

PRIRUČNIK ZA AUDITORNE I LABORATORIJSKE VJEŽBE IZ PREDMETA

OBJEKTNO ORIJENTIRANO PROGRAMIRANJE

Nastavni materijali za:

- **1. auditorna vježba**
- **2. auditorna vježba**
- **1. laboratorijska vježba**



dr.sc. Oliver Jukić, prof.v.š
Ivan Heđi, dipl.ing., v.pred.
Marko Hajba, mag. math., pred.

Virovitica, ožujak 2021.

© Oliver Jukić,
Ivan Heđi,
Marko Hajba, 2021.

Urednik
dr. sc. Oliver Jukić, prof.v.š.

Materijal nije recenziran i služi samo za internu uporabu

Grafička priprema
Ivan Heđi, dipl.ing., v.pred.
Marko Hajba, mag. math., pred.

Niti jedan dio ovog priručnika ne smije se umnožavati i preslikavati na bilo koji način, bez pismenog odobrenja autora



OSNOVNE UPUTE

- Auditorne vježbe (AV) održavat će se tijekom semestra u 10 termina. Uloga im je upoznati studente s temeljnim konceptima objektno orijentiranog programiranja i pripremiti studente za laboratorijske vježbe i ispit.
- Auditorne vježbe su obavezne za sve studente.
- Laboratorijske vježbe (LV) održavat će se tijekom semestra u 6 termina po 3 sata u računalnim praktikumima. Kolokviranje laboratorijskih vježbi obavezno je za sve studente i uvjet je izlaska na ispit. Student će kolokvirati laboratorijske vježbe ukoliko riješi sve obvezne zadatke i točno odgovori na postavljena pitanja vezana uz rješenja zadataka. **Potrebno je moći pojasniti svaku liniju koda.**
- Na kraju semestra održat će se termin laboratorijskih vježbi u kojemu se može popraviti jedna laboratorijska vježba.



1. Auditorna vježba

Sadržaj vježbe:

- Upoznavanje s pojmom klase
- Primjeri klasa
- Povezivanje pojma klase s objektima iz realnog svijeta

2. Auditorna vježba

Sadržaj vježbe:

- Konstruktor, destruktor, kreiranje objekata klase
- Prava pristupa
- Funkcijski članovi klase

1. Laboratorijska vježba

Sadržaj vježbe:

- Kreiranje klasa
- Konstruktor, destruktor, kreiranje objekata klase
- Prava pristupa
- Funkcijski članovi klase

Zadatke je potrebno riješiti na papiru i u MS Visual Studio-u u programskom jeziku C#.

Pojam klase

Klasa je određeno proširenje struktura podataka. Klasa je skup pravila koja opisuju od čega je objekt sagrađen i kakva su njegova svojstva. Klasa je opis, a objekt je stvarna, konkretna realizacija napravljena na temelju opisa (klase). Klasom su u logičku cjelinu zatvorena i opisana svojstva koja karakteriziraju neku skupinu istovrsnih objekata iz stvarnog svijeta (npr. automobil, student, televizor, ...). U programskom kodu moguće je na osnovu opisa (klase) kreirati (instancirati) objekte u programskom kodu, koje onda koristimo da nam reprezentiraju objekte iz stvarnog svijeta u našim programima. Za razliku od strukture koju čine samo podatkovni članovi (kasnije su dodani i funkcijski članovi), unutar klase moguće je definirati i funkcijske članove.

Primjer 1. Klasa Student

Klasa: Student

Podatkovni članovi: ime, prezime, OIB, JMBAG, datum rođenja, prebivalište, studij, godina studija, predmeti, ocjene, visina, težina

Objekti: konkretni studenti kao osobe, npr. Marko Marić sa svim podacima podatkovnih članova klase:

"Marko", "Maric", "12345678123", "0307019999", "01/01/2000", "Virovitica", "Racunarstvo", 2, predmeti_s1, ocjene_s1, 179, 75



Zadatak 1.

Zapišite na papir svoj primjer klase.

Napomena! Svaki student treba imati različiti primjer klase.

Zadatak 2.

Navedite za svoj primjer klase minimalno 5 podatkovnih članova i njihove tipove.

Zadatak 3.

