

JavaScript

Nasleđivanje klasa

Arrow funkcije

DOM

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

Sadržaj

◆ JS Klase:

- ◆ Definicija
- ◆ Instanciranje
- ◆ Nasleđivanje

◆ Arrow funkcije:

- ◆ Princip
- ◆ Sintakse
- ◆ Korišćenje
- ◆ Tajmer

◆ DOM:

- ◆ Struktura
- ◆ Metode
- ◆ Događaji
- ◆ Korišćenje

- ◆ Primeri: codepen.io

ECMA Script

- ◆ JS jezik se kontinuirano usavršava
- ◆ Transponovanje: novi elementi jezika se u pozadini transformišu u postojeće, starije mehanizme
 - Nasleđivanje klasa -> nasleđivanje prototipa
- ◆ Razlika je sintaksna
- ◆ Razlog: kompatibilnost programa sa različitim tipovima i verzijama izvršnih okruženja, tj. pretraživača
- ◆ Moderni okviri za razvoj Web klijentskih aplikacija forsiraju upotrebu novijih elemenata jezika
 - AngularJS nameće upotrebu TypeScript jezika kao proširenja JS-a

JS definicija klase

```
class Car {  
  constructor(wheel, steeringWheel, color) {  
    this.wheel = wheel;  
    this.steeringWheel = steeringWheel;  
    this.color = color;  
  }  
  
  get wheel() { return this._wheel; }  
  set wheel(newWheel) { this._wheel = newWheel; }  
  
  get steeringWheel() { return this._steeringWheel; }  
  set steeringWheel(newSWheel) { this._steeringWheel = newSWheel; }  
  
  get color() { return this._color; }  
  set color(newColor) { this._color = newColor; }  
}
```

JS definicija klase - pravila

- ◆ Ključne reči **class** i **constructor**
- ◆ U konstruktoru se navode parametrima za inicijalizaciju objekata klase
- ◆ **this** se odnosi na **instancu** klase
- ◆ Get/set metode se definišu pomoću ključnih reči **get** i **set**
- ◆ U okviru get/set metoda, atributima objekta se pristupa na specifičan način, *underscore* simbol (**_**) kao prefiks:

_imeAtributa

JS instanciranje klase

```
var carInstance = new Car(4,1,'metallic');  
console.log(carInstance);
```

```
Object {  
  _color: "metallic",  
  _steeringWheel: 1,  
  _wheel: 4  
}
```

JS klase – korišćenje get/set metoda instanci klase

```
carInstance.color = 'red';  
console.log(carInstance.color);
```

```
"red"
```

Kod poziva metoda se navodi samo ime bez zagrada () !

JS nasleđivanje klasa

```
class Car1 extends Car {  
  constructor(wheel, steeringWheel, color, stereo) {  
    super(wheel, steeringWheel, color);  
    this.stereo = stereo;  
  }  
  
  get stereo() { return this._stereo; }  
  set stereo(newStereo) { this._stereo = newStereo; }  
  
  carInfo() {  
    console.log("Wheels:" + this._wheel +  
      " wheels:" + this._steeringWheel +  
      " color:" + this._color +  
      " stereo:" + this._stereo)  
  }  
}
```


JS nasleđivanje klasa - pravila

- ◆ U konstruktoru izvedene klase se mora pozvati konstruktor roditeljske klase – **super**
 - Sa parametrima roditelja
- ◆ Rad sa novim atributima je identičan kao kod roditeljske klase (get/set)
- ◆ Pravila za definisanje metoda su identična kao za proceduralne JS funkcije

JS instanciranje i poziv metode izvedene klase

```
var car1Instance = new Car1(4,1,'red',1);  
car1Instance.carInfo();
```

```
"Wheels:4 wheels:1 color:red stereo:1"
```

Funkcionalno programiranje

Arrow funkcije

Arrow funkcije

- ◆ Koncizniji (sažetiji) način za pisanje JS funkcije
=> (*flat arrow*)
- ◆ Bez upotrebe nekih ključnih reči (*function*) i simbola
- ◆ Uvedene od ES V6 (2015)
- ◆ Sa obzirom da je većina navikla na pisanje funkcije na standardan način u većini jezika, varijante funkcija ćemo obraditi kroz poređenja sa klasičnim načinom pisanja

Osnovna sintaksa

(param1, param2, ..., paramN) => { **statements** }

(param1, param2, ..., paramN) => **expression**

// ekvivalentno sa: => { return expression; }

// Spisak parametara može biti prazan

() => { **statements** }

Osnovna sintaksa se višestrukim parametrima

```
// Osnovna sintaksa sa višestrukim parametrima
// Klasičan način
var multiplyOld = function(x, y) {
    return x * y;
};

// Arrow funkcija
let multiplyNew = (x, y) => { return x * y };
```

Isti rezultat se može postići sa manje kodiranja !

