OBJEKTI, PROTOTIP I NASLJEĐIVANJE

Mrežne usluge i programiranje
8. dio

Prototipno nasljeđivanje

- U programiranju često želimo nešto postojeće proširiti sa dodatnim funkcionalnostima
- Primjerice, za objekt `osoba`, koji ima svoja svojstva i metode, želimo kreirati objekt `student` (ili sl) kao malo izmijenjenu varijanta
- Odnosno želi se koristiti već postojeći objekt (`osoba`), a da se ne kopiraju i ponovno implementiraju njegove metode, već da se povrh postojećeg objekta izgradi novi

Prototipno nasljeđivanje

- Mehanizam nasljeđivanja se u JavaScriptu može izvesti na nekoliko načina
- Može se temeljiti na prototipu, za razliku od drugih OOP jezika gdje se temelji samo na klasama, a gdje klasa predstavlja određeni nacrt
- Kod klasa, instance nasljeđuju od klasa te se klase ne mogu koristiti na isti način kao i instance, odnosno primjerice ne može se pozvati metoda instance preko definicije klase, već se stvara instanca te klase i onda se poziva pripadna metoda, stvarajući hijerarhiju klasa
- Kod prototipa, instance nasljeđuju od drugih instanci, odnosno može se reći da se objekti povezuju sa drugim objektima, pritom ne stvarajući hijerarhije kao kod klasa, a lanac prototipa bi trebao biti što plići.

Prototipno nasljeđivanje

- JS je zapravo nejasan za iskusne programere klasičnih OOP jezika
- Ovaj tip oop-a je zapravo puno snažniji od klasičnog, iako dosta zbunjujući, a smatra se pseudo klasičnim načinom oop
- JS je dinamičan i ne pruža implementaciju klasa
- Klase se uvode u ES6 no zapravo je samo riječ o izrazu koji u jednu ruku olakšava razumijevanje funkcioniranja nasljeđivanja, ali JS je zapravo i dalje prototipno orijentirani jezik

Prototipno nasljeđivanje

- JS objekti imaju skriveno svojstvo `[[Prototype]]` koje ukazuje (referencira) na `null` ili neki drugi objekt, a taj objekt nazivamo `prototype`
- `Objekt -> prototype objekt [[Prototype]]`
- `Null` nema `prototype` i zapravo je zadnja poveznica u lancu prototipa
- Kada želimo pročitati svojstvo objekta ili pozvati metodu objekta, a ono nedostaje, JS ga automatski traži u `prototype-u`
- To nazivamo prototipno nasljeđivanje

Prototipno nasljeđivanje

- Ukoliko se kroz cijeli prototipni lanac ne pronađe tražena vrijednost, vraća se `undefined`
- Postoje više načina kako postaviti svojstvo `[[Prototype]]`
- `__proto__` je starija verzija getter/settera za `[[Prototype]]` i nije ekvivalento `[[Prototype]]`
- Danas je zamijenjeno sa `Object.getPrototypeOf/Object.setPrototypeOf`
- Vrijednost od `__proto__` može biti samo `null` ili objekt

```
let zivotinja = {
  jede: true
};
let ptica = {
  leti: true
};

ptica.__proto__ = zivotinja;  /*postavlja se da je zivotinja
prototype za ptica*/

// kod ptice se nalaze oba svojstva
console.log( ptica.leti ); // true
console.log( ptica.jede ); /* true; prvo se traži unutar ptica, pa
ako ne postoji prati se [[Prototype]] referenca prema zivotinja i
traži svojstvo */
```

`zivotinja` je prototip od `ptica` ili `ptica` prototipno nasljeđuje od `zivotinja`. Svojstva od `zivotinja` postaju dostupna unutar `ptica` (nasljeđena) svojstva.


```
let zivotinja = {
  jede: true,
  info(){console.log("životinja");}
};
let ptica = {
  leti: true,
  __proto__:zivotinja;
};

ptica.info(); /*preko prototipnog lanca dohvaća se metoda
info() iz zivotinja*/
```

