Lekcija 3 - Klase i interfejsi

- **Primer**

- U primeru vidimo konstruktor bez parametara i on se poziva kada se pozove operator new
- Tada se incijalizuje objekat i promenljive u njemu

```
class Person {
    firstName: string;
    lastName: string;
    constructor() {
        this.firstName = " ";
        this.lastName = " ";
    }

    getFullName() {
        return this.firstName + this.lastName;
    }
}
```

```
var aPerson: Person = new Person();
```

Lekcija 3 - Klase i interfejsi

- **Konstruktor sa parametrima**

- **Konstruktor može da ima i parametre** tj. možemo da izvršimo automatski dodelu vrednosti parametrima prilikom inicijalizacije objekta.

```
constructor(firstName: string, lastName: string) {
    this.firstName = firstName;
    this.lastName = lastName;
}
```

```
var aPerson: Person = new Person("Nenad"," Bosanac");
console.log(aPerson.getFullName());
```

Lekcija 3 - Klase i interfejsi

- **U klasi može da postoji samo jedan konstruktor u TS-u**

- Više konstruktora izaziva grešku

```
    constructor() {
        this.firstName = " ";
        this.lastName = " ";
    }
    [ts] Multiple constructor implementations are not allowed.
    constructor(firstName: string, lastName: string) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
```

- U TS nema višetrukih konstruktora jer ne možemo da koristimo overloading konstruktora, tj. jer ne možemo imati overlodovane funkcije

Lekcija 3 – Klase i interfejsi

- **Nasleđivanje klasa**
- TS podržava koncept nasleđivanja tj. mogućnost da osnovu klasu nasledi druga klasa
- Nasleđivanje (inheritance) je jedan od osnovnih OO koncepata
- Svaka klasa naslednica nasleđuje sve atribute i funckije svog roditelja
- Nasleđivanje klasa se vrši uz pomoć ključne reči **extends** koja se definiše pre naziva klase roditelja

Lekcija 3 - Klase i interfejsi

- **Primer:**
- Klasa Programmer treba da nasledi klasu Person1 jer je svaki programer i osoba , definisati objekat klase Programmer i pozvati klasu greet()

- Vidimo u primeru da je pozvana metoda greet klase Person1 jer to znači da se metode naslednice mogu koristiti

```
class Person1 {
    firstName: string;
    lastName: string;
    greet(){
        console.log("Pozdrav gari !");
    }
}
class Programmer extends Person1 {

}
var aProgrammer = new Programmer();
aProgrammer.greet();
```

Lekcija 3 - Klase i interfejsi

- **Metode u roditeljskoj klasi mogu da se overajduju** tj da se ista metoda napiše u klasi detetu i tada će ova metoda biti pozvana

```
class Person1 {
    firstName: string;
    lastName: string;
    greet(){
        console.log("Pozdrav gari !");
    }
}
class Programmer extends Person1 {
    greet(){
        console.log("Zdravo gari !");
    }
}
var aProgrammer = new Programmer();
aProgrammer.greet();
```

Lekcija 3 - Klase i interfejsi

- **Super ključna reč**
- **Koristimo je ako hoćemo da pozovemo promenljivu ili metodu roditeljske klase**
- Super možemo da korisitmo u konstruktoru da pozovemo roditeljski konstruktor

```
class Person1 {
    firstName: string;
    lastName: string;
    greet(){
        console.log("Pozdrav gari !");
    }
}
class Programmer extends Person1 {
    greet(){
        console.log("Zdravo gari !");
    }
    greetFromParent(){
        super.greet();
    }
}
var aProgrammer = new Programmer();
//aProgrammer.greet();
aProgrammer.greetFromParent();
```

Lekcija 3 - Klase i interfejsi

- **Polimorfizam**

- Ideja je da možemo da imamo više instanci od više klasa koje referišu da koristimo roditeljsku klasu

- Npr:

```
}    var aProgrammer: Programmer
var aProgrammer = new Programmer();
```

