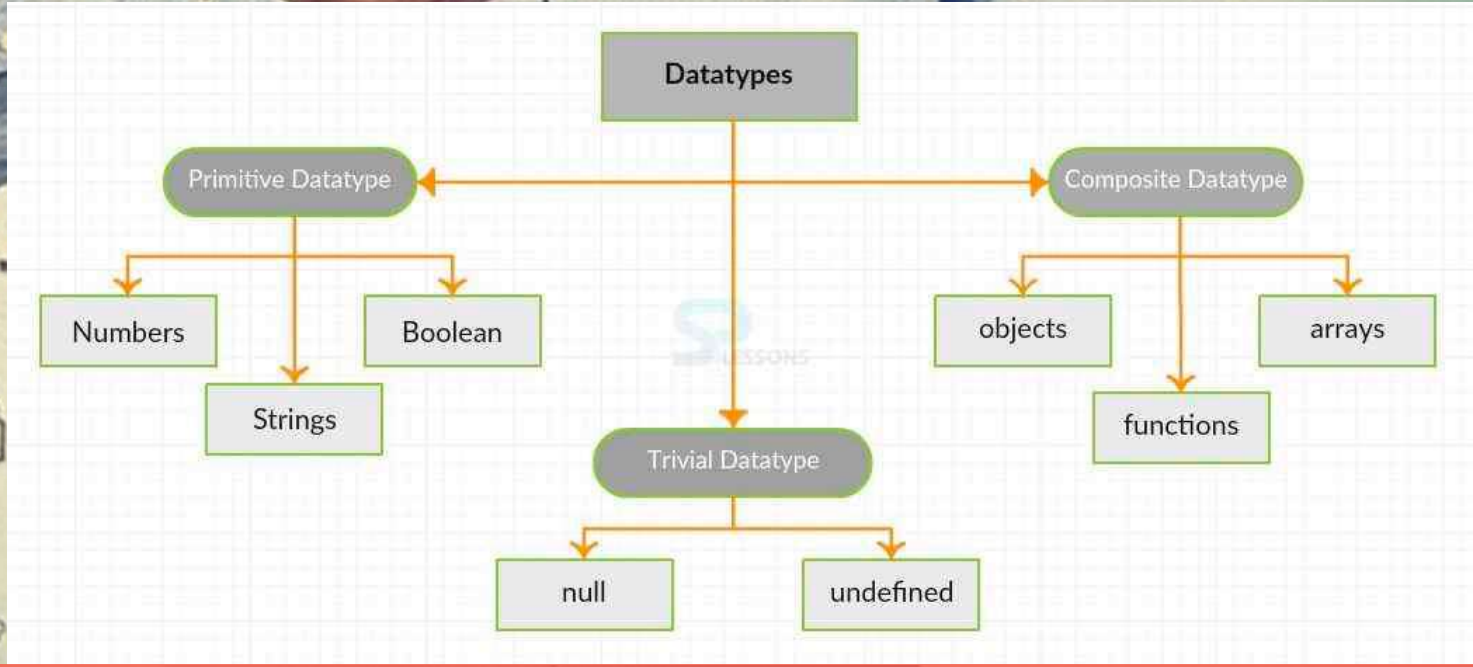
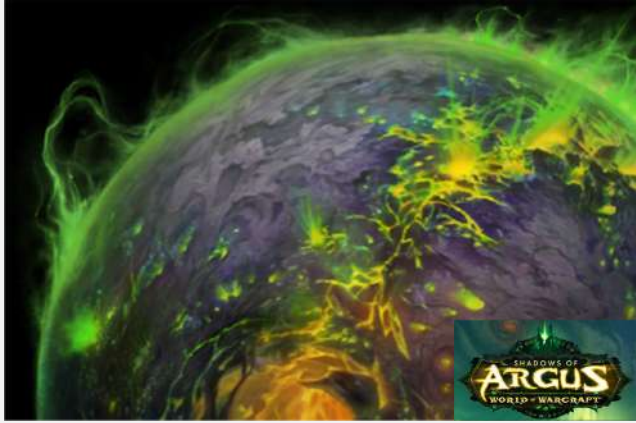
Objektno Orijentirano Programiranje

Četiri stupa Objektno Orijentiranog Programiranja

JavaScript – tipovi podataka

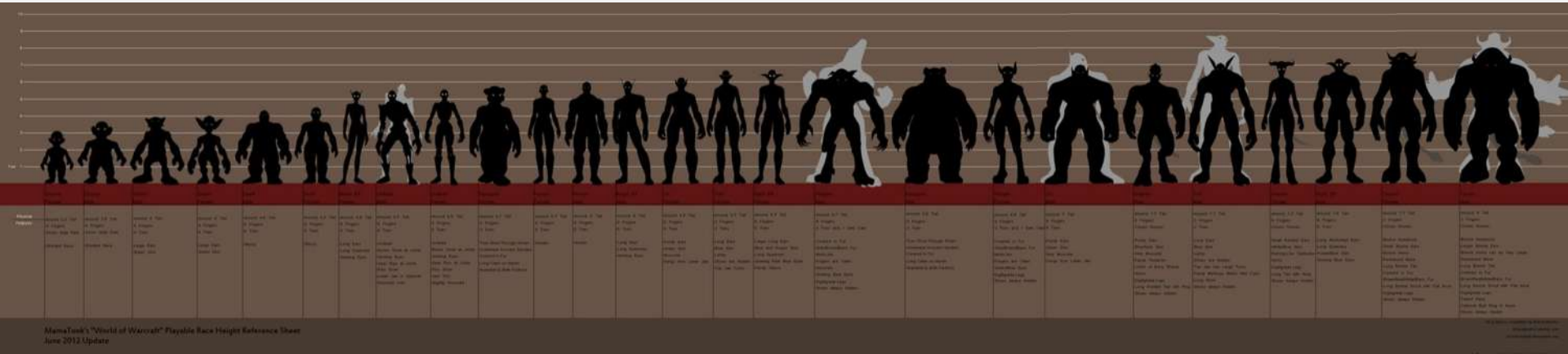


U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...



