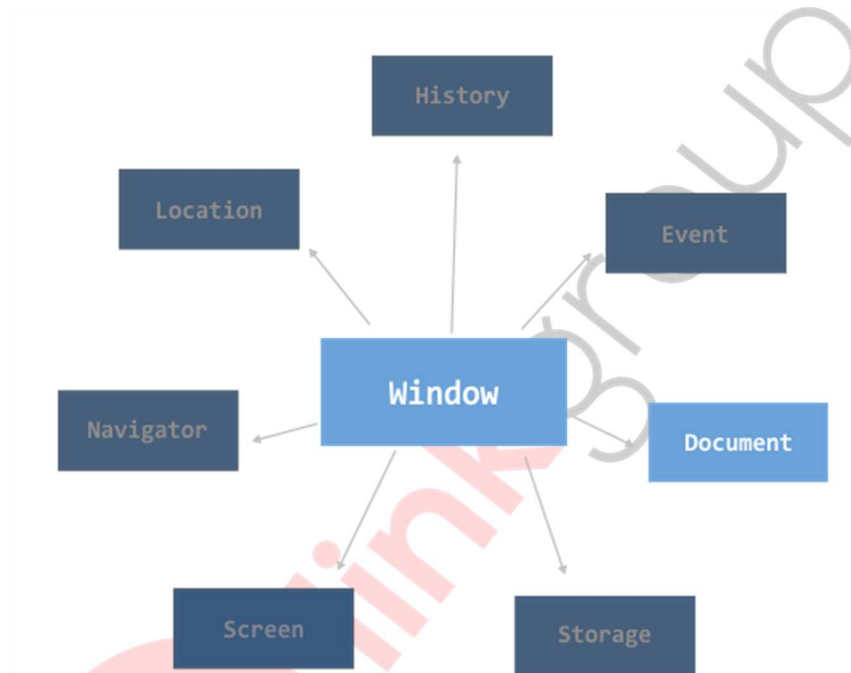


# Objektni model dokumenta

Prethodna lekcija donela je uvodnu priču o funkcionalnostima koje web pregledači stavljaju na raspolaganje JavaScript kodu HTML dokumenta. Tako je u prethodnoj lekciji predstavljan globalni objekat svih web pregledača, odnosno objekat unutar koga su objedinjene sve spomenute funkcionalnosti. Reč je o objektu `Window`.

Unutar objekta `Window`, između ostalih, nalazi se i objekat `Document` (slika 8.1).



Slika 8.1. *Document objekat predstavljen je kao jedno od svojstava globalnog Window objekta*

Unutar objekta `Document` nalazi se jedan od najznačajnijih aplikativnih programskih interfejsa (API-a) weba. Reč je zapravo o objektnom modelu dokumenta, koji se skraćeno naziva DOM (*Document Object Model*).

## Šta je DOM?

DOM je skraćenica koja označava pojam *Document Object Model*. DOM je objektna reprezentacija jednog HTML dokumenta.

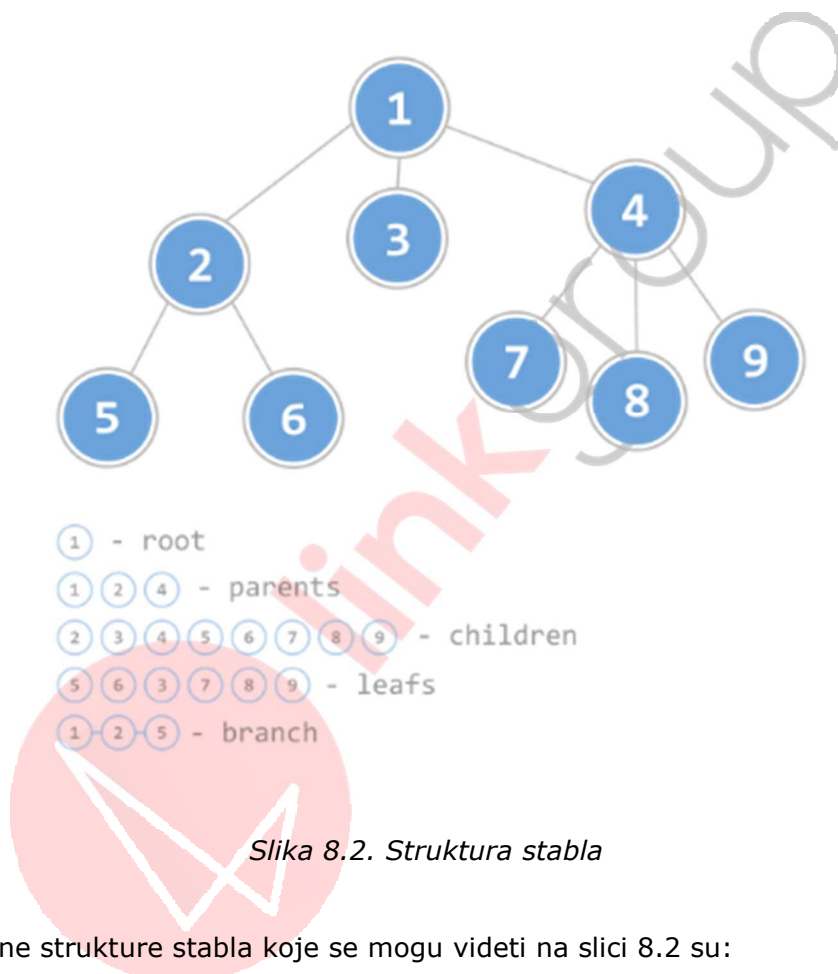
HTML dokument sastoji se iz elemenata koji grade njegovu strukturu. Takvi elementi za JavaScript nemaju posebno značenje. Ali kada se oni prevedu u objekte, dobijaju se entiteti koje JavaScript može da koristi. I upravo to je i osnovna karakteristika DOM-a – reč je o objektima koji predstavljaju HTML elemente.

