Mobil- és webes szoftverek

Osztálykönyvtárak készítése JavaScriptben

BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

Gincsai Gábor
gincsai@aut.bme.hu

window objektum

- Böngészőben futtatott JavaScript kód esetén a `window` objektum segítségével hivatkozhatunk az oldalhoz tartozó ablakra vagy keretre (frame).
- Mivel felső szintű objektumról van szó, nagyon sok általános tulajdonság, metódus és esemény rajta keresztül érhető el, például (nem teljes lista):
  - document, history, location, navigator, screen, status, applicationCache, console, …
  - alert(), confirm(), focus(), open(), close(), print(), scrollTo(), setInterval(), setTimeout(), …
  - onload, onbeforeunload, ondragdrop, onerror, onresize, …

Hol jön létre a változónk

- A var-al létrehozott változók a window objecten jönnek létre.

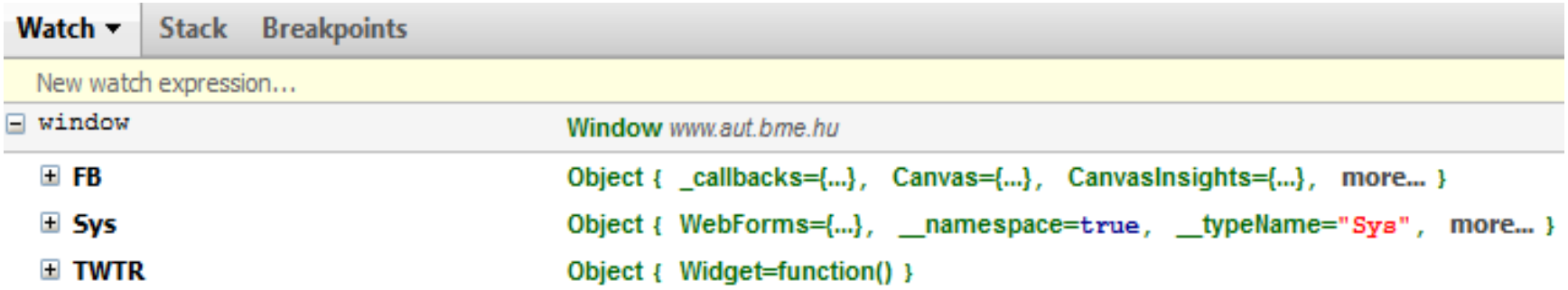
```
var a = 'Szia Világ';
alert(window.a);
```

- A függvények is a window objectre kerülnek

```
function say(name) {
    alert('Szia ' + name)
}
window.say('Világ');
```

Globális névtér szennyezés

- Törekednünk kell arra, hogy elkerüljük a globális névtér beszennyezését (global scope pollution).
- Mindig be kell zárnunk a változóinkat egy objektumba.



Lokális változók

- Más programozási nyelvekhez hasonlóan a függvényeken belül létrehozott változók JavaScriptben is lokális (helyi) változók, azaz csak az adott függvényen belül érhetők el.

```
function f() {
    var x = 8;
    alert('Belül: ' + x);  // Belül: 8
}
f();
alert('Kívül: ' + x);
// ReferenceError: x is not defined
```

function vs block scoping

- A **var** kulcsszóval létrehozott változók function scoping-ot használnak, azaz a teljes függvényben elérhetőek, nem veszi figyelembe a belső blokkokat.
- A **let** kulcsszóval létrehozott változók block scopingot használnak, azaz csak az adott blokkban { } érhetőek el.

Kérdés

- Hol jön létre a változó, ha elhagyjuk a var kulcsszót?
  - Nem jön létre és hibát kapunk
  - A függvényen belül
  - **A window objecten**

Változók láthatósága

- Nincs közvetlen lehetőség a privát és publikus tagok megjelölésére.
- A függvények adta lehetőségeket kell használnunk.
  - JavaScriptben csak és kizárólag a függvény blokkok határozzák meg a láthatóságát, más néven hatókört (functional scoping) a `var` kulcsszóval létrehozott változóknak.

