

Objekty a triedy v JavaScripte

24.02.2022

Marek Nagy

Objekty

- objektový prístup
 - každý prvok je objekt

Špeciálny typ v JavaScripte.

```
let a = new Object ();  
let b = {};
```

Vytvorenie objektu dvoma spôsobmi.

- „metódy“ a vlastnosti

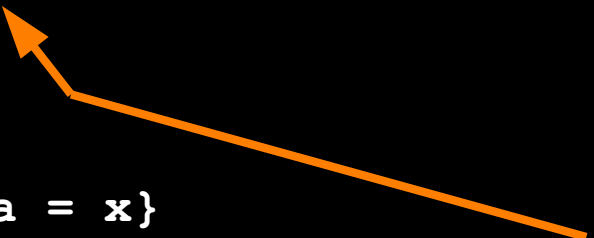
```
let b = {};  
  
b.meno = 'Marek';  
b.vek = 10;  
b.pridaj = function (x) {  
  this.vek += x;  
}  
  
b.pridaj (4);    // b.vek == 14
```

Syntaktická premenná **this** „obsahuje“ aktuálny objekt.

Objekty

- priame vytvorenie vlastností aj metód


```
let a = {x:10, y:5, 's4 s9': 10};  
  
let obj = {  
  a: 3,  
  b: 6,  
  f: function (x) {this.a = x}  
};
```



Meno vlastnosti môže byť aj krkolomné. Vtedy sa uvedie ako reťazec v úvodzovkách alebo apostrofoch.

- zmena a doplnenie

```
obj.a = 50;  
obj[' 3r '] = 40;
```



Možno využiť aj zátvorkovú konvenciu. Hlavne pri krkolomných názvoch vlastností.

- zrušenie vlastnosti

```
delete obj.x;           // obj.x === undefined
```

Prehľadávanie vlastností

in

```
obj = {farba:3};  
  
if ('farba' in obj) console.log (obj.farba);  
if (obj.farba !== undefined) console.log (obj.farba);
```

for in

```
obj = {a:3, c:4, g:5};  
  
for (let x in obj) { console.log (obj[x]) }
```

Správanie sa vlastností

- štandardné
 - t.j. možno čítať, zapisovať, je uvažovaná vo for-in
- zmenené
 - uviesť pri explicitnom konfigurovaní vlastnosti

```
const obj = {};  
  
Object.defineProperty (obj, 'x', {  
  value: 42,  
  writable: false,           // zakázať zmenu hodnoty  
  enumerable: false,        // či bude uvažovaná vo for in  
  configurable: true        // či možno zmazať t.j. aj predefinovať  
});  
  
obj.x = 77;  // hodí error  
  
console.log (obj.x);    // 42
```

Vlastnosť už mohla existovať. Vtedy sa ako keby zmaže a vytvorí nanovo.

Zákaz pridávania vlastností

- do objektu nemožno pridávať ďalšie vlastnosti

```
const obj = {};  
  
Object.preventExtensions (obj);  
  
try {  
    Object.defineProperty (obj, 'x', {value: 42}); // hodí error  
}  
catch (e) {  
    console.log (e);  
}  
  
if (Object.isExtensible (obj)) obj.x = 5; // false
```

Zamrazenie vlastností objektu

- vlastnosti nemožno pridať, zrušiť, modifikovať

```
const obj = {x: 42};  
Object.freeze (obj);  
obj.x = 33;    // hodí error v 'strict mode'  
console.log (obj.x);    // 42  
  
if (!Object.isFrozen (obj)) obj.x = 33;    // x == 42
```


Interné objekty pre základné typy

- interná reprezentácia (t.j. kamufláž na objekty)
 - rýchlejšie, efektívnejšie, prehľadnejšie

```
let a = 1.2,  
    b = false,  
    c = 'Ahoj',  
    d = [1,2,3];  
  
// let e = {};
```

- možno použiť aj všeobecné objekty typov
 - pomalšie

```
let a = new Number (1.2),  
    b = new Boolean (false),  
    c = new String ('Ahoj'),  
    d = new Array (1,2,3);  
  
// let e = new Object ();
```