Objektna reprezentacija strukture dokumenta omogućava JavaScript jeziku da na lak način vrši interakciju sa dokumentom.

## Kako izgleda DOM struktura?

DOM gradi objektnu strukturu u formi stabla. Stablo je specijalna vrsta strukture podataka, kod koje svaki element, osim korenog, ima svog roditelja. Takođe, svaki element može imati i proizvoljan broj potomaka.

Osobine jednog stabla predstavljene su slikom 8.2.



Slika 8.2. Struktura stabla

Osnovne osobine strukture stabla koje se mogu videti na slici 8.2 su:

- stablo poseduje koreni element (*root*);
- svaki element osim korenog ima svog roditelja (*parent*);
- jedan element može imati proizvoljan broj potomaka (*children*);
- element ne mora da ima potomke;
- roditelji i potomci međusobno grade jednu granu (*branch*);
- poslednji element u jednoj grani se naziva list (*leaf*).

Analizom strukture stabla lako je zaključiti da je reč o strukturi koju HTML elementi međusobno grade unutar jednog dokumenta. Kada se ovakva struktura HTML elemenata jednog dokumenta prevede u objektnu strukturu, dobija se DOM takvog dokumenta.

Prve DOM strukture biće ilustrovana kroz primer sledećeg HTML dokumenta:

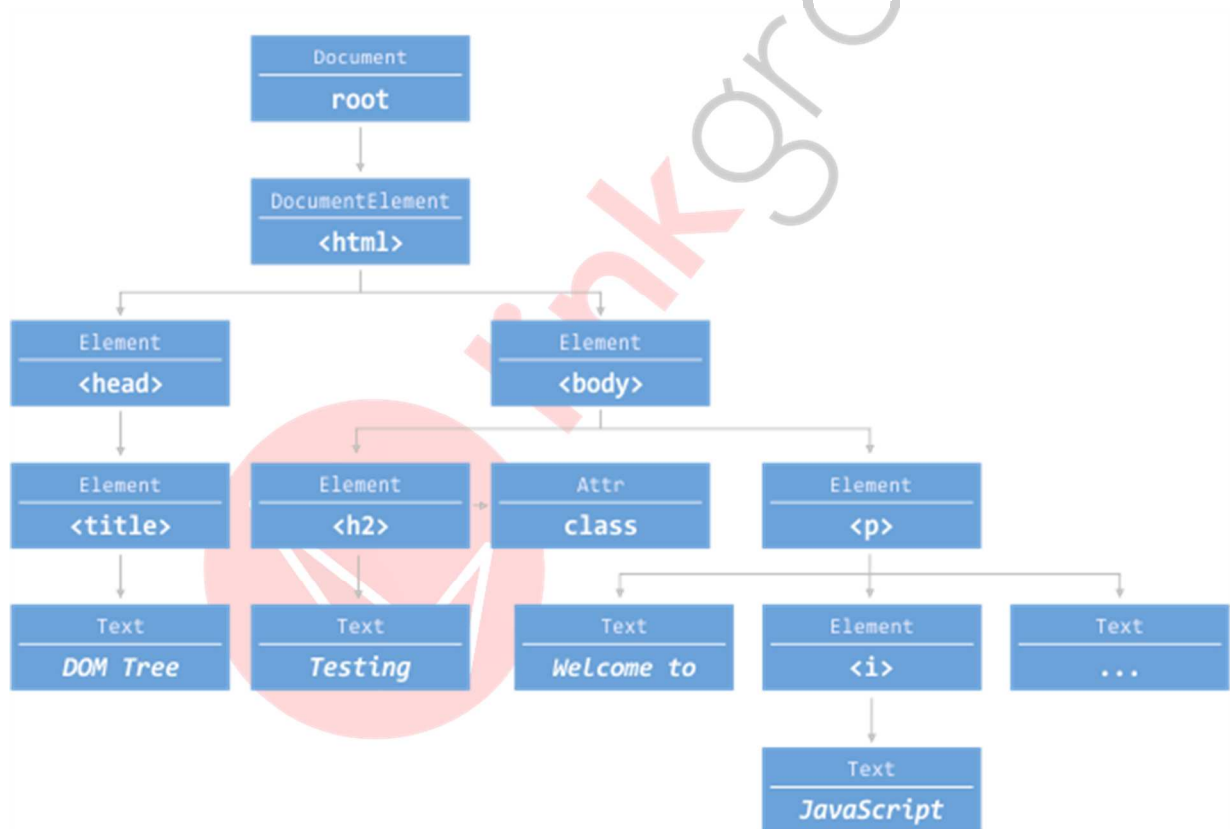
```
<!DOCTYPE html>
<html>

  <head>
    <title>DOM Tree</title>
  </head>

  <body>
    <h2 class="heading">Testing</h2>
    <p>Welcome to <i>JavaScript</i>...</p>
  </body>

</html>
```

Kada se elementi prikazanog HTML dokumenta prikažu u formi stabla, dobija se struktura kao na slici 8.3.



