# Događaji

Jedan od osnovnih pojmova programiranja, bez obzira na to o kojoj platformi je reč, jeste pojam događaja. Događaji su posebno značajni za platforme koje poseduju grafičko okruženje za interakciju sa korisnicima. Upravo zbog toga se i jedna kompletna programerska paradigma naziva programiranje zasnovano na događajima, odnosno *event-driven programming*. S obzirom na to da se web okruženje primarno zasniva na interakciji korisnika i web sajtova korišćenjem grafičkih korisničkih okruženja, pojam događaja zauzima centralnu poziciju i na webu.

Rukovanje događajima na webu, omogućeno je korišćenjem pojmova koji su predstavljeni u prethodne dve lekcije – objektnih modela web pregledača i dokumenta (BOM i DOM). Stoga će u nastavku ove lekcije biti prikazani najznačajniji pristupi za korišćenje i obradu događaja prilikom razvoja modernih web sajtova.

## Šta je događaj?

Događaj je pojam koji opisuje neku pojavu. U prirodi se pojava munje može nazvati događajem. Pojave do kojih dolazi unutar softverskih sistema, a o čijim nastanku nas sam sistem izveštava, mogu se nazvati događajima. Na webu, događaje mogu izazvati web pregledači, ali i korisnici web sajtova. Neki od primera događaja na webu mogu biti:

- klik na dugme;
- prelazak mišem preko HTML elementa;
- završetak učitavanje stranice;
- zatvaranje stranice;
- klik na neko dugme tastature;
- početak učitavanja stranice.

Ovo su samo neki primeri događaja do kojih može doći na webu.

## Pojam obrade događaja

Događaji, sami po sebi, ne unose nikakvu dodatnu vrednost u web sajtove ili aplikacije. Drugim rečima, događaj koji nije detektovan ne poseduje upotrebljivu vrednost za neki sistem. Stoga su događaji stvoreni kako bi bili detektovani, odnosno obrađeni. Tako dolazimo i do pojma obrade događaja.

Obrada događaja podrazumeva definisanje programske logike koja će se aktivirati prilikom pojave nekog događaja. Kod koji se aktivira kada dođe do pojave nekog događaja na engleskom jeziku se naziva **event handler**. Proces kreiranja programske logike koja će se aktivirati prilikom pojave nekog događaja naziva se obrada događaja, prijava, pretplata ili registracija na događaj. Na engleskom jeziku, takav proces se naziva **registering an event handler**.

Obrada događaja, odnosno registracija programske logike koja će se aktivirati prilikom pojave događaja, može se obaviti korišćenjem nekoliko različitih pristupa:

- upotrebom atributa HTML elemenata;
- korišćenjem svojstava DOM objekata koji predstavljaju HTML elemente;
- korišćenjem metode `addEventListener()`, globalnog `Window` objekta.

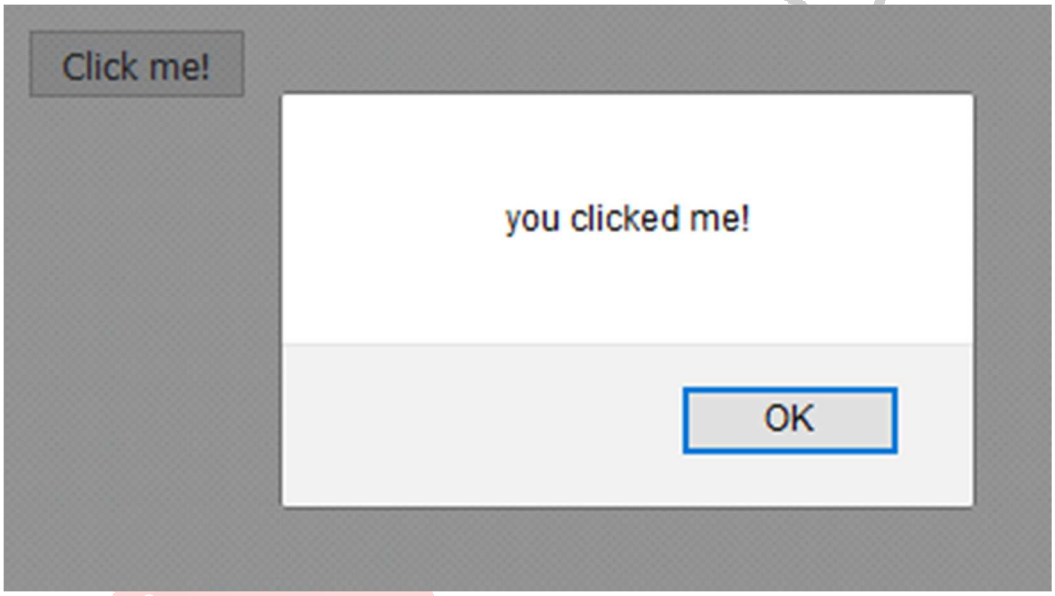
U nastavku ove lekcije prvo će biti ilustrovani upravo navedeni pristupi za registraciju programske logike koja će se aktivirati prilikom pojave nekog događaja. Nakon demonstracije različitih načina za pretplatu, biće ilustrovani i brojni napredniji pristupi rada sa događajima.

## Korišćenje atributa HTML elemenata za pretplatu na događaje

Najjednostavniji način za definisanje JavaScript koda koji će se aktivirati kada dođe do nekog događaja podrazumeva korišćenje atributa HTML elemenata:

```
<button onclick='alert("you clicked me!")'>Click me!</button>
```

Na `button` elementu, definisan je atribut `onclick`, sa vrednošću koja predstavlja logiku koja će se aktivirati u trenutku kada korisnik klikne na ovaj `button` element (slika 9.1).




