Autori: © Divna Krpan, Saša Mladenović, Goran Zaharija

OOP Vježbe 07
Statičke klase

—

Bilješke

Uvod

Ključna razlika između nestatičkih i statičkih klasa je u tome što se statičke klase ne mogu instancirati. U C#-u se statičke klase definiraju pomoću ključne riječi ***static*** što znači da svi članovi klase moraju biti statički. Ako se klasa ne deklarira kao statička, onda se pojedini članovi takve klase ipak mogu deklarirati kao statički.

U JavaScript-u za statičku klasu možemo definirati samo članove kao statičke, ali time nećemo spriječiti instanciranje te klase. Kako bi riješili taj problem, potrebno je koristiti "trik":

```
class StatickaKlasa {
 constructor() {

   if (this instanceof StatickaKlasa) {
     throw "Statička klasa se ne smije instancirati!";
   }

 }

 static metoda() { }

}
```

Prilikom izvršavanja ovog kôda ispisuje se:

```
▸Uncaught Statička klasa se ne smije instancirati!
```

Ako zaboravimo iskoristiti taj način spriječavanja instanciranja objekata klase, onda se greška može pojaviti kasnije kad pokušamo koristiti članove klase koji su statički:

```
class Auto {
 constructor() {
   // ako ne stavimo throw
 }
 static hello() {
   return "Hello!!";
 }
}

let myCar = new Auto();

console.log(Auto.hello());
console.log(myCar.hello()); //greška!
```

Statički članovi klase (metode, svojstva, polja) koriste se kad nije potrebno stvarati instancu klase da bi taj dio kôda mogao raditi.

Nestatički članovi klase korisni su za čuvanje stanja danog objekta koje se mora odvojiti od ostalih objekata klase.

Na primjer, ako imamo klasu ***Player*** u igri koja ima dvije instance (dva objekta tipa *Player*): *igrac1* i *igrac2*, ima smisla spremiti informacije o ostvarenim bodovima posebno za svakog od njih:

```
class Player {
  constructor(imeIgraca){
    this.bodovi = 0;
    this.ime = imeIgraca;
  }
}

let prvi = new Player("Player 1");
let drugi = new Player("Player 2");

console.log(prvi.bodovi);
console.log(drugi.bodovi);
```

Metode koje pripadaju instanci klase omogućuju korištenje polimorfizma, dok kod statičkih klasa to nije moguće. Statička klasa nam često služi kao spremnik za skup povezanih metoda i eventualno konstanti u slučajevima kada nema potrebe za internim podacima već se sve operacije obavljaju nad ulaznim parametrima poslanima u metodu.

Jedan od primjera statičke klase u C#-u je klasa ***Math*** koja sadrži skup metoda koje izvršavaju matematičke operacije te dvije konstante (*Math.E i Math.PI*).

Kad ćemo primijeniti statičke klase?

Na primjer:

- Možemo ih primijeniti kad želimo koristiti ili dijeliti istu metodu ili podatak između više različitih instanci klase (npr. polje u kojem je spremljena informacija je li igra gotova, granice svijeta u kojem se lik kreće, zadane dimenzije nekih likova, ...).
- Za globalni pristup podacima (npr. ako želimo izbjeći globalne varijable i spremiti osnovne postavke ili eventualno likove koje koristimo na jedno mjesto).
- Ako nam trebaju pomoćne funkcije koje ćemo pozivati u projektu.

Statički elementi se nešto sporije izvršavaju.

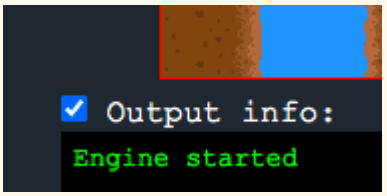
Statička klasa GameSettings

Primjer metoda:

- `colorLog(message, color)` ⇒ Ispis na konzolu u bojama.
  - success, info, error, warning
- `static output(text, reset)` ⇒ Ispis na prostor ispod canvas-a.
  - *text* - predstavlja tekst kojeg želimo ispisati
  - *reset* - ako se postavi na *true* onda će pobrisati prethodno stanje, a ako se ne napiše ili postavi na *false*, onda samo nastavlja ispis u sljedećoj liniji.

