

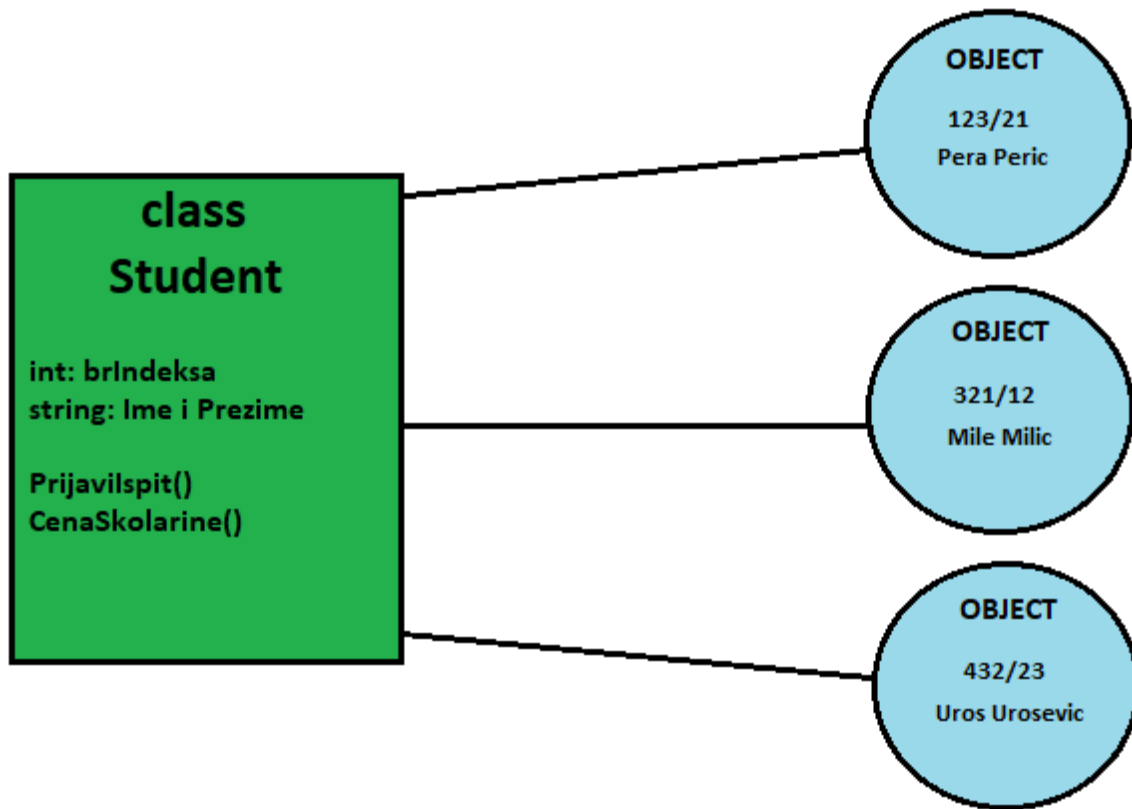
Машински факултет  
Универзитета у Београду

Објектно оријентисана парадигма

Професор: Горан Ђурић  
Група: 1.

# Uvod

Rani programski jezici su bili takozvani Proceduralni, zato što bi programer definisao skup procedura, koje bi računar potom obrađivao. Proceduralno programiranje koristi listu instrukcija da kaže kompjuteru šta da radi korak po korak. Objektno orijentisana paradigma koristi objekte kao osnovu za projektovanje i izradu računarskih programa. Svaka klasa ima u sebi attribute i metode (funkcije), a objekti instanciraju tu klasu i postavljaju svoje vrednosti za te attribute i metode (**slika 1.**). Postoje četiri glavna stuba OOP-a, a to su kapsulacija, nasleđivanje, abstrakcija i polimorfizam. Problem rešavamo tako što identifikujemo objekte i zaključujemo kako se ti objekti ponašaju i utiču jedan na drugi.



