Web programiranje – Vežbe 8
JS prototype i događaji

Primer – web shop

HTML stranica **login.html** sadrži formu za prijavu korisnika. Klikom na dugme "Submit", podaci iz forme se šalju na stranicu **web_shop.html** koja sadrži listu proizvoda.

Dinamičke web stranice se oslanjaju na **događaje** kako bi odreagovali na akcije klijenta, kao što su klik miša ili unos sa tastature. Funkcije koje reaguju na događaje su **obrađivači događaja (event handlers)**. Određenu grupu predefinisanih događaja je moguće dodati kroz HTML atribute (vežbe 7). Problemi kod ovog pristupa su: 1) ne pokriva sve vrste događaja, 2) samo jedan obrađivač je dozvoljen po događaju, 3) nije omogućeno (ili je dosta otežano) dinamičko dodavanje obrađivača događaja. Sve događaje, uključujući i korisnički definisane događaje, je moguće dodavati dinamički kroz kod. Registrovanje obrađivača događaja se vrši metodom **addEventListener(eventName, eventHandler).** Prvi parametar je običan string sa imenom događaja, dok je drugi parametar JavaScript funkcija. Za početak je potrebno u fajl **login.js** dodati obrađivač događaja učitavanja stranice iz koda ispod:

```
//Ubaciti objekte za korisnike ovde. Ključ je korisničko ime.
//Korisnici imaju ime, prezime i lozinku.
var usersMap = {
};
window.addEventListener('load', function(){
	let form = document.forms[0];
	// Dodaj obrađivač događaja slanja sa forme.
	form.addEventListener('submit', function(event) {
		let valid = false;
		// Zameniti pravom proverom
		if (valid) {
			return;
		}
		// Obustavlja podrazumevano ponašanje događaja, što je prelazak na
		// drugu stranicu.
		event.preventDefault();
	});
});
```

Sav kod koji obrađuje DOM stablo se mora nalaziti u ovoj funkciji, jer **DOM objekti postoje tek kada se učita stranica**.

U ovom kodu se takođe registruje obrađivač događaja slanja sa forme (**'submit'**). Prvi parametar funkcije obrađivača je **Event** objekat koji predstavlja sam događaj. Obrađivači obrađuju događaj u obrnutnom redosledu u kojem su registrovani, počevši od poslednje registrovanog. Pošto je naš obrađivač događaja poslednji, on će se prvi izvršiti, a zatim podrazumevani obrađivač koji prelazi na sledeću stranicu. Da bismo sprečili da se pozove podrazumevani obrađivač u slučaju nevalidnog unosa, poziva se **event.preventDefault()** metoda koja prekida lanac izvršavanja. Događaji se takođe obrađuju i po hijerarhiji: prvo se pozivaju obrađivači na objektu koji je okinuo događaj, pa obrađivači

njegovih roditelja itd. Ovo se zove event bubbling. Zbog ovoga bi bilo moguće registrovati 'submit' obrađivač na window objekat koji bi reagovao na slanje forme. Voditi računa o tome.

DODATNO: U JavaScriptu je takođe moguće kreirati korisnički definisane događaje. Više informacija ovde. Ovo nije obavezno za predmet Web programiranje.

JavaScript ima podršku za objekto orijentisano programiranje (OOP). Međutim, OOP u JavaScriptu nije baziran na konceptu klasa, već na prototipovima. ES6 je uveo velike promene u definisanju klasa, tako da liči na nešto iz objektnih programskih jezika. Ovo je sve sintaksni šećer i JavaScript se u pozadini i dalje bazira na prototipovima. Na ovim vežbama ćemo pokriti stari stil definisanja klasa. Informacije o novom stilu su dostupne ovde.

