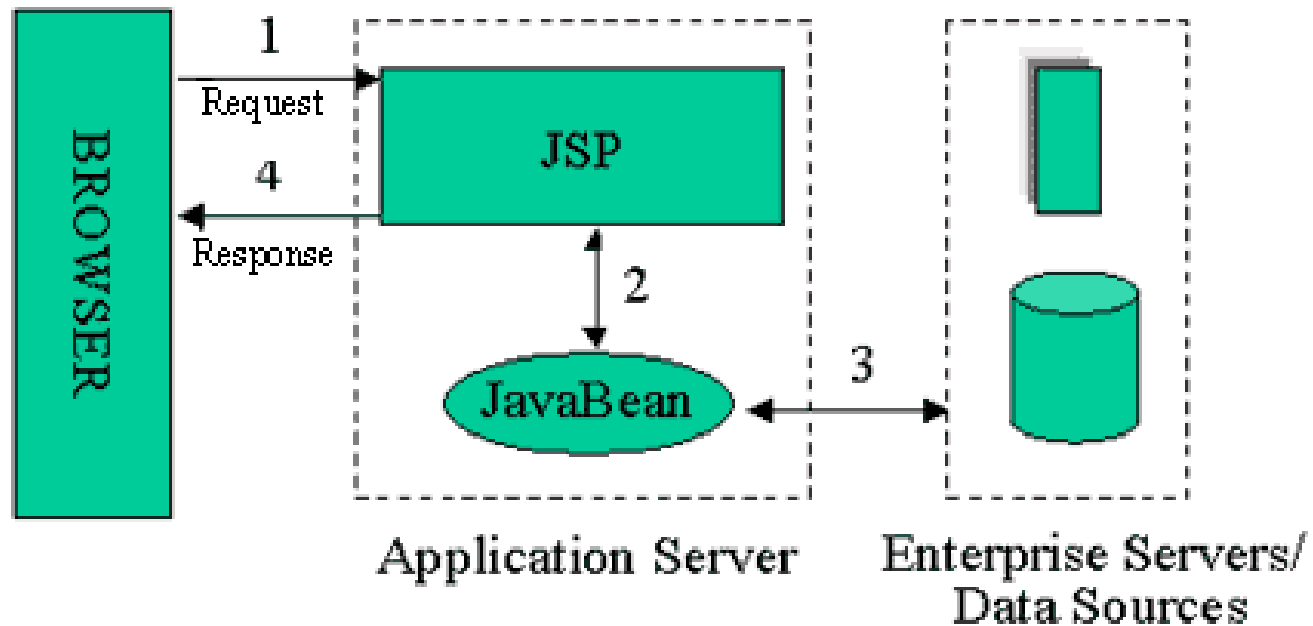
1. JSP stranice

- Sva funkcionalnost je u JSP stranicama
 - Java kod ugrađen u stranice
 - komplikovano testiranje i debugiranje

2. MVC Model 1 (Page-centric)

- Sastoji se iz povezanih JSP stranica
- Prezentacija i kontrola su ugrađeni u JSP stranice
 - podaci su modelirani upotrebom Java beans
- Prelazak na sledeću stranicu je određen:
 - klikom po hiperlinku
 - pritiskom na dugme Submit