*Slika 9.1. Klik na button element*

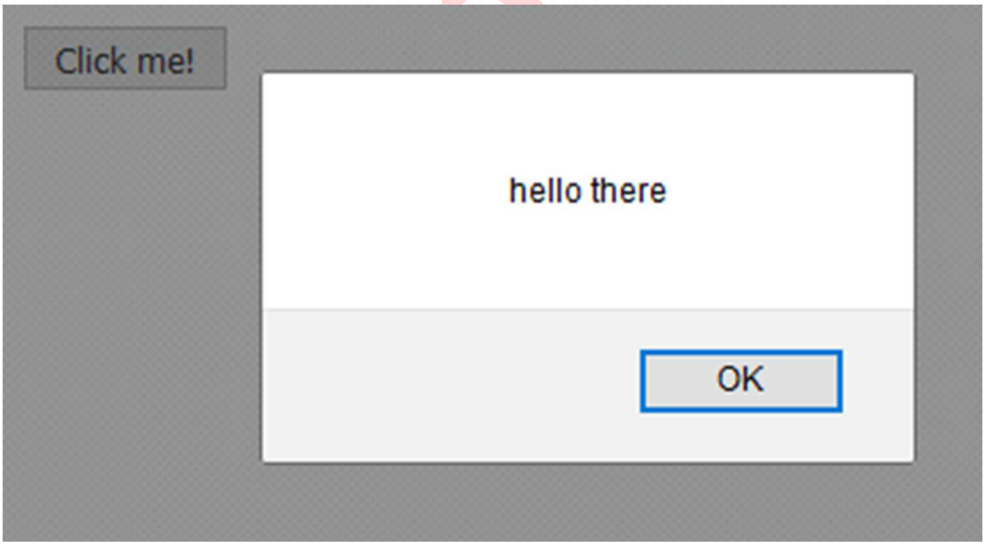
Korišćenje ovakvog pristupa, kojim se piše takozvani linijski JavaScript, nikako se ne preporučuje, zbog ograničenja u pogledu preglednosti, portabilnosti, održavanja... Zato se za definisanje JavaScript koda koji će se izvršiti prilikom pojave nekog događaja preporučuju pristupi koji će biti prikazani u nastavku ove lekcije.

## Reagovanje na događaje korišćenjem događaja kao svojstava

U prethodnom primeru, za obradu događaja korišćeni su atributi HTML elemenata. Identičan efekat može se postići i na nešto drugačiji način. Naime, u prethodnom lekcijama, u kojima je bilo reči o objektnom modelu dokumenta, prikazano je da DOM API omogućava programabilni pristup gotovo svim gradivnim blokovima HTML koda. Upravo zbog toga se atributima koji su prikazani u prethodnom poglavlju može pristupiti i korišćenjem objekata i svojstava DOM strukture:

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

    <body>

        <button id="the-button">Click me!</button>

        <script>

            var button = document.getElementById("the-button");
            button.onclick = displayMessage;

            function displayMessage() {
                alert('hello there');
            }

        </script>

    </body>
</html>
```

Primerom je prikazan HTML dokument sa jednim `button` elementom. JavaScript kodom, prvo se dobija objekat koji predstavlja takav `button` element. Zatim je iskorišćeno svojstvo `onclick`, za čiju vrednost je postavljena referenca na jednu funkciju, koja će se izvršiti kada dođe do pojave događaja. Primer proizvodi rezultat kao na slici 9.2.

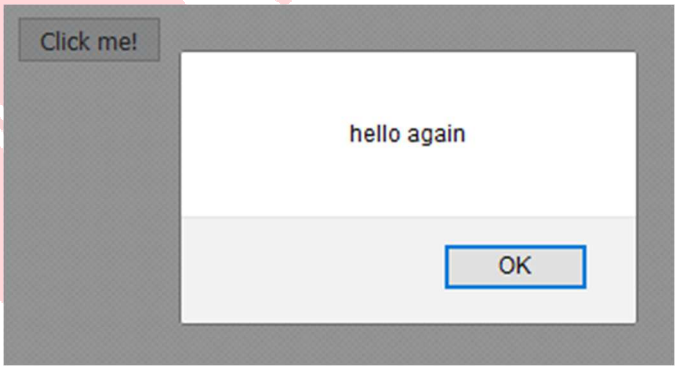



*Slika 9.2. Klik na button element aktivira prikaz modalnog prozora sa porukom*

Prikazana dva načina za registrovanje JavaScript koda koji će se aktivirati kada dođe do pojave događaja potpuno su ravnopravna, odnosno, njima se postiže identičan efekat. Njihov ograničavajući faktor jeste mogućnost registrovanja samo jedne logike koja će se izvršiti prilikom pojave događaja:

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

    <body>

        <button id="the-button">Click me!</button>

        <script>

            var button = document.getElementById("the-button");
            button.onclick = displayMessage;
            button.onclick = displayAnotherMessage;

            function displayMessage() {
                alert('hello there');
            }

            function displayAnotherMessage() {
                alert('hello again');
            }

        </script>

    </body>
</html>
```

U ovom primeru, dva puta je postavljena vrednost svojstva `onclick`. S obzirom na to da je reč o običnom svojstvu, ono može imati samo jednu vrednost. Tako će vrednost svojstva `onclick` biti ona koja je postavljena poslednja, a primer će da proizvede rezultat kao na slici 9.3.