Vratimo se na igru

Uzmimo čovjeka najviše vezanog uz svoj dom.
Čovjek nam je poznat iz stvarnoga svijeta.
Sada se možemo zapitati: Kako ćemo čovjeka opisati u programu/igri?



Klasa i objekt

Klasa i objekt u igri

Odgovor je jednostavan, uz pomoć objekta i klase.

Objekt i klasa su povezani, ali ne i isti koncepti.

Klasu možemo zamisliti kao skicu koju crtamo, a ta skica ne definira točan izgled konačnog crteža nego sadrži samo bitne elemente za prepoznavanje nečega dok objekt možemo zamisliti kao konačan crtež na kojem se jasno vide svi elementi nacrtanoga.

Objekti

Ako pogledamo široku sliku tj. cijeli svijet gdje ovi konkretni heroji žive vidimo ne samo njih nego i druga živa i neživa bića.

Cijeli taj **svijet je opisan uz pomoć klasa**, a kada se igra pokrene i kada se na zaslonu našeg ekrana prikaže taj svijet, tada se **prema tim klasama stvaraju objekti koji imaju svoju ulogu u svijetu**.

Različiti objekti, ista klasa

Bitno je napomenuti kako dva čovjeka različita imena, spola ili jezika predstavljaju dva različita objekta, ali ne i objekte različitih klasa.

Različite vrijednosti tih elemenata ne znače i različit element, npr. kada bi jedan od likova imao krila i mogao letjeti, a drugi ne, to znači kako onaj prvi više nije pripadnik klase čovjek jer mu možemo pridodati dodatne atribute koje običan čovjek ne posjeduje.

Namespace - imenski prostor

Imenski prostor (eng. **namespace**) također je pojam koji se spominje unutar OOP-a.
Namespace možemo zamisliti kao **spremnik** za: klase (**class**), sučelja (**interface**), **struct**, **druge imenske prostore** i dr.
Slično direktorijima na računalu koji sadrže druge direktorije i datoteke.

```
const car = {
        start: () => {
        console.log('start’)        },
        stop: () => {
        console.log('stop') }
}
```

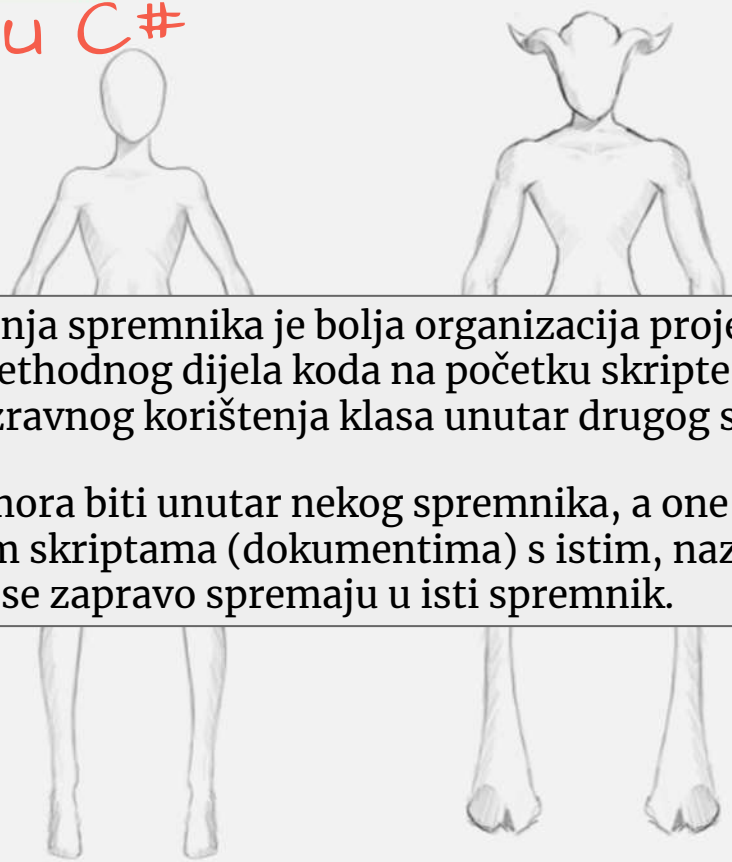
```
car.start()
```

Namespace - imenski prostor

Imenski prostor (eng. **namespace**) također je pojam koji se spominje unutar OOP-a. JavaScript nema klasični imenski prostor, ali možemo koristiti tzv. object literal
Kao što možete vidjeti ideja je ista, ali se razlikuje od C#-a

namespace Svi_Heroji

using - u C#



Svrha korištenja spremnika je bolja organizacija projekta, a svrha upisivanja prethodnog dijela koda na početku skripte je mogućnost izravnog korištenja klasa unutar drugog spremnika.

