Vježba 6 – objektno-orijentirani JS, getteri, setteri

Upute za rješavanje lab. vježbi

Današnja vježba sastoji se od teorijskog dijela i jednog zadatka kojeg je potrebno riješiti za vrijeme vježbi. Kod ispisa vrijednost, poruka ili povratne vrijednosti funkcije potrebno je koristiti literale.

Teorijski dio

JavaScript kao objektno-orijentiran jezik

Nakon zadnjeg predavanja, odlučila sam umjesto zadataka pripremiti teorijski dio s primjerima kako bi bolje razumjeli JavaScript kao objektno-orijentiran jezik. Na predavanju smo raspravljali kako se objekt može instancirati na više načina, što je poprilično zbunjujuće ako poznajete neki drugi objektno-orijentiran jezik kao što su Java, C++, C#, Python itd.

Za pripremu ovog teorijskog dijela koristila sam se različitim literaturama i javnim raspravama drugih programera, stoga ću navesti poveznice na kraju dokumenta kako bi dodatno pročitali ako vas bude zanimalo.

Uvođenjem standarda ES6 2015. godine dolazi do promjena, a ovo su neke od njih:

- uvođenje let i const tipa podatka
- arrow funkcije
- for of
- class
- string i array ugrađene funkcije

Posebno ću istaknuti uvođenje ključne riječi „class“ koja bi trebala olakšati prelazak i prilagodbu programerima koji dolaze s čistih objektno-orijentiranih jezika i postići ono što je bit objektno orijentiranog jezika.

JavaScript koristi više programskih paradigmi. Programska paradigma je stil ili način programiranja. Program napisan funkcionalnom paradigmom, kao što je JavaScript, je čišći, jednostavniji i temelji se na funkcijama (arrow, anonimne, closure, konstruktor itd.). Osim navedene paradigme možemo koristiti objekte, pa kažemo da JavaScript ima i objektno-orijentiranu paradigmu koja se temelji na konceptu objekta. Izbor programske paradigme je ponekad težak zadatak.

Prije svega, potrebno je uzeti u obzir problem koji se treba riješiti. Logično je da različiti problemi zahtijevaju različite pristupe razmišljanja, a samim time i različit programski kôd. Dobrim odabirom programske paradigme se može itekako utjecati na kvalitetu programskog koda što kasnije rezultira jednostavnijim otklanjanjem pogrešaka, testiranjem i održavanjem. Ponekad se neka programska paradigma čini idealnom za rješavanje određenog dijela problema.

Ako se teorija ostavi sa strane, programski kod je samo oruđe za rješavanje problema te mu je tako potrebno i pristupiti. Mnogi jezici podržavaju pisanje koda koristeći više programskih paradigmi.

Na internetu postoje mnogobrojne rasprave o uvedenim promjenama - ES6 standardu. Često se spominje izraz „*syntactic sugar*“, što bi prijevodu značilo uvođenje nečeg novog, a već postoji i funkcionira. Primjer bi bio uvođenje klasa, a ista funkcionalnost se postiže s konstruktor funkcijom.

Također, kažemo da je JavaScript objektno-orijentiran jezik i kao takav koristi objektno-orijentiranu paradigmu. Programi napisani objektno-orijentiranom paradigmom su dizajnirani tako da sadrže objekte koji su u interakciji i temelje se na klasama, što znači da su objekti instanca klase. Na temelju jedne klase se može napraviti više objekata jer je klasa samo predložak za stvaranje objekata. Nažalost, u JS ne postoje klase, iako postoji ključna riječ „class“, već se sve temelji na objektima i prototipiranju.