*Slika 9.3. Klik na button element aktivira prikaz popupa*

Kao što se sa slike 9.3. može videti, prilikom klika na `button` element, samo se izvršava logika funkcije `displayAnotherMessage()`, odnosno logika one funkcije, čija pretplata je definisana poslednja.

## addEventListener()

Još jedan način za registrovanje logike koja je potrebno da se izvrši kada dođe do nekog događaja jeste korišćenje metode `addEventListener()`, globalnog `Window` objekta:

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

    <body>

        <button id="the-button">Click me!</button>

        <script>

            var button = document.getElementById("the-button");
            button.addEventListener("click", displayMessage);

            function displayMessage() {
                alert('hello there');
            }

        </script>

    </body>
</html>
```

Metoda `addEventListener()` koristi se za registrovanje JavaScript logike koja će se aktivirati prilikom pojave nekog događaja. Osnovna osobina ovakvog načina za obradu događaja jeste mogućnost višestruke pretplate na jedan isti događaj:

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

    <body>
        <button id="the-button">Click me!</button>

        <script>

            var button = document.getElementById("the-button");

            button.addEventListener("click", displayMessage);
            button.addEventListener("click", displayAnotherMessage);
            function displayMessage() {
                alert('hello there');
            }
            function displayAnotherMessage() {
                alert('hello again');
            }
        </script>

    </body>
</html>
```

U primeru su izvršene dve registracije različitih logika za isti događaj – `click`. Klikom na `button` element, prvo će se izvršiti logika funkcije `displayMessage()`, a zatim i logika funkcije `displayAnotherMessage()`.

Metoda `addEventListener()` ima sledeću sintaksu:

```
addEventListener(<event name>,<event function>,<optional cascade rule>)
```

Metoda `addEventListener()` može da prihvati tri parametra, od kojih su prva dva obavezna:

- `event name` – naziv događaja;
- `event function` – logika koja se aktivira prilikom pojave događaja;
- `cascade rule` – faza u kojoj će događaj biti obrađen (*više reči o ovom opcionim parametru biće u nastavku lekcije)*.

**Metoda removeEventListener()**

Pored metode `addEventListener()` postoji i metoda `removeEventListener()`. Ona omogućava da se prethodno definisana pretplata na neki događaj otkaže. Na primer, kako bi se otkazala jedna od pretplata na `click` događaj iz prethodnog primera, dovoljno je napisati:

```
button.removeEventListener("click", displayMessage);
```

Na ovaj način biće uklonjena jedna pretplata na `click` događaj, koja podrazumeva aktiviranje funkcije `displayMessage()`. Sve ostale registrovane funkcije i dalje će bivati aktivirane prilikom pojave `click` događaja na `button` elementu.

**Pitanje**

Korišćenjem metode `addEventListener()`, moguće je aktivirati veći broj funkcija za jedan događaj na istom elementu.

**a) Tačno.**
b) Netačno.

**Objašnjenje:**

*Za razliku od ostalih pristupa za obradu događaja, korišćenjem metode `addEventListener()` moguće je izvršiti dodavanje više funkcija za isti događaj na istom elementu.*

## Različiti tipovi događaja

U dosadašnjem toku lekcije prikazano je nekoliko pristupa koji se mogu koristiti za definisanje JavaScript koda koji će se aktivirati prilikom pojave nekog događaja. Pored poznavanja ilustrovanih pristupa, uspešno baratanje događajima podrazumeva i poznavanje njihovih naziva.

Postoji veliki broj događaja koji se mogu slušati i u nastavku lekcije oni će biti podeljeni u nekoliko grupa, radi lakšeg razumevanja:

- događaji prozora i dokumenta;
- događaji miša;
- događaji tastature;
- događaji formi;
- touch događaji.

U nastavku lekcije će biti prikazani neki od najznačajnijih događaja svake navede grupe.

## Događaji prozora i dokumenta

Događaji prozora i dokumenta odnose se na globalne osobine taba unutar koga se web sajt prikazuje, ali i na osobine samog dokumenta. Najznačajniji takvi događaji prikazani su tabelom 9.1.

| Događaj | Svojstvo događaja | Opis |
|---|---|---|
| `close` | `onclose` | poziva se prilikom zatvaranja prozora, odnosno taba |
| `load` | `onload` | događaj do koga dolazi nakon učitavanja svih resursa stranice i generisanja DOM strukture |
| `DOMContentLoaded` | / | događaj koji se aktivira kada se kreira kompletno DOM stablo; ipak, to ne mora da znači da su i svi ostali resursi učitani (stilizacija, skripte, slike...) |
| `readystatechange` | `onreadystatechange` | događaj koji se emituje kada se promeni vrednost svojstva `readyState`, objekta `Document` |
| `resize` | `onresize` | događaj koji se emituje prilikom svake promene veličine prozora |
| `scroll` | `onscroll` | događaj koji se aktivira prilikom pomeranja traka za skrolovanje |
| `unload` | `onunload` | događaj do koga dolazi kada se unutar prozora/taba učita neki drugi dokument ili kada se prozor/tab zatvori |

*Tabela 9.1. Događaji prozora*

Svakako najinteresantniji skup događaja prozora i dokumenta su oni koji omogućavaju dobijanje dojava o učitavanju HTML dokumenta. U ovom trenutku verovatno ne možete da naslutite koliko su informacije o učitavanju dokumenta značajne. Stoga ćemo razmotriti sledeći primer:

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

    <body>
        <script>

            var button = document.getElementById("the-button");
```

```
            button.addEventListener("click", displayMessage);

            function displayMessage() {
                alert('hello there');
            }


        </script>

        <button id="the-button">Click me!</button>

    </body>
</html>
```

Kod prikazanog primera za nas nije novina. Unutar prikazanog HTML dokumenta nalazi se jedan `button` element i JavaScript kod kojim se definiše jednostavna logika koja će se aktivirati prilikom klika na takav `button` element. Ukoliko funkcionisanje primera isprobate unutar web pregledača, videćete da kod ne funkcioniše onako kako se od njega očekuje:

```
Uncaught TypeError: Cannot read property 'addEventListener' of null
```

Unutar konzole se dobija vrlo razumljiva poruka izuzetka – metoda `addEventListener()` ne može se pozvati nad `null` vrednošću. To praktično znači da je vrednost promenljive `button` jednaka `null`, a ne, kako se očekivalo, objekat koji predstavlja referencu na `button` element. Razlog ovakvog ponašanja vrlo je jednostavan – JavaScript kod kojim se pokušava dobiti referenca na `button` element u dokumentu je naveden pre samog `button` HTML elementa. S obzirom na to da se parsiranje dokumenta obavlja od vrha ka dnu, u prikazanom primeru, JavaScript kod pokušava da pristupi DOM-u pre nego što je on uopšte konstruisan od strane web pregledača. Upravo zbog toga, promenljiva `button` dobija vrednost `null`.

Ovakav primer ilustruje jednu veoma važnu osobinu integracije JavaScript koda u HTML dokument – lokacija na koju se JavaScript integriše u dokument veoma je važna. Ipak, događaji koji omogućavaju dobijanje dojava o učitavanju dokumenta mogu pomoći u ovakvim situacijama:

```
<html>
    <head>
        <title>Events</title>
    </head>
    <body>
        <script>
            window.addEventListener("load", function () {
                var button = document.getElementById("the-button");
                button.addEventListener("click", displayMessage);
            });
            function displayMessage() {
                alert('hello there');
            }
        </script>
        <button id="the-button">Click me!</button>
    </body>
</html>
```

Sada je na primeru načinjena mala izmena. Definisano je da će se dobijanje objekta koji predstavlja `button` element obaviti tek kada se aktivira `load` događaj. Jednostavno, obavljena je pretplata na `load` događaj korišćenjem metode `addEventListener()`. Logika koja će se aktivirati prilikom pojave `load` događaja definisana je unutar anonimne funkcije koja je metodi `addEventListener()` prosleđena kao drugi parametar. Tako su naredbe za dobijanje objekta koji predstavlja `button` element i za pretplatu na `click` događaj na takav element premeštene unutar funkcije koja će se aktivirati tek kada se kompletan dokument i sve pripadajuće zavisnosti učitaju. Upravo zbog toga, više nije bitno gde se JavaScript kod nalazi, odnosno da li je u strukturi dokumenta pre ili posle `button` HTML elementa.

U primeru je ilustrovano korišćenje `load` događaja. Međutim, identičan efekat je mogao biti postignut i korišćenjem objekta `DOMContentLoaded`. Reč je o DOM događaju koji se aktivira odmah nakon što web pregledač kreira DOM stablo. Drugim rečima, emitovanje ovog događaja ne čeka na učitavanje svih zavisnosti, već samo objektnog modela. Upravo zbog toga se on može upotrebiti u već prikazanom primeru:

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

    <body>

        <script>

            document.addEventListener("DOMContentLoaded", function () {
                var button = document.getElementById("the-button");
                button.addEventListener("click", displayMessage);
            });


            function displayMessage() {
                alert('hello there');
            }


        </script>

        <button id="the-button">Click me!</button>

    </body>
</html>
```

Sada je u primeru metoda `addEventListener()` pozvana nad `document` objektom, s obzirom na to da se želi obaviti pretplata na jedan od DOM objekata. Događaj za koji se obavlja definisanje pripadajuće logike je `DOMContentLoaded`. Efekat primera je identičan prethodnom.

**DOMContentLoaded ili load**

Kada se JavaScript kod dokumenta oslanja na neku od eksternih zavisnosti (fajl stilizacije, sliku, dokument...), potrebno je koristiti `load` događaj `Window` objekta. Ipak, ukoliko se klijentska skripta oslanja samo na elemente DOM strukture, sasvim je dovoljno koristiti `DOMContentLoaded` događaj.

Pored dva upravo prikazana događaja za dobijanje dojava o završetku učitavanja dokumenta, moguće je koristiti još jedan DOM događaj – `readystatechange`. Reč je o događaju koji se aktivira kada dođe do promene vrednosti svojstva `readyState`, objekta `document`. Svojstvo `readyState` sadrži vrednost koja ukazuje na stanje u kome se dokument nalazi, ukoliko se posmatra kroz prizmu učitavanja. Ovo svojstvo može imati jednu od sledeće 3 vrednosti:

- **loading** – označava da se dokument trenutno učitava;
- **interactive** – označava da je dokument spreman za korišćenje, što praktično znači da je web pregledač kreirao DOM;
- **complete** – stanje koje označava da je kompletan dokument sa svim svojim eksternim resursima učitan.