Osnovna sintaksa se jednim parametrom

```
// Osnovna sintaksa sa jednim parametrom
// Klasičan način
var phraseSplitterOld = function phraseSplitter(phrase) {
  return phrase.split(' ');
};

//Arrow funkcija
var phraseSplitterNew = phrase => phrase.split(" ");
|
console.log(phraseSplitterNew("Osnovna sintaksa sa jednim parametrom"));
```

Parametar se ne mora navoditi u zagradama !

```
["Osnovna", "sintaksa", "sa", "jednim", "parametrom"]
```

Osnovna sintaksa bez parametara

```
//Osnovna sintaksa bez parametara
//Klasičan način
var docLogOld = function docLog() {
    console.log(document);
};

//Arrow funkcija
var docLogNew = () => { console.log(document); };
docLogNew(); // #document... <html> ...
```

Moraju se navesti zagrade () !

Ugrađene JS funkcije za rad sa nizovima

- ◆ **Map**

- ◆ **Filter**

- ◆ **Reduce**

- ◆ Podrazumevano rade po principu *for* petlje, tj. iteratora

 - Mehanizam iteratora je ugrađen u definicije funkcije

- ◆ Prednost je što se petlja ne mora programirati, tj. navoditi u kodu, već samo željene operacije nad elementima kolekcije

Arrow funkcija za manipulaciju nizovima

◆ Mapiranje nizova

- ◆ Izdvajanje specifičnih vrednosti iz niza elemenata tipa *object*

reduced_array = original_array.**map**(object => object.property)

```
const smartPhones = [  
  { name: 'iphone', price: 649 },  
  { name: 'Galaxy S6', price: 576 },  
  { name: 'Galaxy Note 5', price: 489 }  
];
```

Mapiranje nizova tipa elementa object

//Klasičan način

```
var pricesOld = smartPhones.map(function(smartPhone) {  
    return smartPhone.price;  
});
```

```
console.log(pricesOld);
```

//Arrow funkcija

```
var pricesNew = smartPhones.map(smartPhone => smartPhone.price);  
console.log(pricesNew);
```

[649, 576, 489]

[649, 576, 489]

Arrow funkcija za manipulaciju nizovima

◆ Filtriranje nizova

- ◆ Izdvajanje specifičnih vrednosti iz niza elemenata koji zadovoljavaju specifičan kriterijum (tj. filter)

`filtered_array = original_array.filter(element => condition)`

```
const array = [1, 2, 3, 4, 5, 6,  
               7, 8, 9, 10, 11,  
               12, 13, 14, 15];
```

Filtriranje nizova

//Klasičan način

```
var filterOld = array.filter(function (v){  
    return v % 3 === 0;  
});  
console.log(filterOld);
```

//Arrow funkcija

```
var filterNew = array.filter(v => v % 3 === 0);  
console.log(filterNew);
```

[3, 6, 9, 12, 15]

[3, 6, 9, 12, 15]

Arrow funkcija za manipulaciju nizovima

◆ Redukcija nizova

- ◆ Agregacija vrednosti niza (sume, prosečne vrednosti, itd.)

value = original_array.**reduce**((total, current) =>
expression with total and current)

```
var euros = [29.76, 41.85, 46.5];
```

Redukcija nizova - suma

```
//Klasičan način
var sumOld = euros.reduce(function(total, amount){
  return total + amount
});
console.log(sumOld);

//Arrow funkcija
var sumNew = euros.reduce((total, amount) => total + amount);
console.log(sumNew);
```

U početnoj iteraciji, *total* ima vrednost prvog elementa niza, *amount* ima vrednost sledećeg elementa. U svakoj narednoj iteraciji drugi parametar (*amount*) se pomera udesno i dodaje kao vrednost datog elementa niza prvom parametru (*total*).

