Klase i JavaScript

Javni metodi

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

`constructor` je opcionalan.

Parametri konstruktora i metoda se definišu na isti način kao kod globalnih funkcija.

Kod definisanja metoda ne koristi se ključna riječ `function`.

Javni metodi

```
class ClassName {
  constructor(params) {
    ...
  }
  methodOne() {
    this.methodTwo();
  }
  methodTwo() {
    ...
  }
}
```

Unutar klase morate referencirati drugemetode koristeći prefiks **`this.`**

Javni metodi

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

Svi metodi su javni (**public**), i **ne možete** zadati privatne metode… (yet).

Javni metodi

```
class ClassName {
  constructor(params) {
    ...
  }
  methodName() {
    ...
  }
  methodName() {
    ...
  }
}
```

Još se razmatra kako da se napravi specifikacija za njih (figuring out the spec). (A prije toga i private fields.)

Javne članice klase (public fields)

```
class ClassName {
  constructor(params) {
    this.fieldName = fieldValue;
    this.fieldName = fieldValue;
  }
  methodName() {
    this.fieldName = fieldValue;
  }
}
```

Definišemo javne članice klase zadavanjme vrijednsoti pomoću **`this.`*`fieldName`*** u konstruktoru… ili u bilo kojoj drugoj funkciji.

(Postoji predlog (draft) kako (pravilno) dodati javne članice klase u ES.)

Javne članice klase (public fields)

```
class ClassName {
  constructor(params) {
    this.someField = someParam;
  }
  methodName() {
    const someValue = this.someField;
  }
}
```

Unutar klase uvijek morate koristiti prefiks **`this.`** Za pristup članicama klase.

Javne članice klase (public fields)

```
class ClassName {
  constructor(params) {
    this.fieldName = fieldValue;
    this.fieldName = fieldValue;
  }
  methodName() {
    this.fieldName = fieldValue;
  }
}
```

Ne mogu se dodati privatne članice... još uvijek.

(Kao i za metode, postoji predlog (add private fields) u ES.)

Kreiranje objekata

Koristi se ključna riječ new:

```
class SomeClass {
  ...
  someMethod() { … }
}

const x = new SomeClass();
const y = new SomeClass();
y.someMethod();
```

Zašto uvodimo klase?

Zašto uvodimo klase?

Zašto koristimo klase u veb-programiranju?

Zašto prosto ne radimo kako se već radi – globalne funkcije i globalne promjenljive?

Zašto uvodimo klase?

A: Milion razloga

- Za male projekte, globalne promjeljive i funkcije su sasvim OK
- Za velike projekte, kod postje nerazumljiv i podložan greškama, ak nije dobro organizovan
- Upotreba klasa i OOP je najpoznatija strategija organizacije koda

Organizacija koda

Well-engineered software is well-organized software:

- Software engineering se bavi
  1. Što mijenajte (What to change)
  2. Gdje mijenjate (Where to change it)
- Lakše je čitati postojeći kod ako je dobro organizovan
  - *"Zašto bih čitao kod?*" Jer morate modifikovati kod i pronalaziti greške

Drugi problemi sa globalnim promj.

Gomila promjenljivih u globalnom opsegu je veoma opasna za vašu aplikaciju

- Lakše se hakuju
  - Mogu se dohvatiti kroz ekstenzije ili konzolu
  - Can override behaviors
- Globalni opseg postaje zagađen
  - Što ako imamo dvije funkcije sa istim imenom? Jedna o njih bude “pregažena” bez upozorenja i bezgreške
- Veoma lako je modifkovati pogrešno stanje promjenljive

Sve ovo se lakše izbjegava ako koristimo klase

Primjer: Present

Kreirajmo klasu `Present` za naš primjer od ranije.

Starter / Finished

Kako kreirati klase

- Kako da odredimo koje nam klase trebaju?
- Kako da odredimo koje metode dodajemo klasi?

Ovo nije Softversko inženjerstvo niti OOP.

Nećemo posvetiti previše pažnje objašnjavanju kako da organizujete vašu aplikaciju u klase.

Opšta strategija

"Component-based" pristup: Koristite klase da bi doadli funkcionalnost HTML elementima ("components")

Svaka komponenta:

- Ima jedan container element / root element
- Obrađuje attaching/removing event listeners
- Može imati reference na child components / child elements

(Sličnu strategiju ima ReactJS, koji uvodi Custom Elements, i neke druge libraries/frameworks/APIs imaju sličan proistup)

Container element

Jedan šablon:

```
<div id="present-container"></div>

const element =
  document.querySelector('#present-container');
const present = new Present(element);
// Immediately renders the present
```

Container element

Sličan šablon:

```
<div id="present-container"></div>

const element =
  document.querySelector('#present-container');
const present = new Present();
// Renders with explicit call
present.renderTo(element);
```

Web: skoro potpuna sloboda

Za razliku od nekih platformi kao što su Android ili iOS, programer ima (skoro) potpunu slobodu kako da organizuje svoj kod.

