Autori: © Divna Krpan, Saša Mladenović, Goran Zaharija

OOP Vježbe 06
Preopterećene metode

—

Bilješke

Polimorfizam

Dva načina na koje možemo ostvariti polimorfizam:

- Premošćivanje (eng. override)
- Preopterećenje (eng. overload)

Preopterećenje metoda u C#-u se ostvaruje kad imamo dvije metode s istim nazivom, ali različitim "potpisima" (eng. signature), odnosno argumentima:

- Različit broj argumenata
- Različit poredak argumenata
- Različiti tipovi argumenata

Za razliku od C#-a, koncept preopterećenja nije podržan u JavaScript-u, ali ipak ga možemo ostvariti. Ideja je prema broju i/ili tipu argumenata izvršiti određenu metodu (akciju). Međutim, premda ćemo imati samo jednu metodu u klasi, prema vani će izgledati kao da imamo više metoda s istim nazivom.

Parametri funkcije su nazivi varijabli navedeni u definiciji funkcije. **Argumenti funkcije** su stvarne vrijednosti koje se šalju u funkciju (i primaju). Ako se funkcija poziva s manje argumenata od deklariranih, onda će oni koji nedostaju biti ***undefined***. Ponekad takva situacija odgovara, a ponekad ne. Najjednostavnije situacije možemo riješiti opcionalnim parametrima.

Preopterećenje metode u JavaScript-u

Koristit ćemo objekt ***arguments*** kako bi dobili informacije o argumentima koji su poslani u metodu:

- `arguments.length` ⇒ broj argumenata
- `typeof arguments[0]` ⇒ tip prvog argumenta

Pomoću indeksa možemo pristupiti bilo kojem argumentu. U sljedećem primjeru navedeni su parametri *a, b, c*, ali ne moraju biti navedeni da bi funkcija mogla primiti argumente.

```
function func1(a, b, c) {
  console.log(arguments[0]);
  // očekivani ispis: 1

  console.log(arguments[1]);
  // očekivani ispis: 2

  console.log(arguments[2]);
  // očekivani ispis: 3
}

func1(1, 2, 3);
```

Ključnu riječ ***typeof*** možemo koristiti kako bi dobili tip argumenta te na temelju toga odredili što želimo izvršiti. Međutim, to nam ne pomaže ako želimo razlikovati tip objekata:

```
class Sprite {

}

class Animal extends Sprite {

}

let s = new Sprite();
let cat = new Animal();

console.log(typeof s);
console.log(typeof cat);
```

Ispis:

```
object
object
```

Iz ispisa ne možemo razlikovati o kojem objektu se radi. Ako bi nam to u nekom trenutku trebalo možemo koristiti:

```
console.log(s.constructor.name);
console.log(cat.constructor.name);
```

Ispis:

```
Sprite
Animal
```

To bi mogli koristiti ako bi npr. htjeli da se lik ponaša drugačije ovisno o tome je li dotakao drugi objekt tipa *Sprite* ili objekt tipa *Animal*.

Object.constructor vraća referencu na konstruktor klase, a ***name*** je svojstvo koje nam vraća naziv klase.

Primjer za preopterećenje prema broju argumenata:

```
// overloading - preopterecenje sa switch case
function suma() {
  switch (arguments.length) {
    case 0:
      console.log("Niste prenijeli niti jedan argument");
      break;
    case 1:
      console.log("Prenesite barem dva argumenta");
      break;
    default:
      let result = 0;
      let length = arguments.length;

      for (let i = 0; i < length; i++) {
        result = result + arguments[i];
      }
      console.log(result);
      break;
  }
}
```

Ako pozovemo:

```
suma();
suma(1);
suma(1, 2);
suma(4, 2, 1);
```

Dobit ćemo:

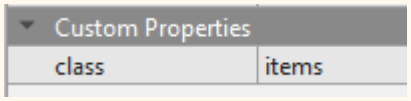
```
Niste prenijeli niti jedan argument
Prenesite barem dva argumenta
3
7
```

Primjena na projektu

U okviru je definirana klasa ***Item*** koja je namijenjena za likove koji nemaju animacije (samo je jedna sličica). Ako pogledati definiciju funkcije vidjet ćete da je za sve položaje lika postavljen broj 1. Klasa ***Item*** nasljeđuje od klase ***Sprite***.