Mit ír ki?

```
function f() {
    if (true) {
        var x = 8;
    }
    alert('Belül: ' + x);
}
f();
```

Mit ír ki?

```
function f() {
    if (true) {
        let x = 8;
    }
    alert('Belül: ' + x);
}
f();
alert('Kívül: ' + x);
```

Lokális vs külső változók

- A változók megtalálása a legmélyebb szintről felfelé történik.
- Ha egy változó helyben nem található, akkor a futtató motor megnézi, hogy a külső függvényben megtalálható-e, és ha ott sem, akkor így halad felfelé egészen a globális névtérig (window).
- Tehát a lokális változók elfedik a külső változókat
  - Shadowing

Mi történik az alábbi esetben?

```
var x = 5;
function f() {
    var x = 8;
    alert('Belül: ' + x);
}
f();
alert('Kívül: ' + x);
```

Kicsit bonyolultabb példa

```
var x = 5;
function kulso() {
    var x = 8;
    function belso() {
        var x = 9;
        alert('Belül: ' + x);
    }
    belso();
    alert('Kívül: ' + x);
}
kulso();
alert('Legkívül: ' + x);
```

Függvények teljes értékű típusok

- A JavaScriptben a függvények teljes értékű típusként viselkednek, ami további izgalmas lehetőségeket biztosít

```
var kulso = function () {
    var x = 8;
    var belso = function () {
        alert(x++);
    };
    return belso;
};
var b = kulso();
b();
```

Closure

- Függvények vannak egymásba ágyazva, de a külső függvény elérhetővé teszi a külvilág számára a belső függvényt.
- A belső függvény megőrzi azt az állapotot, ami a létrehozása pillanatában volt.
- Amikor egy függvény egy belső függvényét láthatóvá teszi a külvilág számára, akkor egy ún. **closure** jön létre, ami nem más, mint az adott belső függvény és a hozzá tartozó állapot együttvéve.

Névtelen függvények

- A korábbi példában felesleges nevet adni a belső függvénynek.

```
var kulso = function () {
    var x = 8;
    return function () {
        x++;
        alert(x);
    }
};
var b = kulso();
b();
```

Self-executing (self-invoking) functions

- Egy másik érdekes lehetőség a JavaScript nyelvben a függvények automatikus futtatása.

```
var fv = function (nev) {
    alert('Szia ' + nev);
};
fv('Világ');
```

- Csak azért adtunk neki nevet, hogy meg tudjuk hívni.

```
(function (nev) {
    alert('Szia ' + nev);
})('világ');
```

Modul tervezési minta

- A modul tervezési minta segítségével a kódunkat egy minden mástól független névtérbe csomagolhatjuk, elkerülve a globális névtér szennyezését.

Modul tervezési minta

```
var N = (function () {
    var priValt = 3;
    var priFv = function () { alert('Privát!'); };
    return {
        pubValt: 5,
        pubFv: function () { alert('Publikus'); }
    };
})();

N.pubFv();
alert(N.pubValt);
```

Modul tervezési minta

- A konstruktor függvény segítségével a modulnak adhatunk bemeneti paramétereket (import), a return kulcsszó után pedig azt határozhatjuk meg, hogy kívülről mi látszik a modulból (export).
- Az import-export lehetőségnek köszönhetően a modulunk kódját akár több fájlba is tehetjük, a publikus tagok el fogják érni egymást:

Akár több fájlra is darabolhatjuk

```
var N = (function (my) {
    my.pubFv = function () { alert('Publikus'); };
    return my;
}(N || {}));
// Másik fájl
var N = (function (my) {
    my.pubValt = 5;
    return my;
}(N || {}));
// Felhasználás
N.pubFv();
alert(N.pubValt);
```

A new operátor

- Az automatikus futtatás elhagyásával és a new operátor használatával osztály-szerű viselkedést is el tudunk érni.

