**Objekti**

**00:23:39**

Jedinica: 26 od 30

**Rezime**

* Objektno orijentisano programiranje omogućava apstrahovanje modela zasnovanih na pojmovima iz realnog sveta.
* Klasa je šablon za kreiranje objekata.
* Kod prototipskog programiranja, pojam klasa ne postoji, zato što objekti nasleđuju objekte.
* Gotovo svi tipovi podataka, izuzev primitivnih tipova string, number, boolean, null i undefined, u JavaScriptu su objekti.
* Objekti su skupovi različitih vrednosti koji opisuju jedan isti pojam.
* Objekti mogu da sadrže svojstva i metode.
* JavaScript omogućava kreiranje objekata uz pomoć specijalne funkcije, koja se naziva objektni konstruktor.
* Prototipovi su osnova nasleđivanja u jeziku JavaScript, i omogućavaju da jedan objekat dobije osobine drugog.
* JavaScript poznaje skup ugrađenih objekata, koji su na raspolaganju programeru, a obavljaju različite funkcije.

Više puta u dosadašnjem toku kursa pomenut je pojam objekta. JavaScript je objektno orijentisan jezik, koji se, za razliku od nekih drugih jezika zasnovanih na klasama, zasniva na prototipovima. Lekcija pred vama biće posvećena osnovama objektnog modela jezika JavaScript.

**Objektno orijentisano programiranje**

Objektno orijentisano programiranje omogućava apstrahovanje modela zasnovanih na pojmovima iz realnog sveta. Tako su objektno orijentisane aplikacije, za razliku od tradicionalnih, proceduralnih aplikacija, zapravo kolekcije objekata koji međusobno sarađuju. Umesto skupa funkcija ili procedura, koje su ništa drugo do jednostavna lista naredbi, objektno orijentisano programiranje omogućava objektu da preuzme centralnu figuru kao nezavisna celina koja može da prima poruke, obrađuje podatke i emituje vrednosti. Na taj način objektno orijentisano programiranje obezbeđuje značajnu fleksibilnost prilikom razvoja softvera, što olakšava održavanje i generalno smanjuje troškove.

Osnovni pojmovi na kojima se zasniva objektno orijentisano programiranje su:

* Modularnost
* Polimorfizam
* enkapsulacija

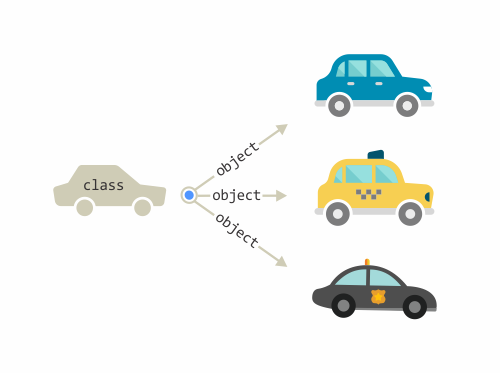
Objekti omogućavaju razdvajanje funkcionalnosti aplikacije na više celina, koje se nazivaju moduli. Otuda objektno orijentisano programiranje omogućava postizanje modularnosti.

Objekat predstavlja zaokruženu celinu, zato što u potpunosti opisuje jedan pojam iz realnog sveta. Svi članovi jednog objekta tiču se isključivo takvog objekta. Tako je objekat idealan primer enkapsulacije, odnosno učaurenja.

Na kraju, objekti omogućavaju da se jednom kreirana logika iskoristi kao osnova za neke druge objekte, kada se govori o polimorfizmu, ili višestrukom značenju jednog istog pojma.

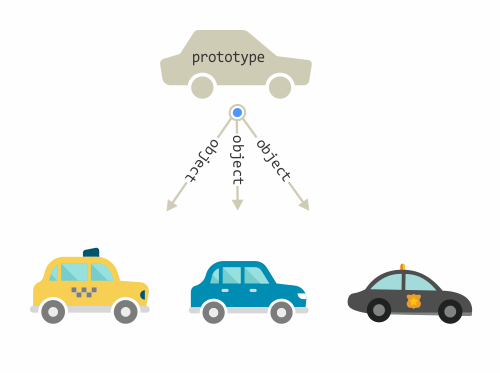
**Prototipsko i klasno objektno programiranje**

Veliki broj najpopularnijih jezika današnjice podržava objektno orijentisano programiranje. Ipak, tradicionalni pristup objektnom programiranju jeste postojanje dva osnovna entiteta takvog modela. Reč je o klasama i objektima. Klase definišu karakteristike objekta, a objekti predstavljaju instance klase. Drugim rečima, klasa je šablon za kreiranje objekata. Odnos između klasa i objekata je prikazan na slici 26.1.