Prototipno nasljeđivanje

- Prototip se koristi samo za radnju čitanja
- Radnje pisanja/brisanja djeluju izravno sa objektom
- Pristupna svojstva su iznimka, budući da je dodjeljivanje upravljano `setter` funkcijom

```
let zivotinja = {
  jede: true,
  info(){}
};
let ptica = {
  leti: true,
  __proto__:zivotinja;
};
ptica.info  = function() {console.log("Ptica");};

ptica.info(); /*dohvaća se metoda info() iz ptica, budući da metodu
nalazi unutar objekta, pa nije potrebno pretraživati prototype*/
```

```
let osoba = {
  ime: "Mate",
  prezime: "Matić",

  set punoImePrezime(vrijednost) {
    [this.ime, this.prezime] = vrijednost.split(" ");
  },
  get punoImePrezime() {
    return `${this.ime} ${this.prezime}`;
  }
};

let student = {
  __proto__: osoba,
  studira: true
};
console.log(student.punoImePrezime); // Mate Matić
student.punoImePrezime = "Ante Antić"; // setter
```

Prototipno nasljeđivanje

- Na vrijednost od `this` ne utječe prototip
- Bez obzira gdje se svojstvo ili metoda nalazi, `this` je uvijek objekt naznačen prije točke
- Stoga kod `student.punoImePrezime = "Ante Antić";` vrijedi da je `this = student`
- Kada objekti izvode naslijeđene metode, oni mijenjaju samo svoja stanja, a ne stanje glavnog izvornog objekta
- Metode se dijele, ali ne i stanje objekta

Prototipno nasljeđivanje

- Petlja `for…in` prolazi i kroz naslijeđena svojstva
- Metoda `Object.keys` vraća samo vlastita svojstva (enumerable)
- Metoda `hasOwnProperty()` razdvaja svojstva objekta od svojstava koja pripadaju prototipu (`prototype`)
- Odnosno `obj.hasOwnProperty(svojstvo)` vraća `true` ako svojstvo pripada `obj`, a ne pripadnom prototipu

```
let osoba = {
  ime: "Mate",
  prezime: "Matić",
  set punoImePrezime(vrijednost) {
    [this.ime, this.prezime] = vrijednost.split(" ");
  },
  get punoImePrezime() {
    return `${this.ime} ${this.prezime}`;
  }
};
let student = {
  __proto__: osoba,
  studira: true
};
```

```
for(let sv in student) {
  let vlastita = student.hasOwnProperty(sv);

  if (vlastita) {
    document.write(`Vlastito: ${sv}`);
    // Vlastito sv: studira
  } else {
    document.write(`Nasljeđeno: ${sv}`);
    // Nasljeđeno sv: ime, prezime, punoImePrezime
  }
}
```

Prototipno nasljeđivanje

- Kao što je već rečeno, uporaba `__proto__` je zastarjela te se preporučuje koristiti metode:
  - `Object.create(proto, deskriptori)` stvara prazan objekt sa danim proto kao `[[Prototype]]`, a deskriptori svojstva su opcionalni kojim se dodaju dodatna svojstva novom objektu
  - `Object.getPrototypeOf(objekt)`
  - `Object.setPrototypeOf(objekt, proto)`

```
let zivotinja = {
  jede: true
};

// stvara novi objekt gdje je objekt zivotinja prototip
let ptica = Object.create(zivotinja, {leti:{value:true}});
console.log(ptica.jede); // true
console.log(ptica.leti); // true
console.log(Object.getPrototypeOf(ptica) === zivotinja); // true
Object.setPrototypeOf(ptica, {}); // promjena prototipa
```

Prototipno nasljeđivanje

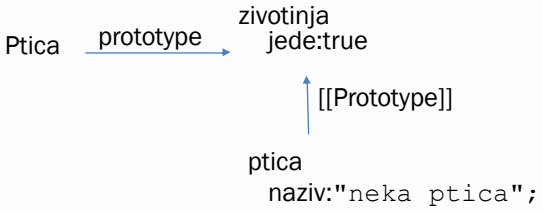
- Metoda `Object.create` se isto može koristiti za kloniranje umjesto kopiranja svojstva sa `for…in`
- Na ovaj način stvara se potpuni pravi klon objekt, sa svim svojstvima (enumerable/nonenumerable), podatkovna svojstva, getter/setter i sa odgovarajućim `[[Prototype]]`

```
let clone = Object.create(Object.getPrototypeOf(objekt),
                Object.getOwnPropertyDescriptors(objekt));
```