Navedite barem tri konkretna primjera objekata za klasu koju ste odabrali.

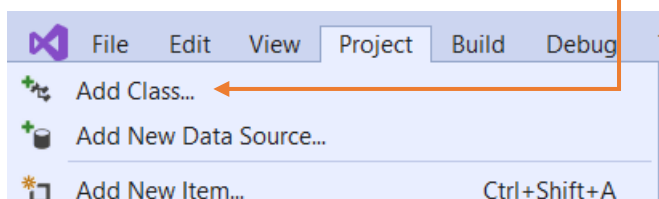
MS Visual Studio i klase u C#

Kreiranje i osnove rada s klasama u C# programskom jeziku demonstrirati ćemo na primjeru.

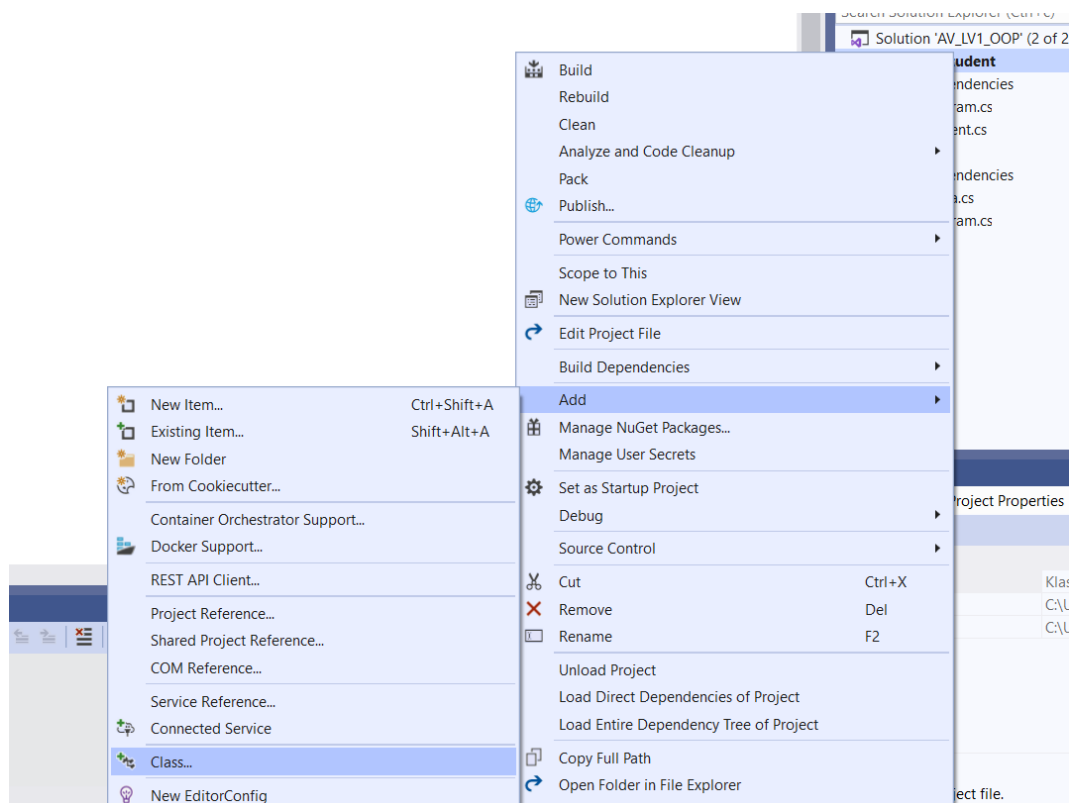
Primjer. Klasa Student

Kreirajmo klasu Student u programu MS Visual Studio 2019 koristeći C#.