Razlozi **Za**:

- Mnogo kontrole!

Razlozi **Protiv**:

- Mnogo mnogo mnogo odluka
- Zato postoji mnogo **Framework-a** : web framework je obično gomila software engineer odluka + neki starter kod

Ne zaboravite `this`

```
// Create image and append to container.
const image = document.createElement('img');
image.src = 'https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/gift-icon.png';
image.addEventListener('click', this._openPresent);
```

Ako koristite metod klase kao event handler funkciju kojoj predajemo `addEventListener`, morate ga predati kao "`this.`***functionName***" (finished)

"Private" sa _

Neki programeri koriste sljedeću konvenciju za JavaScript kodiranje: dodaje se simbol underscore kao prefiks imenu privatnog metoda :

```
_openPresent() {
   ...
}
```

Rješenje: Present

CodePen finished

`this` u event handler-ima

```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

    // Create image and append to container.
    const image = document.createElement('img');
    image.src = 'https://s3-us-west-2.amazonaws.com/s.cdpn.io/1083533/gift-icon.png';
    image.addEventListener('click', this._openPresent);
    this.containerElement.append(image);
  }

  _openPresent(event) {
    const image = event.currentTarget;
    image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
    image.removeEventListener('click', this._openPresent);
  }
}
```

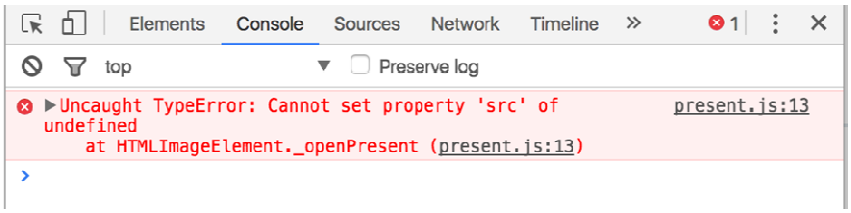
Imamo prpstup slici koja je kreirana u konstruktoru `_openPresent` via `event.currentTarget`.

`this` u event handler-ima

```
  _openPresent(event) {
    this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
    this.image.removeEventListener('click', this._openPresent);
  }
}
```

Q: Što ako napravimo da `image` kako članicu klase i pristupamo mu u `_openPresent` pomoću `this.image` umjesto `event.currentTarget`?

`this` u event handler-ima



Error message!

CodePen / Debug

Što se dešava?

JavaScript `this`

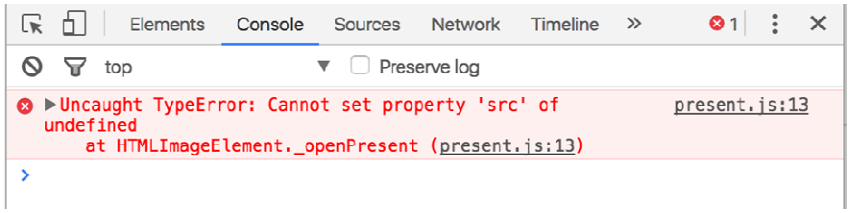
Ključna riječ `this` se **dinamički dodjeljuje**, ili : značenje `this` zavisi od konteksta (mdn list)

- U našem konstruktoru, `this` se odnosi na objekat kalse
- Kada pozivamo event handler, `this` se odnosi na element za koji je zakačen event handler (mdn).

`this` u event handler-ima

```
  _openPresent(event) {
    this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
    this.image.removeEventListener('click', this._openPresent);
  }
}
```

`this` se odnosi na element `<img>` a ne na objekat klase...



...što je razlog zašto dolazi do greške.

Rješenje: `bind`

Da bi realizovali da `this` uvijek ukazuje na trenutni objekat u metodu klase, možete dodati sljedeći red u vaš `konstruktor`:

```
this.methodName = this.methodName.bind(this);
```

```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

    // Bind event listeners.
    this._openPresent = this._openPresent.bind(this);
```

Rješenje: `bind`

Sada se `this` u metodu `_openPresent` odnosi na trenutni objekat (CodePen / Debug):

```
_openPresent(event) {
  this.image.src = 'https://media.giphy.com/media/27ppQUOxe7KlG/giphy.gif';
  this.image.removeEventListener('click', this._openPresent);
}
```

Naravoučenije:

Obavezno dodajte `bind()` u event listenere u vašem konstruktoru!

```
class Present {
  constructor(containerElement) {
    this.containerElement = containerElement;

    // Bind event listeners.
    this._openPresent = this._openPresent.bind(this);
```

Još jednom:

Obavezno dodajte `bind()` u event listenere u vašem konstruktoru!