Slika 1. Primer klase i instanciranih objekata

# Projekat

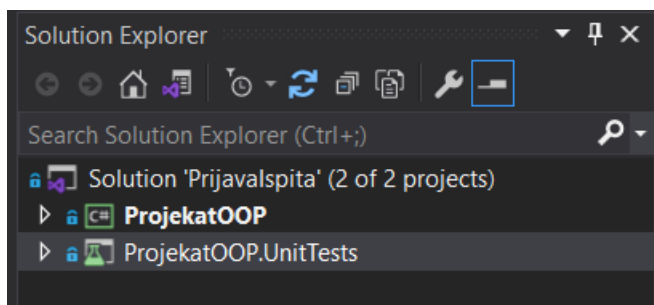
## Opis problema:

Studenti žele da prijave ispit nakon isteka regularnog roka za prijavljivanje ispita. Ispit se održava određenog datuma. Student može da bude u statusu “budžet” i “samofinasiranje”. Cena prijave ispita se formira sabiranjem osnovne cene prijave ispita i proizvoda broja dana zakašnjenja i cene za kašnjenje. Osnovna cena za budžetske studente je 1200 dinara, a za samofinansirajuće je 1300 dinara. Cena za dan kašnjenja je jedinstvena i iznosi 50 dinara. Ispit nije moguće prijaviti ako je kašnjenje veće od 9 dana ili ako je već nastupio dan ispita.

## Rešenje problema:

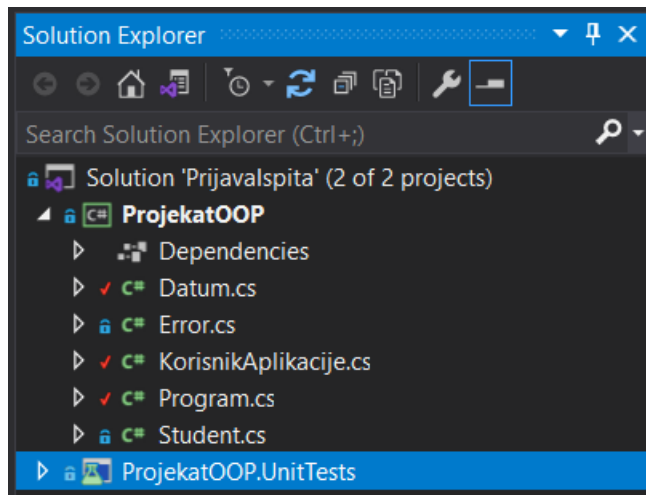
Koristili smo programski jezik C# i razvojno okruženje „Visual Studio“.

Kreirali smo konzolni projekat i projekat za „MS Test” da bi mogli da testiramo kritične metode u našoj aplikaciji (Unit tests) (slika 2.).



Slika 2. Projekti

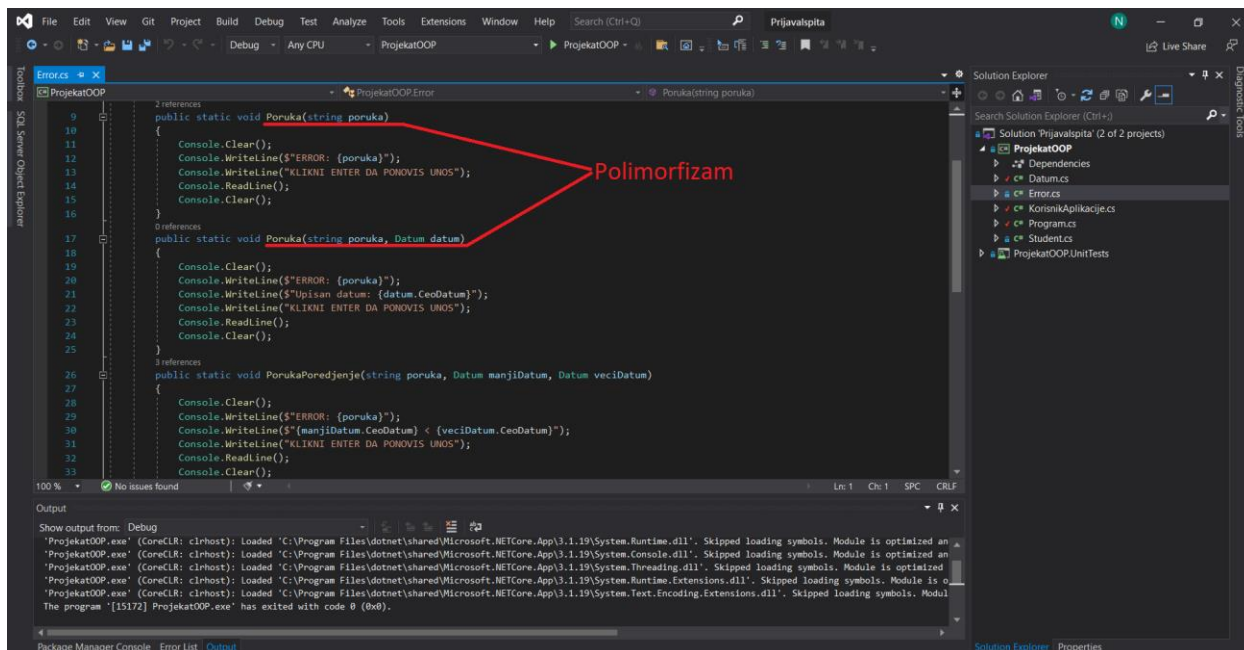
Konstatujemo da ćemo imati 4 nove klase i klasu Program.cs koja je osnovna klasa koja sadrži main() metodu odakle počinje da se izvršava sam program.