118.11

118.11

setInterval kao klasična funkcija

- ◆ Standardna JavaScript tajmer funkcija
- ◆ Tajmer: pozivanje funkcije (**function**) u redovnim vremenskim intervalima (**milliseconds**)

```
var timer = setInterval(function, milliseconds)  
clearInterval(timer)
```

- ◆ Pokretanje tajmera: poziv setInterval funkcije
- ◆ Zaustavljanje tajmera: poziv clearInterval funkcije

setInterval kao arrow funkcija

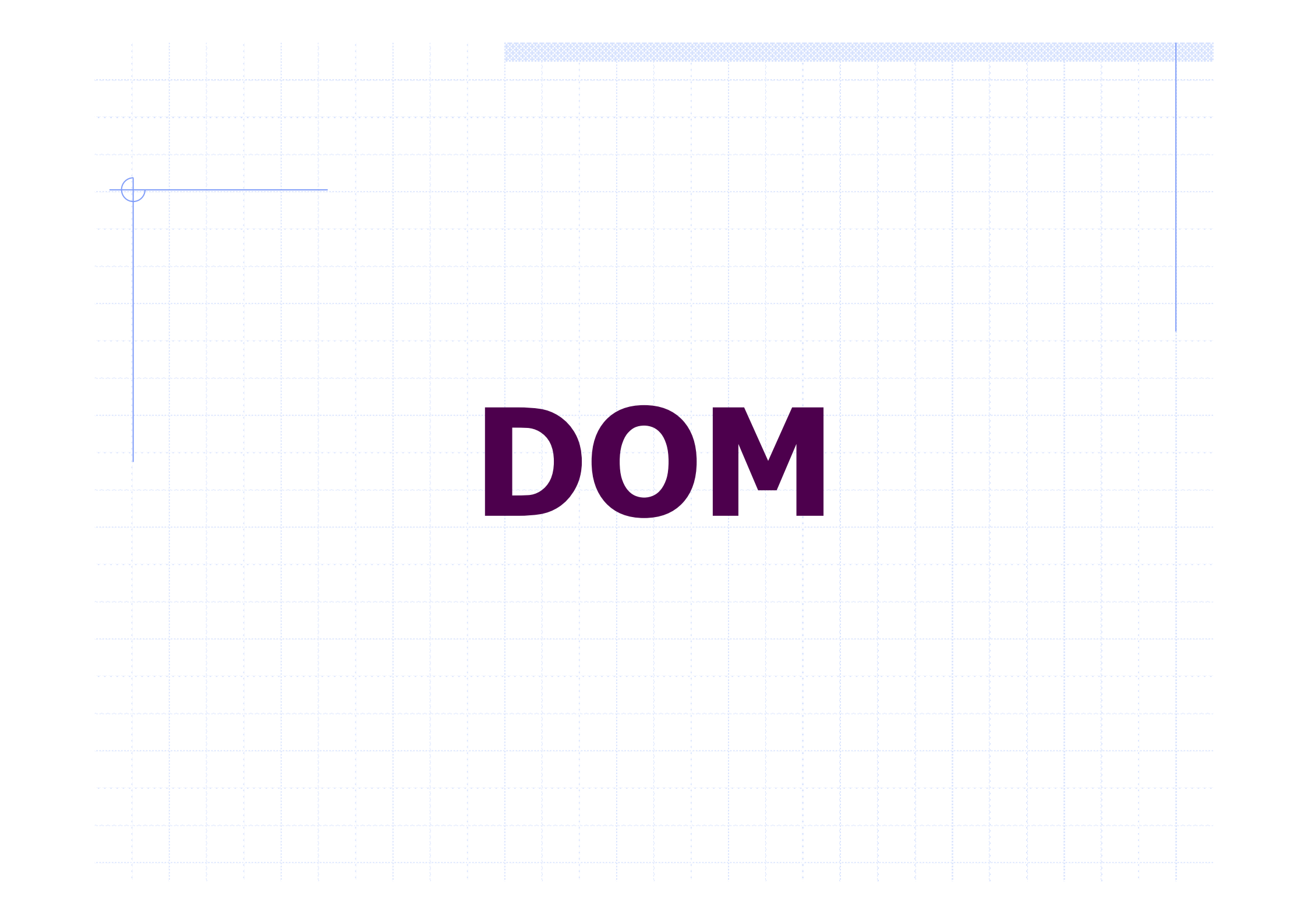
Naredbe (**statements**) se izvršavaju u redovnim vremenskim intervalima (**milliseconds**)

```
var timer = setInterval(( ) => { statements },  
                           milliseconds);
```

clearInterval(timer)

```
function Counter() {  
  var that = this;  
  this.timer = setInterval(() => {  
    console.log(this === that);  
  }, 1000);  
}
```

```
var e = new Counter();  
clearInterval(e.timer);
```