Klasa *Item* se također koristi za prikaz objekata postavljenih u sloj (eng. **layer**) tipa ***object layer***.

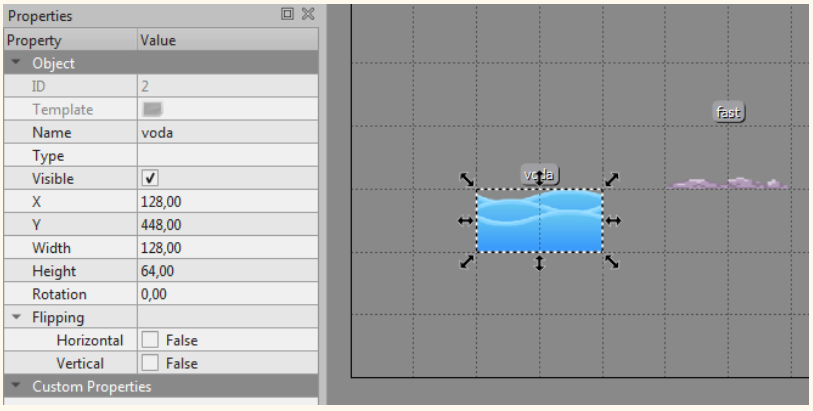
Svakako treba staviti *custom property* kako bi okvir "znao" da se radi o sprite-ovima.



Za razliku od ***tile layer***-a u koje spremamo sve sličice koje se odnose na lika koji će se crtati, objekti postavljeni u *object layer*:

- imaju samo jednu sličicu,
- bit će postavljeni na konačnu poziciju pomoću *Tiled-a*
- imat će dimenzije već postavljene u *Tiled-u*

Prema tome, za instanciranje objekata tog tipa nije potrebno u samom programu unositi koordinate niti dimenzije lika. Svaki objekt bi trebao imati i ime, kako bi mu lakše pristupili. Ako mu ne zadamo ime, onda će nam okvir u konzoli ispisati brojeve kao ID te ćemo morati dohvaćati objekte pomoću tih brojeva. Primjer:



Obzirom da se *object layer* prilično razlikuje od *tile layer-a*, a *Item* nasljeđuje od *Sprite-a*, onda će zapravo okvir obaviti konverziju tako da ćemo za svaki objekt dobiti novi *tile layer* s nazivom samog objekta. Ovo zvuči kompliciranо, ali u praksi znači sljedeće:

- Ne dohvaćamo *layer* tipa *object layer* već u metodi ***getSpriteLayer()*** koristimo nazive objekta.

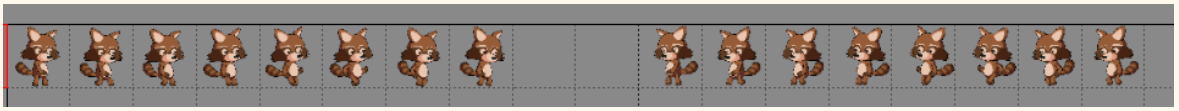
Npr., kako bi dohvatili *layer* koji pripada *vodi* prikazanoj na gornjoj slici, pišemo:

```
let layer = GAME.getSpriteLayer("voda");
```

Od prošlih vježbi imamo klasu ***Animal*** koja nasljeđuje od klase *Sprite*. Klasa *Animal* ima:

- Konstruktor koji postavlja ***frame_sets*** ⇒ sličice za animacije.
- Metodu ***touching(s)*** koja vraća *true* ako lik dira sprite *s* ili *false* ako ga ne dira.
- Metodu ***jump()*** za skakanje koju smo premjestili iz klase *Cat* uz zadanu vrijednost 50.