*Slika 26.1 – Objekti i klase*

Osnovni objektni model JavaScript jezika funkcioniše nešto drugačije od opisanog i zasniva se na prototipovima. Kod prototipskog programiranja pojam klasa ne postoji, zato što objekti nasleđuju objekte. Objekat koji se nasleđuje u takvoj hijerarhiji naziva se **prototip**.



*Slika 26.2 – Prototip i objekti*

Ipak, od [ECMAScript 6](https://www.link-elearning.com/linkdl/opisPojma.php?id=144353) standarda, JavaScript poznaje i pojam klasa, pa tako pored prototipskog modela podržava i klasni objektni model. Ipak, implementiranje podrške za ovakav objektni model je još u toku, tako da je podržan samo od strane browsera poslednjih verzija.

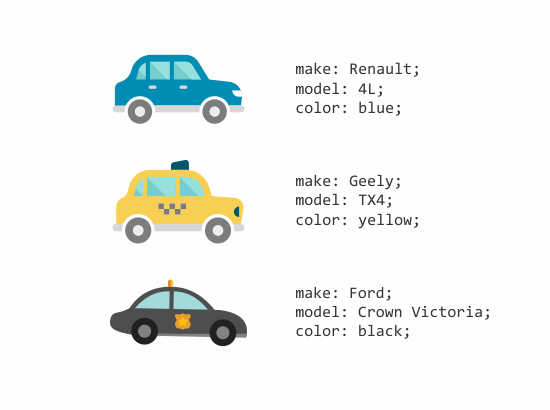
Bez obzira o kojem objektom modelu govorili, centralna figura takve programske paradigme jeste objekat.

**Šta je objekat?**

Osnovni gradivni element jezika JavaScript jeste objekat. Gotovo svi tipovi podataka, izuzev primitivnih tipova string, number, boolean, null i undefined su objekti. Pored toga, kao što je u prethodnoj lekciji prikazano, i funkcije su objekti.

Objekat je programska reprezentacija nekog entiteta. Za razliku od primitivnih tipova podataka, koji mogu da čuvaju samo jednu vrednost, objekti su kolekcija različitih vrednosti koje opisuju jedan entitet.

Primer jednog entiteta koji se može prevesti u objekat jeste automobil. Automobil može imati razne karakteristike, kao što su proizvođač, model, težina, boja i slično. Sve su to kandidati za skup vrednosti koje se mogu naći unutar jednog objekta. Različiti objekti entiteta *automobil* prikazani su na slici 26.3.



*Slika 26.3 – Prototip i objekti*

Na slici 26.3 se mogu videti različite vrste automobila. Svaki od njih ima svoje specifične karakteristike, ali je skup njihovih osobina identičan. Tako je svaki od prikazanih automobila jedan objekat, sa svojom bojom, nazivom proizvođača i nazivom modela. Ipak, svi automobili dele isti skup karakteristika, odnosno, sve je proizveo neki proizvođač, svi imaju svoje nazive, boje i slično. Takođe, određene osobine dele svi automobili: svi imaju četiri točka, dve osovine, jedan motor. Tako se skup naziva karakteristika, kao što su *proizvođač, naziv, boja, motor, broj vrata*, zajednički može nazvati šablon ili prototip. Na osovu takvog prototipa, dobijaju se konkretni objekti sa svojim specifičnim karakteristikama (boja žuta, proizvođač Geely, model TX4…).

**U JavaScript jeziku sve je zapravo:**

 objekat

 funkcija

 procedura

 klasa

**Objektni članovi**

Već je rečno da su objekti skupovi različitih vrednosti koji opisuju jedan isti pojam. *Proizvođač, model, boja i težina* opisuju jedan automobil. Navedene osobine unutar jednog objekta se nazivaju objektna svojstva.