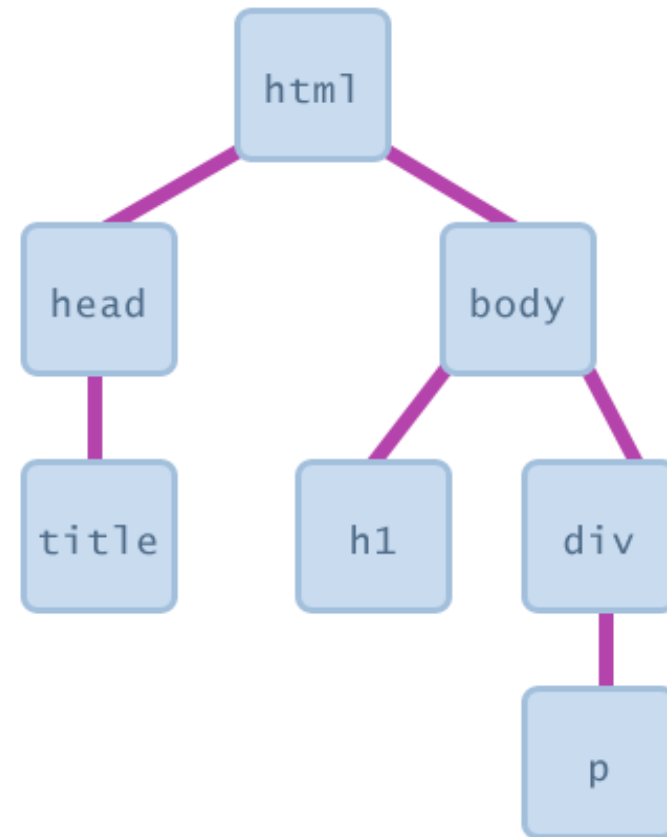


DOM

DOM - Document Object Model

- ◆ Elementi Web stranice u vreme prikazivanja su organizovani kao hijerarhija objekata

```
<html>  
  <head>  
    <title></title>  
  </head>  
  <body>  
    <h1></h1>  
    <div>  
      <p></p>  
    </div>  
  </body>  
</html>
```



DOM

- ◆ DOM je dinamičko stablo HTML objekata, tj. čvorova u vreme izvršavanja programa, tj. prikaza HTML dokument
- ◆ JS može **ispitati** čvorove kako bi ustanovio stanje elementa
 - Na primer, da li je korisnik uneo sadržaj u tekstualno polje
- ◆ JS može **ažurirati** attribute ovih objekata
 - Promena stila ili sadržaja elementa, na primer <p>
- ◆ JS kod može **dodavati** ili **uklanjati** čvorove i time menjati izgled stranice

DOM struktura

- ◆ DOM stablo se sastoji od JS čvorova, instanci tipa **Node**
- ◆ Atributi tipa Node

Atribut	Značenje
textContent	Tekstualni sadržaj čvora i njegove dece
childNodes	Niz čvorova dece
parentNode	Referenca na roditeljski čvor

- ◆ Dva osnovna tipa čvora:
- ◆ **Element** koji predstavlja HTML element (tagovi)
- ◆ **Text** koji predstavlja tekstualni sadržaj tagova, uključujući prazan prostor !
 - Listovi stabla, nemaju elemente decu

```
<head>
</head>
<body>
  <h1>My favorites</h1>
  <section>

    <p>Strawberries</p>
    <p>Chocolate</p>
  </section>
</body>
```

Element čvor predstavlja HTML element (tag)

Text čvor predstavlja tekstualni sadržaj tagova, uključujući prazan prostor !



```
▶ <body>...</body>
▶ NodeList(5) [text, p, text, p, text]
5
```

```
section = document.querySelector('section');
console.log(section.parentNode);
console.log(section.childNodes);
console.log(section.childNodes.length);
```