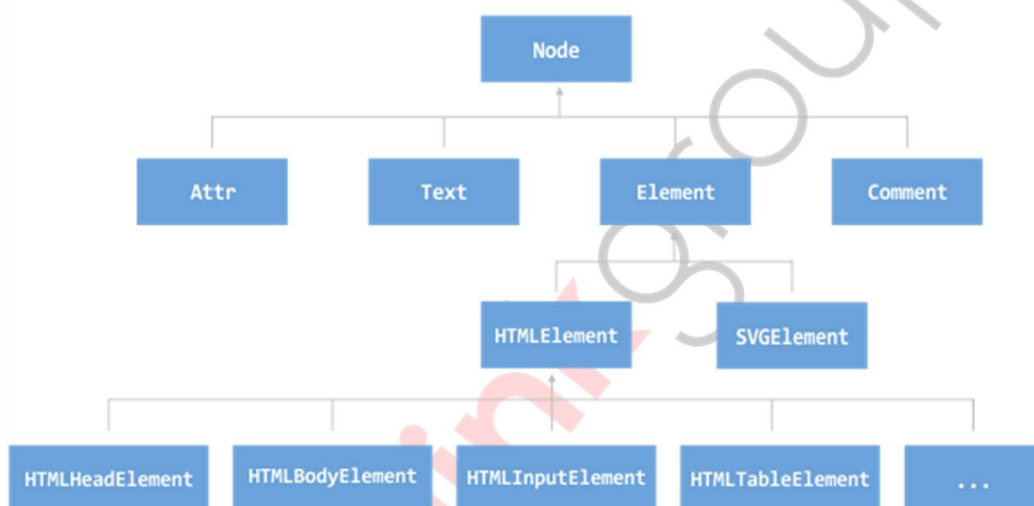
Slika 8.3. DOM struktura jednog HTML dokumenta

Na slici 8.3. prikazani su elementi jednog HTML dokumenta u formi stabla. Na slici se, pored HTML elemenata, koji su prikazani nešto krupnijim tekstom, mogu videti i određene oznake – *Document*, *Element*, *Attr*... O čemu je ovde reč?

Naime, objektna reprezentacija HTML dokumenata podrazumeva korišćenje velikog broja različitih objekata za predstavljanje elemenata. Štaviše, pored HTML elemenata, korišćenjem specijalnih objekata predstavljaju se i atributi, komentari, ali i sadržaj HTML elemenata.

Tako svaka objektna struktura HTML dokumenta započinje objektom `Document`. To je kontejner za sve ostale objekte. Koreni element HTML dokumenta (`html`) predstavlja se objektom `DocumentElement`. Zatim, svi elementi (`head`, `body`, `h2...`) predstavljaju se objektima `Element`, a atributi objektima `Attr`. Na kraju, unutar objektno reprezentacije HTML dokumenta, i sadržaj elemenata se predstavlja korišćenjem posebnog objekta – `Text`.

Iz svega navedenog se može zaključiti da web pregledači prilikom kreiranja objektno strukture HTML dokumenta koriste brojne objekte. Neki od najznačajnijih takvih objekata su prikazani slikom 8.4.



Slika 8.4. Objektna struktura dokumenta (DOM)

Svi elementi unutar objektno HTML strukture drugačije se nazivaju čvorovi (*nodes*). Otuda i objekat **Node**, kojim se uopšteno predstavljaju svi elementi koji se unutar DOM strukture mogu naći. Ipak, čvorovi unutar DOM strukture mogu biti različitog tipa. Takvi različiti čvorovi se unutar DOM-a predstavljaju korišćenjem različitih objekata:

- **Element** – za predstavljanje HTML elemenata;
- **Attr** – za predstavljanje atributa HTML elemenata;
- **Text** – za predstavljanje sadržaja HTML elemenata;
- **Comment** – za predstavljanje komentara.

Baš kao što čvorovi DOM stabla mogu biti različitog tipa, postoji i veliki broj različitih tipova elemenata. Upravo zbog toga postoje objekti:

- **HTMLElement**, kojima se predstavljaju HTML elementi i
- **SVGElement**, kojima se predstavljaju SVG elementi.

Svaki HTML element unutar DOM strukture predstavlja se zasebnim objektom (`HTMLHeadElement`, `HTMLBodyElement`, `HTMLInputElement`...). Na slici 8.4. prikazani su samo neki od takvih objekata, jer je njihov broj zaista veliki.