Slika 3. Klase u projektu

## 1. Klasa Error

Ova klasa je klasa koja sadrži static metode (u aplikaciji pozivamo imeKlase.metoda()), sadrži sve error poruke koje će se korisniku aplikacije izbaciti pri pogrešnom unosu traženih unosa kroz aplikaciju (slika 4.).

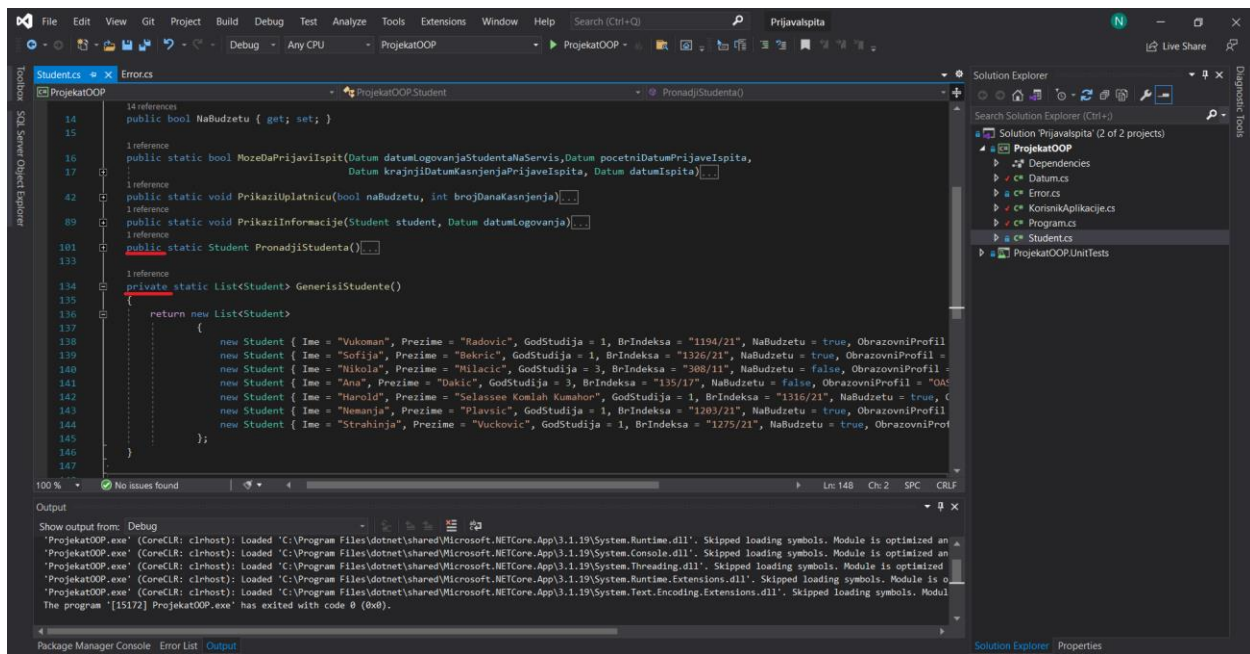


Slika 4. Polimorfizam

Ovde se suočavamo sa jednim od stubova OOP, a to je polimorfizam. Imamo dve metode sa istim nazivom „Poruka” ali one se razlikuju po vrednostima koje primaju, jedna prima samo string, dok druga prima string i Datum (objekat). Ovo se naziva i method overloading.