- Vidimo da je aProgrammer tipa Programmer klase

- Koristeći polimorfizam (poli – više ,morfizam – oblik) – više oblika možemo da aProgrammer bude drugog tipa Person

Lekcija 3 - Klase i interfejsi

- Npr: sada je aProgrammer tipa Person1 ali je i dalje instanca objekta Programmer što znači da će koristiti metode Programmer klase

```
var aProgrammer : Person1 = new Programmer();
```

```
class Programmer extends Person1 {
    greet(){
        console.log("Zdravo gari !");
    }
    greetFromParent(){
        super.greet();
    }
}
//var aProgrammer = new Programmer();
//ova metoda poyiva metodu i ispisuje Zdravo Gari
//aProgrammer.greet();
var aProgrammer : Person1 = new Programmer();
//ova metoda poziva metodu klase Person1 i ispisuje Zdravo Gari
aProgrammer.greet();
```

Lekcija 3 – Klase i interfejsi

- **Interfejsi**
- Za definisanje interfejsa koristi se ključna reč interface zatim sledi naziv interfejsa.
- Uvek se piše početnim velikim slovom i dobra praksa dodati mu slovo I npr IPerson
- Interfejsi sadrže dekleraciju atributa i metoda. Metode se ne implementiraju u interfejsu !
- Metode se samo potpisuju sa svojim nazivom
- Implementacija medode se vrši u klasi koja implementira interfejs

Lekcija 3 - Klase i interfejsi

- **Primer:**

```
interface IPerson{
    firstName: string;
    lastName: string;
    getFullName():string;
}
```

Metoda je samo potpisana ili deklarisana

- Klasa koja implementira neki interfejs to radi sa ključnom rečju implements

- Svaka klasa koja implementira interfejs mora da implementira sve metode iz interfejsa.

- Interfejsi se ne mogu instancirati !
- https://toddmotto.com/classes-vs-interfaces-in-typescript

Lekcija 3 - Klase i interfejsi

- Interfejsi su virtualne strukture u TypeScriptu nema ih u JS kada se uradi transpajlovanje
- **Primer :**

```
interface IPerson{
    firstName: string;
    lastName: string;
    getFullName():string;
}

class Worker1 implements IPerson{
    firstName: string;
    lastName: string;
    getFullName():string {
        return this.firstName + this.lastName;
    }
}
```

- **Klasa može da implementira više interfejsa**

Lekcija 3 - Klase i interfejsi

- **Koncept duck typing**

- "Ako vidiš da izgleda kao patka, kvače kao patka i hoda kao patka onda je sigurno patka"

Lekcija 3 - Klase i interfejsi

- **Vidljivost članica-elemenata-atributa**
- Dot notacija je defaultno ponašanje za pristup elementima ili atributima klase
- Ovo je usvojeno ponašanje iz JS gde se svakom elementu može javno pristupiti , nema koncepta privatnog pristupa
- U TS je uveden koncept privatnosti i dozvoljava da zaštitimo članice promenljive ili atribute u okviru klase u kojoj se nalaze
- To je ustvari koncept enkapsulacije i sakrivanje informacija od ostalih članica

Lekcija 3 - Klase i interfejsi

- TS koristi tri vrste vidljivosti elemenata :
  1. private (članovi su vidljivi u okviru klase)
  2. public i (članovi su vidljivi i van klase)
  3. protected (članovi su vidljivi u okviru klase i klase naslednice)

- U TS ako ne definišemo vidljivost članica klasa podrazumeva se da je public

- Public znači da nema ograničenja u pristupu članici u klasi

- Sve isto važi za atribute i metode.