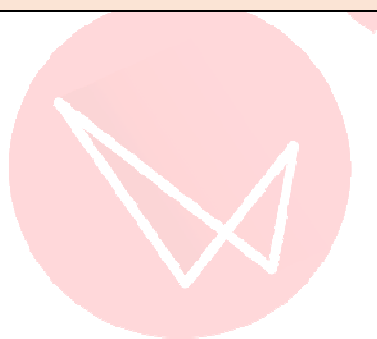
### Prvi primer korišćenja DOM-a

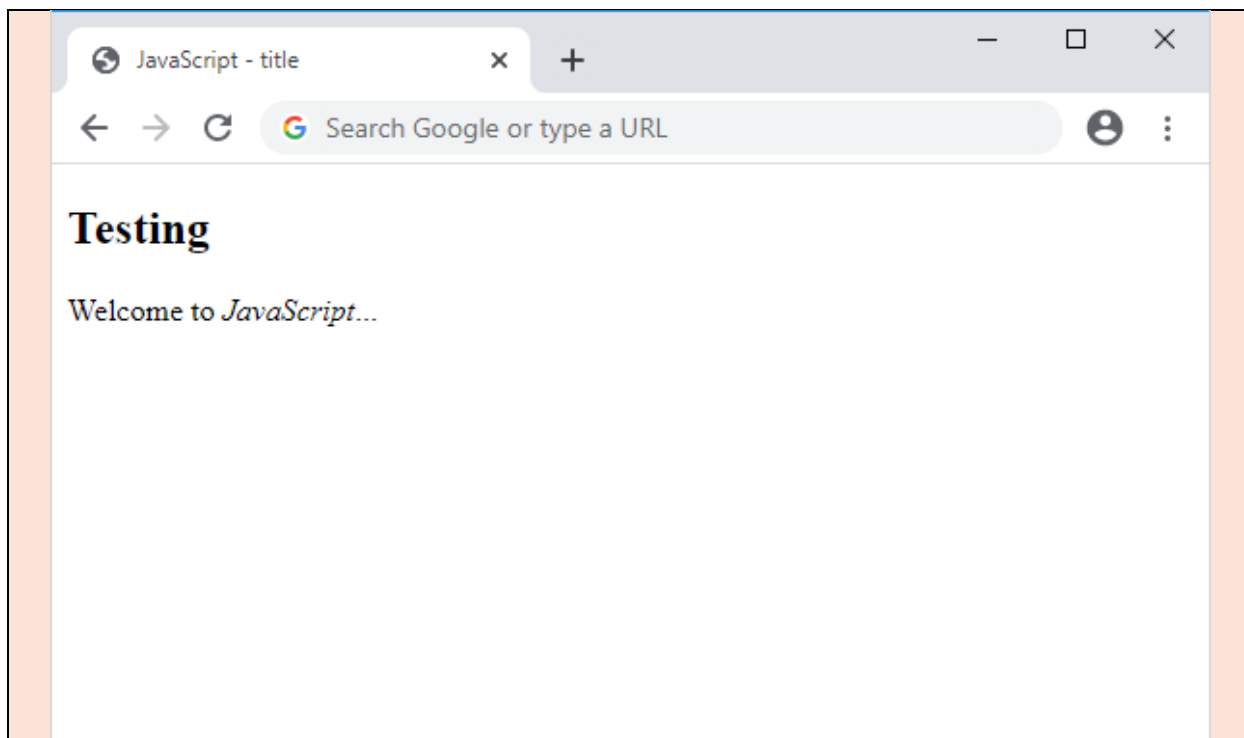
Nakon upoznavanja osnova DOM strukture, moguće je prikazati i prvi primer korišćenja ove objektivne strukture za obavljanje manipulacije nad dokumentom.

Osnovni objekat DOM strukture je `Document`. Njemu se pristupa korišćenjem `document` svojstva `Window` objekta. Unutar `Document` objekta nalazi se svojstvo `title`, kojim je moguće promeniti naslov dokumenta:

```
<!DOCTYPE html>
<html>
  <head>
    <title>DOM Tree</title>
  </head>
  <body>
    <h2>Testing</h2>
    <p>Welcome to <i>JavaScript</i>...</p>
    <script>
      document.title = "JavaScript - title";
    </script>
  </body>
</html>
```

U HTML strukturu dodat je `script` element sa jednom linijom koda, koja za cilj ima postavljanje vrednosti `title` svojstva `document` objekta. Kada se ovakav dokument otvori u nekom pregledaču, dobija se efekat kao na slici 8.5.





*Slika 8.5. HTML dokument sa naslovom koji je definisan programabilno, korišćenjem svojstva title, document objekta*

Na slici 8.5. može se videti da je naslov dokumenta *JavaScript - title*, iako je u samoj strukturi u fajlu naslov postavljen na *DOM Tree*. Dokument uistinu i ima naslov *DOM Tree*, sve do onoga trenutka kada se linija JavaScript koda za promenu vrednosti svojstva *title* ne izvrši.

### **Pitanje**

Elementi DOM strukture koji se nalaze na dnu hijerarhije i ne poseduju potomke nazivaju se:

- grane
- roditelji
- **listovi**
- stabla

### **Objašnjenje:**

*Poslednji element unutar jedne grane DOM strukture naziva se list.*

## Selektovanje elemenata u DOM strukturi

Prethodni primer ilustrovao je veoma jednostavnu intervenciju nad tekućim dokumentom, korišćenjem `document` objekta. Ipak, `document` objekat omogućava mnogo naprednije intervencije nad dokumentom. Kako bismo bili u stanju da ih demonstriramo, prvo je neophodno naučiti kako doći do željenih elemenata DOM strukture.

Postoji nekoliko načina na koje je moguće doći do željenog objekta u DOM strukturi. Svi oni podrazumevaju korišćenje objekta `Document`, kome se može pristupiti korišćenjem svojstva `document`, globalnog `Window` objekta. Nad `Document` objektom mogu se pozivati različite metode koje omogućavaju dobijanje reference na jedan ili više HTML elemenata. Te metode su prikazane tabelom 8.1.

Metoda	Opis
<code>getElementById(id)</code>	obezbeđuje referencu na element sa prosleđenom vrednošću <code>id</code> atributa
<code>getElementsByClassName(name)</code>	obezbeđuje referencu na sve elemente koji imaju prosleđenu vrednost <code>class</code> atributa
<code>getElementsByTagName(name)</code>	obezbeđuje referencu na sve elemente na stranici koji su određenog tipa
<code>querySelector(cssSelector)</code>	obezbeđuje referencu na prvi element koji zadovoljava prosleđeni CSS selektor
<code>querySelectorAll(cssSelector)</code>	obezbeđuje referencu na sve elemente koji zadovoljavaju kriterijum prosleđenog CSS selektora

Tabela 8.1. Funkcije za selektovanje elemenata

U nastavku će biti prikazano korišćenje svake od upravo prikazanih metoda.

### Pretraga elemenata po ID-ju – `getElementById()`

Za pronalaženje elementa po vrednosti atributa `id` koristi se metoda `getElementById()`, koja se poziva nad `document` objektom. Na taj način se web pregledaču govori da je potrebno da pronađe element koji se nalazi u dokumentu i koji ima `id` koji je naveden kao parametar ove metode.

```
<html>
<head>
  <title>DOM</title>
</head>
<body>
  <div id="div-1">
    <p id="p-1">Text 1</p>
    <p id="p-2">Text 2</p>
    <p>Text 3</p>
  </div>
  <script>
    var paragraph = document.getElementById('p-2');
  </script>
</body>
</html>
```

U primeru je kreirano nekoliko HTML elementa koji poseduju definisane vrednosti `id` atributa:

- `div-1` – odnosi se na prvi `div` element;
- `p-1` – odnosi se na prvi `p` element;
- `p-2` – odnosi se na drugi `p` element.

