OOP JAVASCRIPT

Objektno orijentisano programiranje (OOP) je paradigma programiranja u kojoj se programiranje bazira

na objektima koji komuniciraju jedni s drugima kako bi izvršili neku funkciju.

U JavaScriptu, OOP se može postići korišćenjem klase i objekata.

Klasa u JavaScriptu je predstavljena kao funkcija, a objekat je instanca te klase.

Kada se instancira objekat, on dobija svoje sopstvene osobine i metode koje su definisane u klasi.

Klasa može biti nasleđena od druge klase, a objekat može biti instanciran iz ove podklase.

Evo jednog jednostavnog primera kako se koristi OOP u JavaScriptu:

class Osoba {

constructor(ime, prezime, godine) {

this.ime = ime;

this.prezime = prezime;

this.godine = godine;

}

// Metoda koja vraća celo ime osobe

celoIme() {

return `${this.ime} ${this.prezime}`;

}

// Metoda koja vraća broj godina osobe

brojGodina() {

return this.godine;

}

}

// Kreiranje instance klase Osoba

let osoba = new Osoba('Marko', 'Markovic', 30);

// Pozivanje metoda na instanci klase Osoba

console.log(osoba.celoIme()); // 'Marko Markovic'

console.log(osoba.brojGodina()); // 30

U ovom primeru kreiramo klasu Osoba koja ima tri osobine: ime, prezime i godine.

Ova klasa takođe ima dve metode: celoIme koja vraća celo ime osobe i brojGodina koja vraća broj godina osobe.

Nakon toga, instanciramo objekat osoba iz klase Osoba i dodeljujemo vrednosti osobinama.

Kada pozovemo metode celoIme i brojGodina na ovom objektu, one vraćaju odgovarajuće vrednosti.

Ovaj primer je jednostavan, ali OOP u JavaScriptu može biti mnogo složeniji.

Na primer, možete imati više klasa koje nasleđuju jedna od druge, ili

možete koristiti OOP za upravljanje kompleksnijim aplikacijama.

// class Product {

// // title = "Default";

// // imageUrl;

// // description;

// // price;

// // myFunc() {}

// constructor(title, image, desc, price) {

// this.title = title;

// this.imageUrl = image;

// this.description = desc;

// this.price = price;

// }

// }

class Product {

constructor(title, image, desc, price) {

this.title = title;

this.imageUrl = image;

this.description = desc;

this.price = price;

}

}

const myProd = new Product();

console.log(myProd);

// Also possible:"

// myProd.description = "The best product in the world";

const myProd2 = new Product("Tepih", "../../../", "Nema bolji", 130);

console.log(myProd2);

**STATICKE METODE**

Static metode u JavaScriptu su metode koje se mogu pozivati direktno na klasi, bez potrebe za kreiranjem objekta klase.

Ovo ih razlikuje od instance metoda, koje se pozivaju na objektima koji su kreirani na osnovu klase.

Postoji nekoliko razloga zašto biste koristili statičke metode unutar klase u JavaScriptu:

Pristupanje podacima klase: Ako želite pristupiti podacima klase, ali ne želite kreirati objekt klase,

možete koristiti statičku metodu.

Na primer, ako imate klasu koja sadrži konstante koje se koriste u različitim delovima vašeg koda,

možete koristiti statičku metodu za pristup tim konstantama bez kreiranja objekta klase.

class Constants {

static PI = 3.14;

static getPi() {

return Constants.PI;

}

}

console.log(Constants.getPi()); // 3.14

Deljenje funkcionalnosti: Ako želite da delite funkcionalnost između svih objekata kreiranih na osnovu klase,

možete koristiti statičku metodu. Na primer, ako imate klasu koja obrađuje neki tip podataka,

možete kreirati statičku metodu koja vrši validaciju tog tipa podataka i pozvati je iz različitih delova vašeg koda.

class DataProcessor {

static isValid(data) {

// Validacija podataka

}

processData(data) {

if (!DataProcessor.isValid(data)) {

throw new Error('Invalid data');

}

// Procesiranje podataka

}

}

Performanse:

Korišćenje statičkih metoda može poboljšati performanse vašeg koda, posebno ako se metode pozivaju mnogo puta.

Korišćenjem statičkih metoda izbegava se kreiranje novih objekata svaki put kada se metoda pozove.

Ukratko, koristite statičke metode kada želite pristupiti podacima klase,

deliti funkcionalnost između svih objekata kreiranih na osnovu klase ili poboljšati performanse vašeg koda.

**INHERITANCE**

Nasleđivanje omogućava kreiranje nove klase na osnovu postojeće klase, čime se izbegava ponavljanje koda i olakšava održavanje programa.

