Objekti

Objekat je centralna figura JavaScript jezika. Funkcije, nizovi, ali i svi aplikativni programski interfejsi web pregledača, samo su neki od elemenata koji su u JavaScript jeziku zapravo objekti. Zbog toga je JavaScript objektno orijentisan jezik, odnosno jezik koji je baziran na objektima.

Objektno orijentisano programiranje omogućava apstrahovanje modela zasnovanih na pojmovima iz realnog sveta. Tako se objektno orijentisanim programiranjem rukuje skupom objekata koji međusobno sarađuju. Umesto funkcija ili procedura, koje su ništa drugo do jednostavne liste naredbi, objektno orijentisano programiranje omogućava objektu da unutar sebe objedini podatke i operacije i 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 smanjuje troškove razvoja.

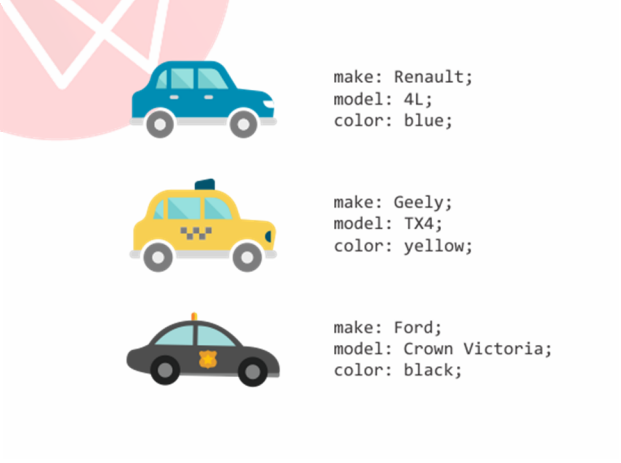
U ovoj lekciji i narednim lekcijama ovoga modula detaljno će biti obrađen objektni model JavaScript jezika. Na početku će biti prezentovani osnovni načini za kreiranje i korišćenje objekata, a zatim i neki napredniji pristupi koji podrazumevaju korišćenje prototipova i klasa.

Šta je objekat?

Objekat je složeni tip podatka, zato što unutar sebe može imati veći broj različitih vrednosti. Pored vrednosti, unutar objekta se mogu naći i funkcionalnosti, pa se tako objekat može doživeti kao omotač srodnih podataka i ponašanja.

Za objekte se veoma često kaže da su programske reprezentacije nekih entiteta. Entiteti su pojave iz realnog sveta – osoba, država, student, olovka, automobil, kompjuter... Stoga su objekti način da se ovakvi i slični entiteti definišu prilikom pisanja JavaScript koda.

Primer jednog entiteta koji se može prevesti u objekat jeste automobil. Automobil može imati razne karakteristike, kao što su proizvođač, model, zapremina motora, težina, boja i slično. Sve su to kandidati za skup vrednosti koje se mogu naći unutar jednog objekta. Različiti objekti entiteta *automobil* prikazani su slikom 1.1.



Slika 1.1. Primeri objekata

Na slici 1.1. prikazano je nekoliko automobila. Svaki od njih ima svoje specifične karakteristike, ali je skup njihovih osobina identičan. Tako je svaki od prikazanih automobila objekat određen sledećim karakteristikama:

- naziv proizvođača;
- naziv modela;
- boja karoserije.

Zbog čega su nam potrebni objekti?

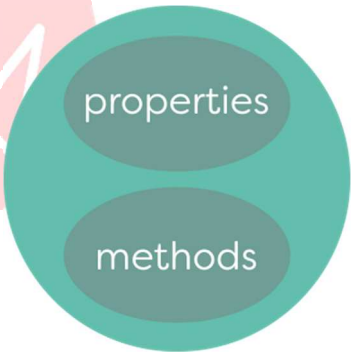
U ovom trenutku za vas je možda teško da razumete osnovnu svrhu objekata unutar nekog programskog jezika i kakve prednosti nam njihovo korišćenje donosi. Objekti zapravo obezbeđuju efikasan način za grupisanje podataka i operacija koje je nad takvim podacima moguće sprovoditi. Zamislite kao primer web aplikaciju koja rukuje studentima neke škole. Objekti omogućavaju da se svi podaci jednog studenta (*ime*, *prezime*, *adresa*, *broj indeksa* i slično), ali i sve operacije koje je korišćenjem aplikacije moguće sprovoditi nad studentima (*upiši*, *izmeni*, *obriši, prijavi ispit*), grupišu unutar nezavisnih komponenata kojima je lako rukovati.