Pored svojstava, objekti mogu da enkapsuliraju i različita ponašanja, iskazana kroz funkcije jezika JavaScript. Funkcija koja se nalazi unutar nekog objekta drugačije se naziva metoda. Tako objekat koji predstavlja neki automobil, može imati metode za pokretanje motora, zaustavljanje, kočenje, promenu brzina…

**Kreiranje objekata**

U JavaScript jeziku objekat može biti kreiran na nekoliko različitih načina:

* kreiranje objekata pojedinačno, korišćenjem objektnog literala, slično tehnici koja se koristi za kreiranje nizova
* kreiranje objekata pojedinačno, korišćenjem ključne reči new
* kreiranje proizvoljnog broja objekata, na osnovu objektnog konstruktora

**Kreiranje objekta korišćenjem objektnog literala**

Najjednostavniji način za kreiranje objekta i definisanje njegovih svojstva jeste korišćenje objektnog literala. Ovaj pristup omogućava kreiranje i inicijalizovanje objekta u jednoj liniji. Sledeći primer ilustruje kreiranje objekta car na ovaj način:

|  |  |
| --- | --- |
| 1 | var car = {make:"Subary", model:"Legacy", weight:1560, color:"black"}; |

Primer može potvrditi ono što je više puta do sada rečeno – da su objekti skupovi više vrednosti koje opisuju jedan pojam. Pritom se te vrednosti navode kao parovi ključeva i vrednosti. Ključ je naziv svojstva, dok je vrednost svojstva vrednost koja se dobija referenciranjem ključa.

**Kreiranje objekta korišćenjem ključne reči new**

JavaScript omogućava da se objekat kreira na sličan način na koji se to čini u objektnim jezicima koji se zasnivaju na klasama, i to korišćenjem ključne reči new. Objekat identičan onom iz prethodnog primera, korišćenjem ključne reči new, može se kreirati na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var car = new Object();  car.make = "Subary";  car.model = "Legacy";  car.weight = 1560;  car.color = "black"; |

Iz primera se može videti da je u prvoj liniji upotrebljena ključna reč new i na taj način je izvršeno kreiranje objekta. Vrlo zgodna osobina JavaScript objekata je da su dinamički, odnosno da omogućavaju dodavanje i uklanjanje svojstava u toku izvršavanja koda. Upravo je to i učinjeno u prikazanom primeru u kome su, nakon linije za kreiranje objekta, takvom objektu dodeljeni nazivi i vrednosti svojstava.

|  |
| --- |
| **Napomena**    *Oba prikazana primera proizvode identičan efekat. Kao što je to bio slučaj i sa ostalim objektnim elementima JavaScript jezika (npr. funkcijama), korišćenje ključne reči new nije obavezno, pa se zbog toga i ne preporučuje.* |

**Kreiranje objekta korišćenjem objektnog konstruktora**

JavaScript omogućava kreiranje objekata uz pomoć specijalne funkcije, koja se naziva objektni konstruktor. Korišćenjem takve funkcije, moguće je kreirati proizvoljan broj objekata koji dele osnovna svojstva. Na primer, svi automobili poseduju identična svojstva koja ih opisuju. Svi su proizvedeni od strane nekog proizvođača, imaju svoje ime, boju, težinu i slično. Ipak, svaki automobil uglavnom ima različite karakteristike, definisane vrednostima zajedničkih svojstava. Jedan automobil je crvene boje, drugi crne, svaki od njih ima svoju težinu, jedinstveno ime i slično. Kada je potrebno modelovati ovakvu situaciju, odnosno, kada je potrebno napraviti više objekata sa istim svojstvima, najprikladnije je za kreiranje objekata koristiti objektni konstruktor:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | function car(make, model, weight, color) {   this.make = make;   this.model = model;   this.weight = weight;   this.color = color;  }    var car1 = new car("Subary", "Legacy", 1563, "black");  var car2 = new car("Ford", "Taurus", 1876, "blue");  var car3 = new car("Porsche", "Panamera", 1963, "grey"); |

Funkcija sa nazivom car je najobičnija JS funkcija. Ipak, njena logika je definisana tako da omogući lak način za kreiranje objekata sa skupom zajedničkih svojstava. To ilustruju poslednje tri linije koda primera, u kojima se kreiraju tri objekta sa identičnim skupom svojstava, a različitim vrednostima. Za ovakva tri objekta se može reći da su istog tipa. Naravno, samo fiktivno, pošto su svi oni zapravo tipa Object.

Osnovna stvar na koju je potrebno obratiti pažnju u prikazanom primeru je ta da se kreirana funkcija car ne poziva na tradicionalan način, već njenom pozivanju prethodi ključna reč new. Na ovaj način, parseru je stavljeno do znanja da je potrebno da funkciju tretira kao konstruktorsku. Stoga će svakim pozivanjem konstruktorske funkcije na ovaj način biti kreiran novi objekat koji će biti dodeljen odgovarajućoj promenljivoj (car1, car2…).