Ukoliko se malo bolje pogledaju vrednosti koje svojstvo `readyState` može imati, lako se može zaključiti da se `readystatechange` događaj može koristiti kao alternativa za oba nešto ranije prikazana događaja. Zapravo, `interactive` vrednost `readyState` svojstva odgovara osobinama `DOMContentLoaded` događaja, pa je tako moguće napisati:

```
document.addEventListener("readystatechange", function () {

    if (document.readyState === "interactive") {
        // DOM is ready
    }

});
```

Vrednost `complete`, svojstva `readyState` po svojim osobinama odgovara događaju `load`, `window` objekta, pa je tako moguće napisati sledeće:

```
document.addEventListener("readystatechange", function () {

    if (document.readyState === "complete") {
        // DOM and all resources are loaded
    }

});
```

## Događaji miša

Web pregledači JavaScript kodu stavljaju na raspolaganje i određeni broj događaja miša. Tako je moguće dobiti dojavu o kliku na miš, ali i o kretanju kursora miša po površini sajta. Tabela 9.2. ilustruje najznačajnije takve događaje.

| Događaj | Svojstvo događaja | Opis |
|---|---|---|
| `click` | `onclick` | događaj do koga dolazi kada korisnik klikne na element |
| `dblclick` | `ondblclick` | događaj do koga dolazi kada korisnik izvrši dupli klik na element |
| `mousedown` | `onmousedown` | događaj do koga dolazi kada korisnik pritisne taster miša sa strelicom iznad elementa |
| `mouseenter` | `onmouseenter` | događaj do koga dolazi kada strelica miša uđe u oblast elementa |
| `mouseleave` | `onmouseleave` | događaj do koga dolazi kada strelica miša napusti oblast elementa |
| `mousemove` | `onmousemove` | događaj do koga dolazi kada se strelica miša pomera, a nalazi se iznad elementa |
| `mouseover` | `onmouseover` | događaj do koga dolazi kada strelica miša uđe u oblast elementa, ali i bilo kojeg njegovog potomka |
| `mouseout` | `onmouseout` | događaj do koga dolazi kada strelica miša napusti oblast elementa, ali i bilo kojeg njegovog potomka |
| `mouseup` | `onmouseup` | događaj do koga dolazi kada korisnik otpusti taster miša, a strelica se nalazi iznad elementa |

*Tabela 9.2. Događaji miša*

Korišćenjem `click` događaja moguće je *uhvatiti* klik na bilo koji HTML element unutar dokumenta. Sledeći primer ilustruje jednu takvu situaciju:

```
<html>
    <head>
        <title>Events</title>
    </head>
    <style>
        #my-div {
            height: 300px;
            background-color: blueviolet;
            font-family: sans-serif;
            font-size: 28px;
            text-align: center;
            color: white;
            line-height: 300px;
        }
    </style>
    <body>
        <div id="my-div">Hello! I am div. You can click me!</div>
        <script>
            var div = document.getElementById("my-div");
            div.addEventListener("click", function () {
                alert("You have clicked on div!");
            });
        </script>
    </body>
</html>
```

Primer ilustruje HTML dokument sa jednim `div` elementom. JavaScript kodom je obavljena pretplata na `click` događaj na takav `div` element. Stoga se klikom na kreirani `div` element obavlja prikaz modalnog prozora korišćenjem metode `alert()`.

Pored `click` događaja, web pregledači izlažu i nekoliko događaja koje je moguće koristiti za praćenje kretanja pokazivača miša u odnosu na prisutne HTML elemente. Tako je događaje `mouseenter` i `mouseleave` moguće koristiti za detekciju ulaska pokazivača miša u oblast nekog elementa i izlaska pokazivača iz te oblasti:

```
var div = document.getElementById("my-div");

div.addEventListener("mouseenter", function () {
    div.innerHTML = 'The mouse pointer entered the div
area!';
});

div.addEventListener("mouseleave", function () {
    div.innerHTML = 'The mouse pointer left the div area!';
});
```

Sada je obavljena pretplata na dva događaja – `mouseenter` i `mouseleave`. Događaj `mouseenter` aktiviraće se kada kursor miša uđe u okvire `div` elementa. Tom prilikom se tekst `div` elementa postavlja na *The mouse pointer entered the div area!*

Kada kursor miša napusti oblast `div` elementa, aktivira se događaj `mouseleave` i tom prilikom se njegov tekst dinamički postavlja na *The mouse pointer left the div area!*

Efekat upravo prikazanog primera ilustrovan je animacijom 9.1.


*Animacija 9.1. Primer korišćenja mouseenter i mouseleave događaja*

Događaj `mousemove` se može koristiti za detekciju kretanja pokazivača miša iznad nekog HTML elementa. Jedan takav primer može da izgleda ovako:

```
var div = document.getElementById("my-div");
let eventCounter = 0;

div.addEventListener("mousemove", function () {
    div.innerHTML = 'The mouse pointer is moving in div
area. (' + eventCounter + ')';
    eventCounter++;
});
```

Događaj `mousemove` emituje se za bilo koje kretanje kursora miša iznad elementa na kome se takav događaj sluša. Zbog toga je logika prikazanog primera nešto drugačija. Pored postavljanja prigodnog teksta unutar `div` elementa, obavlja se i štampanje vrednosti promenljive `eventCounter`, koja služi kao brojač. Ovakav brojač će nam omogućiti da uvidimo u kojim situacijama se `mousemove` događaj aktivira. Naime, događaj `mousemove` se može aktivirati stotinu, pa i hiljadu puta, a sve u zavisnosti od kretanja pokazivača miša iznad elementa. Efekat koda je ilustrovan animacijom 9.2.


*Animacija 9.2. Primer korišćenja mousemove događaja*

Još neki zanimljivi događaji različitih tipova prikazani su tabelama 9.3, 9.4. i 9.5.