Atributi i metode DOM čvora

◆ Atributi:

nodeType – tip čvora (Element, Text, Comment, ...)

nodeName – ime čvora

nodeValue – vrednost čvora (null za Element, sadržaj za Text tip čvora); može se koristiti i za čitanje i za dodelu vrednosti

◆ Kretanje po hijerarhijskoj strukturi:

parentNode() – roditeljski čvor

nextSibling() – naredni čvor u istom nivou hijerarhije

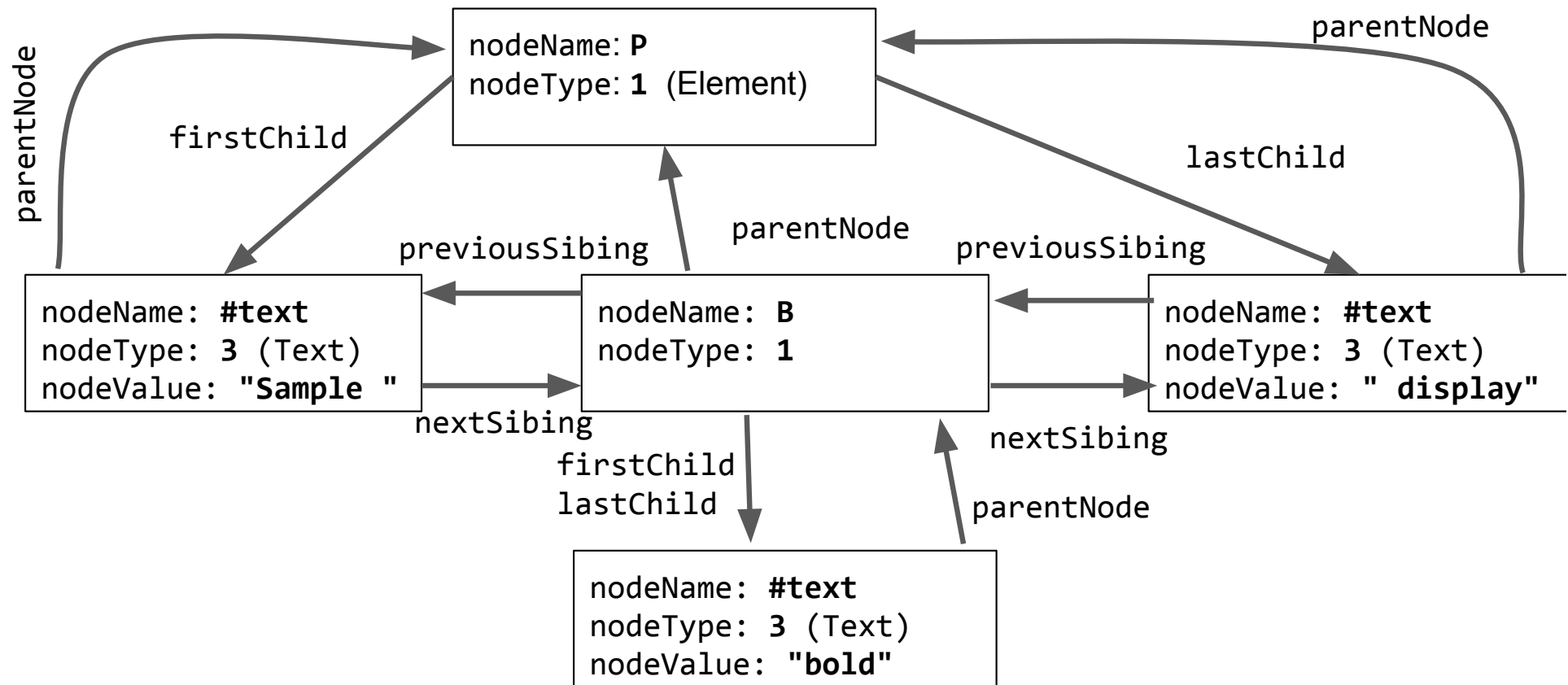
previousSibling() – prethodni čvor u istom nivou hijerarhije

firstChild() – prvi element dete

lastChild() – poslednji element dete

Primer struktura DOM stabla

`<p>Sample bold display</p>`



Kako pristupamo DOM objektu iz JS ?