Svaka klasa mora biti unutar nekog spremnika, a one klase koje su u različitim skriptama (dokumentima) s istim, naznačenim, spremnikom se zapravo spremaju u isti spremnik.

class Covjek

class Draenei

NAMESPACE U C#-U I U JS-U NISU ISTO!

Broj spremnika unutar jednog projekta nije ograničen, a ukoliko imamo neke klase unutar drugog „spremnika“ onda se trebamo pozvati na taj spremnik.

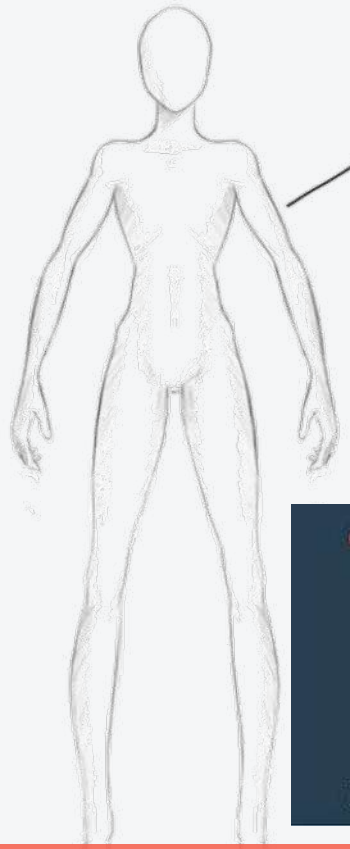
Polja, metode, konstruktor

Kada želimo sve te osobine na neki način klasificirati onda uvodimo pojam polja i metode.

Polje-svojstva

Klasa se zapravo sastoji od podataka koji su predstavljeni kao polja, a mogu biti brojevi, znakovi, skupovi znakova, druge klase ili bilo koji drugi tip podatka.

Te podatke jednim imenom nazivamo **atributi klase**, a to su neka od onih obilježja naših heroja (ime, spol...).



class Covjek
- string Naziv
- int Zlato
- char Spol
- Specijalizacija Spec
- Oklop Oklop
- bool popustNaOruzje

```
class Meal {
  constructor (food) {
    this.food = food
  }

  eat() {
    return '😋'
  }
}
```

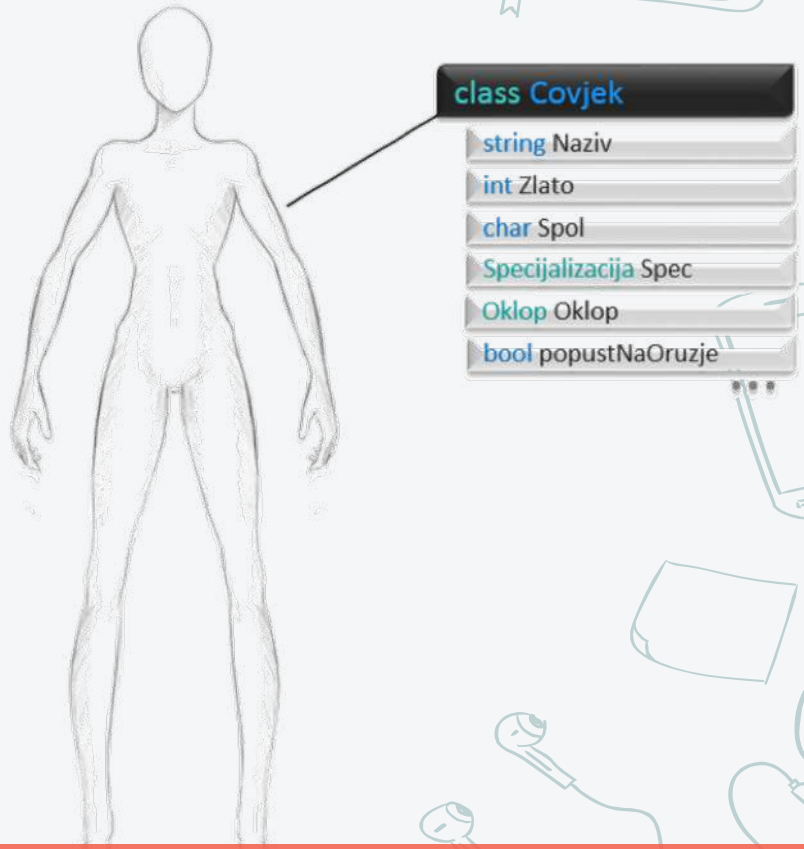
U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...

Polje-svojstva

Poljima možemo pridodavati vrijednosti (inicijalizirati varijable) pri samome pisanju klase tako kada napravimo objekt, on će sadržavati te vrijednosti.