Nové typy Set a Map

Map mapovanie **klúča** na **hodnotu**

Set reprezentuje množinu **prvkov**

– efektívna implementácia

```
a = new Map ()  
  
a.set ('Jurko', 8)    // mapuj  
a.get ('Jurko')      // == 8
```

```
a = new Set ()  
  
a.add ('jablko')     // pridaj  
a.add ('jablko')     // nepridá
```

```
a.size                // počet klúčov/prvkov  
a.clear ()            // vyprázdni obsah  
a.delete (x)          // vyhoď klúč/prvok  
a.has (x)              // testuje, či je tam klúč/prvok
```

```
a.keys ()             // zoznam klúčov/prvkov  
a.values ()           // zoznam hodnôt/prvkov
```


Funkciové typy objektov

- funkcia je tiež „špeciálny“ objekt

```
function Moja (x, y) {  
}  
  
Moja.dlzka = 5;  
Moja.sucet = function (a,b) {return a+b};  
  
Moja.sucet (3, 4);    // 7
```

Konštruktor objektu

- **každá funkcia** je konštruktor objektu
 - okrem šípkových funkcií
 - vytvorí sa nový objekt cez **new Object()** a spustí sa naň konštruktor

```
function Obdlznik (a, b) {  
}  
  
let a = new Obdlznik (3, 5);
```

Hodnota **this** vo funkcii

- kľúčové slovo **odkazujúce** sa na aktuálny objekt

```
function Obdlznik (a, b) {  
    this.a = a;  
    this.b = b;  
}
```

```
let obj = new Obdlznik (3,5);  
console.log (obj.a, obj.b);    // 3, 5
```

```
Obdlznik (1,3);    // this je undefined t.j. vznikne chyba  
                  // keď nie je 'strict mode' odkazuje na globálny objekt
```

- šípkové funkcie **nemajú** vlastné **this** !!!

this mimo funkcií

- odkazuje na globálny kontext

```
this.a = 5;

function Moja () {
  return this.a;    // pri 'strict mode' this je undefined
}

Moja();

// browser
window.a == 5      // this == window

// node.js
global.a == 5      // this == global
```

this a callback funkcie

- napr. metódy poľa every, map, forEach, ...

```
obj.spracuj = function () {  
  // this == obj  
  this.sum = 0;  
  
  [1,2,3].forEach (function (x) {  
    this.sum += x;    // this == [1,2,3]  
  });  
  
  [5,6,7].forEach (function (x) {  
    this.sum += x;    // this == obj  
  }, this);  
  
  [7,8,9].forEach (x => {  
    this.sum += x;    // this == obj  
  });  
  
}  
  
obj.spracuj ();
```


Nastavenie **this** pre volanú funkciu

call

- treba vymenovať argumenty volanej funkcie

apply

- argumenty pre funkciu sú v poli

```
function Sucet (c, d) {  
  return this.a + c + d;  
}  
  
let obj = {a: 3};  
  
Sucet.call (obj, 5, 7);    // 15  
  
Sucet.apply (obj, [1,2]);  // 6
```

Časovače

- jedno zavolanie po uplynutí času

```
let timeoutID = setTimeout ( () => {  
  console.log ('Hurá');  
}, 2000);  
  
clearTimeout (timeoutID); // Zruší časovač
```

- pravidelné spúšťanie funkcie

```
let intervalID = setInterval ( () => {  
  console.log ('Ha');  
}, 500);  
  
// Zruší časovač  
clearInterval (intervalID);
```