- ◆ Prvi način:

- ◆ *querySelector* funkcija

document.querySelector(`css selector`);

vraća **prvi** element koji odgovara datom selektoru

document.querySelectorAll(`css selector`);

vraća **sve** elemente koji odgovaraju datom selektoru

querySelector primer

// Vraća sve DOM čvorova koji predstavljaju HTML elemente sa id="button"

```
let element = document.querySelector('#button');
```

// Vraća listu DOM čvorova koji predstavljaju HTML elemente sa class="quote"

```
let elementList =
```

```
document.querySelectorAll('.quote');
```

Kako pristupamo DOM objektu iz JS ?

- ◆ Drugi način:
- ◆ *DOM lookup* metode

document.getElementById('element id');
vraća element sa vrednošću id atributa kao parametra

document.getElementsByClassName('class name');
vraća sve elemente sa vrednošću class atributa kao parametra

document.getElementsByTagName('tag name');
vraća sve elemente sa vrednošću naziva taga kao parametra

DOM lookup primer

HTML: `<div id="div42">...</div>`

let element = `document.getElementById('div42');`

HTML: `<div class="div42"> ... </div>`

let element =
`document.getElementsByClassName('div42');`

HTML: `<div class="div42"> ... </div>`

let element =
`document.getElementsByTagName('div');`

DOM omogućava promenu HTML elemenata u vreme prikazivanja stranice

- ◆ Slučajevi iz prakse:

- ◆ Promena src atributa img taga:

Promena slike na događaj klika dugmeta
`img.src="newImage.jpg"`

- ◆ Prikazivanje/sakrivanje elementa:

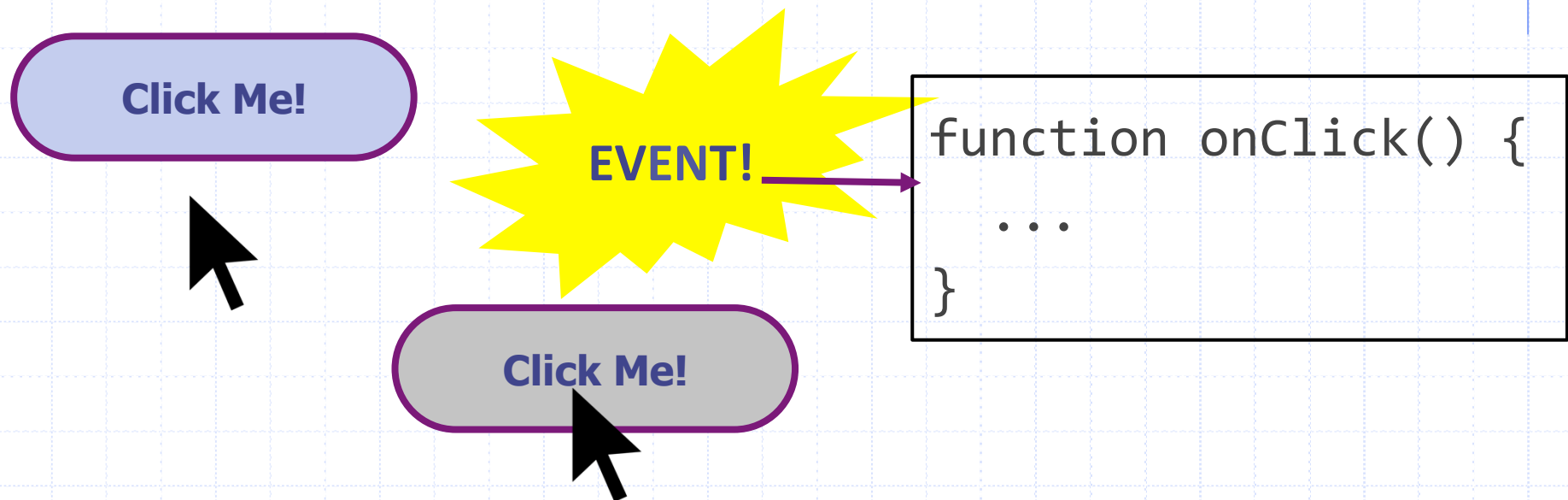
Prikazivanje/sakrivanje generičkih kontejnera (div)