Prototype

- Treći način stvaranja objekata koristi konstruktor funkciju sa ključnom riječju `new` (`new F`)
- Ako je `F.prototype` objekt tada ga operator `new` koristi za postavljanje `[[Protoype]]` za novi objekt
- U ovom slučaju `F.prototype` je zapravo obično svojstvo koje se naziva `prototype`

"prototype" predstavlja standardno svojstvo, a [[Prototype]] predstavlja nasljedstvo ptica od zivotinja



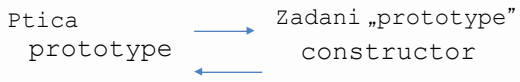

```
let zivotinja = {
  jede: true
};

function Ptica(naziv) {
  this.naziv = naziv;
}

Ptica.prototype = zivotinja;

let ptica = new Ptica("neka ptica"); //  ptica.__proto__ == zivotinja

console.log(ptica.jede); // true
```

Naredba `Ptica.prototype = zivotinja;` znači da kad se stvara `new Ptica`, pridjeljuje se `[[Prototype]]` životinja

Prototype

- `F.prototype` svojstvo se samo koristi kada se poziva `new F`, dodjeljuje `[[Prototype]]` novog objekta
- Ako se kasnije svojstvo `F.prototype` promjeni na drugi objekt, tada novi objekti koji se kreiraju sa novim `F` (`new F`) će imati drugi objekt kao `[[Prototype]]`, a već postojeći objekt će zadržati stare
- Svojstvo `prototype` ima poseban efekt kada se postavi sa konstruktor funkcijom, a pozove sa `new`, dok kod reguarnih objekata nema

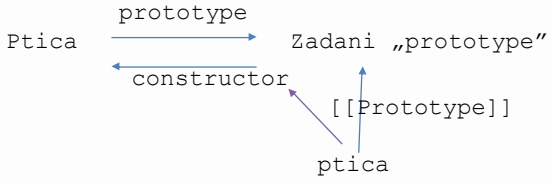
Prototype



- Svaka funkcija uvijek ima svojstvo „prototype”
- Zadani „prototype” je objekt sa jedinim svojstvom `constructor` koji pokazuje na samu funkciju

```
F.prototype = { constructor: F }
```

```
function Ptica() {}

/* zadani prototype
Ptica.prototype = { constructor: Ptica };
*/
console.log(Ptica.prototype.constructor == Ptica ); // true
```




```
function Ptica() {}

/* zadani prototype
Ptica.prototype = { constructor: Ptica };
*/
console.log(Ptica.prototype.constructor == Ptica ); // true


let ptica = new Ptica(); // nasljeđuje od {constructor: Ptica}

console.log(ptica.constructor == Ptica); // true (iz prototype)
```

Svojstvo `constructor` se može koristiti za stvaranje novog objekta u situacijama kada primjerice imamo objekt, ali ne znamo koji se konstruktor koristio

```
function Ptica(naziv) {
  this.naziv = naziv;
  console.log(naziv);
}

let ptica = new Ptica("neka ptica");

let ptica2 = new ptica.constructor("druga ptica");
```

JavaScript ne osigurava automatski pravu „constructor" vrijednost, odnosno iako postoji u zadanom `prototype` za funkcije, sva daljnja događanja ovisi o nama. Odnosno, ako zamijenimo zadani `prototype` kao cjelinu, tada neće biti unutar „constructor". Stoga, kako bi se sačuvao ispravni `constructor` bolje je zadanom `prototype` dodavati i brisati svojstva od opcije da se prebriše kao cjelina

```
function Ptica() {}
Ptica.prototype = {leti:true;}};
let ptica = new Ptica();
console.log(ptica.constructor === Ptica); //false

Odnosno

Ptica.prototype.leti = true
// zadani Ptica.prototype.constructor je sačuvan

ILI

Ptica.prototype = {
  leti: true,
  constructor: Ptica
};
```