## Događaji tastature

| Događaj | Svojstvo događaja | Opis |
|---|---|---|
| `keydown` | `onkeydown` | događaj do koga dolazi kada korisnik pritisne neki taster na tastaturi |
| `keypress` | `onkeypress` | događaj do koga dolazi kada korisnik pritisne neki taster na tastaturi i tako nešto proizvede karakter |
| `keyup` | `onkeyup` | događaj do koga dolazi kada korisnik otpusti neki taster na tastaturi |

*Tabela 9.3. Događaji tastature*

## Događaji form elementa

| Događaj | Svojstvo događaja | Opis |
|---|---|---|
| `blur` | `onblur` | događaj do koga dolazi kada element izgubi fokus |
| `focus` | `onfocus` | događaj do koga dolazi kada element dobije fokus |
| `input` | `oninput` | događaj do koga dolazi kada input element dobije sadržaj |
| `submit` | `onsubmit` | događaj do koga dolazi kada se forma prosledi |

*Tabela 9.4. Događaji form elementa*

## Touch događaji

| Događaj | Svojstvo događaja | Opis |
|---|---|---|
| `touchstart` | `ontouchstart` | događaj do koga dolazi kada korisnik dodirne ekran uređaja osetljivog na dodir |
| `touchmove` | `ontouchmove` | događaj do koga dolazi kada korisnik prevlači prst po ekranu osetljivom na dodir |
| `touchend` | `ontouchend` | događaj do koga dolazi kada korisnik skloni prst sa ekrana osetljivog na dodir |

*Tabela 9.5. Touch događaji*

## Objekat događaja

U dosadašnjem toku lekcije je prikazano da pretplata na neki događaj podrazumeva definisanje logike koja će se aktivirati kada dođe do pojave takvog događaja. Definisanje logike jednostavno se postiže kreiranjem funkcije, koja se drugačije naziva *event handler*. U prethodnim primerima je kreirano nekoliko takvih funkcija, kako imenovanih tako i anonimnih. Ipak, do sada nije rečeno da funkcije koje se pozivaju prilikom pojave nekog događaja mogu da prihvate i jedan parametar. Reč je o objektu događaja.

Objekat događaja je objekat koji sadrži informacije o samom događaju. Na primer, kada korisnik pritisne neki taster na tastaturi, dolazi do pojave `keydown` događaja, a unutar objekta događaja nalaze se informacije o tome koji taster je pritisnut:

```
document.addEventListener("keydown", logKey);

function logKey(e){
      console.log(e.key);
}
```

U primeru je za obradu događaja `keydown` registrovana funkcija sa nazivom `logKey`. Ovoga puta takva funkcija prihvata i jedan parametar sa nazivom `e`. Reč je o objektu događaja. Naziv promenljive koja predstavlja parametar funkcije je proizvoljan, a najčešća praksa jeste korišćenje naziva **e**, **evt** ili **event**.

Unutar funkcije za obradu događaja, čita se svojstvo `key` objekta događaja, i njegova vrednost se ispisuje unutar konzole. Svojstvo `key` ukazuje na taster koji je pritisnut na tastaturi.

**Napomena**

*Različiti događaji imaju objekte događaja sa različitim svojstvima.*

Prethodni primer je ilustrovao korišćenje objekta događaja sa specifičnim svojstvom `key`. Naime, različiti događaji poseduju objekte događaja sa različitim svojstvima. Ipak, zajedničko za sve događaje jeste postojanje svojstva `target`.

Svojstvo **target** poseduje referencu na HTML element koji je proizveo događaj:

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

    <style>
        #my-div {
            height: 300px;
            background-color: blueviolet;
            font-family: sans-serif;
            font-size: 28px;
            text-align: center;
            color: white;
            line-height: 300px;
        }
    </style>

    <body>

        <div id="my-div">Hello! I am div. You can click me!</div>

        <script>

            var div = document.getElementById("my-div");

            div.addEventListener("click", function (e) {
                console.log(e.target);
            });

        </script>

    </body>
</html>
```

Ovo je primer koji je već prikazan nešto ranije u ovoj lekciji. Podrazumeva obradu `click` događaja, `button` elementa. Ipak, sada je unutar anonimne funkcije za obradu događaja obavljeno ispisivanje vrednosti svojstva `target`, objekta događaja. Stoga, kada se klikne na `div` element, svojstvo `target` dobija vrednost tipa `HTMLDivElement`, odnosno objekat koji predstavlja element koji je predmet događaja.

Svojstvo `target` veoma je korisno u situacijama kada se jedna ista funkcija definiše kao logika koja će se aktivirati za događaj na većem broju različitih elemenata. Kako biste razumeli na šta se konkretno misli, dat je sledeći primer:

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

    <body>

        <ul>
            <li>Item 1</li>
```

```
            <li>Item 2</li>
            <li>Item 3</li>
        </ul>

        <script>

            var items = document.getElementsByTagName("li");

            for (let i = 0; i < items.length; i++) {
                items[i].addEventListener("click", function (e) {
                    alert(e.target.innerHTML);
                });

            }

        </script>

    </body>
</html>
```

U prikazanom primeru, HTML dokument poseduje jednu neuređenu listu sa tri stavke. JavaScript kodom prvo se obavlja selektovanje svih elemenata tipa `li`. Tako se unutar promenljive `items` smešta niz sa tri objekta koji predstavljaju reference na tri stavke liste. Zatim se unutar `for` petlje prolazi kroz takav niz i obavlja se pretplata na `click` događaj za svaki od takvih elemenata. Tako se unutar for petlje zapravo definiše jedna ista logika koja će se aktivirati prilikom klika na bilo koju od tri stavke liste. U ovakvoj situaciji se postavlja pitanje – kako da znamo na koju stavku je korisnik kliknuo?