Objektni članovi

Objekti su skupovi različitih vrednosti koje opisuju jedan isti pojam. *Proizvođač*, *model*, *boja* i *težina* opisuju jedan automobil. Navedene osobine unutar jednog objekta nazivaju se **objektna svojstva** (*engl. object properties*).

Pored svojstava, unutar objekata se mogu grupisati 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…

Može se rezimirati da su dva osnovna elementa JavaScript objekata svojstva i metode (slika 1.2).



Slika 1.2. Objekti se sastoje iz svojstava i metoda

U nastavku će prvo biti ilustrovani različiti načini za kreiranje objekata sa nekoliko svojstava, dok će nakon toga biti prikazano i kako se objektima dodeljuju funkcije koje postaju objektne metode.

Kreiranje objekata

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

- korišćenjem objektnog literala;
- korišćenjem konstruktorske funkcije `Object()`
- korišćenjem proizvoljne konstruktorske funkcije.

Kreiranje objekata korišćenjem objektnog literala

Najjednostavniji način za kreiranje objekta i definisanje njegovih svojstva jeste korišćenje objektnog literala. Sledeći primer ilustruje kreiranje jednog objekta:

```
var subaryLegacy = {
    make: "Subary",
    model: "Legacy",
    weight: 1560,
    color: "black"
};
```

Primer može da potvrdi konstataciju da su objekti skupovi više svojstava koji opisuju jedan pojam. Pritom se takva svojstva navode kao parovi ključeva i vrednosti. Ključ je naziv svojstva, dok vrednost svojstva jeste vrednost koja se dobija navođenjem ključa. Svaki par svojstva i vrednosti od narednog se odvaja korišćenjem karaktera zapeta (,).

Kreiranje objekata korišćenjem konstruktorske funkcije Object()

JavaScript omogućava kreiranje objekata korišćenjem specijalne ključne reči **new** u kombinaciji sa posebnom konstruktorskom funkcijom `Object()`, koja je automatski dostupna u jeziku JavaScript.

```
var myCar = new Object();
myCar.make = "Subary";
myCar.model = "Legacy";
myCar.weight = 1560;
myCar.color = "black";
```

Iz primera se može videti da je u prvoj liniji upotrebljena ključna reč `new` ispred posebne konstruktorske funkcije `Object()`, pa je na taj način izvršeno kreiranje objekta. Tako prva naredba prikazanog primera kreira jednu promenljivu tipa `object`, bez nekih posebnih osobina. Može se reći da je prvom naredbom kreiran jedan prazan objekat. Ipak, narednim naredbama kreiranom objektu dodaju se određene osobine – definisanjem svojstava.

U prikazanom primeru se po prvi put koristi takozvana **Dot notacija**. Nakon naziva promenljive navode se tačka, a zatim i naziv svojstva. Na ovaj način objektima je moguće dodeliti nova svojstva, kao što je to urađeno u primeru.

Dot notaciju je moguće koristiti i za čitanje vrednosti postojećih svojstava:

```
console.log(myCar.make + " " + myCar.model);
```

Unutar konzole se dobija:

```
Subary Legacy
```

Kreiranje objekata korišćenjem proizvoljne konstruktorske funkcije

Prethodni primer ilustrovao je kreiranje objekta korišćenjem ugrađene konstruktorske funkcije, čime se dobija prazan objekat bez ikakvih predefinisanih osobina. Ipak, veoma često može se javiti potreba za kreiranjem većeg broja objekata koji dele identičan skup karakteristika (na primer, kada je potrebno kreirati veći broj automobila različitih karakteristika). U takvim situacijama je moguće otići korak dalje i u potpunosti samostalno definisati konstruktorsku funkciju za kreiranje objekata.