Prototype

- Svojstvo `prototype` objekta `function` (uz `length`, `constructor`)
- Svojstvima funkcije se pristupa kao i svojstvima bilo kojeg drugog objekta
- Svojstvo `prototype` se stvara odmah pri stvaranju funkcije (početna vrijednost je prazan objekt) odnosno kao da je napisano `fja.prototype = {}`
- Prazan objekt je moguće popuniti svojstvima i metodama, koje neće utjecati na samu funkciju `fja()`, već će djelovati kada se `fja()` koristi kao konstruktor

Prototype

- Osnovna ideja konstruktor funkcije
  - Unutar funkcije koja je pozvana sa `new` ostvaruje se pristup vrijednosti `this`, koji sadrži objekt kojeg vraća konstruktor
  - Dodavanjem metoda i svojstava `this` objektu se proširuje funkcionalnost stvorenom objektu

```
function Osoba(ime, prezime, godrod)
{
    this.ime=ime;
    this.prezime = prezime;
    this.godrod = godrod;
    this.prikaziPodatke=function(){
        return "<br>Ime i prezime: " + this.ime + " " + this.prezime +
               ".";
    }
}
/*dodaje se novo svojstvo i metoda objektu unutar svojstva prototype*/
Osoba.prototype.radniStatus="nezaposlen";*/
Osoba.prototype.dohvatiRadniSt = function() {
        return 'Radni status: ' + this.radniStatus ;
}
```

Drugi način dodavanja novih svojstava i metoda je potpuno redefiniranje prototype objekta, sa objektom po izboru

```
Osoba.prototype = {
   radniStatus = "nezaposlen",
   dohvatiRadniSt: function() {
        return 'Radni status: ' + this.radniStatus ;
   }
};
```

Prototype

- Nova svojstva i metode unutar `prototype` su dostupne odmah pri stvaranju prvog objekta korištenjem konstruktora
- Ako se stvara objekt `osoba1` korištenjem konstuktora `Osoba()` ostvaruje se pristup svim definiranim metodama i svojstvima
- `var osoba1 = new Osoba('Mate', 'Matić',1982);`
- `osoba1.ime; //Mate`
- `osoba1.radniStatus; //nezaposlen`

Prototype

- Objekti se prenose prema referenci, pa stoga objekt `prototype` nije kopiran sa svakom novom instancom objekta
- Prema tome, `prototype` se može mijenjati bilo kada, a objekti će naslijediti promjene (čak i one prije izmjene)
- `Osoba.prototype.dohvati = function(what){return this[what];}`
- `osoba1.get('ime'); //Mate`
- `osoba1.get('radniStatus'); //nezaposlen`

Prototype

- Funkcija `F1` ima svojstvo `prototype`
- Referenca na `prototype` objekt se kopira u interno svojstvo `[[Prototype]]` nove instance
- Kod `let obj1 = new F1();` JS će nakon stvaranja objekta u memoriji i prije izvođenja funkcija `F1()` sa `this`, postaviti vrijednost `obj1.[[Prototype]] = F1.prototype`
- Kada se pristupa svojstvima instance JS prvo provjerava da li ono postoji unutar tog objekta, te ako ne postoji pretražuje dalje u `[[Prototype]]`

Prototype

- Prema tome, sve što se definira unutar `prototype` se učinkovito dijeli između svih instanci
- Vrijednosti svojstava unutar `prototype` se mogu mijenjati i pritom utjecati na sve instance

Ugrađeni objekti i prototype

- Izvorni prototipovi se mogu mijenjati (npr. `String.prototype`) što utječe na sve stringove
- Prototipovi imaju globalni doseg, stoga nije preporučljivo mijenjati izvorne prototipove
- Njihova promjena je dozvoljena samo u slučajevima kada se želi zamijeniti metoda koja postoji u JS specifikaciji ali još nije podržana od strane JS interpretera (*polyfilling*)

Ugrađeni objekti i prototype

- Neke metode izvornih prototipova se često posuđuju (metode objekta `Array`) iako se ne koriste primjerice nad nizovima
- To je moguće jer te metode gledaju indekse i duljinu, a ne koji tip podatka se stvarno nalazi unutar tog niza
- Posuđivanje metoda dozvoljava kombiniranje funkcionalnosti iz različitih objekata