Primjeri:

```
vjezbe6
Učitan sprite layer racoon
Učitan sprite layer dino
catDog
Učitan sprite layer macka
Učitan sprite layer pas
```



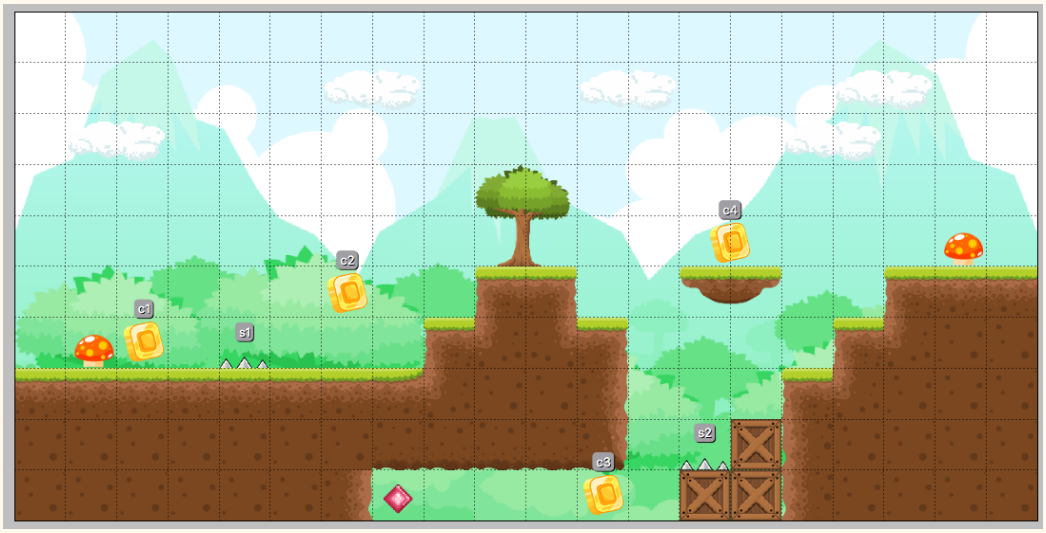
Zadaci

Zadatak koji je pripremljen na elearningu ima novu mapu koja se nalazi u *Platforma3*.
Dodani su novi likovi:

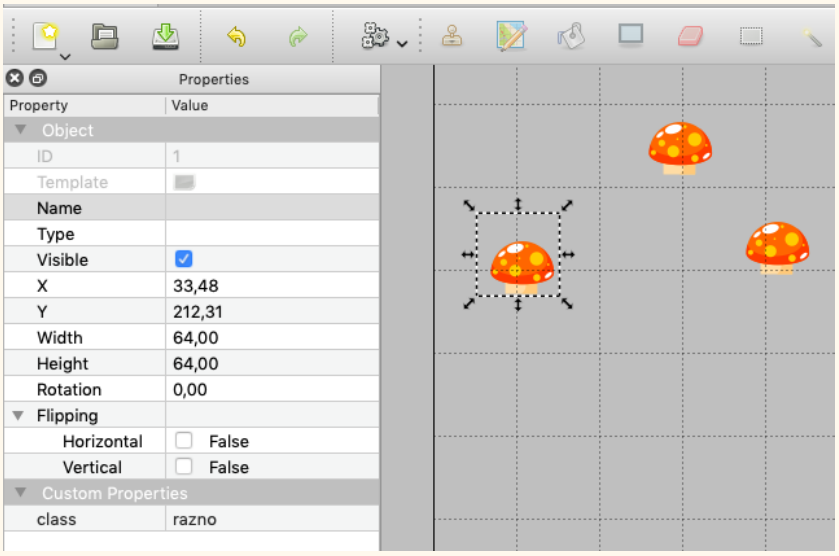
- Coins
- Spikes
- Dinosaur (glavni lik u igri)



Novi likovi: novčići (*coins*) i šiljci (*spikes*) prikazani na slikama predstavljaju objekte koji će se skupljati za vrijeme igre.