Lekcija 3 - Klase i interfejsi

- Primer : Korišćenje vidljivosti kod članica
- Vidimo da je element **firstName** definisan kao **public** i javno je dostupan van klase
- Element **lastName** je definisan kao **private** i nije dostupan van klase Person11, zato mu ne možemo ni pristupiti nakon pravljenje instance sa dot notacijom

```
class Person11 {
    public firstName: string;
    private lastName: string;
    greet(){
        console.log("Pozdrav gari !");
    }
}
class Programmer1 extends Person11 {
    greet(){
        console.log("Zdravo gari !");
    }
    greetFromParent(){
        super.greet();
    }
}

var aProgrammer1 : Person11 = new Programmer1();
console.log(aProgrammer.greet());
console.log(aProgrammer1.firstName);
console.log(aProgrammer1.lastName);
```

Lekcija 3 - Klase i interfejsi

- U okviru public metode možemo da pristupimo privatnim članicama atributa korišćenjem this ključne reči

- Pristup privatnim atributima se pravilno pristupa koristeći **setere i getere** tj. specijalno formiranim funckijama

```
class Person11 {
    private firstName: string;
    private lastName: string;
    greet(){
        console.log("Pozdrav gari !");
    }

    getFullName(){
        this.firstName + " " + this.lastName;
    }
}
```

Lekcija 3 – Klase i interfejsi

- **Setteri i getteri**
- Svaki atribut definisan kao private ima ograničenje u pristupu
- Takvim atributima za pristup i podešavanje vrednosti vršimo koristeći setere i getere
- To su specijalne funkcije koje imaju prefiks set ili get nakon čega sledi naziv imena atributa sa početnim velikim slovom prve reči

Lekcija 3 - Klase i interfejsi

- **Primer:**

```
class Person11 {
    private firstName: string;
    private lastName: string;
    greet(){
        console.log("Pozdrav gari !");
    }
    setFirstName(firstName:string){
        this.firstName = firstName;
    }
    getFirstName(){
        return this.firstName;
    }
```

- **Zadatak : napraviti setere i getere za lastName atribut, zatim napraviti konstruktor za inicijalizaciju obe promenljive**

Lekcija 3 - Klase i interfejsi

- Umesto sledećeg paterna definisanja članica u okviru konstruktora

```
class Person11 {
    private firstName: string;
    private lastName: string;

    constructor(firstName:string,lastname:string){
        this.firstName = firstName;
        this.lastName = lastname;
    }
```

- Možemo da definišemo sledeći patern

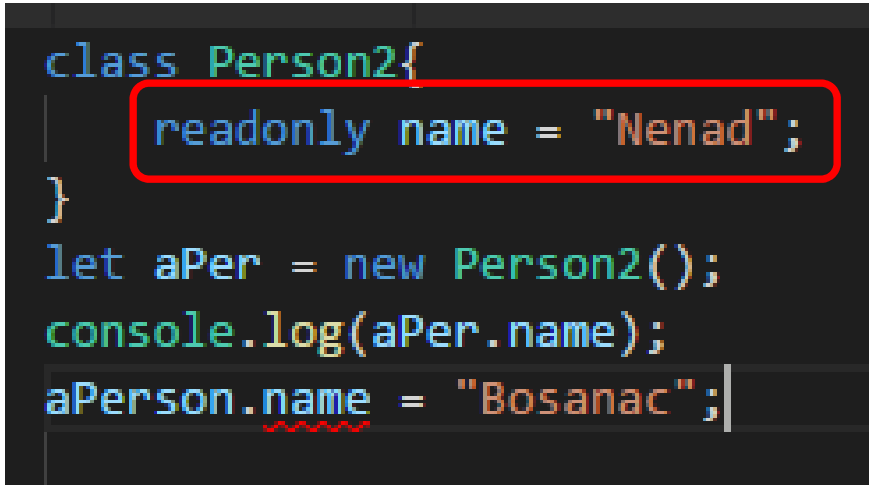
```
constructor(private firstName:string, private lastname:string){
    //ovde ne treba definisati dodelu
}
```

Ovaj patern radi generisanje atributa koji postaju članice atributa, prima argumente u konstruktor i smešta vrednosti u atribute

Lekcija 3 - Klase i interfejsi

- **Readonly ključna reč**

- Ova ključna reč omogućava da se promenljiva definisana kao readonly može samo čitati ,ali posle njene deklaracije i dodele vrednosti ne može da se menja