Nasljeđivanje

- Ukoliko je definirano `obj1 = new F1(); obj2 = new F1();`
- Tada `obj1.uciniNesto()` zapravo je jednako
- `Object.getPrototypeOf(obj1).uciniNesto()` ili
- `F1.prototype.uciniNesto()` ili
- `Object.getPrototypeOf(obj2).uciniNesto()`

```
/*stvaranje objekta obj1 korištenjem funkcije f1 sa svojim svojstvima
a i b*/
let f1 = function () {
   this.a = 1;
   this.b = 2;
}
let obj1 = new f1(); // {a: 1, b: 2}

//dodaju se svojstva u prototipu funkcije f1
 f1.prototype.b = 3;
 f1.prototype.c = 4;
```

```
/*lanac prototipa:
 obj1.[[Prototype]] ima svojstva b i c
 obj1.[[Prototype]].[[Prototype]] je na kraju i zapravo
                                  je Object.prototype
 obj1.[[Prototype]].[[Prototype]].[[Prototype]]  je null
 null nema [[Prototype]]

 {a: 1, b: 2} ---> {b: 3, c: 4}
              ---> Object.prototype
              ---> null
 */
```

```
console.log(obj1.a);

//da li objekt obj1 ima vlastito svojstvo a -> da a=1


console.log(obj1.b);

/*da li objekt obj1 ima vlastito svojstvo b  -> da b=2
prototip također ima svojstvo b,  no budući da se prototip gleda 2. po redu, dolaze do
property shadowing

Vlastita svojstva objekta imaju prednost nad svojstvima prototype-a sa istim imenom*/
```

```
console.log(obj1.c);

/* da li objekt obj1 ima vlastito svojstvo c ->  nema ->
   pretraga se nastavlja u prototipu obj1.[[Prototype]]  i nalazi c = 4   */

console.log(obj1.d); // undefined

/* da li objekt obj1 ima vlastito svojstvo d -> nema ->
pretraga se nastavlja u prototipu obj1.[[Prototype]]  -> nema ->
obj1.[[Prototype]].[[Prototype]]  je null
pa se pretraga zaustavlja i vraća undefined*/
```

Nasljeđivanje

- U slučajevima kada treba paziti na ponovnu upotrebljivost kôda, a i kod bolje izvedbe, bolje je takvu vrstu kôda staviti kao dio prototipa pa je prema tome kod nasljeđivanja, bolja opcija nasljeđivati izravno od prototipa
- Brža je izvedba budući da svi objekti dijele jedan prototip objekt
- Unutar prototipa se uglavnom nalazi kôd koji je uvijek ponovno upotrebljiv

```
function Literatura(){}

Literatura.prototype.vrsta = 'literatura';
Literatura.prototype.toString =
        function() {return this.vrsta;};

function Tisak(){}
Tisak.prototype = new Literatura(); //nasljeđivanje
Tisak.prototype = Literatura.prototype;
Tisak.prototype.constructor = Tisak;
Tisak.prototype.vrsta = 'tisak';
```

```
function Knjiga(autor, naziv, izdanje, godIzdanja){
    this.autor = autor;
    this.naziv = naziv;
    this.izdanje = izdanje;
    this.godIzdanja = godIzdanja;
}


Knjiga.prototype = new Tisak();//nasljeđivanje
Knjiga.prototype = Tisak.prototype;
Knjiga.prototype.constructor = Knjiga;
Knjiga.prototype.vrsta = 'knjiga';
Knjiga.prototype.dohvatiPodatke =
        function(){
                return this.autor + ", " + this.naziv + ", "
                        + this.izdanje + ", "
                        + this.godIzdanja; };
```

```
var knjiga1 =
        new Knjiga("Ivo Ivić", "Autobiografija", "2", "2017");
document.write(knjiga1.dohvatiPodatke());
//Ivo Ivić, Autobiografija, 2, 2017
document.write(knjiga1.hasOwnProperty('godIzdanja'));
//true
document.write(knjiga1.hasOwnProperty('vrsta'));
//false

document.write(Literatura.prototype.isPrototypeOf(knjiga1));
//true
document.write(Tisak.prototype.isPrototypeOf(knjiga1));
//true
document.write((knjiga1 instanceof Literatura));
//true
document.write((knjiga1 instanceof Tisak ));
//true
```