Spomenuti likovi su u Tiled-u nacrtani na sloju mape koji je ***Object layer***. Pri tome su odmah postavljeni na pozicije na kojima bi trebali biti za vrijeme igre (ne moramo kasnije razmišljati gdje ih točno želimo). Također, postavljena im je i veličina. Na slici vidimo kako se ti objekti ne moraju postaviti točno prema mreži. Dodatno, dajemo im imena (***name***) kako bi ih kasnije lakše mogli pozvati. Ako im sami odmah ne zadamo ime, onda ćemo ih morati pozivati po ID broju:



Želimo napraviti igru u kojoj glavni lik (dinosaur) mora skupljati novčiće i izbjegavati šiljke:

- Svaki novčić u ovoj igri vrijedi 10 bodova.
- Dinosaur mora brojati koliko bodova je skupio.
- Dinosaur započinje igru sa 100% zdravlja.
- Svaki šiljak oduzima 50% zdravlja.

Cilj igre:

- Igra završava pobjedom kad lik skupi sve novčiće.
- Igra završava porazom ako lik izgubi zdravlje.

Prisjetimo se situacije od prošlih vježbi u kojima smo imali više dodatnih likova vrste ***Items***. Koristili smo brzu podlogu, vodu, gljivu i strelicu za cilj te smo vidjeli kako smo morali pisati isti kôd za svakog od njih:

- Napravi novi objekt vrste *Item*,
- Postavi vidljivost,
- Dodaj ga u popis svih likova.

Za 4 objekta tipa *Item* smo ponavljali te iste akcije 4 puta. Obzirom da ćemo ih u ovim vježbama imati 6, postaje sve jasnije kako dupliciramo kôd za objekte istog tipa. Kako bi skratili program, koristit ćemo niz (tome niz i služi). Zaključujemo kako se samo naziv *layer-a* mijenja pa ćemo ih staviti u niz:

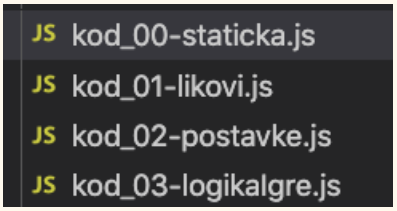
```
let nazivi = ["fast", "voda", "gljiva", "cilj"];
```

Dalje koristimo petlju ***for***:

```
  for (let i = 0; i < nazivi.length; i++) {
    const n = nazivi[i];
    let item = new Item(GAME.getSpriteLayer(n));
    item.visible = true;
    GAME.addSprite(item);
  }
```

Pored činjenice da nam je sad program znatno kraći, ako bude dodatnih izmjena, napravit ćemo ih samo na jednom mjestu.

➢ Zadatak 1.
- Napišite statičku klasu: ***Postavke***.
- Spremite klasu u novu datoteku koju ćete dodati u projekt kako smo već naučili. Npr. *kod_00-staticka.js* spremamo u ***jsKod*** (koristimo taj naziv datoteke kako bi nam u VS Code prikaz datoteka bio pregledniji i poklapao se sa stvarnim redoslijedom učitavanja datoteka).
- Datoteka s novom klasom se mora učitati prije svih ostalih jer će je ostale klase koristiti.



- Statička klasa *Postavke* ima polja:
  - ***coins*** ⇒ predstavlja niz u koji ćemo spremiti sve novčiće
  - ***spikes*** ⇒ niz u koji ćemo spremiti šiljke
  - ***dinosaur*** ⇒ polje u koje ćemo spremiti glavnog lika
  - ***cilj*** ⇒ ukupan broj bodova koje treba ostvariti za pobjedu

➢ Zadatak 2.