Korišćenjem samostalno definisane konstruktorske 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 proizvoljnu konstruktorsku funkciju:

```
function Car(make, model, weight, color) {
      this.make = make;
      this.model = model;
      this.weight = weight;
      this.color = color;
}
```

Funkcija sa nazivom `Car` je naizgled obična JavaScript funkcija. Ona prihvata četiri ulazna parametra i ne poseduje povratne vrednosti. Ipak, postoji nekoliko osobina koje ovu funkciju izdvajaju od drugih. Za početak, ona je imenovana korišćenjem velikog početnog slova. Tako nešto nije obaveza, ali predstavlja veoma korisnu notaciju, kojom se konstruktorske funkcije distanciraju od *običnih*.

I unutar tela prikazane konstruktorske funkcije postoji jedna novina. Reč je o upotrebi jedne posebne ključne reči JavaScript jezika – `this`.

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 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.

Nakon kreiranja konstruktorske funkcije, nju je veoma lako moguće iskoristiti za kreiranje proizvoljnog broja različitih objekata koji poseduju osobine definisane konstruktorskom funkcijom (*marka*, *model*, *težina* i *boja*):

```
var car1 = new Car("Subary", "Legacy", 1563, "black");
var car2 = new Car("Ford", "Taurus", 1876, "blue");
var car3 = new Car("Porsche", "Panamera", 1963, "grey");
```

U primeru se kreiraju tri objekta sa identičnim skupom svojstava, ali različitim vrednostima. Za ovakva tri objekta može se reći da su *istog tipa*. Naravno, samo fiktivno, pošto su svi oni zapravo tipa `Object`.

Osnovna osobina koja neku funkciju čini konstruktorskom jeste način na koji se ona poziva. U primeru se može videti da se konstruktorska funkcija `Car()` ne poziva na tradicionalan način, već njenom pozivanju prethodi **ključna reč new**. Na ovaj način, izvršnom okruženju je stavljeno do znanja da je potrebno da funkciju tretira kao konstruktorsku. Stoga, svakim pozivanjem funkcije na ovaj način biće kreiran novi objekat, koji će biti dodeljen odgovarajućoj promenljivoj (`car1, car2…`).

Koji pristup za kreiranje objekata koristiti?

Potpuno je očekivano da se nakon prethodnih redova pitate koji pristup za kreiranje objekata koristiti. Ukoliko je potrebno brzo i jednostavno napraviti objekat, a da je pritom poznato da neće biti potrebe za kreiranjem većeg broja objekata identičnog skupa karakteristika, najbolje je koristiti objektni literal. Kada unapred znate da će biti potrebe za kreiranjem većeg broja objekata identičnog skupa karakteristika, najbolje je napraviti sopstvenu konstruktorsku funkciju. Korišćenje ugrađene konstruktorske funkcije `Object()` nema neku posebnu upotrebnu vrednost.

Metode

Objektni članovi mogu biti svojstva i metode. Primeri objekata koji su kreirani dosad 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.

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

```
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 metoda koja je dodeljena promenljivoj `startEngine`. Metoda 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:

```
var car1 = new Car("Subary", "Legacy", 1563, "black");
console.log(car1.startEngine());
```

Kod proizvodi sledeći efekat:

```
Engine of Subary Legacy is started.
```

Pitanje

U jeziku JavaScript sve je zapravo:

- **objekat;**
- funkcija;
- procedura;
- klasa.

Objašnjenje:

JavaScript je jezik kod koga se objekti zasnivaju na drugim objektima, a na vrhu takve hijerarhije se nalazi objekat `Object`. Klase su zapravo funkcije, dok su funkcije na kraju objekti. U jeziku JavaScript pojmovi procedura i funkcija mogu se smatrati sinonimima.

Softverski dizajn šabloni

U ovom kursu, uporedo sa predstavljanjem različitih naprednih koncepata jezika JavaScript, biće reči i o veoma bitnom pojmu programiranja – softverskim dizajn šablonima (*engl.* **software design patterns***).*

Softverski dizajn šabloni opisuju rešenja veoma čestih problema, do kojih dolazi prilikom dizajniranja unutrašnje strukture nekog softverskog proizvoda. Reč je zapravo o dokazanim pristupima, odnosno praksama koje su se tokom vremena pokazale kao dobre.

Bitno je razumeti da dizajn šabloni nisu gotova rešenja, već samo generalizovana, koncizna uputstva koja se mogu koristiti za konkretnu implementaciju rešenja. Stoga, dizajn šabloni nisu u vezi ni sa jednim programskim jezikom, te se njihova načela mogu primeniti prilikom kreiranja različitih tipova aplikacija za različite platforme.

Svi softverski dizajn šabloni dele se na tri grupe:

- šabloni kreiranja (*engl. creational patterns*);
- strukturni šabloni (*engl. structural patterns*);
- šabloni ponašanja (*engl. behavioral patterns*).

Postoji velika verovatnoća da ste, kreirajući JavaScript kod, dosad i sami koristili principe nekih dizajn šablona, a da toga niste ni bili svesni. Takva situacija postoji i u ovoj lekciji. Mi smo u prikazanim primerima iskoristili određena načela dva dizajn šablona: **Module Pattern** i **Constructor Pattern.**

Prilikom ilustrovanja kreiranja objekata korišćenjem objektnog literala, iskorišćena su neka od načela *Module dizajn šablona*. Naime, reč je o šablonu kojim se definiše pojam modula kao izolovanih i organizovanih jedinica koda unutar nekog programa. Upravo jedan od načina za kreiranje takvih modula u JavaScriptu podrazumeva korišćenje objektnog literala.

```
            var subaryLegacy = {
                make: "Subary",
                model: "Legacy",
                weight: 1560,
                color: "black",
                startEngine: function () {
                    return "Engine of " + this.make + " " + this.model + "
is started.";
                }
            };
```

Kod ilustruje primer kreiranja jednog objekta. Unutar vitičastih zagrada objedinjeni su podaci i ponašanja koji su karakteristični za takav objekat. Otuda analogija sa pojmom modula – izolovane organizacione jedinice koda unutar nekog programa. Naravno, Module dizajn šablon je mnogo više od onoga što je upravo prikazano. Stoga, u nekim narednim lekcijama priča o Module dizajn šablonu biće nastavljena i proširena.

Primeri iz poglavlja o konstruktorskim funkcijama iz ove lekcije ilustrovali su korišćenje principa *Constructor dizajn šablona*:

```
      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.";
            };
      }
```

Constructor dizajn šablon propisuje način na koji se kreiraju objekti određenog tipa – korišćenjem specijalnih funkcija koje se nazivaju konstruktori. Reč je o šablonu čiju je realizaciju u jeziku JavaScript moguće postići korišćenjem ugrađenih funkcionalnosti jezika. Naime, sam JavaScript poznaje pojam konstruktorskih funkcija. Reč je o funkcijama čijem pozivanju prethodi upotreba ključne reči `new` – čime se na osnovu konstruktorske funkcije kreira novi objekat. Ipak, bitno je znati da se ne može računati da realizacija svakog dizajn šablona bude podržana ugrađenim elementima jezika, kao što je to bio slučaj kod Constructor dizajn šablona u JavaScriptu. U narednim lekcijama moći ćete da vidite primere realizacije nekih drugih šablona, koji nisu direktno podržani od strane samog jezika.

Postulati objektno orijentisanog programiranja

U prethodnim redovima iznete su neke osnovne osobine objekata u JavaScriptu i tehnike za njihovo kreiranje. Ipak, pored iznetih osobina, za adekvatno razumevanje ovog veoma značajnog pojma neophodno je poznavati i osnovne postulate objektno orijentisanog programiranja. Reč je o sledećim pojmovima:

- **apstrakcija**

Kreiranje objekata koji predstavljaju entitete iz realnog sveta naziva se drugačije modelovanje. Modelovanjem se omogućava rukovanje entitetima u programskom kodu JavaScript programa. Ipak, prilikom procesa modelovanja, često se veoma kompleksni pojmovi iz realnog sveta predstavljaju znatno jednostavnije. Tako smo i mi u prethodnim primerima prilikom kreiranja objekata automobila modelovali samo nekoliko osobina i ponašanja takvog realnog pojma. Drugim rečima, jedan automobil, pored proizvođača, modela, boje i zapremine motora, može se definisati i brojnim drugim osobinama – tip menjača, emisiona klasa, broj brzina, tip pogona itd. Takođe, jedan realan automobil, pored mogućnosti startovanja motora, omogućava i brojna druga ponašanja – ubrzavanje, kočenje, upravljanje, promenu brzine itd. Jasno je da smo u prethodnim primerima, prilikom modelovanja pojma automobila, napravili jedan uprošćeni, pojednostavljeni model. Takav proces, kojim se postiže kreiranje jednostavnih modela znatno kompleksnijih entiteta iz realnog sveta, u objektno orijentisanom programiranju naziva se drugačije apstrakcija.

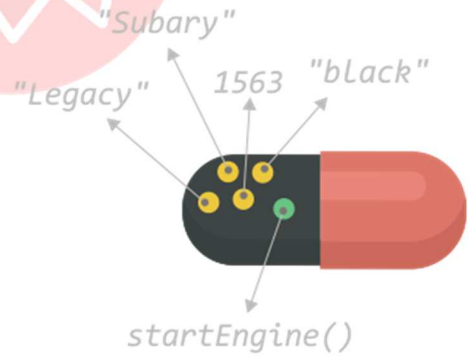
U objektnom programiranju apstrakcija omogućava da se složeni entiteti iz realnog sveta modeluju baš onako kako mi to želimo – sa nivoom detalja koji odgovara potrebama programske logike. Takođe, apstrakcija omogućava i da se u potpunosti skriju unutrašnji detalji nekog entiteta. Tako, na primer, mi ne moramo znati na koji način će biti obavljeno startovanje motora nekog vozila. Na nama je da pozovemo metodu za startovanje, dok je sam način startovanja (korišćenjem tradicionalnog ključa, ključa kartice, dugmeta za startovanje...) skriven unutar objekta i logike kojom je realizovana metoda za startovanje.

Pojam apstrakcije objektno orijentisanog programiranja može se razumeti i na primeru metoda koje web pregledači izlažu na korišćenje JavaScript kodu. Jedna od takvih metoda je i metoda `log()` objekta `Console`. Apstrakcija omogućava da se ova metoda koristi za ispis teksta unutar konzole web pregledača, bez potrebe za poznavanjem načina na koji je takva metoda realizovana. Tako apstrakcija naposletku smanjuje ponavljanje koda, jer nas oslobađa potrebe za ponovnim pisanjem funkcionalnosti koje su već realizovane.

- **enkapsulacija**

Objekat predstavlja zaokruženu celinu, zato što u potpunosti opisuje jedan pojam iz realnog sveta. Unutar objekata objedinjuju se informacije i funkcionalnosti koje se tiču entiteta koji se modeluje. Objedinjavanje takvih informacija i funkcionalnosti drugačije se naziva enkapsulacija.