Nasljeđivanje – kopiranje prototipa

- Ukoliko dijete promjeni vrijednost prototipa, tada se mijenjaju vrijednosti i kod roditelja

```
var x = new Literatura();
document.write(x.vrsta );
```

- Kako bi se takva situacija izbjegla, potrebno je pozvati novi privremeni konstruktor kako bi se razbio lanac prototipa koji je usmjeren na jedan objekt i gdje roditelj ima vrijednosti djece
- Prema tome samo svojstva i metode prototipa bi trebali biti naslijeđeni, a ne vlastita svojstva objekta, budući da su svojstva objekta obično specifična za određene objekte te koji prema tome se ne bi trebali ponovno upotrebljavati

Nasljeđivanje

- Privremeni objekt omogućuje stvaranje objekata bez svojstava objekata, ali koji nasljeđuje prototip roditelja
- Umjesto `Tisak.prototype = Literatura.prototype;`
- definira se sljedeće

```
var Privr = function(){};
Privr.prototype = Literatura.prototype;
Tisak.prototype = new Privr();
```

- Prema tome sljedeći kôd vraća vrijednost „literatura“ a ne „knjiga“

```
var x = new Literatura();
document.write(x.vrsta);
```

Nasljeđivanje

- Korištenjem ovog pristupa, prototipni lanac ostaje netaknut, odnosno djeca ne mogu promijeniti svojstva roditelja
- U isto vrijeme, ovaj pristup podržava ideju da samo svojstva i metode `prototype-a` budu naslijeđene, a ne vlastita svojstva koja su obično prespecifična da bi se ponovno koristila

Nasljeđivanje

- Kad se već govori o upotrebljivosti kôda, tada bi bilo dobro funkcionalnost nasljeđivanja pridijeliti određenoj funkciji koja bi se pozivala u svrhu nasljeđivanja
- Prema tome, prethodno navedeni kôd se može pridijeliti funkciji `naslijedi()`
- U ovom slučaju opet se koristi konstruktori i mehanizam lanca prototipa
- Nasljeđuje samo svojstva prototipa, dok vlastita svojstva kreirana unutar konstruktora nisu naslijeđena
- Svojstvo `uber` ukazuje na roditeljev objekt prototipa

```
function naslijedi(Dijete, Roditelj) {
  var Privr = function(){};
  Privr.prototype = Roditelj.prototype;
  Dijete.prototype = new Privr();
  Dijete.prototype.constructor = Dijete;
  Dijete.uber = Parent.prototype;
}
//poziv
naslijedi(Tisak, Literatura);
```

Nasljeđivanje

- Nasljeđivanje se može provesti i korištenjem metoda `apply` i `call`
- Omogućuju da objekt koristi konstruktor nekog drugog objekta, odnosno dijete poziva konstruktor roditelja te povezuje `this` djeteta sa `this` roditelja
- Posuđivanje konstruktora drugog objekta
- Nasljeđuju se samo vlastita svojstva
- Jednostavan način rješavanja situacije kada dijete nasljeđuje svojstvo koje je zapravo objekt (pa sa time i prenesen prema referenci)

```
function Literatura(sifra) {
        this.sifra = sifra;
}
Literatura.prototype.vrsta = 'literatura';
Literatura.prototype.toString = function(){return this.vrsta;};

function Tisak() {
        Literatura.apply(this, arguments);
}
Tisak.prototype = new Literatura();
Tisak.prototype.vrsta = 'tisak';
var tis1 = new Tisak("ZI-123");
document.write(tis1.sifra);
//ZI-123
document.write(tis1.vrsta);
//tisak
```

```
var FjaA = function() {
    this.naziv = "a";
}
FjaA.prototype.print = function() {
    console.log(this.naziv);
}
var a = new FjaA();
a.print(); //a
var naslijediOd = function (dijete, roditelj) {
    dijete.prototype = Object.create(roditelj.prototype);
};
```

```
var FjaB = function() {
    this.naziv = "b";
    this.vrsta = "dijete";
}
naslijediOd(FjaB, FjaA);
var b = new FjaB();
//b.print(); //b
FjaB.prototype.print = function() {
    FjaA.prototype.print.call(this);
    console.log(this.vrsta);
}
b.print(); //b dijete
```

```
var FjaC = function () {
    this.naziv = "c";
    this.vrsta = "unuče";
}
naslijediOd(FjaC, FjaB);
FjaC.prototype.print = function () {
    FjaB.prototype.print.call(this);
    console.log("prototipno nasljeđivanje!");
}
var c = new FjaC();
c.print(); //c unuče prototipno nasljeđivanje!
```