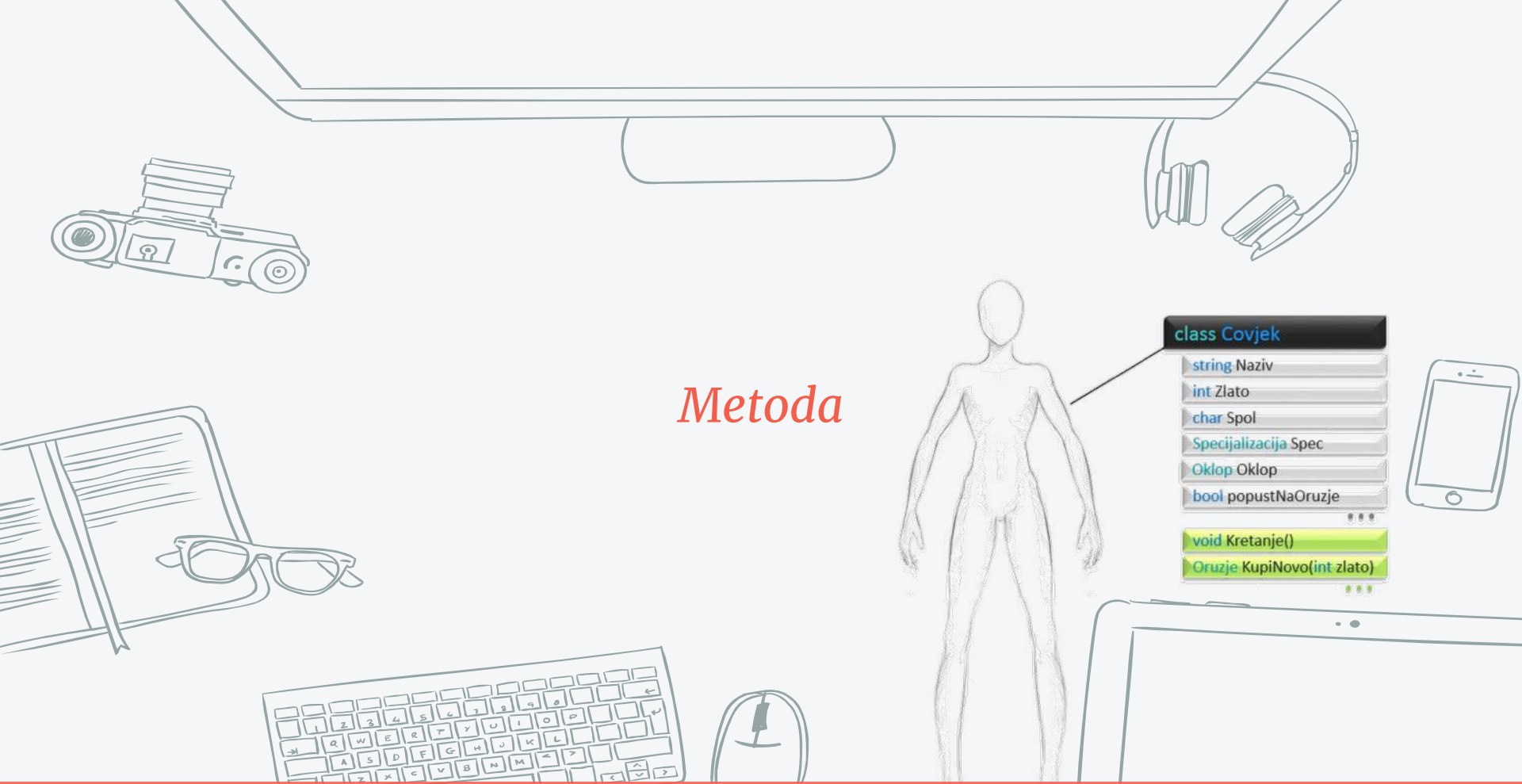
Ukoliko to napravimo kao na slici (samo deklariramo) onda će ta polja imati vrijednost tek kada ih mi sami odlučimo postaviti.

Vrijednosti atributa mogu biti promjenjive u vremenu



U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...

Metoda



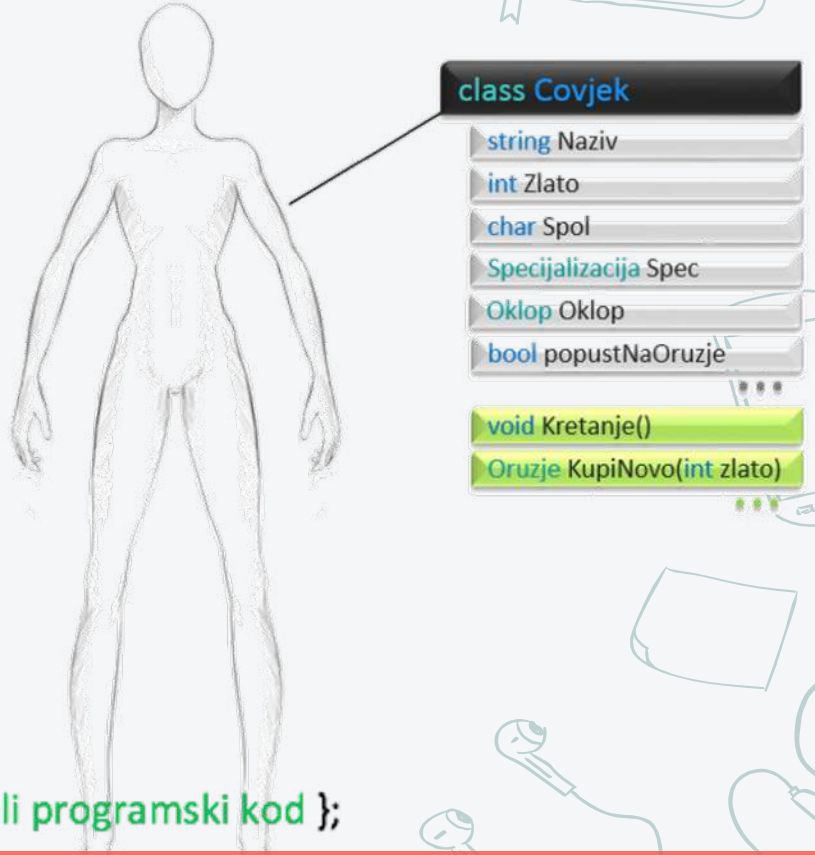
U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...