## 2. Klasa Student

Klasa Student predstavlja podatke koji su vezani za svakog studenta (Ime, Prezime, broj indeksa...), zajedno sa static metodama koje će nam zatrebati tokom rada aplikacije. U ovoj klasi ima posebna private metoda koja se naziva GenerisiStudente(). Mi smo sa ovom metodom oponašali dolazak informacija iz baze podataka sa već postojećim studentima i tako obezbedili metodi PronadjiStudenta da izbacii traženog studenta (slika 5.).

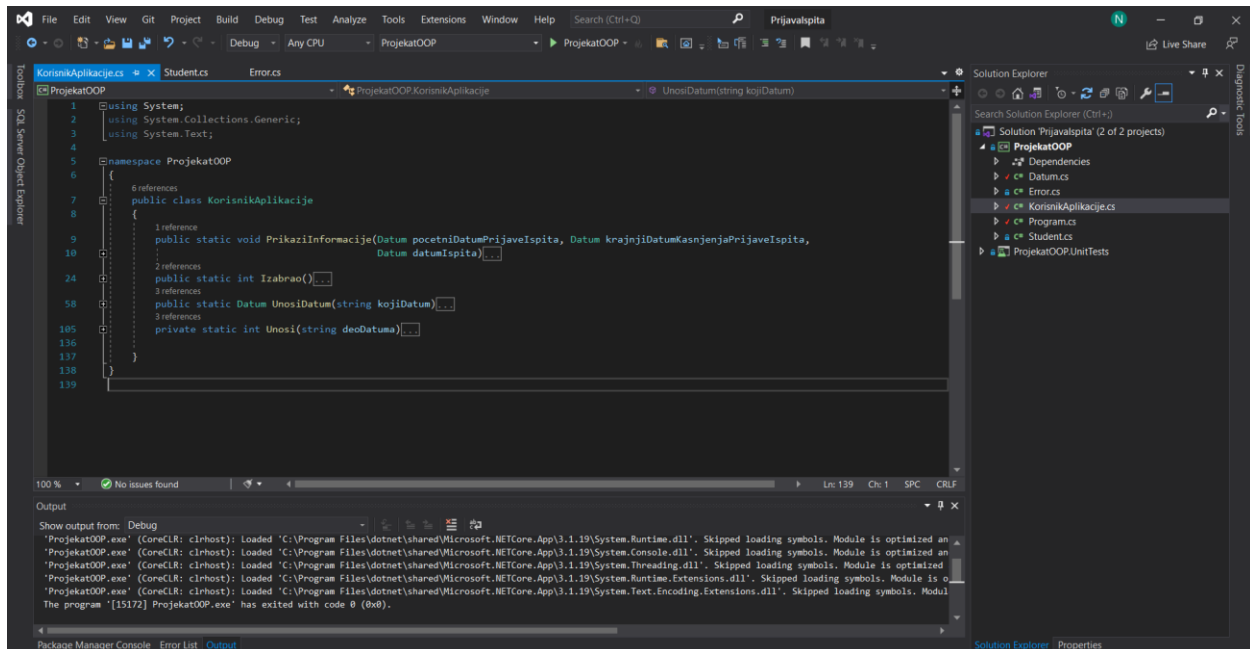


Slika 5. Kapsulacija i generisani studenti

Još jedan detalj je „private” na početku metode GenerisiStudente(), on predstavlja jedan od stubova OOP, a to je kapsulacija. Private, public, protected... se inače nazivaju „access modifiers” (modifikatori pristupa). Private znači da samo naša klasa može da koristi tu metodu (ili atribut ako tako postavimo).