Nakon izvršavanja definisanog JavaScript koda, unutar promenljive `paragraf` biće smeštena referenca na element sa `id`-jem `p-2`, odnosno na drugi paragraf element. To praktično znači da će unutar promenljive `paragraph` biti smešten objekat tipa `HTMLParagraphElement`. Reč je o objektu DOM strukture kojim se predstavljaju paragraf elementi.

#### **Napomena**

*Treći paragraf u primeru ne poseduje definisanu vrednost `id` atributa. Stoga, njega ne bi bilo moguće selektovati na prikazani način.*

#### **Pretraga elemenata po nazivu klase – `getElementsByClassName()`**

DOM API omogućava da se izvrši selektovanje više elemenata koji imaju zajedničku vrednost `class` atributa:

```
<html>

  <head>
    <title>DOM</title>
  </head>

  <body>
    <p id="my-class">Text 1</p>
    <p class="my-class">Text 2</p>
    <p class="my-class some-other-class">Text 3</p>
    <p>Text 4</p>
    <p>Text 5</p>
    <p class="your-class">Text 6</p>

    <script>
      var paragraphs = document.getElementsByClassName('my-
class');
      console.log(paragraphs.length);
    </script>

  </body>

</html>
```

Struktura `body` dela se sastoji iz šest paragrafa. Neki paragrafi imaju definisanu vrednost `id` atributa, drugi `class` atributa, dok su ostali paragrafi bez ikakvih atributa.

JavaScript kodom selektuju se svi elementi koji za vrednost `class` atributa imaju `my-class`.



S obzirom na to da više elemenata može imati jednu istu klasu, metoda `getElementsByClassName()` vraća niz elemenata, pa čak i kada se selektuje samo jedan element. Tako se, nakon izvršenja prve linije prikazanog koda, unutar promenljive `paragraphs` smešta objekat tipa `HTMLCollection`. Reč je o objektu kojim se predstavljaju nizovi, odnosno kolekcija HTML elemenata.

Nakon selektovanja, unutar konzole u primeru se ispisuje broj selektovanih elemenata:

2

Ovo znači da su selektovana ukupno dva elementa sa klasom `my-class`. Ti elementi su drugi i treći paragraf.

Jedan element može imati više klasa, odvojenih praznim mestom, baš kao što je u primeru to slučaj sa trećim paragrafom. Metoda `getElementsByClassName()` selektovaće element ukoliko pronađe podudaranje makar jednog klasnog imena sa nazivom klase koji se traži.

### **Selektovanje elemenata na osnovu naziva taga – `getElementsByTagName()`**

Elemente je moguće selektovati i na osnovu njihovog tipa, odnosno naziva taga koji se koristi za izgradnju elementa:

```
<html>
<head>
  <title>DOM</title>
</head>
<body>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Item 3</li>
  </ul>

  <script>
    var items = document.getElementsByTagName('li');
    console.log(items.length);
  </script>
</body>
</html>
```

U primeru je kreirana jedna neuređena lista sa tri stavke. Da bi se seletovala sva tri elementa koja predstavljaju stavke (`li` elementi), koristi se metoda `getElementsByTagName()`, kojoj se prosleđuje naziv tipa elementa.

Metoda `getElementsByTagName()` vraća niz elemenata tipa `HTMLCollection`. U prikazanom primeru, unutar takve kolekcije nalaze se tri elementa, pri čemu je svaki predstavljen korišćenjem objekta – `HTMLLIElement`. Da je to stvarno tako, može se videti i unutar konzole, u kojoj se ispisuje vrednost 3, što je ukupan broj selektovanih elemenata.

### **Selektovanje elementa korišćenjem CSS selektora – `querySelector()`**

Korišćenjem metode `querySelector()`, moguće je selektovati prvi element koji zadovoljava kriterijum prosleđenog CSS selektora:

```
<html>

  <head>
    <title>DOM</title>
  </head>

  <body>
    <ul>
      <li>Item 1</li>
      <li id="my-id">Item 2</li>
      <li>Item 3</li>
    </ul>

    <script>
      var item = document.querySelector('#my-id');
      console.log(item.innerHTML);

      var oneMoreItem = document.querySelector("li");
      console.log(oneMoreItem.innerHTML);
    </script>
  </body>
</html>
```

Primer prikazuje neuređenu listu sa tri stavke. Korišćenjem metode `querySelector()`, navodi se CSS selektor kojim se selektuje element koji za `id` atribut poseduje vrednost `my-id`. Na taj način promenljiva `item` popunjava se referencom na drugi `li` element, odnosno objektom tipa `HTMLLIElement`.

Metoda `querySelector()` uvek vraća samo prvi element koji zadovoljava prosleđeni CSS kriterijum. Kako bismo se uverili da je to stvarno tako, u prikazanom primeru je definisana još jedna naredba, kojom se selektuju elementi korišćenjem nešto drugačijeg CSS selektora - `li`. Na ovaj način se selektuju elementi po tipu. U dokumentu postoje tri elementa takvog tipa, ali će metoda `querySelector()` vratiti samo prvi element. Tako će promenljiva `oneMoreItem` biti popunjena referencom na prvi `li` element.