Navest ću i opisati nekoliko važnih svojstava koja objektno-orijentiran jezik treba imati:

1. **Enkapsulacija** se povezuje sa skrivanjem informacija, iz tog razloga su nam potrebni privatni atributi ili metode. U nekim jezicima, kao što je C++, objekti imaju public, private i protected metode i svojstva. U JavaScriptu su sva svojstva i metode automatski označene s *public*, ali postoje načini za zaštitu podataka unutar objekta (let, closure funkcije, #).
2. **Agregacija** označava kombiniranje nekoliko objekata u jedan npr. objekt A „koristi“ objekt B.
3. **Nasljeđivanje** je odličan način ponovno korištenja već napisanog programskog koda. Osim što smanjuje vrijeme potrebno za razvoj dodatno povećava i pouzdanost. Uzmimo za primjer klasu Vozilo s metodama kreni() i stani(), koju nasljeđuju klasa Auto i Motor. Kod JavaScripta je situacija nešto drugačija, objekti nasljeđuju od objekata jer klase ne postoje.
4. **Polimorfizam** je pružanje jedinstvenog sučelja entitetima različitih tipova. Zamislite da se u programskom kodu nalazi varijabla prijevoznoSredstvo, a da pri tome ne znamo je li riječ o vozilu, automobilu ili nekom drugom prijevoznom sredstvu. Unatoč navedenom problemu, i dalje možemo pozvati metodu vozi() te neće doći do pogreške prilikom izvršavanja programa.

Potrebno je napomenuti da JavaScript nije čisti objektno-orijentiran jezik zbog navedenih „pravila“, iako koristi objektno-orijentirano paradigmu. Značenje klasa unutar JavaScripta nije jednako onome unutar C# ili Jave. Traženje balansa između objektno-orijentirane i funkcionalne programske paradigme nije jednostavan posao. Kao i uvijek, najbitniji je problem koji se rješava, a onda znanje programera i njegove osobne preferencije.

Kreiranje objekta

Kako je navedeno, objekt je instanca klase, a kako JavaScript nema klase, objekt možemo izvesti na više načina. U primjeru ću pokazati sve načine stvaranja objekta. Također, spomenut ću neke elemente koje nismo radili na predavanju ali možda neki studenti poznaju neki drugi objektno-orijentiran jezik pa će uspjeti povezati.

1. **Object literal**

Jednostavna struktura koju je konceptualno lako razumjeti. Objekt sadrži atribute, ključ i vrijednost odvojenu dvotočkom, i metode. Postavljanjem varijable na prazan objekt je isto kao da ste stvorili new Object(). Za stvaranje više objekata ovaj način nije dobar jer svaki objekt moramo zasebno kreirati, trošiti memoriju i nepotrebno kopirati kôd.

```
const osoba = {
    ime: 'Antonija',
    prezime: 'Baric',
    predmet: 'PWKS',

    info : function(){
        console.log(`${this.ime} ${this.prezime}
        dobrodošli na ${this.predmet}`)
    }
}

console.log(`Predmet ${osoba.predmet}`)
osoba.info()
```

Slika 1. Kreiranje objekta preko Object literal

2. **Factory functions**

Funkcija koja vraća Object literal, type Object. Ovim načinom se od jedne funkcije može instancirati više objekata, a svaki objekt dobiva kopiju funkcije pa pozivajući metode i promjenom parametara ne utječemo na drugi objekt.

Na neobičan način postiže *nasljeđivanje* pa umjesto vraćanja objekta ključnom riječi return , vraća se novi objekta naredbom *return Object.create()* jer se u tom slučaju proslijedi i prototip. Jedan od razloga zašto je dobro koristiti konstruktor funkcije.

```
function kreirajOsobu(ime, prezime) {
    return {
        ime: ime,
        prezime: prezime,
        predmet: 'PWKS',
        info(){
        console.log(`${this.ime} ${this.prezime}
        dobrodošli na ${this.predmet}`)
    }
    };
  }
const student = kreirajOsobu('Antonija', 'Barić')
student.info()
```

Slika 2. Kreiranje objekta factory funkcijom

Factory funkcija je jednostavnija za korištenje. Odlično koristi ideju Closure funkcija s kojima se može postići „*privatnost*“ podataka ali ponekad je potrebno uložiti više truda kako bi se postigle određene funkcionalnosti.

3. Constructor functions

Funkcija koja koristi ključnu riječ „*new*“ i stvara referencu na objekt tipa Osoba (slika 3). Svaki objekt dobiva svoj prototip, kojemu se može dodavati svojstvo i s lakoćom postići nasljeđivanje, za razliku od factory funkcija.

