Autori: © Divna Krpan, Saša Mladenović, Goran Zaharija

OOP Vježbe 09
Apstraktne klase i metode

—

Bilješke

Apstraktne klase i nasljeđivanje

Osnovna klasa (eng. base class) služi kao temelj za nasljeđivanje. Klasa ***Sprite*** sadrži sve element potrebne za prikaz likova (koordinate, dimenzije, slike, animacije, ...) i osnovne akcije (kretanje, skakanje, ...) te služi kao predložak za izradu izvedenih klasa. Kroz izvedene klase nadograđujemo taj predložak dodavanjem novih elemenata ili promjenom postojećih.

Postoje slučajevi kad želimo definirati klasu koja je toliko općenita i koju nikad nećemo instancirati već ćemo je samo koristiti kao temelj za izvedene klase. Takve klase označavamo kao ***apstraktne*** (eng. abstract) te je njihova glavna značajka da se za razliku od "običnih" klasa ne mogu instancirati. U C#-u apstraktne klase označavamo pomoću ključne riječi ***abstract***.

```
//primjer za C#
public abstract class Animal
{
  // članovi apstraktne klase
}
```

U JavaScript-u ne postoji ključna riječ kojom možemo posebno označiti apstraktnu klasu tako da ćemo jednom naredbom u konstruktoru spriječiti njeno instanciranje.

```
// JavaScript
class Animal {

  constructor() {
    if (this.constructor == Animal) {
      throw new Error("Apstraktna klasa Animal se ne može
instancirati");
    }
  }

}
```

Pogledajmo primjer apstraktne klase **Animal** s metodama:

```
class Animal {

  constructor() {
    if (this.constructor == Animal) {
      throw new Error("Apstraktna klasa Animal se ne može
instancirati");
    }
    this.x = 0;
  }

  say() {
    throw new Error("Metoda Say nije implementirana!");
  }

  walk() {
    this.x++;
    console.log("hodam...");
  }

}
```

Klasa **Animal** ima metode *say()* i *walk()*. Napisat ćemo klase **Cat, Dog** i **Mouse** koje nasljeđuju od klase Animal. Sve navedene životinje imaju neki način glasanja koje implementiramo ispisom odgovarajuće poruke na konzolu. Međutim, općenito, životinja nema što ispisati tako da ćemo za metodu *say()* programski zahtijevati specifičnu implementaciju. Programer ne mora implementirati metodu ako se neće koristiti.

Sve spomenute klase hodaju pomicanjem po osi X za jednu točku.

```
class Cat extends Animal {
  say() {
    console.log("Mijau");
  }
}

class Dog extends Animal {
  say() {
    console.log("Vau");
  }
}

class Mouse extends Animal {}
```

Ako pokušamo instancirati klasu Animal:

```
let a = new Animal();
```

Dogodit će se iznimka koju smo definirali u konstruktoru:

```
▸Uncaught Error: Apstraktna klasa Animal se ne može instancirati
```

Za metodu *walk()*:

```
let c = new Cat();
let d = new Dog();
let m = new Mouse();

c.walk();
d.walk();
m.walk();
```

Rezultat je:

```
hodam...
hodam...
hodam...
```

Za metodu *say()*:

```
c.say();
d.say();
m.say();
```

Rezultat je:

```
Mijau
Vau
▸Uncaught Error: Metoda Say nije implementirana!
    at Mouse.say (kod.js:12:11)
```

Apstraktna klasa može imati i **apstraktne metode**. Apstraktne metode nemaju implementaciju.

U primjeru klase *Animal*, apstraktna metoda je ***Say()*** jer nema implementaciju već se mora implementirati u klasi koja je nasljeđuje, ako je mislimo koristiti. Tako u primjeru klase *Mouse* dobivamo pogrešku jer nismo implementirali metodu.

Ne možemo izravno stvarati objekt ili instancu apstraktne klase. Koristimo je kako bi grupirali zajednička svojstva i akcije. Na primjer, ako bi razmišljali o osobama uključenima u rad studentske referade možemo najprije izdvojiti: studente, nastavnike i zaposlenike.

- Student ima: ime, prezime, adresu, email, godinu upisa, predmete koje upisuje, broj ECTS-ova, …
- Nastavnik ima: ime, prezime, adresu, kontakt email, godinu zaposlenja, popis predmeta koje predaje, …
- Zaposlenik ima: ime, prezime, adresu, adresu e-pošte, broj telefona u uredu, opis posla, …

Izdvojimo atribute s istim nazivima: *ime, prezime, adresa*. Međutim, *email, kontakt email, adresa e-pošte* se također odnose na istu stvar: *email*. Umjesto definiranja tih istih polja za svaku od tri moguće klase, možemo oblikovati klasu ***Osoba*** koja će sadržavati: *ime, prezime, email*. Student, nastavnik i zaposlenik će naslijediti te zajedničke atribute te ih proširiti sa svojim specifičnostima. Svi dijele iste nazive zajedničkih atributa.

- ➢ Zadatak 1.
  - Preuzmite projekt (“zip” datoteku) sa stranice kolegija.
  - U projektu je već definirana klasa ***Animal***. Napravite od nje apstraktnu klasu prema uputama.
  - Neka klasa ***Racoon*** nasljeđuje od *Animal*. Napravite novog lika tipa *Racoon*.
  - Napisati metodu *start()* koja glavnog lika vraća na početnu poziciju
  - Ako lik padne na dno, vraća se na početnu poziciju i gubi život
  - Napraviti override metode updatePosition()
- ➢ Zadatak 2.
  - U klasu *Racoon* dodajte polje ***bodovi*** i svojstvo ***zivoti***
  - Početno stanje bodova je 0
  - Početno stanje života je 3
  - Ako lik izgubi sve živote, igra završava
  - Možete pozvati *btnStop_click()*
- ➢ Zadatak 3.
  - Napraviti apstraktnu klasu od *Collectable*
  - Napisati klasu ***Coin*** koja nasljeđuje *Collectable* i ima:
    - Polje *value*
    - Početna vrijednost 10
  - Napisati metodu ***start()*** koja postavlja coin na slučajni x te y=0
- ➢ Zadatak 4.
  - Dodati metodu ***collect()*** u klasu *Racoon*
    - Skuplja bodove i vraća coin na početnu poziciju
  - Postaviti likove
  - Ako lik dira novčić, pokupiti bodove pomoću *collect()*
- ➢ Zadatak - domaći rad
  - Dodati novi lik (gljivu) koja se pojavljuje nakon svakog trećeg novčića
  - Ako glavni lik pokupi gljivu, dobit će dodatni život
  - Ukupan broj života glavnog lika ne može biti veći od 4

_