U prikazanom primeru se može primetiti i upotreba jedne rezervisane reči, o kojoj u dosadašnjem toku kursa nije bilo reči. To je ključna reč this.

Ključna reč this uvek se odnosi na objekat koji je vlasnik koda u kome se takva reč nalazi. Kada se this navede unutar funkcije, odnosi se na objekat koji je vlasnik te funkcije.

Kada se this navede unutar objekta, this se odnosi na taj objekat.

Kada se this navede unutar konstruktorske funkcije, kao u primeru, odnosi se na konkretan objekat koji će korišćenjem takve funkcije biti kreiran. Upravo zbog toga, vrednosti prosleđene konstruktorskoj funkciji bivaju upisane u svojstva upravo tog (this) objekta.

Konstruktorska funkcija se, baš kao i svaka druga funkcija, može dodeliti nekoj promenljivoj. U takvoj situaciji, ta promenljiva postaje identifikator konstruktorske funkcije:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | var Car = function(make, model, weight, color) {   this.make = make;   this.model = model;   this.weight = weight;   this.color = color;  } |

**Metode**

Već je rečeno da se objekti sastoje iz članova definisanih u vidu parova ključeva i vrednosti. Objektni članovi mogu biti svojstva i metode. Primeri objekata koji su kreirani do sada oslikavali su objekte koji nisu sadržali metode, već isključivo svojstva.

Metoda je ništa drugo do funkcija koja se nalazi unutar nekog objekta. I kao što svojstva opisuju objekat i njegove karakteristike, metode govore šta objekat može da uradi i definišu logiku kojom se tako nešto postiže. Tu se razlike između svojstava i metoda završavaju, s obzirom da se metodama rukuje na vrlo sličan način na koji se rukuje svojstvima.

Kako bi se jednom objektu dodala metoda, potrebno je kreirati funkciju unutar takvog objekta i nju dodeliti novoj promenljivoj:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var Car = function(make, model, weight, color) {   this.make = make;   this.model = model;   this.weight = weight;   this.color = color;     this.startEngine = function(){    return "Engine of " + this.make + " " + this.model + " is started.";   };  } |

Konstruktorskoj funkciji dodata je funkcija koja je dodeljena promenljivoj startEngine. Funkcija kao rezultat vraća poruku o tome da je motor određenog automobila startovan. Sada je moguće kreirati jedan objekat korišćenjem ovakve konstruktorske funkcije i pozvati kreiranu metodu:

|  |  |
| --- | --- |
| 1  2 | var car1 = new Car("Subary", "Legacy", 1563, "black");  console.log(car1.startEngine()); |

Kod proizvodi sledeći efekat:

|  |  |
| --- | --- |
| 1 | Engine of Subary Legacy is started. |

**Prototip**

U uvodnom delu lekcije, rečeno je da JavaScript jezik koristi prototipski objektni model. Prototipovi su osnova nasleđivanja u jeziku JavaScript, i omogućavaju da jedan objekat dobije osobine drugog.

Svaki JavaScript objekat poseduje svoj prototip, a prototipovi su u JavaScriptu takođe objekti. U dosadašnjem toku lekcije, kreirana je jedna konstruktorska funkcija:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | var Car = function(make, model, weight, color) {   this.make = make;   this.model = model;   this.weight = weight;   this.color = color;     this.startEngine = function(){    return "Engine of " + this.make + " " + this.model + " is started.";  };  } |

S obzirom da su i funkcije u JavaScriptu objekti, to praktično znači da i ovakva konstruktorska funkcija mora imati svoj prototip. To je naravno istina.

Funkcije u JavaScriptu poseduju svojstvo prototype, koje omogućava da se definiše prototip svih objekata koji će biti kreirani kada se takva funkcija koristi kao konstruktorska. Tako je moguće prikazani primer modifikovati na sledeći način:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | var Car = function(make, model, weight, color) {   this.make = make;   this.model = model;   this.weight = weight;   this.color = color;    }    Car.prototype.startEngine = function(){  return "Engine of " + this.make + " " + this.model + " is started.";  };    var car1 = new Car("Subary", "Legacy", 1563, "black");    console.log(car1.startEngine()); |