MVC Model 1



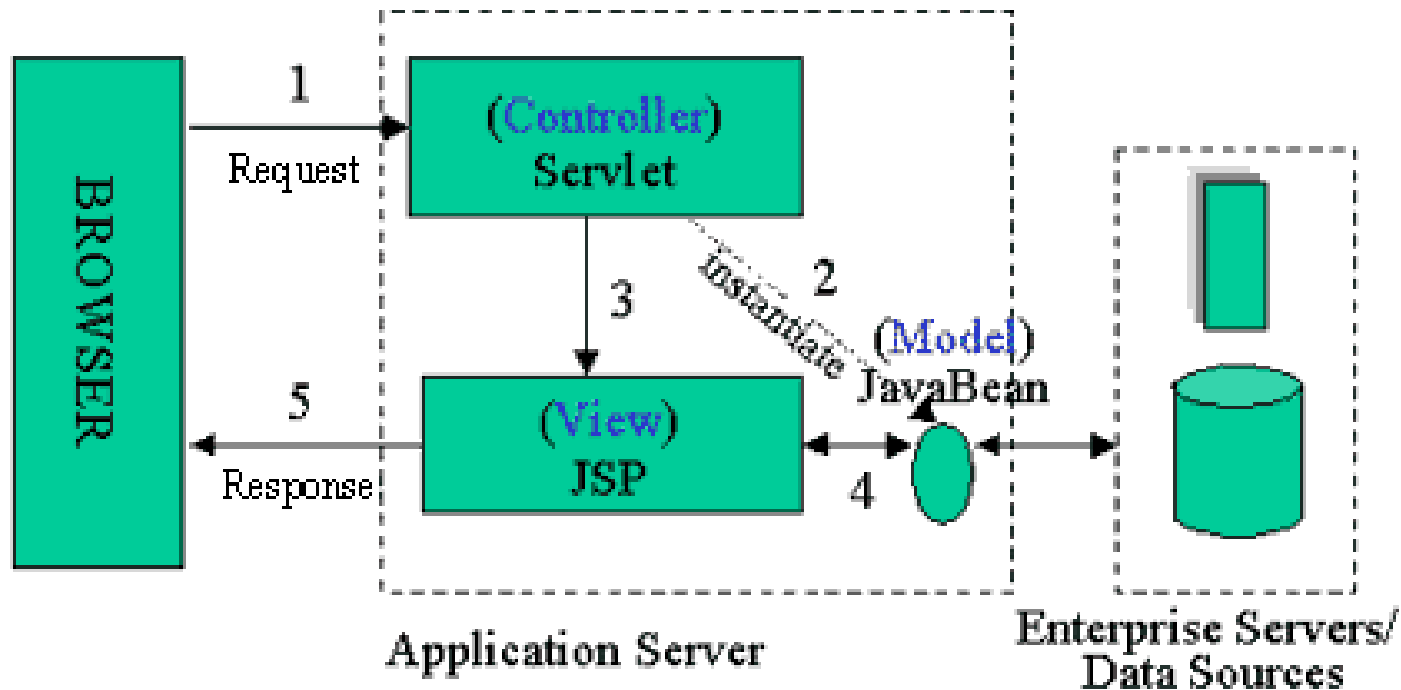
MVC Model 1

- Mane:
 - podstiče špageti-kod u JSP stranicama
 - odabir stranice za prikaz je unutar JSP skriptleta
 - pronalaženje i ispravljanje grešaka može biti teže nego kod servleta
 - oslanjamo se na log aplikacionog servera

2. MVC Model 2

- Razdvaja model, prezentaciju i kontrolu
- Bolje upravlja tokom prezentacije
 - omogućuje lakši prikaz različitih JSP stranica u zavisnosti od ulaznih podataka
- Servleti obavljaju posao kontrolera
 - upravljaju stanjem web aplikacije
 - određuju stranicu za prikaz redirekcijom
- JSP stranice su prezentacioni sloj
- Java beans modeliraju podatke

MVC Model 2



Količna servleta u Model2 tehnologiji

- Jedan servlet (Master Servlet)
- Jedan servlet po funkciji
- Kombinacija
 - master servlet obavlja opšte poslove
 - za specifične poslove, master servlet preusmerava kontrolu na funkcijske servlete

Scenario upotrebe MVC modela 2

- Servlet obrađuje zahteve
 - servlet čita zahteve, proverava validnost parametara, ...
- Servlet inicira obradu podataka (u modelu)
 - rezultati se smeštaju u Java beans
 - Java beans se smeštaju u zahtev (request), sesiju ili aplikaciju (servlet context)
 - page scope se ne koristi u MVC modelu 2
 - postoji samo na JSP stranicama
- Servlet aktivira prezentacioni sloj uz pomoć *RequestDispatcher* klase i njenih metoda *forward* i *include*
- Prezentacioni sloj preuzima podatke iz Java beans uz pomoć `<jsp:useBean>` taga.