Komunikacija između klasa

Više klasa

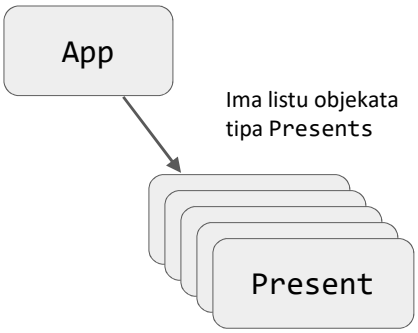
Imamo više poklona (CodePen):

Click a present to open it:

Više klasa

Dovoljne su nam dvije klase :

- App: predstavlja cijelu stranicu
  - `Present`: predstavlja jedan poklon



CodePen

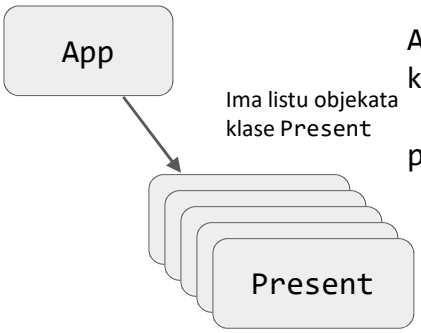
Komunikacija između klasa

Što ako želimo da promijenimo naslov kada se otvori sljedeći pokon? (CodePen)

Enjoy your presents!

Komunikacija između klasa

Komunikacija u smjeru App → `Present` je laka, jer App ima lista objekata klase `Present`.

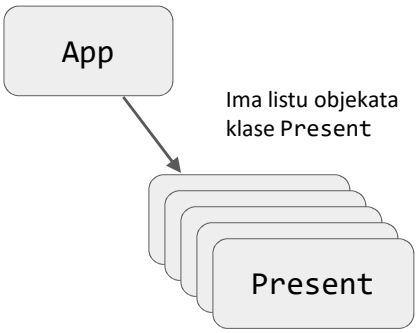


App može pozivati metode iz klase `Present`:

```
present.doWhatever();
```

Komunikacija između klasa

Komunikacija u smjeru `Present` → App nije tako jer klasa `Present` nema referencu na App



Komunikacija između klasa

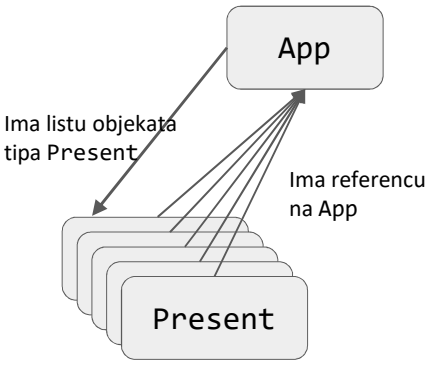
Postoje tri pristupa:

1. Dodati referencu na App u klasu Photo (**Ovo nije dobra praksa**,)
2. Okidanje korisiničkog događaja **OK (ne zaboravite na bind)**
3. Dodati "callback funkciju" u onOpened za klasu Present

Best option (don't forget to bind)

Loš pristup: Presents own App

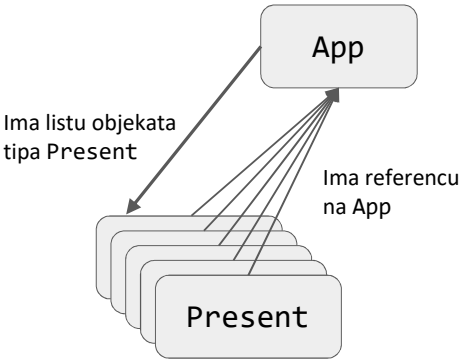
Naivan način jeste da se u konstruktor klase `Present` doda referenca na App : CodePen



(Please don't do this.)

Loš pristup: Presents own App

Ovo je najlakši pristup ali je **veoma loša programerska praksa.**



- Logički besmisleno: `Present` nema App
- Sada `Present` ima "previše pristupa" u App
- Posebno je loše jer JS još uvijek nema privatne fčlanice/ metode

Korisnički događaji (Custom events)

Custom Events

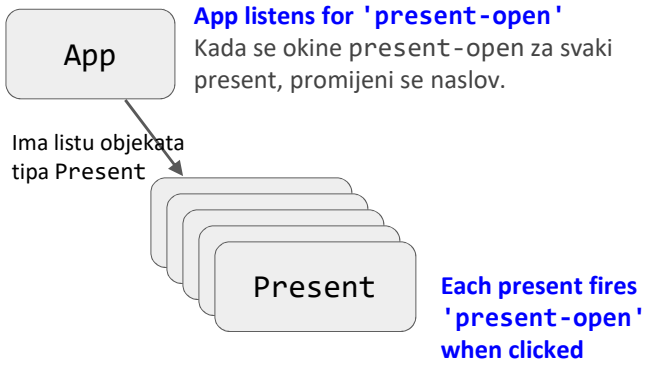
Možete osluškivati (listen to) i raspoređivati (dispatch) korisničkim događajima u cilju komunikacije (mdn):

```
const event = new CustomEvent(
    eventNameString, optionalParameterObject);
element.addEventListener(eventNameString);
element.dispatchEvent(eventNameString);
```

Međutim, CustomEvent se **može osluškivati ili raspoređivati na HTML elementima,** a ne na običnim objektima klase.