Ovoga puta je metoda startEngine izmeštena unutar prototype svojstva konstruktorske funkcije. Na taj način, svi objekti koji budu kreirani korišćenjem ovakve funkcije, imaće metodu startEngine. Zbog toga primer proizvodi sledeći rezultat:

|  |  |  |
| --- | --- | --- |
| 1 | | Engine of Subary Legacy is started. |
| **Napomena**    *Svojstvo prototype koje poseduje svaka funkcija u jeziku JavaScript inicijalno ima vrednost null. Ipak, to ne znači da takva funkcija ne poseduje prototip. Jednostavno, prototip nekog objekta i svojstvo prototype koje imaju funkcije nisu iste stvari.* | |

**Nasleđivanje**

Nasleđivanje je jedan od osnovnih koncepata objektno orijentisanih jezika. Nasleđivanje omogućava da se karakteristike jednog objekta iskoriste za izgradnju drugog. Za lakše razumevanje nasleđivanja, najlakše je napraviti analogiju sa pojavama u prirodi. Svako od nas ima svog roditelja, tako da se može reći da je proces nastanka svake jedinke zapravo nasleđivanje. Pritom, naslednici dobijaju karakteristike svojih roditelja, koje mogu da prošire ili izmene. Identična situacija je i kod objektnog modela.

U JavaScript jeziku jedan objekat može da nasledi drugi i da pritom dobije sve njegove osobine. U takvom procesu, prototipovi predstavljaju osnovni mehanizam za postizanje nasleđivanja.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | var Person = function(firstname, lastname, age){   this.firstname = firstname;   this.lastname = lastname;   this.age = age;  }    Person.prototype.info = function(){   return this.firstname + " " + this.lastname;  }    var Student = function(firstname, lastname, age, university) {   Person.call(this, firstname, lastname, age);   this.university = university;  }    Student.prototype = Object.create(Person.prototype);  Student.prototype.constructor = Student;    Student.prototype.info = function(){   return this.firstname + " " + this.lastname + ", from University " + this.university;  }  var student1 = new Student("Ben", "Clock", 22, "Link");  console.log(student1.info()); |

Rezultat:

|  |  |
| --- | --- |
| 1 | Ben Clock, from University Link |

U primeru je prvo izvršeno kreiranje Person konstruktorske funkcije. Iskorišćeno je svojstvo prototype, kako bi se unutar prototipa Person objekta upisala jedna metoda sa nazivom info, koju će imati svi objekti koji budu nasledili objekat Person.

Nakon kreiranja konstruktorske funkcije Person, kreirana je i konstruktorska funkcija Student. Unutar konstruktorske funkcije Student, vrši se pozivanje konstruktorske funkcije objekta Person, korišćenjem metode call. Ova ključna reč omogućava da se neka funkcija prepiše i pozove bez potrebe za njenim redefinisanjem od nule, te se stoga često koristi prilikom nasleđivanja. Na primer, unutar konstruktorske funkcije objekta Person, već je definisana logika za postavljanje vrednosti svojstava koja se proslede kao parametri funkciji. Zašto bi se sada takva logika definisala opet? Upravo zbog toga se koristi funkcija call, kojoj se kao prvi parametar prosleđuje objekat koji predstavlja pozivaoca ove funkcije. Ostali parametri su regularni parametri funkcije. Nakon pozivanja funkcije call i prosleđivanja osnovnog seta parametara, u liniji ispod se vrši dodavanje jednog svojstva koje je karakteristično za objekte tipa Student. To je svojstvo university. Ovo je svojstvo koje će razlikovati objekte tipa Student od objekata tipa Person.

Nakon kreiranja konstruktorske funkcije za objekte tipa Student, definiše se i prototip za objekte ovog tipa. Prototip objekata Student će biti objekat tipa Person i za definisanje prototipa se koristi svojstvo prototype, koje, kao što je rečeno, poseduje svaka konstruktorska funkcija. Na taj način je definisano da će prototip svih objekata Student biti objekat Person.

Za kreiranje objekta prototipa u primeru se koristi funkcija create, objekta Object. Ova funkcija omogućava kreiranje novog objekta koji ima prototip koji se prosleđuje kao parametar. Tako se kreira novi objekat koji kao prototip ima Person objekat. Pored definisanja prototipa, prototipu objekta Student se dodeljuje i konstruktorska funkcija na sledeći način:

|  |  |
| --- | --- |
| 1 | Student.prototype.constructor = Student; |

Na ovaj način će se ubuduće za kreiranje objekata tipa Student koristiti konstruktorska funkcija Student.