Nasljeđivanje – kopiranje svojstava

- U JS-u koncept nasljeđivanja izgleda slično nasljeđivanju klasa kod klasičnih OO jezika, no u JavaScriptu se u biti nasljeđuju objekti a ne klase
- Ukoliko se objekt definira kao literal objekta, tada je dovoljno kopirati svojstva postojećeg objekta u novi objekt (koristi objekte a ne `prototype`, kopira po vrijednosti)
- Problem nastaje ukoliko je neki od postojećih svojstava objekt, tada bi trebalo razriješiti i kopiranje i njegovih svojstava
- Prema tome funkcija kopiranja svojstava bi trebala ispitivati takve situacije
- Bilo kakve promjene vrijednosti u objektu ne utječu na izvorni objekt
- Ovaj način ne koristi prototip

```
function kopiranje(roditelj, dijete) {
    var dijete = dijete || {};
    for (var i in roditelj) {
        if (typeof roditelj [i] === 'object') {
            dijete [i] =
                (roditelj [i].constructor === Array) ? [] : {};
            kopiranje(roditelj [i], dijete [i]);
        } else {
                dijete [i] = roditelj [i];
        }
    }
    return dijete;
}
/*petljom se prolazi kroz sva svojstva i kopiraju se jedno po jedno, a
ukoliko je svojstvo objekt onda se opet poziva ista fja*/
```

```
var roditelj = {
brojevi: [1, 2, 3],
slova: ['a', 'b', 'c'],
obj: {
x: 1
}
};


var dijete = kopiranje(roditelj);
dijete.brojevi.push(4,5,6);
console.log(dijete.obj.x);   // 1
dijete.obj.x = 5;
console.log(dijete.brojevi); //[1,2,3,4,5,6]
console.log(dijete.obj.x);  //5
console.log(roditelj.brojevi);//[1,2,3]
console.log(roditelj.obj.x);  //1
```

Nasljeđivanje

- Prototipno nasljeđivanje je nasljeđivanje gdje se roditeljski objekt postavlja kao prototip objekta djeteta
- Ta vrsta nasljeđivanja se može izvesti i korištenjem metode `object()` koja prihvaća i vraća objekt, odnosno vraća novi objekt kojemu je za vrijednost prototipa postavljen objekt roditelja
- Koristi se lanac prototipa
- Radi sa objektima

```
function object(o) {
    function F() {}
    F.prototype = o;
    return new F();
}
var knjiga = object(tisak);
```

Nasljeđivanje

- Kod nasljeđivanja istodobno se mogu koristiti metode prototipnog nasljeđivanja i kopiranje svojstava
- JavaScript dozvoljava da jedan objekt može istovremeno naslijediti svojstva i metode od više objekata
- Iako nema izričiti način kako se izvodi višestruko nasljeđivanje, može se izvesti prema prethodno navedenim postupcima, odnosno kopiranjem svojstava koja se mogu proširiti sa neograničenim brojem objekata
- Kopiranje svojstava roditeljskih objekata se izvodi redoslijedom kako je navedeno u pozivu

```
function naslijediVise() {
        var naslij = {}, objekti, i=0;
        var brojObjekata = arguments.length;
        for (i=0; i < brojObjekata;i++)
        {
                objekti= arguments[i];
                for (var sv in objekti)
                {
                        naslij[sv] = objekti[sv];
                }
        }
        return naslij;
}
```

```
var literatura = {
        vrsta: 'literatura',
        toString: function() {return this.vrsta;}
};
var tisak = {
        vrsta: 'tisak'
};
var knjiga = naslijediVise(literatura, tisak, {
        autor: "Ivo Ivić",
        naziv: "Autobiografija",
        izdanje: "2",
        godIzdanja: "2017",
        dohvatiPodatke:
                function(){
                        return this.autor + ", " + this.naziv + ", "
                        + this.izdanje + ", " + this.godIzdanja; }
});

document.write(knjiga.dohvatiPodatke());
//Ivo Ivić, Autobiografija, 2, 2017
document.write(knjiga.vrsta); //tisak
```