- Zadana je klasa ***Dinosaur*** koja nasljeđuje od *Animal*.
- **Napomena!** Metode *jump()* i *touching()* su prebačene u klasu *Sprite*.
- Dodati polja:
  - ***health*** s početnom vrijednosti: **100**.
  - ***points*** s početnom vrijednosti: **0**.
- Lik dovoljno skače u visinu, ali mu je skok prekratak po osi x, tako da neće nikako moći preskočiti šiljke.
- Napraviti premošćivanje metode ***moveRight()*** koja će omogućiti veći skok.
  - Zadana brzina u klasi *Sprite* je 0.5.
  - Neka lik tipa *Dinosaur* ima ubrzanje 1.5.

➢ Zadatak 3.

- Morat ćemo koristiti predmete koji nemaju animacije ⇒ klasa *Item*.
- Problem? Objekti koje skupljamo imaju više infromacija od klase *Item*.
- Napisati klasu *Collectable* koja nasljeđuje od klase *Item*.
  - **Napomena:** Nemojte mijenjati postojeće klase okvira!

➢ Zadatak 4.

- U postavkama: ***setupVjezbe7()***:
- Zadani su nizovi: ***c*** i ***s*** koji sadrže nazive novih likova.
- Učitati sve novčiće i šiljke u igru kao objekte tipa *Collectable*.
- Spremiti dinosaura u polje statičke klase.
- Kolizije su za sad isključene:

```
//GAME.activeWorldMap.setCollisions("platforme");
```

- Pokrenite igru sa svim likovima.
- **Što se dogodilo?** Zašto likovi padaju na dno?

Metoda ***updatePosition()*** nalazi se u klasi ***Sprite***, a brine se o osvježavanju pozicije lika. Zadano ponašanje lika u našim platformskim tipovima igara je padanje ovisno o gravitaciji. Obzirom da klasa *Item* nasljeđuje od klase *Sprite*, svi objekti vrste *Item* (i budućih podklasa)

imat će istu metodu. Premda se objekti tipa *Item* mogu pomicati kao svi ostali likovi (samo nemaju animacije), u ovoj igri nam ne odgovara njihovo pomicanje niti padanje zbog gravitacije. Želimo ih zadržati na mjestima gdje smo ih postavili. Prošli put to nismo primijetili jer su likovi bili postavljeni izravno na platforme pa nisu ni mogli pasti na dno. M

Potrebno je napraviti premošćivanje (*override*) metode *updatePosition()*:

```
class Collectable extends Item {
  constructor(layer) {
    super(layer);
  }

  updatePosition() {
    //ne kreću se!
  }

}
```

Metoda *updatePosition()* koja pripada klasi *Collectable* ne radi ništa pa se likovi neće moći pomaknuti.

- Dodajte kretanje dinosaura koristeći novu statičku klasu u kojoj je spremljen dinosaur.

Nakon što smo riješili problem padanja likova, razmislimo malo detaljnije o našim predmetima koje lik skuplja ili dodiruje:

- skupljanje novčića donosi bodove, a
- dodirivanje šiljka smanjuje zdravlje.

Premda se radi o minimalnim razlikama (samo jedno polje), ipak ćemo samo radi vježbanja sintakse definirati dvije nove klase:

- Klasa ***Coin***:
  - Sadrži polje ***value*** sa zadanom vrijednosti 10.
- Klasa ***Spike***:
  - Sadrži polje ***damage*** sa zadanom vrijednosti 50.

Umjesto stvaranja objekata tipa *Collectable* promijenite u vašem rješenju na odgovarajućim mjestima u *Coin* odnosno *Spike*.

Napomena: ne zaboravite dodati sve likove tipa *Coin* u ***Postavke.coins*** te sve likove tipa ***Spike*** u ***Postavke.spikes***.