Nevidljivo: `element.style.display = "none";`

Vidljivo: `element.style.display = "";`

DOM događaji

- ◆ Veći deo JS koda se pokreće tako što nastane događaj



Kada korisnik klikne dugme generiše se programski objekat događaja i poziva funkcija koja osluškuje nastanak tog tipa događaja (*event handler*)

Dinamičko dodavanje oslušivača događaja

◆ Svaki DOM objekat ima sledeću metodu:

addEventListener(event name, function name);

event name je naziv događaja čiji nastanak osluškujemo

uobičajeni tipovi: click, focus, blur, ...

function name je ime funkcije koja će biti pozvana kada događaj nastane

Dinamičko uklanjanje oslušivača događaja

- ◆ Da bi uklonili oslušivač događaja, koristimo sledeću metodu:

removeEventListener(event name, function name);

event name je naziv događaja čiji nastanak prestajemo da osluškujemo

function name je ime funkcije koja će biti pozvana kada događaj nastane

HTML atributi i atributi DOM objekata

- ◆ Svaki atribut HTML elementa se preslikava u atribut DOM objekta u vreme prikazivanja stranice

HTML

```

```

JavaScript

```
const element = document.querySelector('img');  
element.src = 'newImage.png';
```

Click for a present:



**DOM dinamička
promena
sadržaja čvora na
primeru img taga –
primer 1**

```
function openPresent() {  
  const image = document.querySelector('img');  
  image.src =  
  'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';  
  image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

Funkcija u kojoj pronalazimo img tag i menjamo vrednost src atributa

```
function openPresent() {  
  const image = document.querySelector('img');  
  image.src =  
  'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';  
  image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

Dodela osluškivača događaja img tagu koji na događaj klika dugmeta poziva funkciju iznad

```
<html>
  <head>
    <meta charset="utf-8">
    <title>DOM primer 1</title>
  </head>
  <body>
    <h1>Click for a present:</h1>
    
  </body>
</html>
```

```
body {
  font-family: monospace;
  text-align: center;
}

img {
  height: 200px;
}
```

Click for a present:



Click for a present:



Atributi Element tipa čvora

Atribut	Značenje
id	Vrednost id atributa HTML elementa kao string
textContent	Tekstualni sadržaj čvora i njegove dece
innerHTML	Originalan HTML sadržaj između početnog i krajnjeg taga
classList	CSS klase primenjene na dati element

Click for a present:



DOM dinamička promena sadržaja čvora na primeru img taga i h1 teksta – primer 2

```
function openPresent(event) {  
  const image = document.querySelector('img');  
  image.src = 'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';  
  
  const title = document.querySelector('h1');  
  title.textContent = 'Hooray!';  
  
  image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

```
<html>
  <head>
    <meta charset="utf-8">
    <title>DOM primer 1</title>
  </head>
  <body>
    <h1>Click for a present:</h1>
    
  </body>
</html>
```

```
body {
  font-family: monospace;
  text-align: center;
}

img {
  height: 200px;
}
```

Click for a present:



Hooray!



Element – izvor događaja

- ◆ Identične naredbe na dva mesta u primeru

```
function openPresent(event) {  
    const image = document.querySelector('img');  
    image.src = 'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';  
  
    const title = document.querySelector('h1');  
    title.textContent = 'Hooray!';  
  
    image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```


JS Event objekat

- ◆ **event.target** – element **izvor** događaja (na primer, slika na koju se klikne). U slučaju da elementu nije pridružen osluškivač, događaj se prenosi kroz DOM čvorove ka roditelju dok se ne nađe na osluškivač registrovan za taj tip događaja
- ◆ Konkretnan element ne mora imati pridružen osluškivač
- ◆ Mehanizam se naziva delegiranje događaja (*event delegation*)
- ◆ **event.currentTarget** – element kojem je pridružen **osluškivač** za konkretan događaj
- ◆ U slučaju da je konkretnom HTML elementu pridružen osluškivač događaj, dva atributa se odnose na isti element
- ◆ U suprotnom mogu vratiti dva različita elementa

Pristup DOM elementu kao izvoru događaja

```
function openPresent() {  
  //const image = document.querySelector('img');  
  const image = event.currentTarget;  
  image.src =  
  'https://media.giphy.com/media/27ppQU0xe7KlG/giphy.gif';  
  image.removeEventListener('click', openPresent);  
}  
  
const image = document.querySelector('img');  
image.addEventListener('click', openPresent);
```