Evo jednog jednostavnog primera nasleđivanja u JavaScriptu:

class Osoba {

constructor(ime, prezime) {

this.ime = ime;

this.prezime = prezime;

}

pozdrav() {

console.log(`Pozdrav, ${this.ime} ${this.prezime}!`);

}

}

class Student extends Osoba {

constructor(ime, prezime, brojIndeksa) {

super(ime, prezime);

this.brojIndeksa = brojIndeksa;

}

prijaviIspit(predmet) {

console.log(`${this.ime} ${this.prezime} je prijavio ispit iz predmeta ${predmet}.`);

}

}

let student = new Student('Marko', 'Markovic', '12345');

student.pozdrav(); // Pozdrav, Marko Markovic!

student.prijaviIspit('Matematika'); // Marko Markovic je prijavio ispit iz predmeta Matematika.

U ovom primeru, Student klasa nasleđuje Osoba klasu korišćenjem ključne reči extends.

To znači da Student klasa ima sve osobine i metode koje ima Osoba klasa, plus dodatne osobine

i metode koje definira sama klasa.

U konstruktoru klase Student, vidite da se prvo poziva super(ime, prezime) kako bi se pozvao konstruktor klase Osoba

i inicijalizovali ime i prezime osobine. Zatim se inicijalizuje brojIndeksa.

U klasi Student, dodali smo novu metodu prijaviIspit, koja prikazuje poruku o tome koji predmet student prijavljuje.

Kada instanciramo objekat student i pozovemo prijaviIspit i pozdrav, primetite da se pozdrav metoda poziva iz

Osoba klase, dok se prijaviIspit poziva iz Student klase.

Super metoda se koristi za pozivanje metoda klase iz koje se vrši nasleđivanje.

U ovom primeru, super metoda se koristi u konstruktoru klase Student kako bi se pozvao konstruktor klase Osoba.

Super metoda može se koristiti i za pozivanje metoda klase iz koje se vrši nasleđivanje.

U nasleđenoj metodi, super metoda se može koristiti za pozivanje metode iz klase iz koje se vrši nasleđivanje.

Na primer, ako klasa Osoba ima metodu pozdrav, a klasa Student nasleđuje tu metodu,

možemo koristiti super metodu kako bismo pozvali originalnu pozdrav metodu iz `Osoba

**PRIMERI**

Kreiranje objekta sa svojstvima i metodama:

const osoba = {

ime: "Ana",

prezime: "Anić",

godine: 25,

dobrodosli: function() {

console.log("Dobrodošli, ja sam " + this.ime + " " + this.prezime + "!");

},

rodjendan: function() {

this.godine++;

}

};

osoba.dobrodosli(); // ispisuje "Dobrodošli, ja sam Ana Anić!"

console.log(osoba.godine); // ispisuje 25

osoba.rodjendan();

console.log(osoba.godine); // ispisuje 26

**Konstruktori i nasleđivanje:**

function Osoba(ime, prezime, godine) {

this.ime = ime;

this.prezime = prezime;

this.godine = godine;

}

Osoba.prototype.dobrodosli = function() {

console.log("Dobrodošli, ja sam " + this.ime + " " + this.prezime + "!");

};

function Student(ime, prezime, godine, fakultet) {

Osoba.call(this, ime, prezime, godine);

this.fakultet = fakultet;

}

Student.prototype = Object.create(Osoba.prototype);

Student.prototype.constructor = Student;

Student.prototype.poloziIspit = function() {

console.log("Položio sam ispit!");

};

const student = new Student("Ana", "Anić", 20, "Fakultet računarstva");

student.dobrodosli(); // ispisuje "Dobrodošli, ja sam Ana Anić!"

console.log(student.fakultet); // ispisuje "Fakultet računarstva"

student.poloziIspit(); // ispisuje "Položio sam ispit!"

Privatne metode i svojstva uz pomoć zatvaranja:

javascript

function Osoba(ime, prezime, godine) {

this.ime = ime;

this.prezime = prezime;

this.godine = godine;

let bankovniRacun = 0;

function dodajNaRacun(iznos) {

bankovniRacun += iznos;

}

this.stanjeRacuna = function() {

return bankovniRacun;

};

this.uplati = function(iznos) {

dodajNaRacun(iznos);

console.log("Uplaćeno " + iznos + " kuna.");

};

}

const osoba = new Osoba("Ana", "Anić", 30);

console.log(osoba.stanjeRacuna()); // ispisuje 0

osoba.uplati(100);

console.log(osoba.stanjeRacuna()); // ispisuje 100