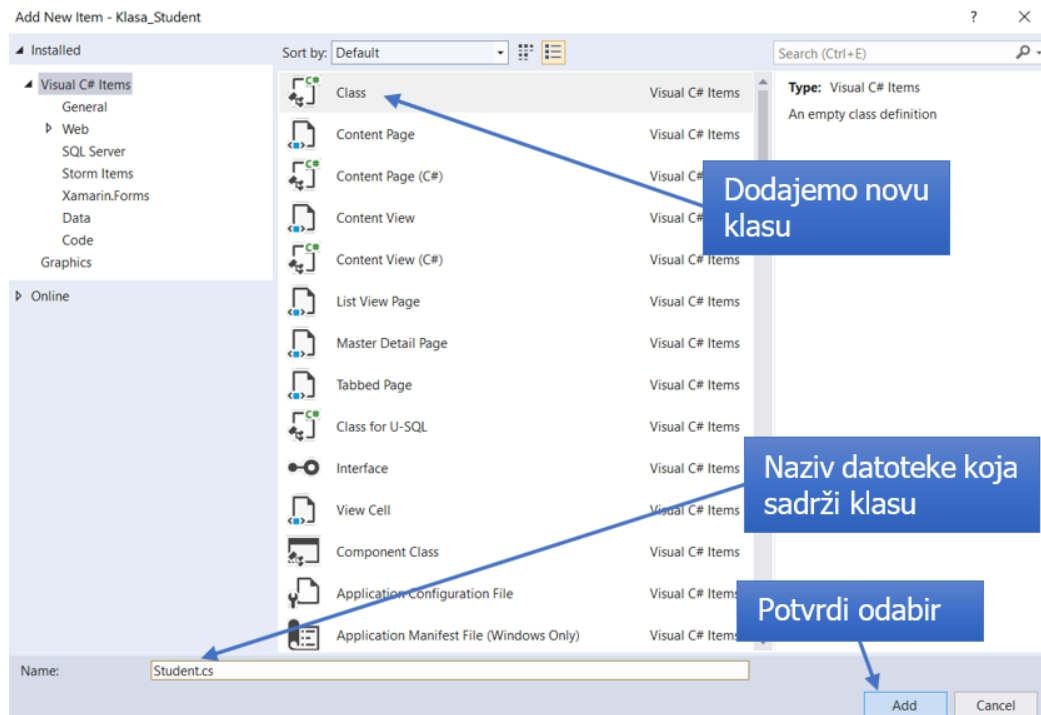
1. Kreiramo novi projekt
2. Novu klasu možemo dodati odabirom **Project > Add Class**



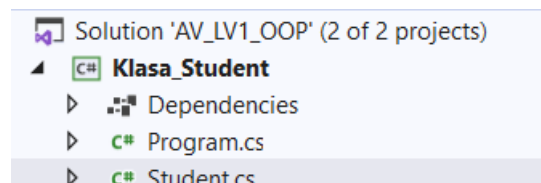
Ili **desni klik mišem na projekt > Add > Class**



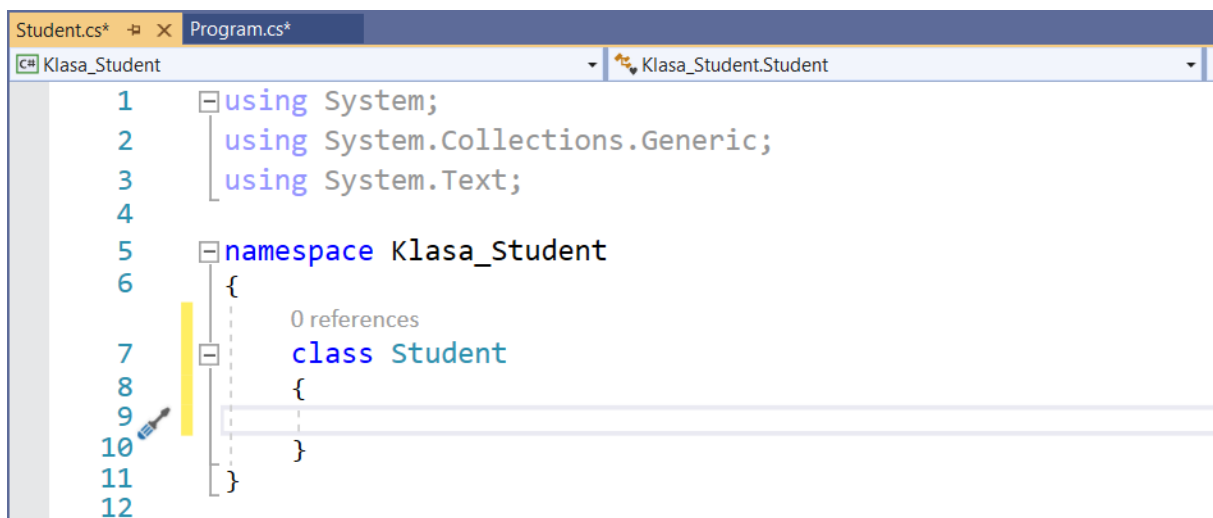
Zatim nam se otvara novi prozor u kojemu biramo klasu i zadajemo joj naziv:



Nakon toga će se datoteka Student.cs pojaviti unutar projekta.