Naziv konstruktor funkcije piše se velikim početnim slovom. Ključna riječ „new“ kreira novi objekt, postavlja prototip objekta u svojstva konstruktor funkcije, povezuje s „this“ i vraća novi objekt (this). Ovaj način se razlikuje od prethodnih jer je sličan konstruktoru koji se inače koristi u nasljeđivanju.

Općenito, svrha funkcija je kreirati logičko rješenje koje se može primijeniti na više objekata s dinamičkim informacijama koje se prosljeđuju kao parametri.

```
function Osoba(ime, prezime){
    this.ime = ime,
    this.prezime = prezime,
    this.predmet ='PWKS',

    this.info = function(){
        console.log(`${this.ime} ${this.prezime}
        dobrodošli na ${this.predmet}`)
    }

}

const studnet = new Osoba("Antonija", "Barić")
studnet.info()
```

Slika 3. Kreiranje objekta preko konstruktor funkcije

Izrazito poznati programer Eric Elliot 2016. godine napisao je članak i iznio svoje stajalište o kreiranja objekta na različite načine.

4. Class constructors

Koristi ključnu riječ „class“ i „new“ što je najbliže objektno-orijentiranoj sintaksi. Gotovo je identična konstruktor funkciji, što se može vidjeti u primjeru na slici 4. Postavlja se pitanje što dobivamo koristeći ovaj način kreiranja objekta. Koje su to „velike“ razlike koje drugi načini nisu postigli? Zapravo ih nema. Isto je, samo ćete možda jednim načinom brže doći do rješenja ili nekome više odgovara funkcionalna povrh objektno-orijentirane paradigme. Stvar je izbora. Iz tog razloga uvođenje ključne riječi „class“ kao neke novosti nazivamo „syntactic sugar“.

```
class Osoba {
    constructor(ime, prezime) {
        this.ime = ime,
            this.prezime = prezime,
            this.predmet = 'PWKS'
    }
    info = function () {
        console.log(`${this.ime} ${this.prezime}
        dobrodošli na ${this.predmet}`)
    }
}

const student = new Osoba('Antonija', 'Barić')

console.log(student.ime)
student.info()
```

Slika 4. Kreiranje objekta preko klase

Izdvojena stajališta programera o uvođenju klase:

- *„Konceptualno, klasa u JS ne postoji. Sve se postiže funkcijama i prototipiranjem!“*
- *„Pozitivna promjena uvođenja svojstva class je jednostavnije sintaksa i nenametljivost kod izrade objekta.“*
- *„Udaljava programere od funkcionalne paradigme koja je korisnija.“*
- *„Dakle, riječ class je samo syntactic sugar. Drugim riječima, konceptualno class daje identičan kôd i jedina je razlika čitljivost.“*
- *„Forsiranje JS da izgleda kao drugi jezik može se činiti poželjno s naivnog gledišta ali neće pomoći u dugoročnom održavanju.“*

Navedena stajališta i rasprave možete pročitati na poveznicama stavljenim pod Literatura.

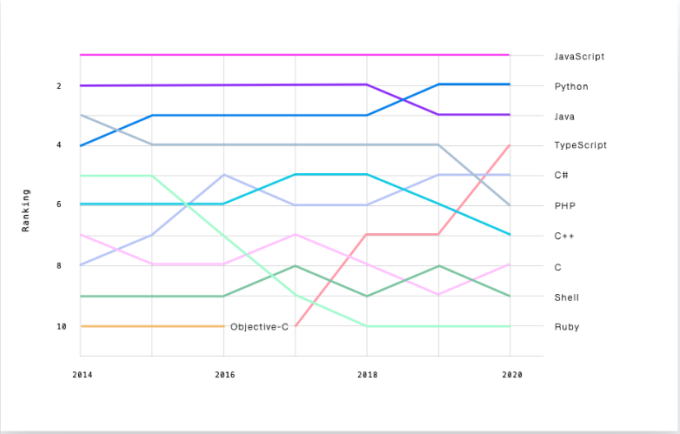
Ako dobro pogledate primjere, vidjet ćete da su izvedena na sličan način pa se postavlja pitanje čemu uvođenje novog svojstva „class“ u standardu ES6, osim što asocira na klasu i sve njezine funkcionalnosti, ali nema ono što objektno -orijentirani jezici imaju već simulira razinu prava atributa ili metoda, nasljeđivanja itd.