### 3. Klasa KorisnikAplikacije

Ova klasa pomaže korisniku pri interakciji sa aplikacijom tako što obrađuje svaki vid korisnikovog unosa, da li korisnik unosio datum, broj indeksa studenta, izbor koji korak želi da preduzme. Možda bi mogao da se smisli bolji naziv klase ali zbog logike njenog postojanja ona je stavljena ovako, lako je razumeti šta radi kada se pozove sa metodom, primera radi KorisnikAplikacije.UnosiDatum(), vrlo je lako razumeti šta će uraditi (slika 6.).



Slika 6. Korisnik Aplikacije

Takođe ova klasa ima metodu koja će predstaviti informacije na početku, slično kao neki korisnički interfejs (slika 7.).

### **\*Slika 7. Metoda PrikaziInformacije()**

```
1 reference
public static void PrikaziInformacije(Datum pocetniDatumPrijaveIspita, Datum krajnjiDatumKasnjenjaPrijaveIspita,
                                         Datum datumIspita)
{
    Console.Clear();
    Console.WriteLine("-----");
    Console.WriteLine($"Datum pocetka prijave ispita:           |{pocetniDatumPrijaveIspita.CeoDatum}");
    Console.WriteLine($"Krajnji datum prijave ispita (sa kasnjenjem): |{krajnjiDatumKasnjenjaPrijaveIspita.CeoDatum} ");
    Console.WriteLine($"Datum ispita iz OOP:                       |{datumIspita.CeoDatum}");
    Console.WriteLine("-----\n");
    Console.WriteLine("(1)--> Postavi datum pocetka prijave ispita");
    Console.WriteLine("(2)--> Postavi datum ispita iz OOP");
    Console.WriteLine("(3)--> Udji kao student");
    Console.WriteLine("(0)--> Iskljuci aplikaciju");
    Console.Write("UNESI BROJ OPCIJE KOJU ZELIS:");
}
```

Slika 7. Metoda PrikaziInformacije(...)

Takođe jedna od glavnih metoda koja će se koristiti u aplikaciji je UnosiDatum(). Datum će biti vraćen (return-ovan) samo ako korisnik unese tačno predviđen format (slika 8.).

```
3 references
public static Datum UnosiDatum(string kojiDatum)
{
    int dan, mesec, godina;
    Datum novDatum = new Datum();

    bool tacanFormatUnetogDatuma = false;

    while (tacanFormatUnetogDatuma == false)
    {
        Console.WriteLine($"Unesi datum {kojiDatum}:");
        Console.WriteLine("(dd/mm/yyyy)");
        dan = Unosi("dan");

        Console.WriteLine($"Unesi datum {kojiDatum}:");
        Console.WriteLine($"({dan}/mm/yyyy)");
        mesec = Unosi("mesec");

        Console.WriteLine($"Unesi datum {kojiDatum}:");
        Console.WriteLine($"({dan}/{mesec}/yyyy)");
        godina = Unosi("godinu");

        try
        {
            novDatum = new Datum(dan, mesec, godina);
        }
        catch (Exception)
        {
            novDatum = new Datum();
        }

        if (novDatum.Godina == 0001)
        {
            Error.Poruka("Uneli ste pogresan format datuma!");
        }
        else if (novDatum.Godina < 2021)
        {
            Error.PorukaPoredjenje("Uneti datum je manji od danasnjeg datuma!", novDatum, Datum.GenerisiDanasnjiDatum());
        }
        else
        {
            tacanFormatUnetogDatuma = true;
        }
    }
    return novDatum;
}
```