A new használata

```
var MyClass = function () {
    var priValt = 3;
    var priFv = function () { alert('Privát!'); };
    return {
        pubValt: 5,
        pubFv: function () { alert('Publikus'); }
    };
};

var c = new MyClass();
c.pubFv();
alert(c.pubValt);
```

A this kulcsszó

- JavaScriptben is létezik this kulcsszó, és a legtöbb esetben arra az objektumra mutat, amin a függvényt meghívjuk.
- Az alábbi esetben például a függvény a window objektumhoz tartozik, ezért a this a window-ra mutat

```
var F = function () {
    this.A = 1;
};
F();
alert(window.A);
```

A this kulcsszó

- Ha az előző kódot kicsit módosítjuk és használjuk a new operátort, akkor megváltozik a viselkedés

```
var f = new F();
alert(window.A);    // undefined
alert(typeof f);    // object
alert(f.A);         // 1
```

- A new operátor a megadott konstruktor függvény segítségével létrehoz egy új Object-et, és beállítja a this-t erre az objektumra.
  - A new operátor ezzel az objektummal tér vissza (ha a konstruktor függvénynek nincs visszatérési értéke).

A this-el egyszerűsíthetjük a kódunkat

```
var MyClass = function () {
    var priValt = 3;
    var priFv = function () { alert('Privát!'); };

    this.pubValt = 5;
    this.pubFv = function () { alert('Publikus'); }
};

var c = new MyClass();
c.pubFv();
alert(c.pubValt);
```

A this nem mindig az a this

- Előfordul, hogy a this kulcsszó nem arra az objektumra mutat, amin belül használjuk.
  - Pl: egy gomb eseménykezelőjében a this gyakran a gombhoz tartozó DOM elemre mutat.
  - Ilyen esetekben zavaró, hogy az eseménykezelő függvényben a `this` mást jelent, mint az objektum más függvényeiben.
- A hibás működés elkerülése érdekében ezért a this használatát kerülni szoktuk, és az osztály tagjainak elérését más nevű változón (pl. that, self) keresztül végezzük.

A this elmentése

```
var MyClass = function () {
    var self = this;
    var priValt = 3;
    var priFv = function () { self.pubFv(); };
    self.pubValt = 5;
    self.pubFv = function () {
        alert('Privát: ' + priValt);
        alert('Publikus: ' + self.pubValt);
    };
    return self;
};
var c = new MyClass();
c.pubFv();
```

Osztályok létrehozása ES6-ban

- Az ECMAScript6 lehetőséget ad osztályok létrehozására is, a `class` kulcsszó segítségével. Ez a szintaxis egy kicsivel közelebb áll az OO nyelvekhez, mint a korábban megismert.

A class kulcsszó ES6-ban

```
class Point {
  constructor(x, y) {
    this.x = x; this.y = y;
  }
  toString() {
    return '(' + this.x + ', ' + this.y + ')';
  }
}
var p = new Point(25, 8);
console.log(p.toString())          // '(25, 8)'
```

Backtick `

- A fenti példában a toString()-et átláthatóbban is megírhatjuk az alábbi módon.

```
return `(${this.x}, ${this.y})`;
```

- Így nincs szükség a szintring összefűzésekre.
- Ügyeljünk rá, hogy itt nem aposztróffal kell kezdeni a sztringet hanem backtick (`-tel)

Fontos a sorrend

- Bár függvényeknél megtehetjük, hogy korábban hívjuk meg, mint deklaráljuk, mert a deklarálást kiemeli a kódból. Viszont ugyanez az osztályokra már nem igaz.

```
foo();                              // Működik
function foo() { alert("hi"); }


new Foo();
// ReferenceError: can't access lexical
declaration `Foo' before initialization
class Foo {}
```

Származtatás

- Arra is lehetőségünk van, hogy az osztályok származzanak egymásból. Ehhez az extends kulcsszót kell használnunk. Az ősosztály függvényét például a super.toString() a kontruktorát pedig a super() segítségével tudjuk meghívni.
- Ügyeljünk arra, hogy a konstruktornál az előtt kell meghívni a super()-t mielőtt a this-t használjuk!