Lekcija 3 - Klase i interfejsi

- Može se deklarisati u okviru promenljive kao atribut ili u okviru konstruktora

- Ali tada jedina inicijaizacija promenljive može da se uradi kroz instancu klase

```
//skraceni yapis
constructor(readonly name:string){

}
```

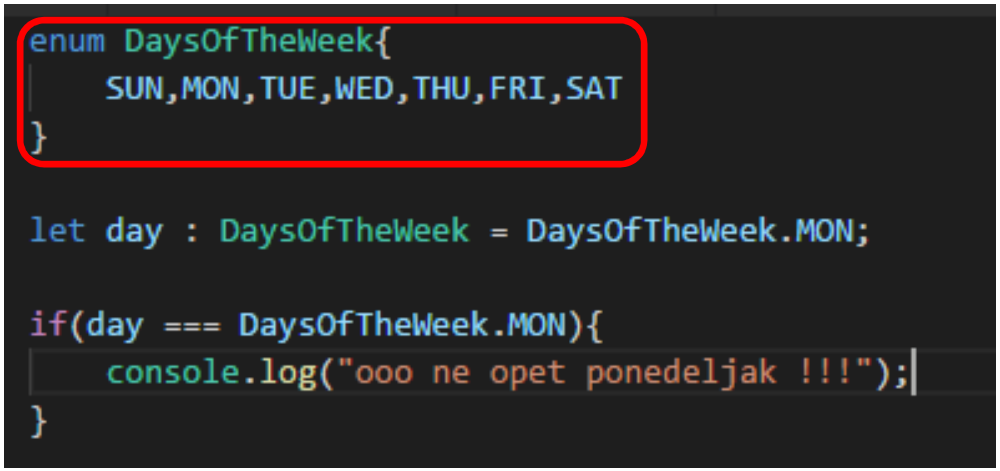
```
class Person2{
    readonly name: string;

    constructor(name:string){
        this.name =name;
    }
}
let aPer = new Person2("Nenad");
console.log(aPer.name);
aPerson.name = "Pera";
```

Lekcija 3 - Klase i interfejsi

- **Enums**
- Ukoliko imamo veliki broj konstanti u našem kodu možemo da koristimo enumeraciju
- Enumeracije se označavaju ključnom rečju enum
- Kod je sličan kao kada pravimo klasu i enumerator se ustvari ponaša kao tip podataka .
- JS enume prepoznaje kao niz sa dodeljenim ključ –vrednost opcijom

Lekcija 3 - Klase i interfejsi

- **Primer:**



- U primeru gore vidimo definisan enum sa konstantama
- Konstantama u enumu pristupamo sa dot notacijom

Lekcija 3 - Klase i interfejsi

- **Primer:**

```
enum DaysOfThe (enum member) DaysOfTheWeek.MON = 101
    SUN = 100,MON,TUE,WED,THU,FRI,SAT
}

let day : DaysOfTheWeek = DaysOfTheWeek.MON;

if(day === DaysOfTheWeek.MON){
    console.log("ooo ne opet ponedeljak !!!");
}
```

- Enumeratori mogu da imaju definisanu početnu vrednost, koje se posle toga automatski inkrementuju za 1

- Naravno svaka konstanta u enumu može da ima svoju vrednost

Lekcija 3 – Klase i interfejsi

- **Generički tipovi**
- Generički tipovi e koriste kao zamena za striktno definisanje tipova
- Ključna reč za definisanje nekog elementa da je generički jeste **<T> (slovo t okruženo sa manje i veće)**
- **Generičke tipove možemo gledati kao zamenu za overlodovanje iste funkcije sa različitim tipovima**

Lekcija 3 - Klase i interfejsi

- **Primer**

```
function echo<T>(arg: T): T{
    return arg;
}
let myStr = echo(1);
let myStr1 = echo("Nenad");
let b1 = echo(true);
let myStr2 : String = echo(100);
```