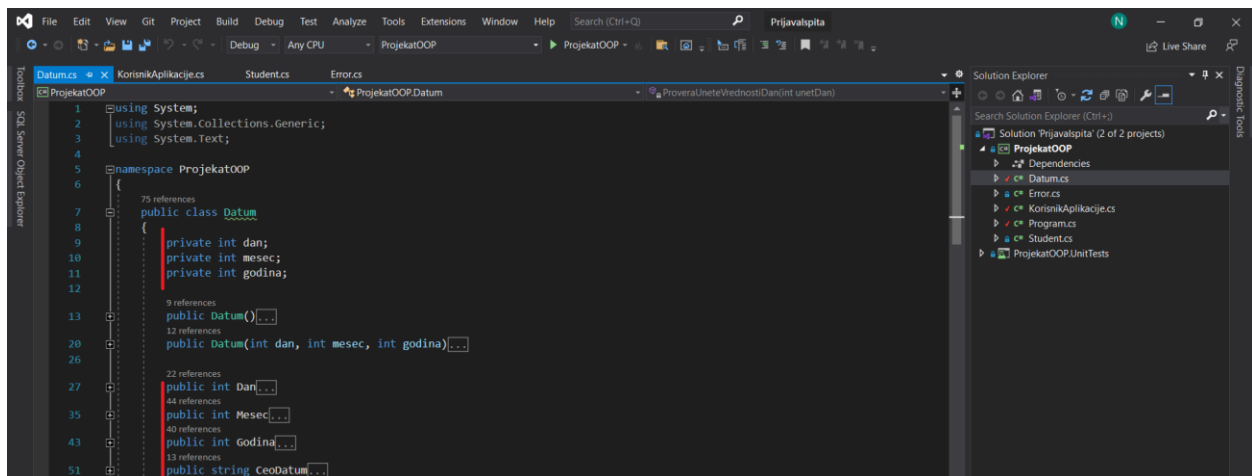
Slika 8. Metoda UnosiDatum()



## 4. Klasa Datum

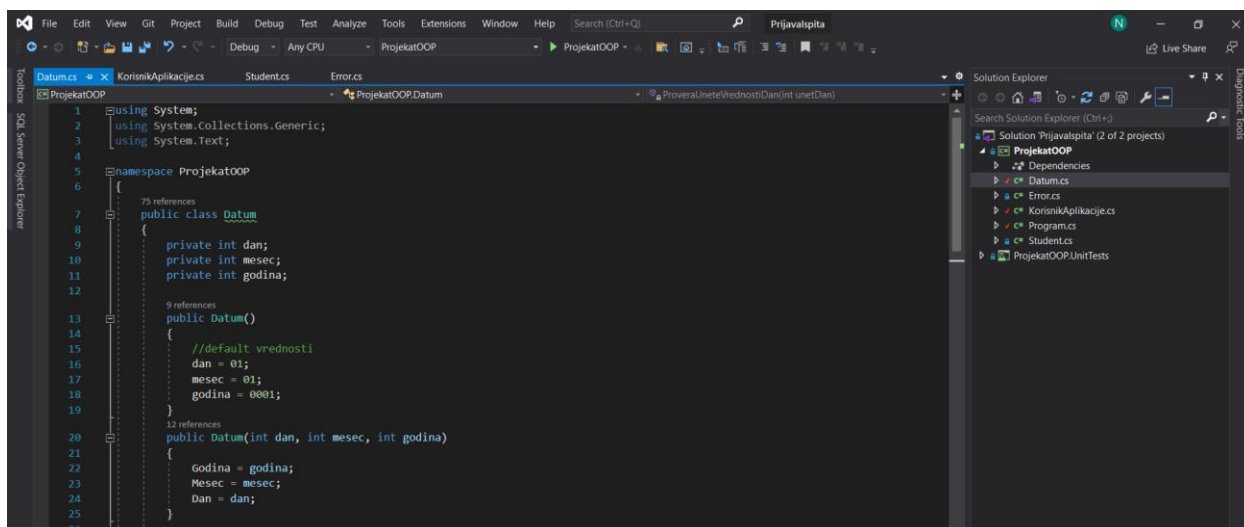
Ovo je najkomplikovanija klasa ujedno i srce ove aplikacije. Moglo je da se koristi već postojeća biblioteka DateTime, ali bila je suština napraviti od 0 sličnu klasu.

Prvo što treba da uočimo to je da ova klasa ima 3 sakrivena (private) atributa: dan, mesec, godinu. Ona će nositi ključne vrednosti našeg datuma. Takođe imamo i 4 property-a (svojstva), koja imaju get i set metode, pomoću kojih pristupamo i menjamo vrednosti naših private atributa. Ovo je vrlo bitno zbog samog unosa, jer su vrednosti ograničene (primer: korisnik aplikacije ne može da unese 30. Dan ako je 2. Mesec jer februar ima 28 dan ili u prestupnoj godini 29) (slika 9.).