Prednastavenie **this**

bind

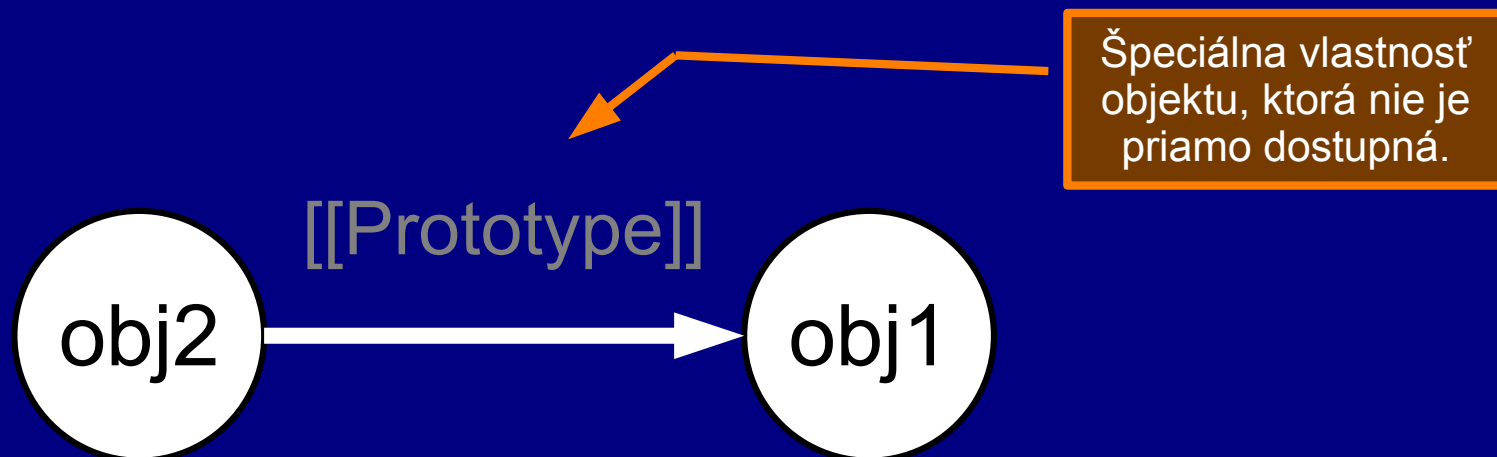
- obalí funkciu
- vytvorí novú funkciu s prednastaveným **this**

```
function Moja () { return this.a }
s = {a:5};
let obalMoja = Moja.bind (s);
x = obalMoja();    // x == 5

obj = {sucet:90};
obj.tikac = function () {console.log (this.sucet)};
obj.nastav = function () {
  // cez bind
  setInterval (this.tikac.bind (this), 1000);

  // cez pomocnú premennú
  let self = this; setInterval (function (){self.tikac()}, 1000);
  // cez šípkovú funkciu
  setInterval (() => {this.tikac()}, 1000);
}
```


Reťazenie objektov

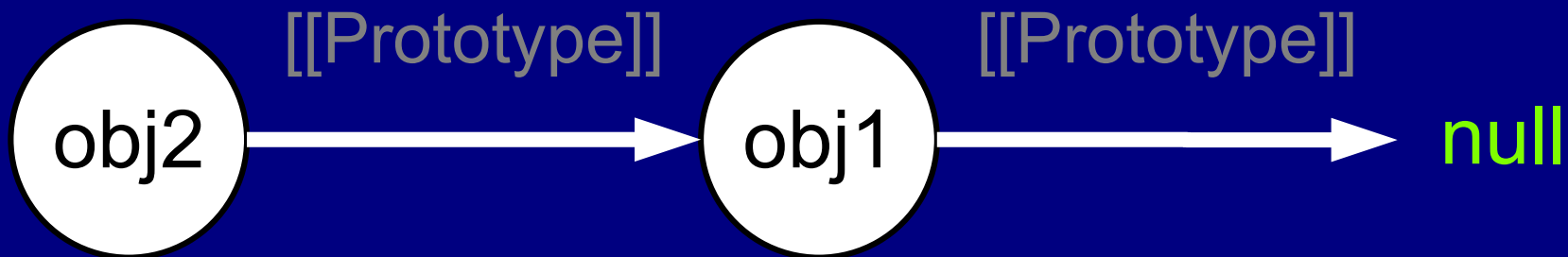


```
obj1 = {a: 9};  
  
obj2 = Object.create (obj1);  
obj2.x = 3;  
  
console.log (obj2.x, obj2.a); // 3, 9
```

Vlastnosť **a** neexistuje v objekte **obj2**, ale pomocou reťazenia sa nájde v **obj1**.