Pripremljena je nova mapa s novom životinjom te će biti potrebno napisati klasu ***Racoon*** koja nasljeđuje od *Animal*.



➢ Zadatak 1.
- Napraviti objekt tipa *Racoon* i postaviti ga na koordinate (0, 0).
- Moramo riješiti sljedeći problem:
  - Kad lik naiđe na posebnu podlogu, treba se kretati brže ili sporije.
  - Za tu namjenu iskoristit ćemo koncept preopterećenja.
- Napisati preopterećenu metodu ***moveRight()*** koja radi na sljedeći način:
  - Ako nema argumenata, pozvat će metodu osnovne klase za pomak desno
  - U protivnom, prima samo brzinu v
    - Postavlja smjer na 90
    - Povećava brzinu po x za v
- Dodati dva nova objekta tipa ***Item***: voda i fast.
- Pogledajte u mapi kako su postavljeni i kako se zovu.
- Logika igre
  - Kad lik ide desno, ako dotakne brzu podlogu, pozvati metodu moveRight(5)
  - U protivnom, pozvati metodu moveRight()
  - Lijevo ide normalno
  - Ako se pritisne strelica gore skače

Sad imamo novi problem, a to je da za razliku od mačaka rakuni ne mrze vodu. Također, znaju plivati pa će moći prijeći preko vode na mapi, ali u vodi nije moguće skakati kao što skaču na čvrstoj podlozi.

➢ Zadatak 2.

- Napišite preopterećenu metodu ***jump()*** koja radi na sljedeći način:
  - Ako nema argument, skače normalno (kao u klasi *Animal*)
  - Ako primi argument tipa *number*, onda taj broj predstavlja visinu koliko će skočiti.
  - Ako primi argument tipa *string* i ako je to voda, onda će ispisati poruku na konzolu "Ne mogu skakati" i neće skočiti.
  - U protivnom skače normalno

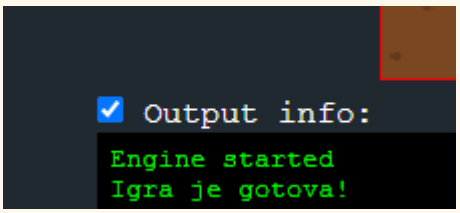
➢ Zadatak 3.

- Ako lik dira gljivu koja je na dnu mape, onda može skočiti jako visoko tako da dosegne platformu koja stoji u zraku.
- Uputa: bit će dovoljno 100 točkica.
- Kako će znati je li dotaknuo gljivu?
  - Pazite, gljiva je trenutno u sloju s ukrasima, što znači da nemamo način za otkrivanje pozicije i dodira.
  - Obzirom da gljiva ima samo jednu sličicu, prebacit ćemo je odnosno nacrtati kao objekt u sloju *items* tipa *object layer*.
  - Napraviti export.
  - Zatim ćemo je instancirati u dijelu *postavke* te ispitati dodiruje li je rakun u dijelu *logika igre*.
  - U implementaciji metode *jump()* ćemo dodati uvjet za gljivu.

➢ Zadatak 4.

- Ako lik dođe do strelice, neka se ispiše poruka: "Igra je gotova", ali na prostor ispod mape namijenjen ispisu.
- Upute: opet će biti potrebno prebaciti strelicu u odgovarajući sloj mape.

Za ispis:



Koristimo naredbu:

```
GameSettings.output("Igra je gotova!");
```

U gornjoj liniji možemo primijetiti da smo pozvali **metodu klase**, a ne objekt!

Prisjetimo se kako smo u C#-u pozivali:

```
Console.WriteLine("pozdrav");
```

Za razliku od:

```
Random g = new Random();
```

```
int br = g.Next(10, 100);
```

Console je klasa, a *WriteLine()* metoda klase. *Random* je klasa, ***g*** je objekt klase *Random*, a *.Next()* je metoda koju pozivamo uz objekt ***g***.

Potražite na webu što su to statičke klase i statički članovi (eng. members) klase.

_