- Generički tip se postavlja kao oznaka posle naziva funkcije ili klase .
- Svaki naziv tipa se zamenjuje u funkciji ili klasi sa slovom T
- U primeru vidimo da se vrši implicitna konverzija svih tipova na osnovu unete vrednosti u funkciju

Lekcija 3 - Klase i interfejsi

- Generičke tipove možemo primeniti i na klase odnosno da deklarišemo tipove klasa kroz generički tip
- **Primer: generičkiTip.ts**

```
class Person3 {
    firstName: string;
    lastName: string;

    constructor(firstName: string, lastName: string) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    getFullName() {
        return this.firstName + this.lastName;
    }
}
class Admin extends Person3{

}
class Manager extends Person3{

}
```

```
let admin = new Admin('aca','aleksov');
let manager = new Manager('boki','bonusic');

function personEcho<T>(person:T):T{
    return person;
}

var foo = personEcho(admin);
var foo1 = personEcho(manager);
```

Prosleđujemo tipove klasa u funkciiju

Lekcija 3 - Klase i interfejsi

- **Moduli**

- Moduli omogućavaju deljenje source koda u više fajlova po logičkim celinama

- Moduli se dele na interni i eksterne module

- Moduli olakšavaju i pregled koda jer se sve ne nalazi u jednom fajlu

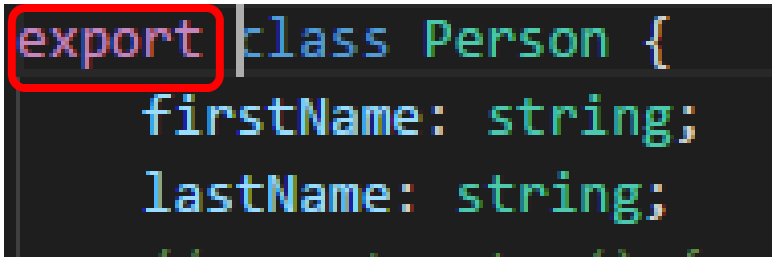
- U TS klase ,interfejse ili funkcije možemo da eksportujemo importujemo iz jednog u drugi fajl i to je koncept modula.

Lekcija 3 - Klase i interfejsi

- **Primer:** Pogledajmo prethodni primer generickiTip.ts gde imamo klasu Person koju smo već definisali u klase.ts fajlu i možemo da koristimo ovu klasu u generički Tip.ts fajlu .

- **Sve što je potrebno jeste da napravimo export klase Person u klase.ts fajlu i importujemo je u generickiTip.ts**

- Potrebno je da ispred naziva klase definišemo ključnu reč **export**



Lekcija 3 - Klase i interfejsi

- Sada je potrebno da u generickiTip.ts importujemo klasu klase.ts koja ima eksportovanu klasu Person

```
import {Person} from './klase';
```

- Koristimo ključnu reč **import** , zatim u uglastim zagradama navodimo jednu ili više klasa koje importujemo.

- Koristimo ključnu reč **from** koja nam govori iz kojeg fajla importujemo klase, interfejse ili funkcije

- **Ne navodimo ekstenziju .ts na kraju naziva fajla !**

- Sa import ključnom rečju rekli smo TS-u koje klase su od sada dostupne u celom fajlu koj iga importuje

Lekcija 3 - Klase i interfejsi

- Upamtite da se prilikom eksportovanja i importovanja klase izvršava ceo TS fajl koji je importovan bez obzira što importujemo samo klasu !

- Tako da rezultat pokretanja genrickiTip.js inicijalizuje objekat i ispiše ono što je definisano u console.log

```
export class Person {
    firstName: string;
    lastName: string;
```

```
var aPerson: Person = new Person("Nenad"," Bosanac");
console.log(aPerson.getFullName());
```

```
D:\Kursevi\NSZ\TypeScript\Radni folder\Primer-tsconfig\js>node generickiTip.js
Nenad Bosanac
```