Reťaz objektov

- čas hľadania vlastností závisí od dĺžky reťaze
- nájde sa vždy najbližšia vlastnosť s daným menom
- zatieňovanie** metód a vlastností otca



```
obj1 = {f: function () {return 9}};  
obj2 = Object.create (obj1);  
obj2.f = function () {return 3};  
console.log (obj2.f()); // 3
```

Prístup ku [[Prototype]] hodnote

- čítanie

```
const obj1 = {};  
const obj2 = Object.create (obj1);  
  
console.log (Object.getPrototypeOf (obj2) === obj1); // true
```

- zmena

- príliš časovo náročná operácia !!!

```
const obj1 = {};  
const obj2 = {};  
  
Object.setPrototypeOf (obj2, obj1);
```

Prehľadávanie vlastností objektov

- **for in** a **in** prehľadáva aj „zdedené“ vlastnosti !!!
- ku „vlastným“ vlastnostiam treba testovanie

```
o = {a:1};
```

```
obj = Object.create (o);
```

```
obj.x = 4;
```

```
obj.y = 6;
```

```
for (let w in obj) if (obj.hasOwnProperty(w)) obj[w]++;
```

```
// a == 1, x == 5, y == 7
```

```
let s = Object.keys (obj); // s == ['x', 'y']
```

Testuje, či je to vlastná
nezdedená vlastnosť
objektu.

Vráti pole názvov všetkých
vlastných vlastností objektu,
ktoré sú enumerable.

Reťazenie cez konštruktory

- každá funkcia je konštruktor
- má vlastnosť **prototype**, ktorej hodnota bude nastavená novovytvorenému objektu ako otec
- tento spôsob sa už nepoužíva, lebo sú už triedy

```
function Obdlznik (a, b) {  
    this.a = a;  
    this.b = b;  
}  
Obdlznik.prototype = {x:9, y:20};  
  
let obj = new Obdlznik (3,5);  
console.log (obj.x, obj.y);    // 9, 20
```


Triedy

- prehľadnejšia syntax vytvárania objektov

```
class Obdlznik {  
  constructor (a, b) {  
    this.a = a;  
    this.b = b;  
  }  
}
```

```
let obj = new Obdlznik (3,5);  
console.log (obj.a, obj.b); // 3,5
```

Vytvorí sa prázdny objekt, na ktorý sa spustí metóda **constructor**. Názov je vyhradený pre tento účel.

- defaultne je konštruktor „prázdna“ funkcia

```
class Obdlznik {  
  constructor () {}  
}
```

Metódy (inštancie) triedy

- zjednodušenie syntaxe

```
class Obdlznik {  
  constructor (a, b) {  
    this.a = a;  
    this.b = b;  
  }  
  
  obvod () {  
    return 2*(this.a + this.b);  
  }  
}  
  
let obj = new Obdlznik (3,5);  
console.log (obj.obvod ()); // 16
```