Odgovor na ovo pitanje upravo se krije u upotrebi svojstva `target`. Zbog toga je u primeru `target` svojstvo iskorišćeno kako bi se došlo do konkretnog elementa koji je predmet događaja. Svojstvom `innerHTML` čita se tekst stavke koja je predmet `click` događaja. Sve to će na kraju omogućiti da se klikom na stavku liste unutar modalnog prozora prikaže njen tekst (*Item 1*, *Item 2*...).

## Otkazivanje podrazumevanog ponašanja događaja

Određene radnje na HTML stranici proizvode događaje koje pregledač automatski obrađuje. Dva osnovna primera takvih događaja su:

- klik na linkove (a elemente) i
- prosleđivanje forme.

Web pregledači i JavaScript jezik omogućavaju da se obavi otkazivanje podrazumevane akcije koju proizvodi pojava nekog događaja. Tako je moguće otkazati prosleđivanje forme ili preusmeravanje web pregledača na adresu definisanu nekim linkom.

U nastavku će otkazivanje podrazumevanog ponašanja događaja biti ilustrovano na primeru link elemenata. Klikom na neki link, web pregledač automatski vrši preusmeravanje korisnika na adresu koja je definisana unutar `href` atributa. Takvo podrazumevano ponašanje se može otkazati na sledeći način:

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

    <body>

        <a href="http://www.link-group.eu/" id="the-link">Link Group</a>

        <script>

            var link = document.getElementById("the-link");
            link.addEventListener("click", preventDefaultEvent);

            function preventDefaultEvent(e) {

                e.preventDefault();
                console.log("Default event is prevented!");
                return false;

            }


        </script>

    </body>
</html>
```

U primeru je kreiran jedan a element, koji klikom vodi na eksternu stranicu. JavaScript kodom obrađen je `click` događaj na takav link element. Za logiku koja će se aktivirati prilikom klika definisana je logika unutar funkcije sa nazivom `preventDefaultEvent`. Unutar funkcije `preventDefaultEvent()`, nad objektom događaja pozvana je metoda **preventDefault()**. Ovo je metoda koja otkazuje podrazumevano ponašanje događaja. U prikazanom primeru to će biti preusmeravanje na stranicu definisanu `href` atributom. Rezultat prikazanog primera će biti ispisivanje poruke: `Default event is prevented!`, bez preusmeravanja na ULR definisan `href` atributom.

## Propagiranje događaja

Jedna od osnovnih osobina JavaScript događaja jeste propagiranje. Propagiranje događaja je pojam koji je usko povezan sa strukturom roditelj-dete, koju elementi međusobno grade unutar HTML dokumenata.

Za demonstraciju propagiranja događaja, biće korišćena sledeća HTML struktura:

```
<html>
<head>
      <title>Events</title>
      <style type="text/css">
            #div1{
                  width: 400px;
                  height: 400px;
```

```
                  background-color: #ABD486;
                  display: flex;
                  justify-content: center;
                  align-items: center;
            }

            #div2{
                  width: 300px;
                  height: 300px;
                  background-color: #5DC2B0;
                  display: flex;
                  justify-content: center;
                  align-items: center;

            }

            #div3{
                  width: 200px;
                  height: 200px;
                  background-color: #F06270

            }
      </style>
</head>
<body>

      <div id="div1">
            <div id="div2">
                  <div id="div3">
                  </div>
            </div>
      </div>

</body>
</html>
```

Prikazani HTML dokument sastoji se iz tri `div` elementa, koji su postavljeni jedan unutar drugog. Roditeljski element je `div1`, njegov direktan potomak `div2`, dok je potomak elementa `div2` element `div3`. Prikazani kod na stranici proizvodi efekat kao na slici 9.4.

*Slika 9.4. Tri div elementa jedan unutar drugog*

Pitanje je sledeće:

*Kada se izvrši klik na element koji je u strukturi elemenata najdublje (div3), da li će se takav klik tretirati samo kao klik na div3 element ili kao klik i na elemente div2 i div1?*

Elementi `div2` i `div3` svakako su članovi elementa `div1`, pa je klik na `div3` element zapravo klik i na `div2` i na `div1`. Da li je to tako?

Da bi se odgovorilo na postavljena pitanja, biće definisana odgovarajuća logika koja će se aktivirati prilikom klika na bilo koji od tri `div` elementa:

```
var div1 = document.getElementById("div1");
var div2 = document.getElementById("div2");
var div3 = document.getElementById("div3");

div1.addEventListener("click", eventHandler);
div2.addEventListener("click", eventHandler);
div3.addEventListener("click", eventHandler);

function eventHandler(e){

      console.log("Hello from " + this.id);

}
```

Ukoliko se izvrši klik na `div3` element, u konzoli se dobija sledeći ispis:

```
Hello from div3
Hello from div2
Hello from div1
```

Analizom izlaza se može zaključiti da se klik na `div3` element tretira kao klik i na `div3` element, ali i na `div2` i na `div1` elemente.

**this, target i currentTarget**

U upravo prikazanom primeru može se primetiti upotreba ključne reči `this` unutar funkcija koje se aktiviraju kada dođe do pojave događaja. Može se videti i to da se korišćenjem ove ključne reči dolazi do vrednosti `id` atributa koji su definisani na `div` elementima. Kao logično se može postaviti pitanje zbog čega u ovakvoj situaciji nije iskorišćeno nešto ranije prikazano `target` svojstvo objekta događaja?

Svojstvo `target` objekta događaja uvek se odnosi na konkretan element koji je predmet događaja.

Ključna reč `this` unutar funkcija za obradu događaja odnosi se na element na kome je obavljena konkretna pretplata.

Stoga, da je u prikazanom primeru umesto ključne reči `this` iskorišćeno svojstvo `target` objekta događaja, prilikom klika na `div3` element unutar konzole bi se dobilo:

```
Hello from div3
Hello from div3
Hello from div3
```

S obzirom na to da se `target` uvek odnosi na konkretan element koji je izazvao događaj, sva tri event handlera bi kao predmet događaja imali upravo `div3` element.

Alternativa ključnoj reči `this` može biti jedno posebno svojstvo objekta događaja – **currentTarget.** Naime, objekti događaja, pored svojstva `target`, poseduju i svojstvo `currentTarget`. Ono uvek obezbeđuje referencu na element za koji je obavljena pretplata, baš kao i ključna reč `this`.

Upravo prikazani primer ilustrovao je jednu vrlo važnu osobinu JavaScript događaja – propagiranje. Iz primera se moglo videti da propagiranje predstavlja prenošenje jednog događaja sa dece na roditelje. Ipak, pojam propagiranja krije još neke zanimljivosti. Naime, JavaScript događaji prolaze kroz dve etape nakon aktiviranja na nekom elementu. To su faze:

- capturing i
- bubbling.

Događaj se prvo propagira od najvišeg elementa u hijerarhiji do najnižeg elementa, koji je u suštini i izazvao događaj. Ta faza se naziva *capturing faza*. Nakon što je događaj stigao na najniži, izvorišni element, on započinje fazu koja se naziva *bubbling*, odnosno propagiranje događaja nazad na vrh. Sve ovo je ilustrovano slikom 9.5.


*Slika 9.5. Capturing i bubbling faze propagiranja događaja*

Svi događaji se u JavaScript jeziku podrazumevano *hvataju* u bubbling fazi, i baš zbog toga je u prethodnom primeru prvo aktiviran event handler `div3` elementa.

Sada se postavlja pitanje kako *uhvatiti* događaje tokom *capturing* faze?

Odgovor na ovo pitanje leži u sintaksi nešto ranije obrađene metode za registrovanje logike koja će se izvršiti prilikom pojave nekog događaja:

```
addEventListener(<event name>,<event function>,<optional cascade rule>)
```

Reč je, naravno, o metodi `addEventListener()`, koja kao treći parametar prihvata jednu `boolean` vrednost. Ovaj parametar podrazumevano ima vrednost `false`, a definiše da li će događaji biti obrađeni tokom *capturing* ili *bubbling* faze.

```
div1.addEventListener("click", eventHandler, true);
div2.addEventListener("click", eventHandler, true);
div3.addEventListener("click", eventHandler, true);
```

Ovoga puta je prilikom registrovanja funkcija koje će se aktivirati prilikom pojave događaja naveden i treći parametar `addEventListener()` metode. Za njegovu vrednost je postavljeno `true`. To znači da će događaji biti obrađeni tokom *capturing* faze. Ukoliko se nakon ove male izmene ponovo pokrene prethodni primer i izvrši klik na `div3` element, rezultat izvršavanja će biti sledeći:

```
Hello from div1
Hello from div2
Hello from div3
```

S obzirom na to da se događaji sada obrađuju tokom *capturing* faze, prvo se obrađuje događaj na `div1` elementu, zatim na elementu `div2`, a na kraju na `div3` elementu.

**Zaustavljanje propagiranja događaja**

Web pregledači obezbeđuju i mehanizam za zaustavljanje procesa propagiranja događaja. Naime, objekat događaja poseduje metodu `stopPropagation()`. Pozivanjem ove metode zaustavlja se propagiranje događaja, bez obzira na to da li se oni obrađuju u *capturing* ili *bubbling* fazi:

```
function eventHandler(e) {
    console.log("Hello from " + this.id);
    e.stopPropagation();
}
```

Unutar funkcije za obradu događaja, sada je obavljen poziv metode `stopPropagation()`, nad promenljivom `e`, koja predstavlja objekat događaja. Time se zaustavlja propagiranje događaja, pa će događaj uvek biti aktiviran samo na jednom elementu.

## Rezime

- Događaj je pojam koji opisuje neku pojavu.
- Na webu, događaje može proizvesti web pregledač, ali i korisnici web sajtova.
- Web pregledači obezbeđuju mehanizam za reagovanje na događaje definisanjem programske logike koja će se aktivirati kada do neke pojave dođe.
- Logika koja se aktivira kada dođe do nekog događaja naziva se *event handler*.
- Metoda `addEventListener()` koristi se za registrovanje logike koja će se aktivirati prilikom pojave nekog događaja.
- Metoda `removeEventListener()` omogućava da se prethodno definisana pretplata na neki događaj otkaže.
- Objekat događaja je objekat koji sadrži informacije o samom događaju.
- Metoda `preventDefault()` otkazuje podrazumevano ponašanje događaja.
- JavaScript događaji poseduju osobinu propagiranja, što se odnosi na prenošenje događaja sa dece na roditelje.
- Metoda `stopPropagation()` zaustavlja proces propagiranja događaja.