Objekat Person poseduje metodu info koja je priključena njegovom prototipu. Na taj način i objekat Student će imati ovu metodu. Ipak, ova metoda ne odgovara u potpunosti osobinama objekta Student, zato što ne štampa informaciju o univerzitetu koji student pohađa. Podsetimo se da je svojstvo university karakteristično samo za objekte tipa Student. Da bi se ovaj problem rešio, odnosno, da bi metoda info štampala odgovarajuće informacije i za objekte Person i za objekte Student, u primeru se vrši redefinicija ove metode za objekte Student:

|  |  |
| --- | --- |
| 1  2  3 | Student.prototype.info = function(){   return this.firstname + " " + this.lastname + ", from University " + this.university;  } |

Na ovaj način je unutar prototipskog objekta objekata Student upisana metoda info, sa nešto drugačijom logikom u odnosi na onu koja postoji u objektu Person.

Na kraju, kada se kreira jedan objekat tipa Student i pozove metoda info, dobijaju se adekvatne informacije.

**Prostori imena**

Posebnu važnost prilikom rada sa objektima imaju i prostori imena. Prostori imena omogućavaju grupisanje određenih  srodnih funkcionalnosti. U JavaScript jeziku, za razliku od nekih drugih jezika, prostori imena se kreiraju korišćenjem objekata.

Prilikom deklarisanja promenljivih ili funkcija direktno u fajlu, oni postaju članovi globalnog objekta. Globalni objekat je onaj objekat koji se prvi kreira, čim JavaScript parser počne sa obavljanjem posla. Globalni objekat sadrži sve ugrađene metode i svojstva.

Kada se korisnički definisane funkcije i promenljive deklarišu izvan bilo koje funkcije, one postaju članovi globalnog objekta. Ukoliko u kodu postoji veliki broj korisnički definisanih funkcija i promenljivih, globalni objekat može da postane prenatrpan, a javlja se i rizik od sukoba imena.

Naime, prostor imena je jedan leksički sklop, pa se tako unutar njega ne mogu naći dve promenljive ili metode sa istim nazivom. Zbog svega ovoga je najbolje korisnički definisan kod smestiti unutar posebno kreiranog prostora imena:

|  |  |
| --- | --- |
| 1 | var my\_namespace = {}; |

Prikazanom linijom koda vrši se kreiranje jednog prostora imena. Ukoliko postoji opasnost da prostor imena sa navedenim imenom već postoji, može se napisati:

|  |  |
| --- | --- |
| 1 | var my\_namespace = my\_namespace || {}; |

U ovom slučaju, ukoliko prostor imena (odnosno promenljiva) već postoji, biće korišćen, a u protivnom će biti kreiran.

Unutar kreiranog prostora imena, sada se može kreirati jedna funkcija:

|  |  |
| --- | --- |
| 1  2  3  4  5 | var myNamespace = {        foo: function() {      }  }; |

Pozivanje funkcije:

|  |  |  |
| --- | --- | --- |
| 1 | | myNamespace.foo(); |
| **Napomena**    *Ovo je samo jedan od načina za kreiranje prostora imena u JavaScriptu, a za postizanje nečeg takvog mogu se koristiti i drugi pristupi, kao što je samopozivajuća funkcija i slično.* | |

**Ugrađeni JS objekti**

Sve do sada, u ovoj lekciji smo se bavili korisnički definisanim objektima. Pored korisnički definisanih objekata, JavaScript poznaje i skup ugrađenih objekata, koji su na raspolaganju programeru, a obavljaju različite funkcije.

Skup osnovnih ugrađenih JavaScript objekata prikazan je u tabeli 26.1.

|  |  |
| --- | --- |
| **Objekat** | **Opis** |
| Object | Osnovni objekat u jeziku JavaScript; svi objekti nasleđuju svojstva i metode koje su upisane unutar Object.prototype svojstva |
| Function | Globalni objekat za kreiranje funkcija |
| Boolean | Objektni omotač za boolean prost tip |
| Number | Objektni omotač za number prost tip |
| Math | Objekat koji sadrži matematičke funkcije i konstante |
| Date | Objekat koji se koristi za predstavljanje jednog trenutka u vremenu |
| String | Objekat za rad sa tekstom |
| RegExp | Objekat za kreiranje regularnih izraza |
| Array | Objekat za kreiranje nizova |
| Map | Objekat koji predstavlja kolekciju ključeva i vrednosti |
| Set | Objekat koji predstavlja kolekciju jedinstvenih vrednosti |
| JSON | Objekat koji sadrži logiku za parsiranje JSON-a i konverziju vrednosti u JSON |

*Tabela 26.1 – Osnovni ugrađeni objekti jezika JavaScript*