Redirekcija (flow control)

- Relativni URL:

```
RequestDispatcher disp =  
    request.getRequestDispatcher(relativanURL);
```

- Apsolutni URL:

```
RequestDispatcher disp =  
    getServletContext().getRequestDispatcher(apsolutniURL);
```

- Klasa *RequestDispatcher* ima dve metode:
 - *forward* – koristi se kada kompletno prebacimo kontrolu na odredišnu stranicu
 - *include* – koristi se kada ubacujemo sadržaj odredišne stranice i nastavljamo sa tekućom
 - originalna stranica može da generiše sadržaj pre i posle uključene stranice
- *RequestDispatcher* omogućuje da na odredištu “vidimo” **request** i **response** objekte iz polazne stranice (servleta)

Preuzimanje parametara u servletu

- Tekući zahtev:
 - parametar *request* u *doGet*, *doPost*,... metodama ima metodu: *getParameter()*:
`String param = request.getParameter("bla");`
- Tekući zahtev (request-scoped JavaBean):
`String param = (String)request.getAttribute("bla");`
- Tekuća sesija (session-scoped JavaBean):
 - parametar *request* u *doGet*, *doPost*,... metodama ima metodu: *getSession()* koja vraća objekat klase *HttpSession*, a koji ima metodu *getAttribute()*:
`String param = (String)request.getSession().getAttribute("bla");`
- Globalni nivo (application-scoped JavaBean):
 - Klasa *HttpServlet* ima metodu *getServletContext()*, koja vraća kontekst servleta, a koji ima metodu *getAttribute()*:
`String param = (String)getServletContext().getAttribute("bla");`

Prenos parametara iz servleta u JSP stranicu, u zahtevu

- U servletu:

```
BeanClass value = new BeanClass(..);  
request.setAttribute("bean", value);
```

- U JSP strani:

```
<jsp:useBean id="bean" type="BeanClass"  
    scope="request" />
```

```
<jsp:getProperty name="bean"  
    property="someProperty" />
```


Prenos parametara iz servleta u JSP stranicu, u sesiji

- U servletu:

```
BeanClass value = new BeanClass(..);  
HttpSession session =  
    request.getSession(true);  
session.setAttribute("bean" , value);
```

- U JSP strani:

```
<jsp:useBean id="bean" type="BeanClass"  
    scope="session" />  
<jsp:getProperty name="bean"  
    property="someProperty" />
```

Prenos parametara iz servleta u JSP stranicu, u aplikaciji

- U servletu:

```
BeanClass value = new BeanClass(..);  
getServletContext().setAttribute("bean",  
    value);
```

- U JSP strani:

```
<jsp:useBean id="bean" type="BeanClass"  
    scope="application"/>  
  
<jsp:getProperty name="bean"  
    property="someProperty" />
```

<jsp:useBean> tag

- Razlika između <jsp:useBean type="..."/> i <jsp:useBean class="...">
 - class instancira ako ne postoji instanca
 - type koristi postojeći; puca ako ne postoji instanca

Alternativna redirekcija

- Umesto *RequestDispatcher*-a, može da se koristi metoda *response.sendRedirect(url)*
- Posledica:
 - korisnik vidi novi url,
 - klijent se dva puta obraća serveru.
- Prednost:
 - korisnik može da ode na odredišnu stranu nezavisno od servleta.
- Mana:
 - korisnik može da ode na odredišnu stranu nezavisno od servleta 😊

Kada koristiti Model 1 a kada Model 2

- Model 1
 - ako je aplikacija jednostavna i tok aplikacije ide od stranice na stranicu (poput Wizard-a)
- Model 2
 - ako svaki klik po linku ili dugmetu zahteva procesiranje podataka
 - ako prikaz sledeće stranice zavisi od stanja aplikacije