Sve što je napisano u prethodnim redovima potvrđuje ispis koji se dobija unutar konzole:

```
Item 2
Item 1
```

### innerHTML

U upravo prikazanom primeru, prvi put je iskorišćeno jedno specijalno svojstvo - `innerHTML`. Naime, svi objekti unutar DOM strukture kojima se predstavljaju različiti HTML elementi poseduju svojstvo `innerHTML`. Reč je o svojstvu kojim je moguće pročitati ili postaviti tekst nekog HTML elementa.

## Selektovanje elemenata korišćenjem CSS selektora – `querySelectorAll()`

U prethodim redovima je ilustrovana metoda `querySelector()` i tom prilikom je prikazano da ova metoda uvek vraća samo jedan element na osnovu definisanog CSS selektora. Ukoliko je potrebno obaviti selektovanje većeg broja DOM elemenata korišćenjem CSS selektora, potrebno je koristiti metodu `querySelectorAll()`.

```
<html>

  <head>
    <title>DOM</title>
  </head>

  <body>
    <ul>
      <li>Item 1</li>
      <li id="my-id">Item 2</li>
      <li>Item 3</li>
    </ul>

    <script>
      var items = document.querySelectorAll("li");

      for (let i = 0; i < items.length; i++) {
        console.log(items[i].innerHTML);
      }
    </script>
  </body>
</html>
```

S obzirom na to da metoda `querySelectorAll()` vraća više elemenata, njena povratna vrednost je niz vrednosti. U prikazanom primeru je definisan kod za prolazak kroz takav niz. Unutar `for` petlje se ispisuje unutrašnji tekst svakog `li` elementa, korišćenjem `innerHTML` svojstva. Unutar konzole se dobija:

```
Item 1
Item 2
Item 3
```

## Izmene na HTML elementima

Nakon što se obavi selektovanje jednog ili više elemenata, može se preći na manipulaciju njihovim sadržajem i vrednostima atributa. Tako će u nastavku lekcije biti prikazane tehnike za:

- izmenu sadržaja elemenata;
- dodavanje, brisanje i izmenu vrednosti atributa;
- kreiranje nove i izmenu postojeće stilizacije.

### innerHTML

Svojstvo koje se koristi za rukovanje tekstualnim sadržajem HTML elementa je `innerHTML`. Ono omogućava da se tekstualni sadržaj pročita ili da se upiše novi umesto postojećeg:

```

<html>
  <head>
    <title>DOM</title>
  </head>
  <body>
    <p id="the-paragraph">Text 1</p>

    <script>
      var paragraph = document.getElementById('the-paragraph');
      //reading innerHTML
      console.log("Original paragraph content was: " +
paragraph.innerHTML);
      //writing innerHTML
      paragraph.innerHTML = "New text set using innerHTML
property";
    </script>

  </body>
</html>

```

Primer ilustruje korišćenje `innerHTML` svojstva za čitanje i postavljanje teksta HTML elementa. U primeru, rukuje se paragraf elementom sa id-jem `the-paragraph`.

Prvo se čita originalni tekst ovog paragrafa i informacija o tome se upisuje u konzolu:

```
Original paragraph content was: Text 1
```

Nakon čitanja, vrednost `innerHTML` svojstva se menja, dodeljivanjem nove vrednosti, tako da paragraf na stranici izgleda ovako:

```
New text set using innerHTML property.
```

## Čitanje vrednosti atributa

Objekti koji predstavljaju HTML elemente unutar DOM strukture poseduju svojstva koja je moguće koristiti za pristup vrednostima atributa:

```

<html>
  <head>
    <title>DOM</title>
  </head>
  <body>
    <p id="the-paragraph" class="my-class">Text 1</p>

    <script>
      var paragraph = document.getElementById('the-paragraph');
      console.log("Id: " + paragraph.id);
      console.log("Class: " + paragraph.className);
    </script>

  </body>
</html>

```

Primer će emitovati rezultat:

```
Id: the-paragraph  
Class: my-class
```

## Postavljanje vrednosti atributa

Za postavljanje vrednosti atributa nekog elementa može se koristiti metoda `setAttribute()`. Ova metoda ima mogućnost ne samo da promeni vrednost atributa već i da obavi njihovo kreiranje u situacijama kada oni ne postoje.

Sintaksa `setAttribute()` metode je sledeća:

```
element.setAttribute(attributename, attributevalue);
```

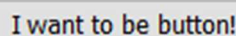
Prvi parametar se odnosi na naziv atributa, čija vrednost se postavlja, dok je drugi parametar vrednost atributa.

```
<html>  
  <head>  
    <title>DOM</title>  
  </head>  
  <body>  
    <input type="text" name="firstname" id="first-name" value="I want  
to be button!">  
    <script>  
      var input = document.getElementById('first-name');  
      input.setAttribute('type', 'button');  
    </script>  
  </body>  
</html>
```

U primeru, definisan je jedan `input` element tipa `text`. Ipak, JavaScript kodom menja se vrednost njegovog atributa `type`, sa `text` na `button`. Pre izvršavanja linije koda u kojoj se to postiže, element na stranici izgleda kao na levoj polovini slike 8.6. Nakon izvršavanja linije koda u kojoj se poziva metoda `setAttribute()`, `input` element zbog nove vrednosti `type` atributa dobija novu vizuelnu reprezentaciju na stranici, pa tako izgleda kao na desnoj polovini slike 8.6.



I want to be button!



I want to be button!