Metoda

Klasa se sastoji i od metoda, koje neki nazivaju i funkcijama prema nekim drugim programskim jezicima i paradigmama.

Metode nisu ništa drugo nego sposobnosti koje heroj/čovjek posjeduje (način na koji govori, hodanje, skakanje, rukovanje mačem...).

Metode se sastoje od 3 dijela:




```
tip Naziv (argumenti tj. ulazni parametri) { ostali programski kod };
```

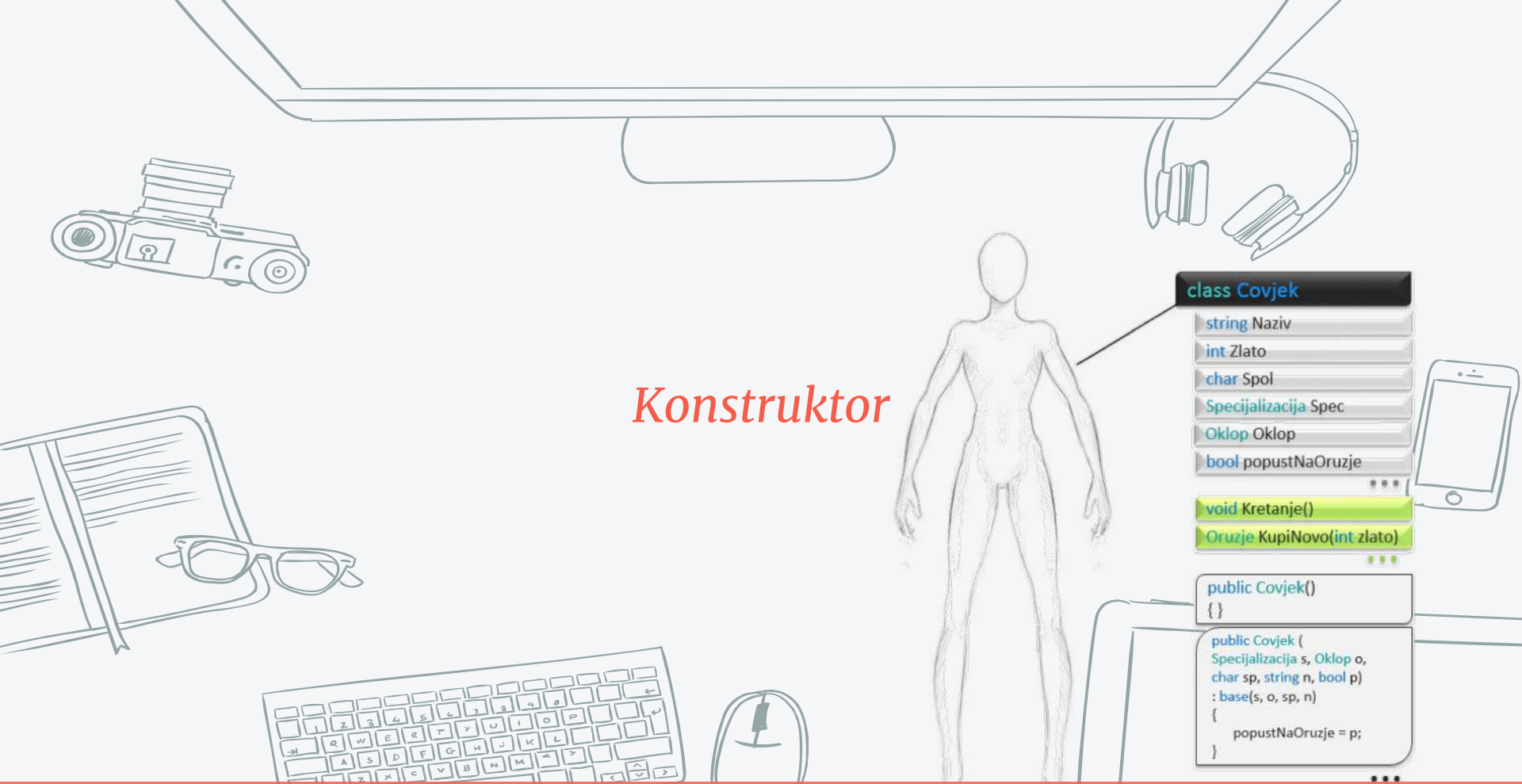
U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...

Metoda s nazivom klase i "prazna" metoda

Na slici vidimo 4 različita oružja. Kada bi metode opisivali preko kupovine i pregleda oružja, **pregled** bi bila metoda koja ne prima nikakve parametre niti vraća išta kao rezultat dok metoda za oružje treba sadržati **tip Oružje** jer je to ono što dobijemo kupovinom (metodom) te treba imati ulazni parametar, tj. novac kojim kupujemo oružje.