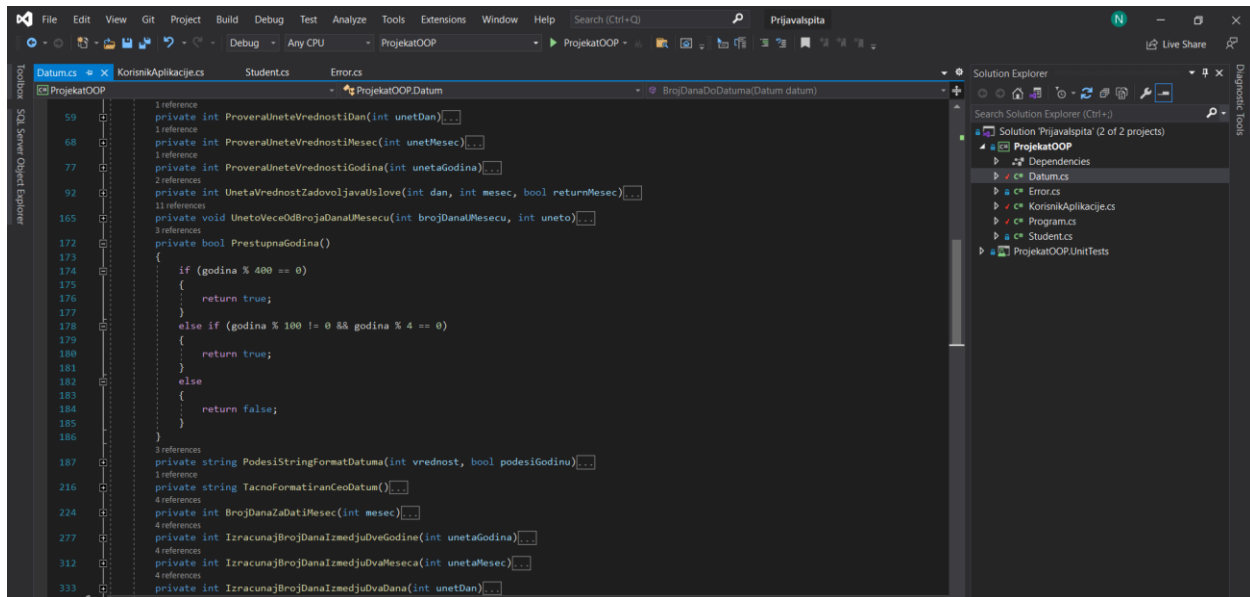
Slika 9. Atributi i property Datum klase

Takodje imamo 2 konstruktora klase. Konstruktor se odma izvršava čim se napravi novi objekat, tj instancira klasa. Primetićemo opet polimorfizam ili construtor overloading (u našem primeru imamo 2 overloada). Konstruktor koji nema unete vrednosti će podesiti default vrednosti za nas datum, dok konstruktor koji zahteva unos int vrednosti za dan, mesec i godinu će prvo uneti vrednosti za property Godinu pa Mesec i na kraju Dan, zato što unos dana zavisi od meseca i od godine (da li je godina prestupna ili ne i koliko dana može da ima mesec).