Slika 8.6. Promena `type` vrednosti `input` elementa korišćenjem JavaScripta

## Rukovanje stilizacijom

Stilizacijom jednog elementa se može rukovati korišćenjem `style` svojstva. Sledeći primer ilustruje način na koji je moguće promeniti boju teksta jednog paragrafa:

```

<html>

  <head>
    <title>DOM</title>
  </head>

  <body>
    <p id="the-paragraph" class="my-class">Text 1</p>

    <script>
      var paragraph = document.getElementById('the-paragraph');
      paragraph.style.color = "blue";
    </script>

  </body>

</html>

```

U primeru se koriste svojstva `style` i `color`. Vrednost svojstva `color` se postavlja na `blue`. Na taj način se tekst paragrafa boji u plavo tako što JavaScript postavlja linijsku (*inline*) stilizaciju na element:

```

<p id="the-paragraph" class="my-class" style="color: blue;">Text 1</p>

```

## Dodavanje klasa

Upravo prikazani način za rukovanje stilizacijom podrazumeva direktnu intervenciju nad CSS opisima i njihovo upisivanje u `style` atribut elementa (*inline stilizacija*). Pored ovog pristupa, radi promene stilizacije jednog elementa često se pribegava rešenju koje podrazumeva dodavanje i uklanjanje klasa sa HTML elemenata. Jednostavno, definiše se nekoliko CSS opisa koji se zatim dodeljuju ili uklanjaju sa elemenata korišćenjem vrednosti atributa `class` ili `id`.

Prethodni primer se sada može transformisati na sledeći način:

```

<html>

  <head>
    <title>DOM</title>
  </head>
  <style type="text/css">
    .blue-text {
      color: blue;
    }
  </style>

  <body>
    <p id="the-paragraph" class="my-class">Text 1</p>

    <script>
      var paragraph = document.getElementById('the-paragraph');
      paragraph.classList.add("blue-text");
    </script>

  </body>

</html>

```

Ovoga puta je unutar `style` elementa kreiran CSS opis za stilizovanje svih elemenata sa klasom `blue-text`. Kada se ova klasa doda paragraf elementu, boja njegovog teksta postaje plava. Upravo to se i postiže prikazanim primerom, tako što se nad svojstvom `classList`, koje predstavlja skup klasa elementa, poziva metoda `add()`, za dodavanje nove klase elementu. Tako će nakon izvršavanja prikazanog koda HTML kod paragraf elementa izgledati ovako:

```
<p id="the-paragraph" class="my-class blue-text">Text 1</p>
```

Bitno je primetiti da element poseduje dve klase: `.my-class` i `.blue-text`. Klasa `.my-class` je definisana unutar samog dokumenta, dok je klasa `.blue-text` dinamički dodata korišćenjem JavaScript koda.

## Uklanjanje klasa

Da bi se jedan naziv klase uklonio sa nekog elementa, moguće je koristiti sledeći pristup:

```
<html>

  <head>
    <title>DOM</title>
  </head>
  <style type="text/css">
    .blue-text {
      color: blue;
    }
  </style>

  <body>
    <p id="the-paragraph" class="my-class blue-text">Text 1</p>

    <script>
      var paragraph = document.getElementById('the-paragraph');
      paragraph.classList.remove("blue-text");
    </script>
  </body>
</html>
```

Ovoga puta, paragraf element inicijalno ima dve klase: `.my-class` i `.blue-text`. Naravno, zbog toga je boja teksta paragrafa plava. Da bi se boja teksta vratila na podrazumevanu, moguće je ukloniti klasu `.blue-text` i to se u primeru postiže pozivanjem metode `remove()` nad svojstvom `classList`.

## Dodavanje i brisanje elemenata

U dosadašnjim primerima, prikazano je kako se rukuje elementima koji već postoje unutar DOM strukture. Vrlo moćna funkcionalnost DOM API-a, podrazumeva dinamičko dodavanje i uklanjanje elemenata. Ovo otvara mogućnost za čitav splet različitih interaktivnosti.

Za kreiranje i uklanjanje elemenata koriste se metode prikazane tabelom 8.2.

Metoda	Opis
<code>createElement(element)</code>	kreira HTML element
<code>removeChild(element)</code>	uklanja HTML element
<code>appendChild(element)</code>	dodaje HTML čvor
<code>replaceChild(element)</code>	menja postojeći HTML element
<code>insertBefore(newNode, referenceNode)</code>	omogućava dodavanje elementa pre nekog drugog elementa u DOM strukturi

*Tabela 8.2. Metode za kreiranje i uklanjanje elemenata*

Da bi se neki HTML element dodao u postojeći dokument, njega je prvo potrebno kreirati korišćenjem metode `createElement()`:

```
var newParagraph = document.createElement("p");
```

Na ovaj način, kreira se jedan novi paragraf element. Ipak, ovakav paragraf još nije dodat stranici, a i ne poseduje nikakav sadržaj. Njegov sadržaj se može kreirati ovako:

```
var newContent = document.createTextNode("This is dynamically generated paragraph!");
```

U DOM strukturi, element i njegov tekstualni sadržaj kreiraju se zasebnim objektima. Upravo zbog toga se čvor tekstualnog sadržaja zasebno kreira. Ovako kreiran tekstualni sadržaj još nije povezan ni sa jednim konkretnim elementom. To se postiže upotrebom metode `appendChild()`:

```
newParagraph.appendChild(newContent);
```

Na kraju, preostaje da se paragraf sa sadržajem doda stranici:

```
document.body.appendChild(newParagraph);
```

Kod kompletnog primera je sledeći:

```
<html>
  <head>
    <title>DOM</title>
  </head>
  <body>
    <p id="the-paragraph" class="my-class blue-text">Non-dynamic
    paragraph.</p>
    <script>
      var newParagraph = document.createElement("p");
      var newContent = document.createTextNode("This is dynamically
      generated paragraph!");
      newParagraph.appendChild(newContent);
      document.body.appendChild(newParagraph);
    </script>
  </body>
</html>
```



Primer proizvodi rezultat:

```
Non-dynamic paragraph.  
This is dynamicly generated paragraph!
```

Analizom primera se može videti da se korišćenjem metode `appendChild()` element dodaje na začelje roditeljskog elementa. Ukoliko je potrebno preciznije odrediti lokaciju ubacivanja elementa, može se koristiti metoda `insertBefore()`. Ova metoda prihvata dva parametra, odnosno čvor koji se dodaje i referentni čvor, pre koga će element biti dodan.