Unutar te metode bi se izvršavao kod koji bi tražio odabir oružja, prema cijeni provjerio stanje računa, te smanjio količinu našeg zlata za cijenu oružja.

Konstruktor



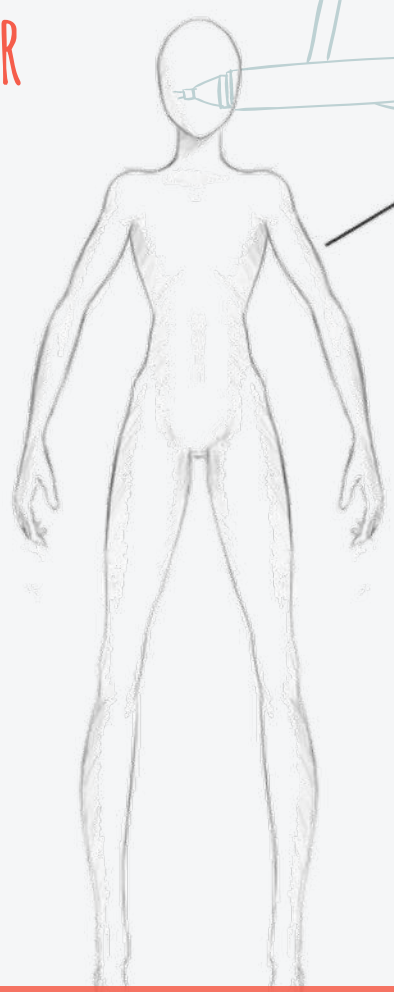
U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...

Konstruktor

Treći element svake klase je **konstruktor** koji **je zapravo poseban oblik metode**.

Dosta je sličan običnoj metodi s jednom razlikom, **uvijek treba sadržati naziv klase**.

Konstruktor je ključni element koji je potreban za instanciranje (stvaranje objekta), a to je onaj dio koda koji deklariranje varijabli razlikuje od instanciranja objekta.



```
class Covjek
    string Naziv
    int Zlato
    char Spol
    Specijalizacija Spec
    Oklop Oklop
    bool popustNaOruzje
    ...
    void Kretanje()
    Oruzje KupiNovo(int zlato)
    ...
```

```
public Covjek()
{ }
```

```
public Covjek (
Specijalizacija s, Oklop o,
char sp, string n, bool p)
: base(s, o, sp, n)
{
    popustNaOruzje = p;
}
```

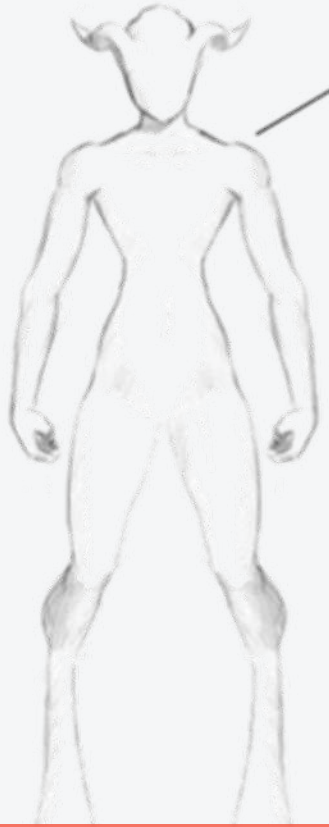
U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...

Nasljeđivanje

U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...

Nasljeđivanje

- Odmah nakon klase i objekta, nasljeđivanje je jedan od najbitnijih koncepata OOP-a.
- Kao što sama riječ govori nasljeđivanje je preuzimanje osobina roditelja.
- Već smo uveli klasu čovjeka kojeg smo opisali raznim svojstvima i sposobnostima, a sada uvedimo klasu draeneia.



```
class Draenei : Humanoid
    string Naziv
    int Zlato
    char Spol
    Specijalizacija Spec
    Oklop Oklop
    int ubrzanjeIzradeNakita

    void Kretanje()
    Oruzje KupiNovo(int zlato)
    void Regeneracija()

    public Draenei()
    { }

    public Draenei(
    Specijalizacija s, Oklop o,
    char sp, string n, float u)
    : base(s, o, sp, n)
    {
        ubrzanjeIzradeNakita = u;
    }
```

U JavaScript programskom jeziku imamo podršku za neke, ali ne za sve na koje smo navikli u C#u...