A datoteka Student.cs ima oblik



Primijetite `namespace Klasa_Student`. Polje imena `namespace` vrlo je važna odrednica vidljivosti klase unutar projekta i potrebno je biti pažljiv s definicijom polja imena. Npr., u ovom slučaju je



`namespace` isti kao naziv projekta pa će se klasa vidjeti unutar cijelog projekta bez eksplicitnog dodavanja klase unutar drugih datoteka unutar projekta. Neka je klasa `Student` definirana unutar `namespace` `klasa_Student`:

```
namespace klasa_Student
{
    class Student
    {
    }
}
```

Sada je polje imena `namespace` `klasa_Student`, a ne `namespace` `Klasa_Student` i tada se klasa `Student` ne vidi u drugim dijelovima projekta bez uključivanja `namespace` `klasa_Student` u datotekama gdje je to potrebno. Navedeno polje imena uključujemo u druge datoteke pomoću ključne riječi `using` iza koje slijedi naziv polja imena, konkretno `using` `klasa_Student`.

```
using System;
using klasa_Student;

namespace Klasa_Student
{
    class Program
    {
        static void Main(string[] args)
        {
            Student Marko = new Student();
        }
    }
}
```

Vratimo se klasi `Student` i navedimo primjer deklaracije podatkovnih članova u datoteci *Student.cs*:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace klasa_Student
{
    class Student
    {
        public string ime;
        public string prezime;
        public string OIB;
        public string JMBAG;
        public string datum_rodenja;
        public string prebivaliste;
        public string studij;
        public int godina;
        public float visina;
        public float tezina;
        public List<string> predmeti;
        public List<int> ocjene;
    }
}
```


Klase mogu imati pravo pristupa `public` ili `internal`, pri čemu je zadano `internal`. Unutarnje klase su dostupne samo unutar komponente koja ju definira, unutar jednog projekta. Klase koje su definirane kao javne, ne moraju nužno dati pristup svim svojim podatkovnim i funkcijskim članovima drugim dijelovima programa, pristup im se i dalje može posebno definirati.

Primijetite da ne koristimo riječ `static` u deklaraciji podatkovnih članova klase `Student`. Varijable koje bi bile definirane kao `static` odnosile bi se na cijeli tip, tj. klasu. To znači da varijable koje su određene kao `static`, ne ovise o objektu, već je to zajednička odrednica svih objekata klase. Kada bismo željeli znati ukupni broj svih studenata, mogli bismo dodati cjelobrojnu varijablu određenu kao `static` koja bi predstavljala ukupni broj studenata, i pri svakom pozivu konstruktora novog studenta povećati ju za jedan. Odmah je jasno da npr. ime ne smije biti tipa `static`, jer svaki student (objekt klase `Student`) ima svoje ime.

Konstruktor

Kako bismo kreirali/inicijalizirali objekte neke klase koristimo konstruktor. Konstruktor je specijalna metoda/funkcija koja mora imati isto ime kao klasa i ne smije imati povratni tip. Sve klase imaju konstruktor. Ako ne definirate konstruktor sami, jezik C# će ga kreirati, ali tada ne možete postaviti inicijalne vrijednosti varijabli odmah, tj. podatkovnih članova klase. Konstruktori daju fleksibilnost, ali i štede vrijeme. Npr. ako ne definiramo konstruktor, trebali bismo za svaki objekt klase inicijalizirati sve podatkovne članove "ručno". Klasa može imati više definicija konstruktora.

Npr. za klasu `Student` želimo dodati novi objekt, tj. novog studenta. Bez definiranog konstruktora to može izgledati ovako (za primjer uzmimo prva četiri podatkovna člana klase `Student`):