Klase se definišu u nekoliko koraka. Prvi korak je definisanje konstruktora za klasu. Konstruktor je obična JavaScript funkcija koja postavlja polja klase. Kod ispod definiše klasu Korisnik i treba ga dodati u fajl **proto.js**.

```
function Korisnik(ime, prezime, lozinka){
	this.ime = ime;
	this.prezime = prezime;
	this.lozinka = lozinka;
};
```

U konstruktoru je moguće dodati metode, ali nije preporučljivo jer će se onda za svaki objekat formirati posebna referenca na metodu. Rezultat ovoga je dvojak: 1) više memorije se troši jer svaki objekat ima posebnu metodu, 2) nemoguće je promeniti ili naslediti metodu. Da bi se ovo izbeglo, metode se kače na **prototype** polje klase. Polja koja se kače na **prototype** se ponašaju kao statička polja u Javi, tj. svi objekti jedne klase imaju istu referencu na to polje. Kod ispod definiše metode **ispis** i **stampaj** klase Korisnik:

```
Korisnik.prototype.ispis = function(tekst){
	var p = document.querySelector('#poruka');
	p.innerText = tekst;
};

Korisnik.prototype.stampaj = function(){
	// call metoda funkcije kao prvi parametar prima objekat koji će
	// biti referenca 'this' unutar funkcije. Npr. ako se kao prvi
	// argument prosledi {ime: 'pera'},  unutar funkcije ispis 'this'
	// će biti referenca na {ime: 'pera'}
	Korisnik.prototype.ispis.call(this, 'Hello ' + this.ime + '!');
};
```

- ispis: u paragraf #poruka upisuje prosledjenu poruku
- stampaj: prosledjuje funkciji ispisu Hello poruku sa imenom logovanog korisnika

Dodati ovaj kod u fajl **proto.js**.

Nasleđivanje u JavaScriptu se u velikoj meri oslanja na prototype. Za početak u fajl **proto.js** dodati definiciju klase MaloprodajniKorisnik:

```
function MaloprodajniKorisnik(ime, prezime){
	Korisnik.call(this,ime,prezime);
}
```

Metoda **Korisnik.call** poziva konstruktor klase **Korisnik** i može se posmatrati kao **super konstruktor** iz Jave. Ako kao prvi argument prosledimo **this**, konstruktor Korisnika će mu dodeliti sva svoja polja. Međutim, ovim nismo nasledili metode klase Korisnik. Da bismo i to postigli, potrebno je za prototip klase MaloprodajniKorisnik postaviti kopiju prototipa Korisnik (**proto.js** fajl):

```
MaloprodajniKorisnik.prototype = Object.create(Korisnik.prototype);
```

Zatim za konstruktor klase MaloprodajniKorisnik postavljamo prethodno definisanu funkciju MaloprodajniKorisnik:

```
MaloprodajniKorisnik.prototype.constructor = MaloprodajniKorisnik;
```

Redefinicija (override) metode se vrši tako što se jednostavno redefiniše vrednost polja iz prototipa. Kod ispod redefiniše metodu **ispis**:

```
MaloprodajniKorisnik.prototype.ispis = function(input){
	let vrednost = 'TODO';
	let poruka = 'Cena narudžbine je: ' + vrednost;
	Korisnik.prototype.ispis.call(this, poruka);
}
```

Zadaci

1. Definisati klasu **VeleprodajniKorisnik** koja nasleđuje klasu **Korisnik**.
   Na stranici **login.html** omogućiti logovanje dva korisnika (lozinke proizvoljne):
   - Pera Perić
   - Mika Mikić

   Ukoliko podaci prilikom logovanja nisu ispravni, ispod forme za logovanje (paragraf #poruka) ispisati crvenim slovima: "Neispravno korisničko ime i/ili lozinka"
2. Ukoliko u URLu na stranici **webshop.html** ne postoje podaci o korisniku, prebaciti se **login.html**. Prilikom učitavanja stranice preuzeti korisnika iz URLa i u zavisnosti od logovanog korisnika, kreirati objekat korisnik tipa:
   - MaloprodajniKorisnik ukoliko se loguje Pera
   - VeleprodajniKorisnik ukoliko se loguje Mika
3. Na stranici **webshop.html** input poljima je **onchange** podešen na **ispis** metodu ulogovanog korisnika. Definisati metodu **ispis** tako da:
   - Ispisuje cenu narudžbine ukoliko je MaloprodajniKorisnik
   - Ispisuje cenu narudžbine sa popustom od 15% ukoliko je reč o VeleprodajniKorisnik