Dinamičko dodavanje/uklanjanje stilova

◆ Mogu se kontrolisati CSS klase primenjene na elemente

classList.add('class1', 'class2') – dodavanje specifičnih CSS klasa koje će se primenjivati na element

classList.remove('class1', 'class2') – uklanjanje specifičnih klasa koje se se primenjuju na element

classList.toggle('class1', 'class2') – ako se specifične klase već primenjuju one se uklanjaju; ako se ne primenjuju dodaju se elementu

Dinamičko dodavanje/uklanjanje stilova

JS:

```
const image = document.querySelector('img');
```

```
// Adds a CSS class called "active"
```

```
image.classList.add('active');
```

```
// Removes a CSS class called "hidden"
```

```
image.classList.remove('hidden');
```

```
//Toggle between active and hidden
```

```
image.classList.toggle('hidden');
```

CSS:

```
.hidden {  
    display: none;  
}
```

```
.active {  
    display: inline;  
}
```

```
.hidden {  
  display: none;  
}
```

DOM dinamička promena sadržaja korišćenjem display – primer 3

```
<head>  
  <meta charset="utf-8">  
  <title>DOM primer 3 display</title>  
</head>  
<body>  
  <div id="gift-outside">  
    <h1>Click for a present:</h1>  
      
  </div>  
  <div id="gift-inside" class="hidden">  
    <h1>Hooray!</h1>  
      
  </div>  
</body>
```

```
function openPresent(event) {  
  const image = event.currentTarget;  
  image.removeEventListener('click', openPresent);  
  
  const giftOutside = document.querySelector('#gift-outside');  
  const giftInside = document.querySelector('#gift-inside');  
  giftOutside.classList.add('hidden');  
  giftInside.classList.remove('hidden');  
}  
  
const image = document.querySelector('#gift-outside img');  
image.addEventListener('click', openPresent);
```

Vidljivost div kontejnera se menja tako što se dodaje i uklanja klasa hidden koja kontroliše display atribut.

Sažetak – promene izgleda stranice pomoću DOM

- ◆ Promena sadržaja postojećeg elementa
 - Primeri 1 i 2: promena sadržaja img i h1 taga
 - Jednostavnije i manje promene sadržaja
- ◆ Kontrola prikaza sekcija stranice organizovanih kao blok elementi grupisanja (div, section) pomoću display
 - Primer 3: naizmeničan prikaz slika
 - Krupnije promene sadržaja stranice
- ◆ Dodavanje/uklanjanje elemenata pomoću naredbi
document.createElement(**tag string**);
element.appendChild(**element**);
element.remove();

Linkovi ka primerima

JavaScript objektno-orijentisano nasleđivanje:

<https://codepen.io/mljovanovic/pen/qvVeVY>

DOM dinamičko dodeljivanje osluškivača i promena sadržaja elementa (img):

<https://codepen.io/mljovanovic/pen/WmzWEL>

DOM dinamičko dodeljivanje osluškivača i promena sadržaja elementa (img, h1):

<https://codepen.io/mljovanovic/pen/PLeqJr>

DOM dinamičko dodeljivanje osluškivača i prikaz/sakrivanje elementa pomoću CSS klase (display):

<https://codepen.io/mljovanovic/pen/wOjWWz>

Arrow funkcije:

<https://codepen.io/mljovanovic/pen/Qordag>

Tajmer (setInterval) kao arrow funkcija:

<https://codepen.io/mljovanovic/pen/PLeOmN>

Sažetak

◆ JS Klase:

- ◆ Definicija
- ◆ Instanciranje
- ◆ Nasleđivanje

◆ Arrow funkcije:

- ◆ Princip
- ◆ Sintakse
- ◆ Korišćenje
- ◆ Tajmer

◆ DOM:

- ◆ Struktura
- ◆ Metode
- ◆ Događaji
- ◆ Korišćenje

◆ Primeri: codepen.io

JavaScript

Nasleđivanje klasa

Arrow funkcije

DOM

Dr. Mlađan Jovanović
mjovanovic@singidunum.ac.rs

U pripremi prezentacije korišćene su ilustracije i primeri sa
<https://developer.mozilla.org>