Možemo zaključiti da svaki od navedenih načina ima svojih prednosti i mana, a na programeru je da odabere svoj stil. Čitajući članke i gledajući YouTube videa, nema najboljeg načina iako većina kreira objekt konstruktor funkcijom.

Važno je napomenuti kod izrade aplikacija gotovo se uvijek koristi neko razvojno sučelje za frontend i backend kao npr. React, Node.js, Angular, Vue.js. Prije React-a 2011. godine programeri su razvijali i gradili korisnička sučelja isključivo pomoću JavaScript-a. To je označavalo dulje vrijeme razvoja i više mogućnosti za različite pogreške. Kako bi se izbjegla složenost interakcije i potreba za naknadnom obradom, React u potpunosti pruža potpunu obradu aplikacije.

U većini literature će vas upozoriti kako treba biti oprezan kod korištenja novih značajki standarda ES6 jer neki mrežni preglednici ili aplikacije ne podržavaju nove standarde, ali ako koristite razvojno sučelje napisan kôd se prevodi u ranije verzije JavaScripta koje podržavaju većina mrežnih preglednika. Uzmimo za primjer razvojno sučelje React koji koristi Babel (compiler) za prevođenje programskog koda napisanog modernim standardom u verziju koja je kompatibilna s većinom mrežnih preglednika.

Bez obzira na izmjene i rasprave, JavaScript je jedan od najkorištenijih programskih jezika, posebno u svijetu web tehnologija. Postoji 1.8 bilijun mrežnih stranica na svijetu i 95% stranica koristi JavaScript. Možemo se složiti kako je izrazito popularan, dinamičan jezik koji se odlično prilagođava današnjim standardima.



Slika 5. JavaScript je daleko najkorišteniji jezik prema Github-ovom Octoverse Reportu za 2020.

Zadatak

Kreirati datoteku *get_set.js*.

Instancirati dva objekta koristeći konstruktor funkciju koja će opisivati Auto s atributima i metodama po želji. Potrudite se da vaše metode imaju neku svrhu osim tekstualnog ispisa. U svoje metode uključite atribute kako bi bolje vidjeli primjenu privatnih atributa npr. izračunajte potrošnju goriva.

Važno je imati barem **jedan privatan atribut** i **jednu privatnu metodu** koje ćete dohvatiti u globalnom scopeu (get, set ili Object.definepropetry()).

Jednom atributu postavite default vrijednost koju ne trebate prosljeđivati kroz konstruktor.

Nakon što napravite objekt preko konstruktor funkcije, svoj kôd iskoristite i izradite još jedan objekt, ali preko factory funkcije ili klase. Nemojte izostaviti privatne atribute i metode.

Literatura:

[1] file:///C:/Users/Korisnik/Downloads/ziterbart_bruno_pmfst_2017_diplo_sveuc%20(1).pdf

[2] https://rajaraodv.medium.com/is-class-in-es6-the-new-bad-part-6c4e6fe1ee65

[3] https://rajaraodv.medium.com/is-class-in-es6-the-new-bad-part-6c4e6fe1ee65

[4] https://everyday.codes/javascript/please-stop-using-classes-in-javascript/

[5] https://generalassemb.ly/blog/what-makes-javascript-so-popular/

[6] https://medium.com/swlh/understanding-inheritance-in-javascript-through-object-creation-df2f9e891a09

[7] https://www.youtube.com/watch?v=fbuyliXlDGI&t=1243s

[8] https://octoverse.github.com/2022/top-programming-languages

[9] https://bootcamp.berkeley.edu/blog/most-in-demand-programming-languages/

[10] https://repozitorij.foi.unizg.hr/islandora/object/foi%3A5903/datastream/PDF/view