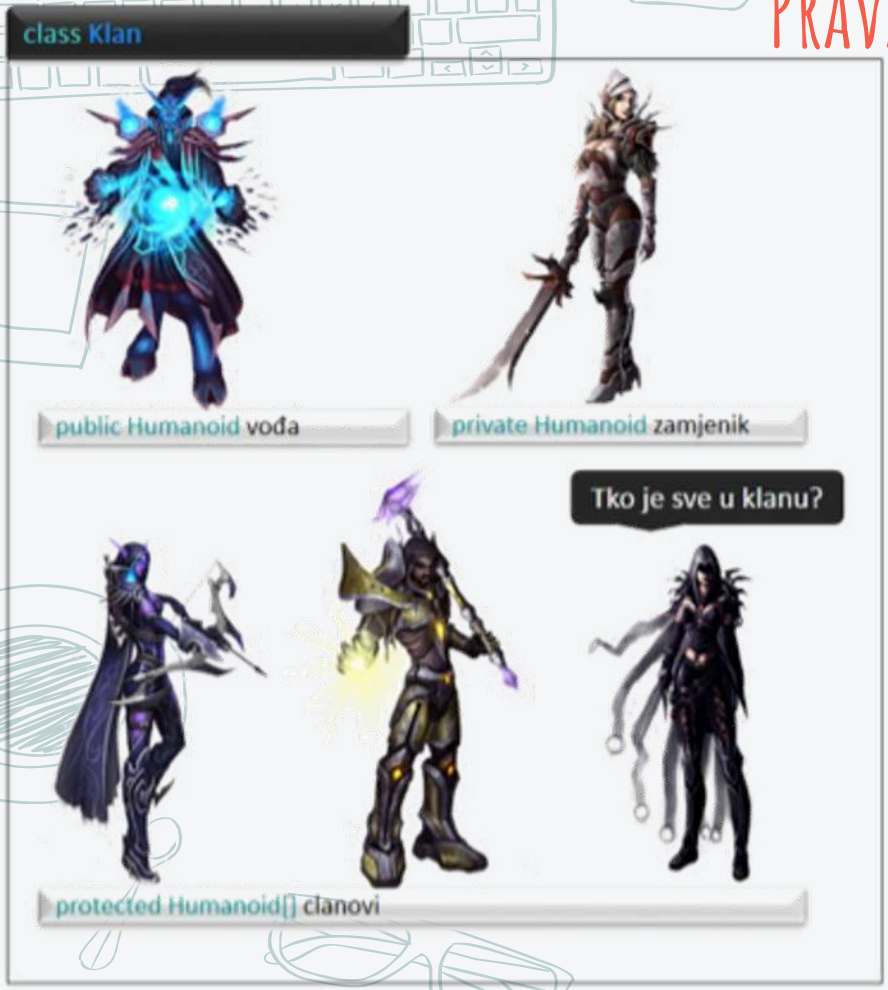
Prava pristupa i učahurivanje

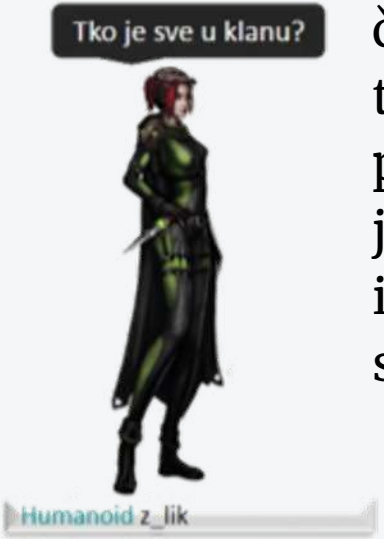
- Kada smo uspješno opisali našeg heroja, od imena pa sve do njegovih sposobnosti i uloge u svijetu, trebamo postaviti prava pristupa određenim obilježjima. Za to imamo par pitanja:
  - Tko bi sve trebao znati naše planove?
  - Tko bi sve trebao znati sve o nama?
  - Tko sam ja i želim li da drugi utječu na to tko sam ja?
  - Kako ću se zaštititi od negativnog utjecaja drugih?

Ujedinili smo se i osnovali klan, bolje ćemo funkcionirati kao skupina nego kao pojedinci. Kada imamo svoj klan trebamo ga zaštiti od drugih i držati neke informacije tajnima. Za to nam, u programiranju, služe prava pristupa.

Navesti ćemo samo dva osnovna, a to su: public, private.

Prava Pristupa





Heroj koji je član klana dobro poznaje sve članove klana dok onaj tko nije član klana može poznavati samo vođu jer je vođa javan (public), a informacije o ostalima su privatne (private).

Objektno Orijentirano Programiranje

Četiri stupa Objektno Orijentiranog Programiranja

JOŠ JEDAN OD STUPOVA OOP-A

Što je polimorfizam?

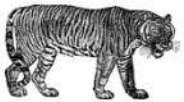
Pogledajmo primjer

Without Polymorphism

Required Type: “lion”

add_lion()

Required Type: “tiger”

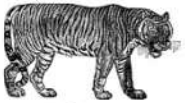


add_tiger()

Required Type: “bear”

add_bear()

With Polymorphism



Required **Behavior**: “growl()” function

add_growler()

Polimorfizam

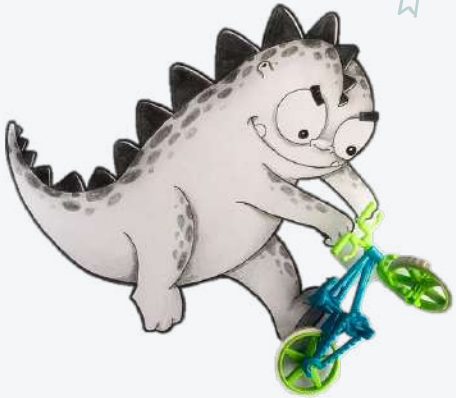
Polimorfizam (“višeobličje”) je važno svojstvo objektno orijentiranog programiranja jer omogućuje definiciju metoda u osnovnoj klasi koje će biti zajedničke za sve izvedene klase, ali uz slobodu implementacije tih metoda u izvedenoj klasi.