Példa

```
class ColorPoint extends Point {
  constructor(x, y, color) {
      super(x, y); this.color = color;
  }
  toString() {
      return super.toString() + ' in ' + this.color; }
}
let cp = new ColorPoint(25, 8, 'green');
console.log( cp.toString() );     // '(25, 8) in green'
console.log(cp instanceof ColorPoint);  // true
console.log(cp instanceof Point);       // true
```

Arrow function

- dfs

Arrow function

- Függvényeket tömörebben leírhatjuk vele, így olvashatóbb lesz a kód.

```
nev => console.log(nev)

() => console.log("alma")

(varos, utca) => {
  var cim = varos + utca
  console.log(cim)
}
```

Arrow function példák

```
materials.map(function(material) {
    return material.length;
}); // [8, 6, 7, 9]

materials.map((material) => {
    return material.length;
}); // [8, 6, 7, 9]

materials.map(material =>
    material.length); // [8, 6, 7, 9]
```

Nincs saját this-e

- JavaScriptben minden függvénynek van saját this –e, ami időnként megnehezíti a kódolást
  - Alapértelmezés szerint a this a függvény kontextusára mutat
  - jQuery-s eseménykezelőben a this a DOM objektum, mert átállítja a jQuery.
- Arrow function esetény nincs saját this, tehát nem lehet átállítani, hogy magára mutasson

this példa (hibásan)

```
function Person() {
// Person() constructor defines `this` as an instance of itself.
this.age = 0;


setInterval(function growUp() {
// In non-strict mode, the growUp() function defines `this`
// as the global object, which is different from the `this`
// defined by the Person() constructor.
            this.age++;
    }, 1000);
}
```

this példa (javítva)

```
function Person() {
    var that = this;
    that.age = 0;

    setInterval(function growUp() {
    // The callback refers to the `that` variable of which
    // the value is the expected object.
        that.age++;
    }, 1000);
}
```

Arrow functionben használható a this

```
function Person(){
    this.age = 0;

    setInterval(() => {
        this.age++;
    // |this| properly refers to the person object
    }, 1000);
}
```

Arrow function törzse

- Nem kell kiírni a returnt

```
> var func = x => x * x;
```

- Ha több soros a kód, akkor kell return

```
> var func = (x, y) => { return x + y; };
```

- Objektum nem adható vissza simán

```
> var func = () => { foo: 1 };
> var func = () => { foo: function() {} };
```

- Zárójelezni kell

```
> var func = () => ({foo: 1});
```

Visszahívás szerver oldalról

Long poll, WebSockets

Visszahívás szerver oldalról

- Hagyományos pollozás
- Long-polling
- Web Sockets
- ASP.NET SignalR

Hagyományos pollozás

- Adott időközönként Ajax-szal lekérdezzük a szervertől a változásokat.
  - **`setInterval`**
- Előnyök
  - Egyszerű megvalósítás.
  - Működik minden böngészőben.
- Hátrányok
  - Nem azonnal jelennek meg az adatok → gyakori lekérdezés.
  - Sok felesleges kommunikáció.
- Ha a kérés tovább tart, mint a pollozási idő, egymásra csúsznak a kérések.
  - Nem fix pollozási idő, az előző kérés feldolgozása után újabb timer felhúzása.
  - **`setInterval`** helyett **`setTimeout`** függvény
  - Véd az egymásra futás ellen, de nem ismerjük a frissítési gyakoriságot.

Long-polling

- A kliens sokáig nyitva tartja a kapcsolatot.
  - Ha van változás, a szerver vissza tudja küldeni.
    - A szerver folyamatosan is írhatja a választ (streaming), de mivel HTTP válaszról van szó, azt egyes tűzfalak és proxy szerverek bufferelik → nem célszerű.
  - Ha nincs változás, timeout bontja a kapcsolatot.
- Előnyök
  - Egyszerű megvalósítás.
  - Működik minden böngészőben.
  - A változásról azonnal értesítést kap a kliens.
- Hátrányok
  - Bonyolultabb szerver oldali implementáció.
  - Jobban terheli a szerver oldalt, mert foglalni kell a kapcsolathoz tartozó erőforrásokat.

Web Sockets

- Full-duplex, kétirányú TCP kommunikáció egyetlen socketen keresztül.
  - HTTP-től független TCP kommunikáció.
    - **ws://** és **wss://** (biztonságos) URI scheme.
    - A kliens egy **Connection:Upgrade** fejléccel kéri a protokoll váltást (handshake).
    - Nem csak web → bármilyen alkalmazásban használható.
  - 80-as és 443-as TCP portokon megy
    - nincs tűzfal probléma.
  - Nem bájt folyam, hanem üzenet folyam.