Statické metódy triedy

- sú to metódy samotnej triedy a nie objektov

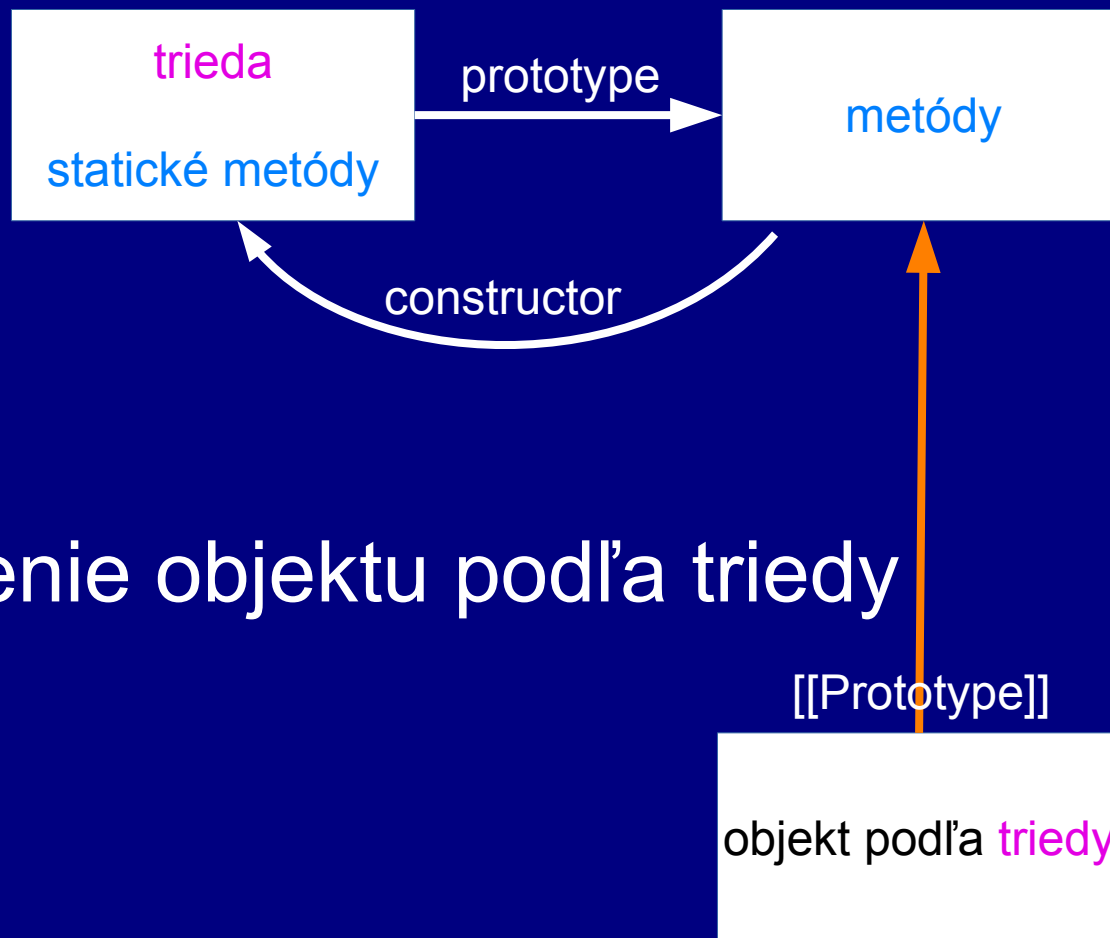
```
class Obdlznik {  
  constructor (a, b) {  
    this.a = a;  
    this.b = b;  
  }  
  
  obvod () {  
    return 2 * Obdlznik.sucet (this.a, this.b);  
  }  
  
  static sucet (a, b) {return a + b}  
}
```

```
let obj = new Obdlznik (3,5);  
console.log (obj.obvod ()); // 16
```

```
Obdlznik.sucet (10,15); // 25
```

Trieda ako špeciálny typ objektu

- definícia triedy je vlastne dvojica objektov



- vytvorenie objektu podľa triedy

Triedy cez výrazy

- bez mena

```
let Obdlznik = class {  
  constructor (a, b) {  
    this.a = a;  
    this.b = b;  
  }  
}
```

- s menom

```
let Obdlznik = class Obdlznik {  
  constructor (a, b) {  
    this.a = a;  
    this.b = b;  
  }  
}
```

Public / Private fields / methods

Verejné položky možno zapísať aj takto priamo v triede namiesto v konštruktore.

```
class Obdlznik {  
  a = 3;  
  b = 4;  
  static N = 30;  
  
  // privátne  
  #d = 4;  
  #moja () {}  
  
  obvod () {return this.#d * (this.a + this.b)}  
  
  static #M = 3;  
  static #Dalej () {return ++this.#M}  
}
```

Privátne položky a metódy možno využiť iba v rámci definície tejto triedy.

```
let obj = new Obdlznik ();  
console.log (obj.a, obj.b);    // 3, 4  
console.log (this.#d)         // hodí chybu  
  
console.log ( Obdlznik.N );    // 30
```


setter a getter metódy

- ako keby klasická vlastnosť objektu
 - za priradením sa odohrá postup
 - zisťovanie hodnoty je tiež postup

```
class Obdlznik {  
    #active = false;  
  
    set active (v) {this.#active = v}  
    get active () {return this.#active}  
}
```

```
obj = new Obdlznik ()
```

```
obj.active = true;
```

```
if (obj.active) console.log ('Obdlžnik je aktívny');
```