Custom Events: Present primjer

Neka App osluškuje događaj `'present-open'` event...



CodePen attempt

`this` u event handler-ima

```
▶Uncaught TypeError: Cannot read property 'length' of undefined    app.js:24
    at HTMLDocument._onPresentOpened (app.js:24)
    at Present._openPresent (present.js:19)
```

Naš prvi pokušaj ponovo ima grešku! (CodePen attempt)

Rješenje: `bind`

Kako smo ranije rekli – ne zaboravite na bind:

```
this.methodName = this.methodName.bind(this);
```

```
this._onPresentOpened = this._onPresentOpened.bind(this);
```

CodePen solution

Funkcije prvog reda

Podsjećanje: `addEventListener`

Već smo koristili **funkcije** kao parametre u `addEventListener`:

```
dragon.addEventListener(
    'pointerdown', onDragStart);

image.addEventListener(
    'click', this._openPresent);
```

First-class functions

JavaScript je jezik koji podržava funkcije prvog reda (**first-class functions**) tj. funkcije su kao promjenljive tipa Function:

- Mogu biti parametri
- Mogu se sačuvati u promjenljivim
- Mogu se definisati bez imena /identifikatora
  - Pod imenom **anonymous function**
  - Pod imenom **lambda function**
  - Pod imenom **function literal value**

Funkcijske promjenljive

Više načina definisanja funkcija:

```
function myFunction(params) {
}

const myFunction = function(params) {
};

const myFunction = (params) => {
};
```

Funkcijske promjenljive

```
function myFunction(params) {
}

const myFunction = function(params) {
};

const myFunction = (params) => {
};
```

Funkcije se pozivaju na način kako ste navikli, bez obzira kako se definisane:

```
myFunction();
```

Jednostavan primjer

```
function greetings(greeterFunction) {
  greeterFunction();
}

const worldGreeting = function() {
  console.log('hello world');
};

const hawaiianGreeting = () => {
  console.log('aloha');
};

greetings(worldGreeting);
greetings(hawaiianGreeting);
```

CodePen

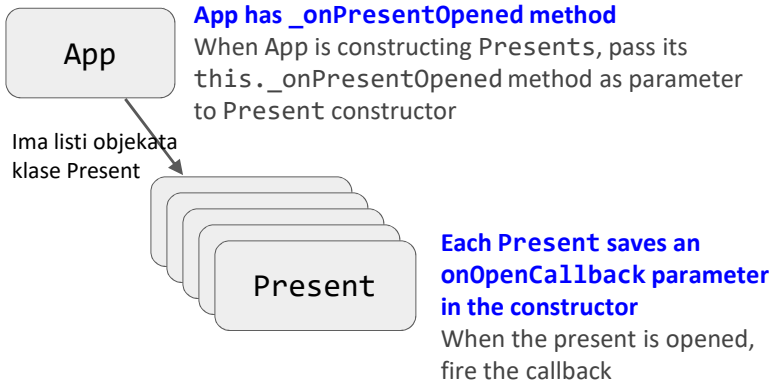
Malo realniji primjer: Callbacks

Još jedan način komunikacije između klasa jeste pomoću tzv. callback functions:

- **Callback**: Funkcija koja je predata kao parametar drugoj funkciji, najčešće kao odgovor na "nešto".

Callback: Present primjer

Pokažimo kako `Present` komunicira sa App preko callback `parameter`: (CodePen attempt)



`this` u event handler-ima

```
▶Uncaught TypeError: Cannot read property 'length' of undefined    app.js:21
    at Present._onPresentOpened [as onOpenCallback] (app.js:21)
    at Present._openPresent (present.js:20)
```

Opet greška...

Rješenje: opet `bind`

Sjetite se što smo govorili o bind: vjerovatno morate dodati sljedeću liniju koda u konstruktor:

```
this.methodName = this.methodName.bind(this);
```

```
this._onPresentOpened = this._onPresentOpened.bind(this);
```

CodePen solution

Object-oriented photo album

Pogledajmo kako izgleda objektno-orientisana verzija albuma za fotografije: CodePen / Debug