Handshake

Kliens oldalról

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://example.com
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
```

A szerver válasza

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: chat
```

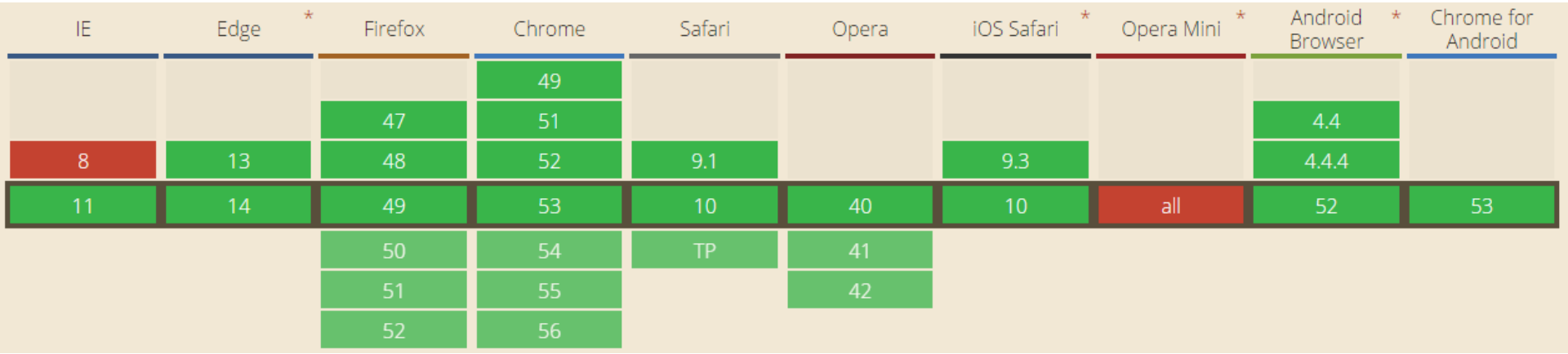
Web Sockets előnyök / hátrányok

- Előnyök
  - Full-duplex, kétirányú kommunikációra tervezve.
  - Nem HTTP
    - nincsenek a HTTP fejlécekkel járó overheadek
    - gyorsabb
- Hátrány
  - A szabvány sok változáson ment keresztül, az RFC-nek megfelelőt csak az újabb böngészők támogatják.
  - Nem minden webszerver támogatja (pl. IIS 8 kell).

Web Socket használata

```
var socket = new WebSocket('ws://localhost:8080/');
socket.onopen = function () {
      console.log('Connected!');
};
socket.onmessage = function (event) {
      console.log('Received data: ' + event.data);
      socket.close();
};
socket.onclose = function () {
      console.log('Lost connection!');
};
socket.onerror = function () {
      console.log('Error!');
};
socket.send('hello, world!');
```

Web sockets támogatottsága



Forrás: http://caniuse.com/websockets

Minta alkalmazás: http://demos.kaazing.com/echo/

Események leírása:
https://webplatform.github.io/docs/apis/websocket/

ASP.NET SignalR

- Nyílt forráskódú osztálykönyvtár ASP.NET alkalmazások számára, valósidejű funkciók megvalósítására.
  - Teljes Microsoft terméktámogatás.
  - Nem csak web, hanem desktop, mobil és Windows Store alkalmazások számára.
- Server-to-client-RPC: JavaScript függvény meghívása .NET kódból.
- WebSocketet használ, ha a böngésző támogatja, ha nem akkor van fallback
  - Server sent events (EventSource IE nem támogatja).
  - Forever frame (csak IE-ben).
  - Ajax long-polling
- Támogatás:
  - IE 8+ (WebSockethez IE 10 kell), egyéb böngészőknél utolsó -1 verzió.
  - Szerveren .NET 4 (WebSockethez .NET 4.5 kell).

http://www.asp.net/signalr http://signalr.net/