Primer identičan prethodnom, ali ovoga puta sa ubacivanjem elementa na početak roditeljskog elementa, izgleda ovako:

```
<html>  
  
  <head>  
    <title>DOM</title>  
  </head>  
  
  <body>  
  
    <p          id="the-paragraph"          class="my-class">Non-dynamic  
paragraph.</p>  
  
    <script>  
      var newParagraph = document.createElement("p");  
      var newContent = document.createTextNode("This is dynamically  
generated paragraph!");  
      newParagraph.appendChild(newContent);  
  
      var oldParagraph = document.getElementById("the-paragraph");  
      document.body.insertBefore(newParagraph, oldParagraph);  
  
    </script>  
  
  </body>  
</html>
```

Ovoga puta, primer proizvodi rezultat:

```
This is dynamically generated paragraph!  
Non-dynamic paragraph.
```

## Uklanjanje elemenata

Da bi se neki element uklonio iz DOM strukture, moguće je koristiti metodu `removeChild()`:

```

<html>

  <head>
    <title>DOM</title>
  </head>

  <body>

    <ul id='myList'>
      <li>Item 1</li>
      <li>Item 2</li>
      <li>Item 3</li>
    </ul>

    <script>
      var list = document.getElementById("myList");
      list.removeChild(list.children[1]);

    </script>

  </body>

</html>

```

Primer proizvodi rezultat:

- Item 1
- Item 3

U primeru je kreirana jedna HTML lista sa tri stavke. U JavaScript kodu, dolazi se do reference na listu u DOM strukturi, a zatim se nad takvim objektom poziva metoda `removeChild()`. Ovoj metodi se prosleđuje objekat koji predstavlja element koji je potrebno ukloniti iz strukture. U primeru se uklanja središnja stavka, a referenca na nju se dobija korišćenjem svojstva `children`.

### Svojstva `children` i `childNodes`

Svojstvo `children` omogućava pristup kolekciji elemenata koji su naslednici nekog elementa. Pri tome se pod pojmom elemenata misli na HTML objekte, a ne na čvorove. Tako ovo svojstvo ima različit efekat od svojstva `childNodes`, koje vraća kolekciju čvorova jednog elementa. Već je rečeno da se elementi i njihov tekstualni sadržaj u DOM strukturi uglavnom predstavljaju različitim čvorovima.

### Zamena elemenata

Pored metoda za kreiranje, dodavanje i uklanjanje, DOM API poseduje i metode za zamenu čvora nekim drugim. Reč je o metodi `replaceChild()`:

```
replaceChild(newChild, oldChild);
```

Metoda `replaceChild()` prihvata dva parametra. Prvi se odnosi na element koji se dodaje, a drugi na element umesto koga se novi element dodaje:

```
<html>

  <head>
    <title>DOM</title>
  </head>

  <body>

    <p          id="the-paragraph"          class="my-class">Non-dynamic
paragraph.</p>

    <script>
      var newParagraph = document.createElement("p");
      var newContent = document.createTextNode("This is dynamically
generated paragraph!");
      newParagraph.appendChild(newContent);

      var oldParagraph = document.getElementById("the-paragraph");
      document.body.replaceChild(newParagraph, oldParagraph);

    </script>

  </body>

</html>
```

U primeru je kreiran HTML dokument sa jednim paragrafom čiji tekst je *Non-dynamic paragraph*. Korišćenjem JavaScript koda, vrši se dinamičko kreiranje novog paragrafa sa tekstom *This is dynamically generated paragraph!*. Dinamički kreirani paragraf se zatim postavlja unutar HTML dokumenta, umesto paragrafa koji je kreiran HTML jezikom, i to korišćenjem metode `replaceChild()`. Na kraju, unutar HTML dokumenta se dobija:

This is dynamically generated paragraph!

## Rezime

- DOM je skraćenica za pojam Document Object Model, što je objektna reprezentacija jednog HTML dokumenta.
- DOM podrazumeva objektnu strukturu u formi stabla.
- Kako bi se rukovalo elementima (čvorovima) DOM strukture, neophodno je prethodno dobiti referencu na objekte koji ih predstavljaju.
- Metoda `getElementById()` obezbeđuje referencu na element sa prosleđenom vrednošću `id` atributa.
- Metoda `getElementsByClassName()` obezbeđuje referencu na sve elemente koji imaju prosleđenu vrednost `class` atributa.
- Metoda `getElementsByTagName()` obezbeđuje referencu na sve elemente na stranici koji su određenog tipa.
- Metoda `querySelector()` obezbeđuje referencu na prvi element koji zadovoljava prosleđeni CSS selektor.

- Metoda `querySelectorAll()` obezbeđuje referencu na sve elemente koji zadovoljavaju kriterijum prosleđenog CSS selektora.
- Svojstvo koje se koristi za rukovanje tekstualnim sadržajem HTML elementa je `innerHTML`.
- Za postavljanje vrednosti atributa nekog elementa može se koristiti metoda `setAttribute()`.
- Stilizacijom jednog elementa se može rukovati korišćenjem `style` svojstva koje svaki DOM objekat poseduje.
- Metoda `createElement()` kreira HTML element.
- Metoda `removeChild()` uklanja HTML čvor.
- Metoda `appendChild()` dodaje HTML čvor.
- Metoda `replaceChild()` menja postojeći HTML element.
- Metoda `insertBefore()` omogućava dodavanje elementa pre nekog drugog elementa u DOM strukturi.