Web Application Frameworks

- Web aplikacije imaju zajednički skup funkcionalnosti
 - postoji mogućnost da se jedan deo zajedničkih funkcionalnosti izdvoji u apstraktni sloj
 - apstraktni sloj se može proširiti
- Zasnivaju se na Model 2 arhitekturi

Web Application Frameworks

- Razdvajanje prikaza od poslovne logike
- Omogućuju centralizovanu kontrolu
- Olakšavaju razvoj, testiranje i održavanje
- Postoji snažna podrška
- Omogućavaju jednostaviju internacionalizaciju
- Omogućuju automatsku validaciju unetih podataka
- Pružaju mogućnost korišćenja komponenti
 - komponente je neko napravio i ponudio na upotrebu
 - reusability

Odabrani WAF

- Apache Struts
- JavaServer Faces
- Tapestry
- Ajax

Java Server Pages Expression Language

Expression Language

- Problem sa dosadašnjim prikazom podataka:
 - komplikovan prikaz vrednosti iz JavaBeans (iz komponente Model)
 - nije moguće pristupiti delovima atributa (user.address.street)
 - skriptleti umanjuju mogućnost održavanja
 - ugrožava koncept MVC

Expression Language

- Expression Language (EL) je uveden da bi se rešili navedeni problemi – JSP 2.0 (servleti 2.4)
- Opšti oblik: `${izraz}`
- Prednosti:
 - kraći zapis: `${username}` umesto `<jsp:getProperty>` taga
 - može da pristupa delovima atributa: `${user.address.street}`
 - jednostavan pristup kolekcijama i nizovima: `${niz[indeks_ili_ključ]}`
 - automatska konverzija tipova
 - empty values umesto grešaka
- Novi format web.xml datoteke:

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">
```

Šta ako...

- JSP strana sadrži izraze \${
 - izbaciti EL totalno – koristiti stari format web.xml
 - izbaciti EL iz grupe JSP stranica
 - tag jsp-property-group u web.xml
- ```
<jsp-property-group>
 <url-pattern>*.jsp</url-pattern>
 <scripting-invalid>true</scripting-invalid>
</jsp-property-group>
```
- deaktiviranje EL u pojedinačnim JSP stranicama
    - `<%@ page isELIgnored="true" %>`
  - zamena alternativnim karakterima
    - u JSP 1.2: umesto \$ staviti `&#36;`,
    - u JSP 2.0: `\${`

# Pristup JavaBean komponentama

- U izrazu `${varijabla}` se traži JavaBean komponenta sa imenom *varijabla*, i po redosledu:
  - PageContext (ne koristi se u MVC),
  - HttpServletRequest,
  - HttpSession,
  - ServletContext.

- Primer:

```
${user}
```

*ili*

```
<%
```

```
User user = (User)pageContext.findAttribute("user");
```

```
%>
```

```
<%= user %>
```

*ili*

```
<jsp:useBean id="user" type="beans.User" scope="..."> 5
```

```
<%= user %>
```

# Pristup atributima komponenti

- U izrazu `${varijabla.attr}` se traži JavaBean komponenta sa imenom *varijabla*, i pristupa se njenom atributu *attr*.

- Primer:

```
${user.username}
```

*ili*

```
<%
```

```
User user =
```

```
(User)pageContext.findAttribute("user");
```

```
%>
```

```
<%= user.getUsername() %>
```

*ili*

```
<jsp:useBean id="user" type="beans.User" scope="..." />
```

```
<jsp:getProperty name="user" property="username" />
```

- Bez EL nije moguće jednostavno pristupiti delovima atributa (user.address.street)

# Predefinisane varijable

- PageContext objekat: pageContext.
  - `${pageContext.session.id}`
- parametri stranice iz request objekta: param i paramValues
  - `${param.custID}`
- Zaglavlje HTTP zahteva: header i headerValues
  - `${header.Accept}` ili `${header["Accept"]}` ili
  - `${header["User-Agent"]}`
- Cookie objekat: cookie
  - `${cookie.userCookie.value}` ili
  - `${cookie["userCookie"].value}`
- Context inicijalizacioni parametar (tag `<context-param>` u web.xml):  
initParam
  - `${initParam.inicijalizacioniParametar}`
- Umesto da prepustimo sistemu da pretražuje bean po opsezu važnosti: pageScope, requestScope, sessionScope, applicationScope
  - `${sessionScope.user.niz1[0]}`

# Operatori

- Aritmetički: + - \* / div % mod
- Relacioni: == eq != ne < lt > gt <= le >= ge
- Logički: && and || or ! not
- Unarni operator empty daje
  - true za null, empty string, empty array, empty list, empty map.
  - false u suprotnom.
- Koristiti oprezno da se ne naruši MVC model
- Primer:

```
disabled =
"$ {grupaklasifikacijeBean.mode=='browse'} "
```



# Uslovni izraz

- `${ test ? expression1 : expression2 }`
- Mane:
  - komplikuje izraze
  - mami na ubacivanje poslovne logike u JSP strane
- Primer:

```
<p
 style=${valutaManagedBean.selectedValuta.
 id == valutaRow.id?'background-
 color:#A1A1A1;':'background-
 color:#FFFFFF'} ; cursor:pointer;" >
```

# Funkcije

- Predefinisane funkcije za rad sa stringovima i nizovima

`<%@ taglib prefix="fn"  
uri="http://java.sun.com/jsp/jstl/functions" %>`

- Primer:

**Duzina niza messages je:**  
`${fn:length(messages)}`

# Funkcije

- `fn:escapeXml(string)` – zamenjuje sve tagove eskejp kodovima
- `fn:contains(string, substring)`
- `fn:containsIgnoreCase(string, substring)`
- `fn:endsWith(string, suffix)`
- `fn:indexOf(string, substring)`
- `fn:join(array, separator)` – vraća string sastavljen iz elemenata array-a, odvojenih separator-om
- `fn:length(item)`

# Funkcije

- `fn:replace(string, before, after)`
- `fn:split(string, separator)`
- `fn:startsWith(string, prefix)`
- `fn:substring(string, begin, end)`
- `fn:substringBefore(string, substring)` - vraća deo stringa koji ide pre prosleđenog podstringa
- `fn:substringAfter(string, substring)` – vraća deo stringa koji ide posle prosleđenog podstringa

# Funkcije

- `fn:toLowerCase(string)`
- `fn:toUpperCase(string)`
- `fn:trim(string)`

# Funkcije

- Primer:

```

```

```
<c:forEach var="message" items="${messages}">
```

```
<c:if test="${fn:containsIgnoreCase(message,
 param.filter) or empty param.filter}">
```

```
<c:out value="${message}"/>
```

```
</c:if>
```

```
</c:forEach>
```

```

```

```
<form action="test2.jsp">
```

```
 <input type="text" name="filter">
```

```
 <input type="submit" value="Filtriraj">
```

```
</form>
```

# Zaključak

- Jednostavan pristup
  - bean-ovima
  - kolekcijama
  - standardnim elementima HTTP protokola preko predefinisanih varijabli
- Izbegavati složenije izraze koji narušavaju MVC

# JSP Standard Tag Library (JSTL)



# JSTL

- JSTL je specifikacija dodatnih tagova
- Tagovi za iteracije, grananja, pristup bazi podataka, itd.
- Referentna implementacija na adresi:  
<http://jakarta.apache.org/builds/jakarta-taglibs/releases/standard/>
- Instalira se kopiranjem *jstl.jar* i *standard.jar* u WEB-INF/lib folder aplikacije

# Korišćenje

- Navede se deskriptor biblioteke i prefiks:

```
<%@ taglib prefix="c"
 uri="http://java.sun.com/jsp/jstl/core" %>
```

- Koristi se tag:

```
<c:forEach var="item" begin="1" end="10">
 ${item}
</c:forEach>
```

# Variable

- `<c:set>` postavlja varijablu na zadatu vrednost
- Primer:  
`<c:set var="mojaVarijabla" value="123" />`

# Grananje

- Grananje tipa if

```
<c:if test="$ {someTest}">
```

```
 Content
```

```
</c:if>
```

- Višestruki izbor – choose:

```
<c:choose>
```

```
<c:when test="test1">Content1</c:when>
```

```
<c:when test="test2">Content2</c:when>
```

```
...
```

```
<c:when test="testN">ContentN</c:when>
```

```
<c:otherwise>Default Content</c:otherwise>
```

```
</c:choose>
```

# Petlje

- Klasična for petlja za brojanje

```
<c:forEach var="item" begin="1" end="10">
 ${item}
</c:forEach>
```

- Foreach petlja za iteriranje kroz kolekciju:

```

<c:forEach var="message"
 items="${messages}">
<c:out value="${message}" />
</c:forEach>

```

# Petlje

- Foreach petlja za iteriranje kroz listu stavki odvojenih zarezom  
`<c:forEach var="country"`  
`items="Australia,Canada,Japan,USA">`  
    `<c:out value="\${country}" />`  
`</c:forEach>`
- ForTokens petlja za iteriranje kroz listu stavki odvojenih proizvoljnim delimiterom  
`<c:forTokens var="color"`  
`items="(red (orange) yellow)(green)((blue) violet)"`  
`delims="( )">`  
    `<c:out value="\${color}" />`  
`</c:forTokens>`

# DB tagovi

- Poseban taglib:  
**<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>**
- **<sql:setDataSource>**
  - definiše data source (poželjno je da se ovo podešava iz konfiguracione datoteke)
- **<sql:query>**
  - izvršava upit nad bazom podataka i smešta rezultat u promenljivu tipa ResultSet
- **<sql:update>**
  - atributi:
    - var – rezultat rada, tipa Integer
    - sql – sql update string
- **<sql:param>** - postavlja parametar za sql izraz
  - atribut – value vrednost parametra
- **<sql:dateParam>** - parametar tipa date, time ili timestamp
  - atributi
    - value
    - type – date, time ili timestamp
- **<sql:transaction>**
  - sve obuhvaćene **<sql:query>** i **<sql:update>** akcije tretira kao jednu transakciju

# Tagovi za manipulaciju URL-ovima

- `<c:import>`
  - čita sadržaj sa proizvoljnog URL-a i ubacuje u stranicu
  - za razliku od `<jsp:include>`, nije ograničen na sopstveni sistem
- `<c:redirect>`
  - redirektuje odgovor na navedeni URL
- `<c:param>`
  - enkodira request parametar i dodaje ga na URL
  - koristi se unutar tagova `<c:import>` i `<c:redirect>`



# Tagovi za formatiranje

- **Poseban taglib:**

**<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>**

- **<fmt:setLocale>**

- postavlja trenutni lokal

- **<fmt:formatNumber>**

- formatira numeričku vrednost kao broj, valutu ili procenat, u skladu sa lokalnim podešavanjima (Locale)

- **<fmt:parseNumber>**

- čita string kao broj, valutu, procenat ili po zatom obrascu

- **<fmt:formatDate>**

- formatira datum

- **<fmt:parseDate>**

- čita string kao datum

- Trebalo bi izbegavati tagove parseXXX – pogodnije mesto za to je biznis logika

# Funkcije

- Predefinisane funkcije za rad sa stringovima i nizovima  
`<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>`

- Primer:

```
<c:forEach var="message" items="${messages}">
 <c:if test="${fn:contains(message,
 param.filter) or empty param.filter}">
 <c:out value="${message}" />
 </c:if>
</c:forEach>
```

# Zaključak

- Uz upotrebu Expression Language, doprinosi čitljivijem kodu
- Veliki broj dodatnih tagova
- Pažljivo koristiti da se ne naruši MVC model