Prema trenutnim zahtjevima koje smo postavili na igru, možemo zaključiti kako *dinosaur* ne smije dotaknuti oba šiljka.

➢ Zadatak 5.

- Ako dinosaur dotakne novčić:
  - Novčić će nestati.
  - Dinosauru se uvećavaju bodovi za vrijednost novčića.
- Ispisati trenutne bodove pomoću: `GameSettings.output()`.

Kako ćete provjeriti dodiruje li sve likove neke vrste? Koristit ćete nizove *Postavke.coins* i *Postavke.spikes* i petlju. Metoda *touching()* vraća *false* ako je lik kojeg provjeravamo nevidljiv.

➢ Zadatak 6.

- Ako dinosaur dotakne šiljak:
  - Gubi zdravlje, ovisno o šteti koju nanosi šiljak.
  - Svaki šiljak može djelovati samo jednom! (Ovo smo odlučili kako nam se ne bi dogodilo da lik ostane bez zdravlja prije nego što ga uspijemo odmaknuti.)
  - To znači da nakon gubitka zdravlja, lik može bez problema prolaziti preko tog istog šiljka.
  - Ali nije imun na drugi…
- Probajte ispisati trenutno zdravlje na običnu konzolu pomoću statičke klase *GameSettings* i metode ***colorLog()***.

```
GameSettings.colorLog("Zdravlje: " + Postavke.dinosaur.health, "error");
```

Dodatno:

- Ako je ostvaren cilj igre, zaustaviti sve.
- Možete pozvati funkciju: ***btnStop_click()***. To je metoda koja upravlja događajem klik na Stop.

➢ **Domaći rad ***

- Napravite premošćivanje i/ili preopterećenje metode ***touching***() za dinosaura tako da:
  - Na temelju tipa lika zaključuje što će napraviti (je li *Coin* ili *Spike*).
  - Ako je dotaknuti lik tipa *Coin*, onda ga skriva i povećava bodove dinosauru.
  - Ako je dotaknuti lik tipa *Spike*, onda oduzima zdravlje dinosauru.

* nije obavezno

Napomena:

- Ova igra nije primjer projekta!

Za razmišljanje…

- Novčići imaju istu vrijednost, što bi trebalo napraviti da igra radi ako su vrijednosti slučajne?
- Probajte više puta pokrenuti igru klikanjem na "Setup" (ne osvježavanjem).
    - Radi li sve kako treba? Probajte ispisati u konzolu polja klase Postavke nakon svakog pokretanja.
        - Dovoljno je samo napisati *Postavke.dinosaur* kako bi vidjeli trenutno stanje polja.
    - Jeste li možda nešto propustili napraviti?

Kod stvaranja novog dinosaura, svi podaci će se vratiti na početno stanje jer su podaci vezani za instancu, ali to neće biti slučaj s podacima spremljenima u polja statičke klase *Postavke*.

Prvo pokretanje (klik na Setup) i sva ostala pokretanja igre na isti način bi trebala imati ovakvo stanje za polje *coins*:

```
Postavke.coins
▶ (4) [Coin, Coin, Coin, Coin]
```

A ne ovakvo:

```
Postavke.coins
▶ (8) [Coin, Coin, Coin, Coin, Coin, Coin, Coin, Coin]
```

- Kako možemo doznati je li neki objekt tipa *Coin* ili *Spike*?
- Ako smo zaboravili u kojim bilješkama se to nalazi, možemo iskoristiti svojstvo iako nije najbolja ideja:

```
get type() {
  return "Coin";
}
```

Svojstvo *type* je prema tome *samo za čitanje* (eng. read-only) jer ima samo ***get*** te ne sprema podatak u polje. Ako promijenimo naziv klase, moramo promijeniti i konstantu "Coin".

Napišite svojstvo u klasi *Collectable* koje će vratiti naziv klase:

```
Postavke.coins[0].type
'Coin'
Postavke.spikes[0].type
'Spike'
```