Enkapsulacija se najbolje može razumeti na nešto ranije prikazanim primerima kreiranja objekata automobila. Jedan automobil modelovan korišćenjem JavaScript objekta ilustrativno se može prikazati kao na slici 1.3.



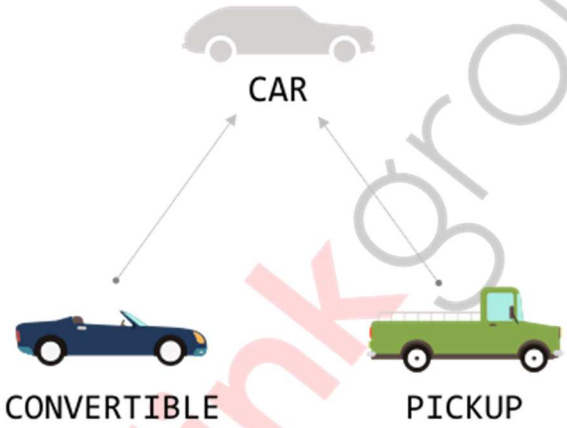
Slika 1.3. Enkapsulacija

Slika 1.3. dočarava pojam enkapsulacije na primeru jednog JavaScript objekta. Na slici nije slučajno odabrana analogija sa kapsulom tablete – svojstva i metode objedinjuju se unutar objekta, baš kao i različiti sastojci unutar jedne tablete. Enkapsulacijom se različita svojstva i metode jednog objekta prezentuju kao celina, kojom se rukuje korišćenjem naziva objekta.

- **nasleđivanje i polimorfizam**

Primeri u ovoj lekciji ilustrovali su kreiranje nekoliko objekata koji su predstavljali različite automobile. Ipak, svi takvi automobili predstavljali su generički uopšten tip automobila. Ukoliko se javi potreba za modelovanjem i konkretnijih tipova automobila (kabriolet, limuzina, SUV i slično), objektno orijentisani model omogućava korišćenje pojmova nasleđivanja i polimorfizma.

Nasleđivanje omogućava kreiranje objekata koji osobine i ponašanja nasleđuju od nekog drugog objekta.



Slika. 1.4 Nasleđivanje

Slika 1.4. ilustruje jedan primer nasleđivanja u objektno orijentisanom svetu. Objekti *Convertible* i *Pickup* nasleđuju osnovni skup svojstava i ponašanja od objekta *Car*. Unutar objekta *Car* objedinjene su osnovne karakteristike koje poseduju svi automobili. Objekti *Convertible* i *Pickup* nasleđuju takav skup osnovnih osobina, ali pored njih poseduju i određene osobine koje su karakteristične samo za njih.

Prilikom realizovanja nasleđivanja u objektno orijentisanom programiranju veoma često dolazi do potrebe za redefinisanjem nekih nasleđenih ponašanja. Tako roditeljski objekat može definisati neku funkcionalnost koja ne zadovoljava u potpunosti njegove naslednike. Ipak, naslednici imaju mogućnost da nasleđeno ponašanje prilagode, dopune ili u potpunosti izmene. Na taj način se stvaraju različite verzije jednog istog ponašanja, odnosno verzije koje odgovaraju svakom pojedinačnom tipu objekta. Takva osobina objektno orijentisanog programiranja drugačije se naziva polimorfizam, odnosno višestruko značenje jednog istog pojma.

Pojmovi nasleđivanja i polimorfizma nešto su teži za razumevanje, pogotovu ukoliko nisu propraćeni realnim primerima. Stoga, naredne lekcije ovoga modula ilustruju brojne primere za realizaciju dva spomenuta postulata objektno orijentisanog programiranja u JavaScriptu.

Rezime

- objektno orijentisano programiranje omogućava apstrahovanje modela zasnovanih na pojmovima iz realnog sveta;
- gotovo svi tipovi podataka, izuzev primitivnih tipova `string`, `number`, `boolean`, `null` i `undefined`, u JavaScriptu su objekti;
- objekti objedinjuju vrednosti i ponašanja koji se odnose na jedan pojam;
- unutar objekata, vrednosti se čuvaju unutar svojstava, a ponašanja se definišu metodama;
- JavaScript omogućava da se objekti kreiraju upotrebom objektnih literala i konstruktorskih funkcija;
- konstruktorska funkcija jeste funkcija čijem pozivanju prethodi ključna reč `new`, što na kraju rezultuje stvaranjem novog objekta;
- ključna reč `this` odnosi se na objekat koji je vlasnik koda u kome se takva reč nalazi;
- softverski dizajn šabloni opisuju rešenja veoma čestih problema, do kojih dolazi prilikom dizajniranja unutrašnje strukture nekog softverskog proizvoda;
- osnovni postulati objektno orijentisanog programiranja su: apstrakcija, enkapsulacija, nasleđivanje i polimorfizam;
- apstrakcija omogućava da se složeni entiteti iz realnog sveta modeluju sa nivoom detalja koji odgovara potrebama programske logike;
- enkapsulacija se odnosi na postupak objedinjavanja informacija i funkcionalnosti unutar jedne celine – objekta;
- nasleđivanje omogućava kreiranje objekata koji osobine i ponašanja nasleđuju od nekog drugog objekta;
- polimorfizam se odnosi na višestruko značenje jednog istog pojma, što se često koristi za kreiranje više metoda istog naziva, ali nešto drugačije logike.