Polimorfizam

Polimorfizam (“višeobličje”)

Premošćovanje (Override)

zamijeni postojeću metodu novom



Polimorfizam

Polimorfizam (“višeobličje”)

Premošćovanje (Override)

zamijeni postojeću metodu novom

Preopterećenje (Overload)

prema broju i tipu argumenata izvrši odgovarajuću metodu (nije podržano u sintaksi JavaScript-a, ali podržano je u C#-u)

Dva vrste polimorfizma

Polimorfizam

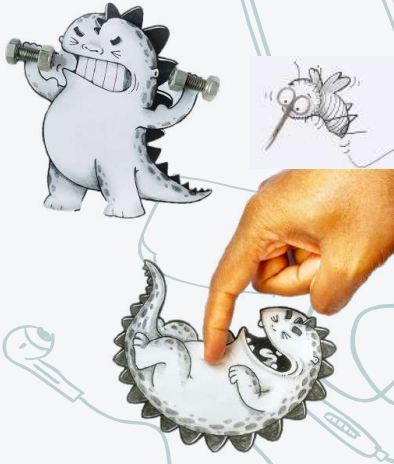
Polimorfizam (“višeobličje”)

Premošćovanje (Override)

zamijeni postojeću metodu novom

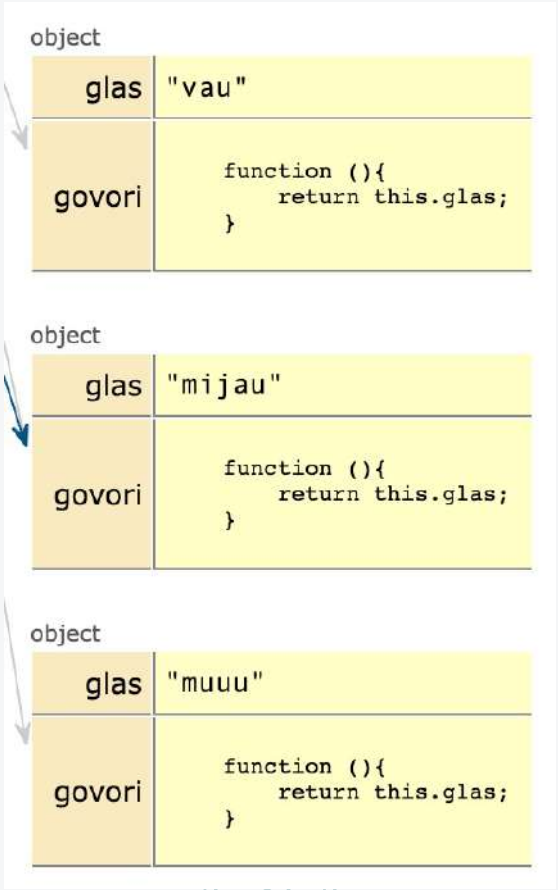
Preopterećenje (Overload)

prema broju i tipu argumenata izvrši odgovarajuću metodu (nije podržano u sintaksi JavaScript-a, ali podržano je u C#-u)



Dva vrste polimorfizma

Primjer - premošćivanje




```
function Zivotinja(glas){
this.glas=glas;
this.govori=function(){
return this.glas;
}
}
//jedna metoda
function pokaziInfo(obj){
console.log(obj.govori());
}
//razliciti objekti
var pas = new Zivotinja("vau");
var macka = new Zivotinja("mijau");
var krava = new Zivotinja("muuu");
//odgovaraju na razlicite nacine
pokaziInfo(pas);
pokaziInfo(macka);
pokaziInfo(krava);
```

Premošćivanje

Ako ipak želimo pozvati implementaciju koja se nalazi u osnovnoj klasi, jer nam je to SUPER, potrebno je koristiti ključnu riječ

super.

Preopterećenje

Preopterećenje nije podržano u JavaScript sintaksi, ali ideja je:

Prema broju argumenata izvršiti neku, točno određenu metodu

Kako do rješenja?

TRIK

Postoji objekt argument...

```
1  function funkcija1(a, b, c) {
2    console.log(arguments[0]);
3    // očekivani izlaz: 1
4
5    console.log(arguments[1]);
6    // očekivani izlaz: 2
7
8    console.log(arguments[2]);
9    // očekivani izlaz: 3
10 }
11
12 funkcija1(1, 2, 3);
```

```
> 1
> 2
> 3
```

Preopterećenje

```
// preopterecenje - overloading
function suma() {
// ako nema argument
if (arguments.length === 0) {
console.log("Niste prenijeli niti jedan argument");
}

// ako ima samo jedan argument
else if (arguments.length === 1) {
console.log("Prenesite barem dva argumenta");
}

// vise argumenata
else {
let result = 0;
let length = arguments.length;

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

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

Preopterećenje

```
// 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;
}
}
```

```
suma();
suma(5);
suma(5, 9);
suma(1, 2, 3, 4, 5);
```

Pogledajmo kako bi naučeno primijenili

Započnimo s kodom naše igre Avantura, ali prije toga oblikujmo (dizajnirajmo) aplikaciju!

Što radimo

Radimo dijagram klasa (dio UML-a) koji je vrsta strukturnog dijagrama koji opisuje strukturu sustava objašnjavajući klase unutar sustava, njihove atribute i odnose.

DO IDUĆI PUTA!

ste slobodni