Reťazenie cez triedy

- dedenie...

```
class Stvorec extends Obdlznik {  
  constructor (a) {  
    super (a, a);    // volanie konštruktora otca  
  }  
}
```

```
let obj = new Stvorec (6);  
console.log (obj.obvod ());    // 24
```

- ak konštruktor nie je, realizuje sa implicitný:

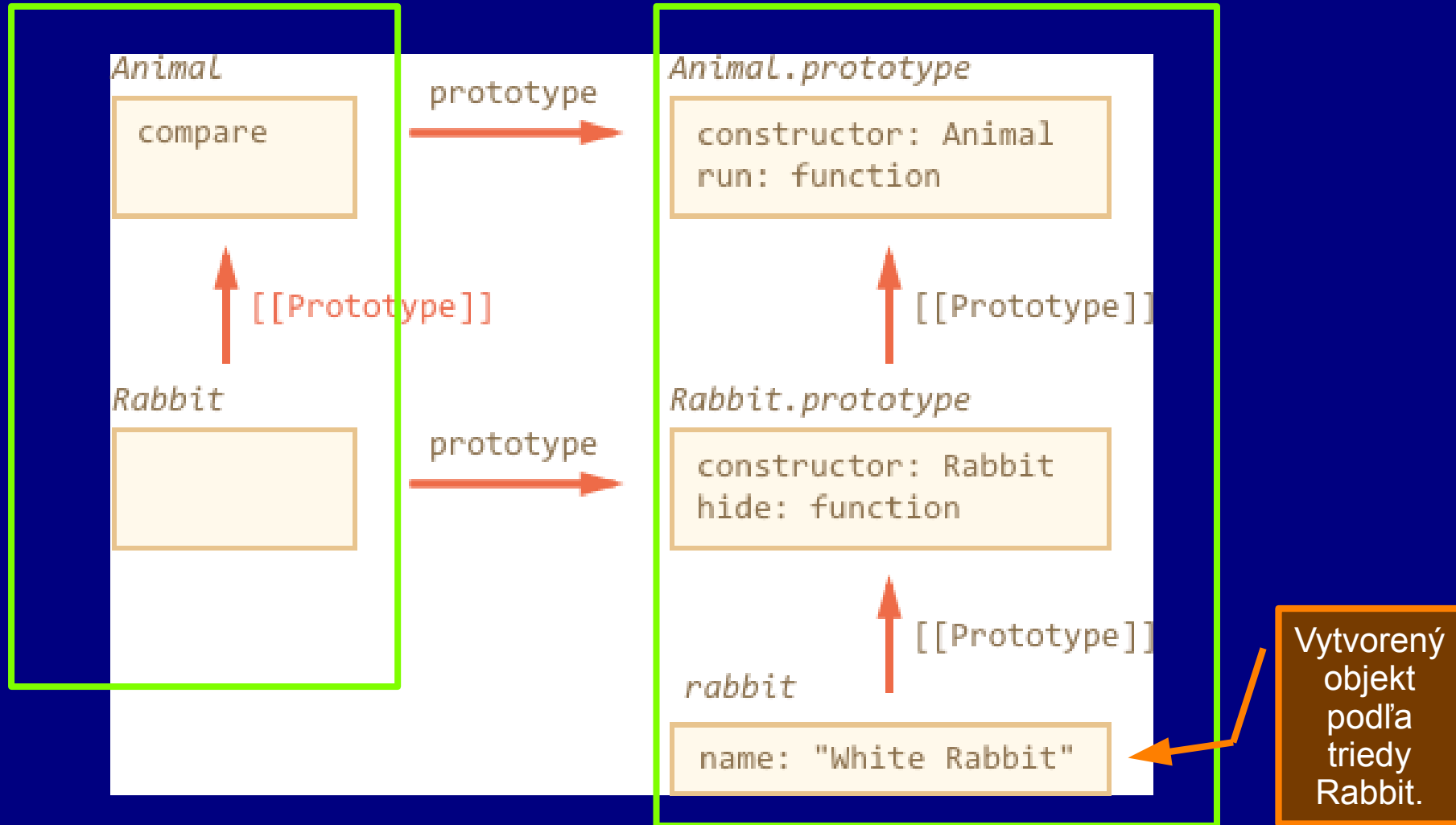
```
class Stvorec extends Obdlznik {  
  constructor (...args) {  
    super (...args);  
  }  
}
```