Nasljeđivanje

- Metoda `Object.create()` omogućava stvaranje objekata, odnosno odabiranje prototip objekta za stvaranje novog objekta bez korištenja konstruktor funkcije i ključne riječi new
- Metoda `Object.assign()` omogućuje da joj se prosljeđuje ciljni objekt i neograničen broj izvornih objekata koji su odvojeni zarezom te tako kopira svojstva iz svih izvora u ciljni objekt u kojem prioritet uvijek ima zadnji navedeni objekt

```
let tisak = {
        vrsta: 'tisak'
};

let knjiga = Object.assign(Object.create(tisak), {
        autor: "Ivo Ivić",
        naziv: "Autobiografija",
        izdanje: "2",
        godIzdanja: "2017",
        dohvatiPodatke: function()
                {return this.autor + ", " + this.naziv + ", " +
                        this.izdanje + ", " + this.godIzdanja; }

});
```

ES6 klase

- JS ne podržava pravi pojam klasa koji se pojavljuje kod standardnih OOP jezika, već ga simulira korištenjem prototipa
- Iako ES6 uvodi jednostavniju sintaksu koja podsjeća na OO jezike, ipak se u pozadini i dalje izvodi prototipno nasljeđivanje
- ES6 jednostavnija sintaksa se ostvaruje definiranjem konstruktor funkcija, korištenjem klasa i `constructor` te izostavljanjem ključne riječju `function` i zareza između članova klase
- ES6 nasljeđivanje se izvodi korištenjem ključne riječi `extends` (izvodi nasljeđivanje) i `super` (poziva konstruktor roditelja)

```
//ES5 sintaksa
function Osoba(imePrez) {
    this.imePrez = imePrez;
}
Osoba.prototype. prikaziInfo = function () {
    return 'Ime i prezime: ' + this.imePrez;
};

//ES6 sintaksa
class Osoba {
    constructor(imePrez) {
        this.imePrez = imePrez;
    }
    prikaziInfo() {
        return 'Ime i prezime: ' + this.imePrez;
    }
}
```

```
//ES5 sintaksa
function Osoba(imePrez) {
    this.imePrez = imePrez;
}
Osoba.prototype.prikaziInfo = function () {
    return 'Ime i prezime: '+this.imePrez;
};

function Student(imePrez, studij) {
    Osoba.call(this, imePrez); // super(imePrez)
    this.studij = studij;
}

Student.prototype = Object.create(Osoba.prototype);
Student.prototype.constructor = Student;
Student.prototype.prikaziInfo = function () {
    return Osoba.prototype.prikaziInfo.call(this)
    // super.prikaziInfo ()
    + ' (' + this.studij + ')';
};
```

```
//ES6 sintaksa
class Osoba {
    constructor(imePrez) {
        this.imePrez = imePrez;
    }
    prikaziInfo() {
        return 'Ime i prezime: '+this.imePrez;
    }
}
class Student extends Osoba {
    constructor(imePrez, studij) {
        super(imePrez); //poziva konstruktor od Osoba
        this.studij = studij;
    }
    prikaziInfo() {
        return super.prikaziInfo() + ' (' + this.studij + ')';
    }
}
```

Nasljeđivanje

- Prototipno nasljeđivanje predstavlja jedno od složenijih područja JavaScripta, čija dobra efikasnost upravo i dolazi iz objektne strukture i nasljeđivanja
- Koristi se kod korištenja svojstava i metoda ugrađenih JS objekata, web API i drugo
- Nije preporučljivo imati puno razina nasljeđivanja
- Ukoliko je potrebno koristiti objekt samo kao kolekciju podataka odnosno koristiti eventualno jednu instancu objekta, tada je dovoljno kreirati objekt kao literal, a ne uz pomoć složenijih mehanizama
- U kontekstu funkcionalnog programiranja, ne preporučuje se koristiti pojam nasljeđivanja koji se ostvaruje objektima („klasama“), već se preporučuje korištenje kompozicije, kao bolji pristup funkcionalnog programiranja