```
Student s1 = new Student();
s1.ime = "Marko";
s1.prezime = "Maric";
s1.OIB = "12345678123";
s1.JMBAG = "0307019999";
```

Dok s pravilno definiranim konstruktorom koji inicijalizira podatkovne članove klase, studenta možemo kreirati pomoću naredbe (za primjer uzmimo prva četiri podatkovna člana):

```
Student s1 = new Student("Marko", "Maric", "12345678123", "0307019999");
```

Definicija klase `Student` sadrži brojne podatkovne članove pa korištenje konstruktora ima vidnu prednost. Definicija konstruktora klase `Student`:

```
public Student(string i, string p, string oib, string jmbag, string date,
string preb, string st, int g, float v, float t, List<string> pr, List<int> o)
{
    ime = i;
    prezime = p;
    OIB = oib;
    JMBAG = jmbag;
    datum_rodenja = date;
    prebivaliste = preb;
    studij = st;
    godina = g;
    visina = v;
    tezina = t;
    predmeti = pr;
    ocjene = o;
}
```

Dakle, objekt klase Student možemo kreirati pozivom konstruktora:

```
Student s1 = new Student("Marko", "Maric", "12345678123", "0307019999",  
    "01/01/2000", "Virovitica", "Racunarstvo", 2, 179, 75, predmeti_s1, ocjene_s1);
```

Zbog definicije konstruktora, program zna da je ime objekta klase student Marko, prezime Maric, OIB 12345678123, JMBAG 0307019999, ...

Prije poziva konstruktora potrebno je kreirati listu predmeta i listu ocjena. Bolji pristup bi nalagao da predmeti budu objekti klase Predmet, ali tada bismo imali objekte klase Predmet unutar klase Student, pa zbog jednostavnosti u ovom primjeru koristimo liste za predmete i ocjene.

Destruktor

Destruktor je metoda/funkcija koja se koristi za oslobađanje memorije prilikom uklanjanja objekata klase. Destruktor je jedinstven unutar klase, tj. ne može biti više različitih destruktora unutar jedne klase. Nema povratni tip, a ima isto ime kao klasa. Sadrži prefiks tilda ~ po kojemu ga prepoznavamo i definiramo. Destruktor ne može primiti argumente i ne može se naslijediti od neke druge klase. Načelno se poziva po završetku izvođenja programa.

Općenito ima oblik:

```
~Klasa()  
{  
    // naredbe prilikom poziva destruktora  
}
```

Često želimo znati ako je destruktor pozvan:

```
~Klasa()  
{  
    Console.WriteLine("Poziv destruktora.");  
}
```

Za klasu Student destruktor se definira na sljedeći način:

```
~Student()  
{  
    Console.WriteLine("Poziv destruktora.");  
}
```

Prava pristupa

Prava pristupa određuju vidljivost članova klase, podatkovnih i funkcijskih. Postoje prava tipa **public**, **private**, **protected**, **internal**, **protected internal**, **private protected**. Često želimo zaštititi direktan pristup podatkovnim članovima klase i tada koristimo **private** ili **protected** pravo pristupa nad određenim podatkovnim članovima, a pristupa se tim podacima pomoću funkcijskih članova klase. Npr. ne bismo željeli omogućiti slučajnu ili namjernu promjenu OIB-a ili JMBAG-a bilo gdje u programu za studenta svakom korisniku, jer je to opasno. Pravo pristupa **protected** vezano je uz paradigmu nasljeđivanja i o tome će biti više kasnije na predmetu. Dakle, ako želimo zaštititi podatkovne članove od promjena u dijelovima programa gdje to nije poželjno koristimo pravo pristupa **private**.

Funkcijski članovi klasa

Klase uz podatkovne članove imaju i funkcijske članove u svojoj definiciji. Ako se prisjetimo predmeta Osnove programiranja, strukture nisu imale funkcijske članove. Morali smo pisati posebno funkcije izvan definicije strukture koje su radile sa strukturama, tj. instancama struktura. U klasama funkcije možemo direktno pozivati nad objektima klase, što je u praksi iznimno važno. Npr. ne želimo da nekoj važnoj identifikacijskoj varijabli imamo direktan pristup, točnije ne želimo dati mogućnost manipulacije vrijednosti izvan definicije klase kako bi se izbjegle slučajne i nemjerne pogreške. Ako je ta varijabla deklarirana kao `private` ili `protected`, nemožemo joj pristupiti npr. u funkciji `main()` ili nekoj drugoj funkciji izvan definicije klase. Kako bismo onda mogli npr. ispisati tu varijablu ili pretraživati po njoj? Jedan od načina je pomoću funkcijskog člana.

Klasa `Student` je imala sve varijable deklarirane kao `public`. To znači da će se za objekt `s1` klase `Student` u program `Program.cs` u funkciji `main()` izvršavanjem dijela programa:

```
Student s1 = new Student("Marko", "Maric", "12345678123", "0307019999", "01/01/2000", "Virovitica", "Racunarstvo", 2, 179, 75, predmeti_s1, ocjene_s1);
```

```
Console.WriteLine(s1.JMBAG);
```

Ispisati na ekran 0307019999. Nadalje, izvršavanjem

```
s1.JMBAG = "0000000000";  
Console.WriteLine(s1.JMBAG);
```

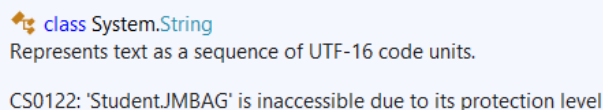
Podatkovni član `JMBAG` objekta `s1` postaje `0000000000`, što se vidi ispisivanjem na ekran nakon pridruživanja. `JMBAG` je jedinstveni identifikacijski broj svakog studenta i ne bismo željeli da ga se može tako jednostavno mijenjati. Kako bismo spriječili ovu situaciju postavimo u definiciji klase podatkovni član `JMBAG` da bude `private string` `JMBAG`; Pokušamo li sada izvršiti dio koda:

```
Student s1 = new Student("Marko", "Maric", "12345678123", "0307019999", "01/01/2000", "Virovitica", "Racunarstvo", 2, 179, 75, predmeti_s1, ocjene_s1);
```

```
Console.WriteLine(s1.JMBAG);
```

Odmah je vidljivo da ne možemo pristupiti `JMBAG`-u.

```
Console.WriteLine(s1.JMBAG);
```



Kako bismo mogli "vidjeti" podatkovni član `JMBAG` objekata izvan definicije klase možemo napisati jednostavnu funkciju koja će biti `public` i vraća `string` (`JMBAG`)

```
public string VратиJMBAG()  
{  
    return JMBAG;  
}
```

Sada u funkciji `main()` možemo ispisati `JMBAG` studenta `s1` na sljedeći način:

```
Console.WriteLine(s1.VратиJMBAG());
```

Dakle, pozvali smo funkcijski član klase `Student` koji se naziva `VratiJMBAG()` nad objektom `s1`. Iz ovog primjera vidimo da funkcijskim članovima pristupamo jednako kao i podatkovnim.

Sada ne možemo mijenjati podatkovni član JMBAG studenta `s1` npr. u funkciji `main()` u nekoj drugoj datoteci:

```
s1.JMBAG = "0307010000";
```



class System.String

Represents text as a sequence of UTF-16 code units.

CS0122: 'Student.JMBAG' is inaccessible due to its protection level

Show potential fixes (Alt+Enter or Ctrl+.)

Nekoliko drugih primjera funkcijskih članova klase `Student`:

1. Ispiši ime i prezime studenta

```
public void IspisiIP()
{
    if(ime == "" || prezime == "")
    {
        Console.WriteLine("Nedostaju podaci!");
    }
    else
    {
        Console.WriteLine(ime + " " + prezime);
    }
}
```

- U slučaju da je ime ili prezime prazan string, ispisujemo poruku upozorenja, a inače se ispisuje ime i prezime odvojeni razmakom.
- Poziv funkcije `IspisiIP()` nad objektom `s1`:

```
s1.IspisiIP();
```

2. Ispis svih predmeta željenog studenta

```
public void IspisiPredmete()
{
    if(predmeti.Count == 0)
    {
        Console.WriteLine("Student nema upisanih predmeta!");
    }
    else
    {
        foreach (string s in predmeti)
        {
            Console.WriteLine(s);
        }
    }
}
```

- U slučaju da je lista predmeta prazna, ispisujemo prikladnu poruku, a inače ispisujemo sve predmete željenog studenta.
- Poziv funkcije `IspisiPredmete()` nad objektom `s1`:

```
s1.IspisiPredmete();
```

3. Izračunaj prosjek ocjena studenta

```
public float ProsjekOcjena()
{
    if (ocjene.Count > 0)
    {
        float sum = 0;
        foreach (int o in ocjene)
        {
            sum += o;
        }
        return sum / ocjene.Count;
    }
    else
    {
        Console.WriteLine("Student nema upisanih ocjena!");
        return 0;
    }
}
```

- U slučaju da student nije još položio niti jedan predmet, ispisat će se poruka Student nema upisanih ocjena! i funkcija vraća 0, a inače će funkcija vratiti prosječnu ocjenu studenta nad kojim pozivamo funkciju.
- Primjeri poziva funkcije `ProsjekOcjena()` nad objektom `s1`:

```
float prosjek = s1.ProsjekOcjena();
Console.WriteLine(s1.ProsjekOcjena());
```

4. Promijeni JMBAG

```
public void PromijeniJMBAG(string s)
{
    JMBAG = s;
}
```

- Iako je podatkovni član JMBAG privatni, možemo ga promijeniti koristeći funkcijski član klase Student.
- JMBAG je specifičan niz brojeva i poželjno je provjeriti zadovoljava li unos `s` kriterije kako bi bio valjani JMBAG, npr., duljina, svi znakovi trebaju biti znamenke, svaka ustanova ima fiksni dio JMBAG-a i sl.

Zadatak 5.

Testirajte opisan način definicije klase Student u aplikaciji MS Visual Studio. Dodajte klasu Student u novu datoteku Student.cs, navedite podatkovne članove, konstruktor, destruktora, funkcijske članove. Funkciju `public void PromijeniJMBAG(string s)` doradite tako da provjerite zadovoljava li unos `s` kriterije da postane JMBAG. Kreirajte barem jednog studenta i testirajte pozive funkcijskih članova u funkciji `main()` u programu Program.cs.

Zadatak 6.

Kreirajte svoju klasu iz zadatka 1. i deklarirajte joj podatkovne članove iz zadatka 2. Napišite definicije konstruktora, destruktora i napišite definicije barem 5 funkcijskih članova klase. Kreirajte barem 3 objekta klase u funkciji `main()` u datoteci Program.cs i testirajte



funkcionalnost funkcijskih članova klase na objektima koje ste kreirali. Testirajte prava pristupa `public` i `private` nad podatkovnim i funkcijskim članovima. Za Vašu klasu obrazložite koji podatkovni i/ili funkcijski članovi trebaju imati pravo pristupa `private`.

Zadatak 7. Dodatni zadatak

- a) Doradite klasu `Student` tako da svaki predmet bude objekt klase `Predmet`. Klasa `Predmet` će sadržavati podatkovne članove `ID`, `naziv`, `nositelj`, `ECTS`, `ocjena`. U slučaju da student nije položio predmet, `ocjena` je jedan.
- b) Kreirajte listu studenata s barem 5 članova.
- c) Ispišite sve studente iz liste u obliku:
 1. Student: Ime i Prezime,
OIB:
JMBAG:
Prebivaliste:
Studij i godina:
Predmeti i ocjene:
 Predmet1 – ocjena1
 Predmet2 – ocjena2
 ...Ako predmet nije položen, umjesto ocjene (1) ispisati poruku „Predmet nije položen!“
2. Student: Ime i Prezime ...
- d) Napišite funkciju koja prima listu studenata i sortira ih po prosjeku položenih predmeta. Ako studenti imaju isti prosjek, prednost ima student koji je položio više ECTS-a. Funkcija ispisuje sortiranu listu studenata s njihovim podacima i vraća sortiranu listu. Možete li sortirati studente bez korištenja funkcija za sortiranje?
- e) Nakon sortiranja iz d) dijela zadatka, izradite rang listu studenata koji su položili barem 30 ECTS bodova.