super

- prepojenie na otca „pri dedení“
- v **konštrukto**re
 - **super()** → volá konštruktor otca, ktorý nastaví hodnotu **this**
 - vždy musí byť zavolaný pred prvým použitím hodnoty **this** !
- v **metóde**
 - **super.metoda()** → volá metódu otcovského objektu
 - nezmení sa hodnota **this** !
 - metóda si pamätá príslušnosť k objektu triedy, aby vedela zavolať otcovské metódy v reťazi ...
- v **statickej metóde**
 - **super.statMetoda()** → volá statickú metódu konšuktora otca
 - aj konštruktory vytvoria reťaz a statické metódy sa tiež dedia

```
Obdlznik.sucet (10,15); // 25  
Stvorec.sucet (10,15); // 25
```

Prepojenie pri reťazení class-ov



Ret'az tried

Ret'az objektov

Operátor instanceof

- zisťovanie, či je objekt inštanciou danej triedy
 - t.j. pohľadá, či má objekt v reťazi prototype objekt z danej triedy

```
class Obdlznik {  
}  
class Stvorec extends Obdlznik {  
}  
  
let obj = new Stvorec ();  
if (obj instanceof Svorec) a = 20; // true  
if (obj instanceof Obdlznik) a = 10; // true
```

- vhodné aj pre testovanie typu Array

```
let a = [1,2,3];  
typeof a; //=== 'object'  
a instanceof Array; //=== true nefunguje skrze viac iframes  
Array.isArray (a); //=== true vhodné, ak sa používajú iframes
```

Dedenie od dvoch tried „naraz“

- do reťaze objektov je zaradený objekt jednej triedy a potom druhej

```
ToolApp = base => class extends base {  
  constructor (hidden, ...rest) {  
    super (...rest);  
    this.hidden = hidden;  
  }  
  show () {console.log ('Ukáž')};  
  hide () {console.log ('Skry')};  
}  
  
class AppObdlznik extends ToolApp (Obdlznik) {  
  constructor (...rest) {  
    super (...rest);  
  }  
}  
  
let a = new AppObdlznik (true, 3, 5);  
a.obvod ();      // 16  
a.show ();       // 'Ukáž'
```


... syntax

...rest

- zlúči elementy do objektu (od ES2018)

...spread

- expanduje elementy z objektu (od ES2018)

```
let o = {x:10, y:30};  
let q = {...o, r:9};
```


Deštrukturalizácia objektu

```
let o = {a: 42, b: 100};
let {a, b} = o;           // a=42, b=100

// Iné názvy
let {a: x, b: y} = o;    // x=42, y=100

// Bez var, let, const treba zátvorky!
let a, b;
({a, b} = o);           // a=42, b=100

// Defaultné hodnoty
let {a, c=1} = o;        // a=42, c=1

// V parametroch funkcie
function Draw ( {coords = {x:0,y:0}, radius = 25} = {} ) {}
Draw ( {cords: {x:10,y:30}, radius:90} )
Draw ()

// Zvyšok
let {a, ...rest} = o;    // a=42, rest={b:100}

// Aj viacúrovňové, aj kombinované s poľom
// Aj pri premennej cyklu for of
```

Cyklus **for of**

- prechádza po **iterovateľných** objektoch
 - napr. reťazec, pole, DOM kolekcie, generátory, ...
 - pri poliach máme aj `forEach()`
 - rozdiel je oproti **for in** (, ktorý ide cez enumerable vlastnosti objektu)

```
deti = ['Janko', 'Marienka', 'Jurko'];  
  
for (let osoba of deti) console.log (osoba);  
  
mn = new Set ()  
for (let p of mn) console.log (p)  
  
mp = new Map ()  
for (let [k, v] of mp) console.log (`${k} => ${v}`)
```


Ďakujem za pozornosť