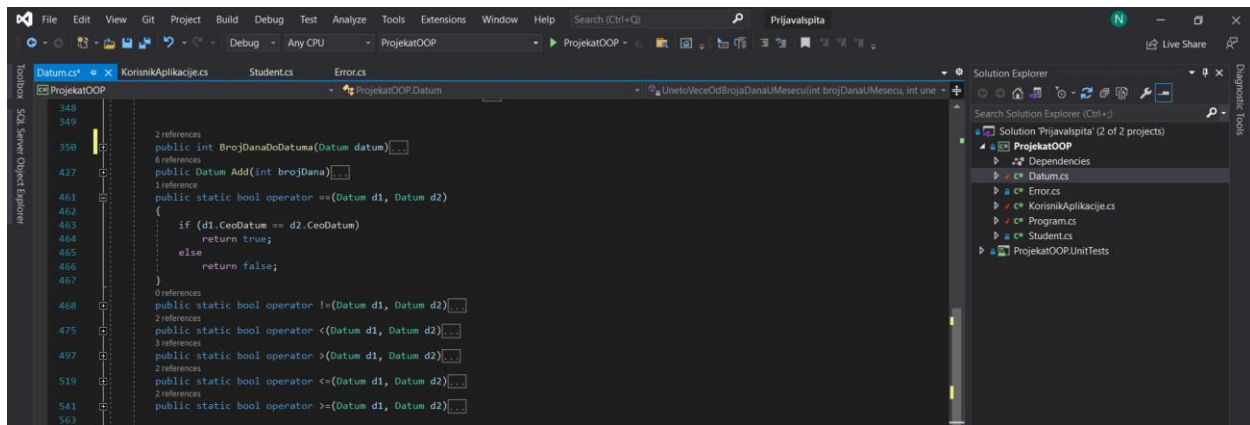
Slika 10. Konstruktori Datum klase

Ova klasa poseduje dosta private metoda koje nam pomažu da obradimo podatke, pokušavali smo što više da se držimo DRY(don't repeat your self) principa, tj da ne ponavljamo previše puta kod, nećemo zalaziti u detalje jer bi nam trebalo puno vremena da objasnimo svaku metodu. Ali jedna od zanimljivih i lakše objašnjivih je računanje prestupne godine (slika 11.).



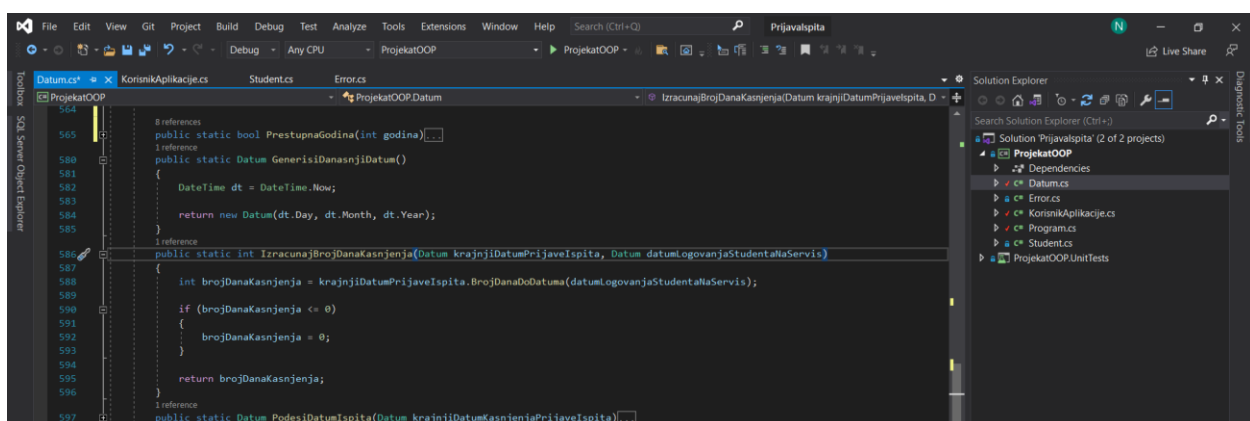
Slika 11. Private metode unuta Datum klase

Segmentno smo odvojili public metode koje su dostupne van naše klase, to su metode za računanje broja dana do nekog datuma, dodavanje broja dana na datum, static metoda koja vraća bool promenljivu u zavisnosti da li je uneta godina prestupna. Takodje tu su i operatori za naše objekte; ovo su posebne metode koje kada definisemo možemo da upoređujemo dva objekta (>, <, ==, <=, =) (slika 12.).



Slika 12. Public metode Datum klase

Poslednji segment su static metode, koje nam koriste da generisemo danasnji datum (pozivajući Datum.GenerisiDanasnjiDatum()), izračunavamo broj dana kašnjenja i podešavamo datum ispita(ova metoda može i da se premesti u KorisnikApplikacije ali zato što vraća Datum objekat mi smo je zadržali ovde).

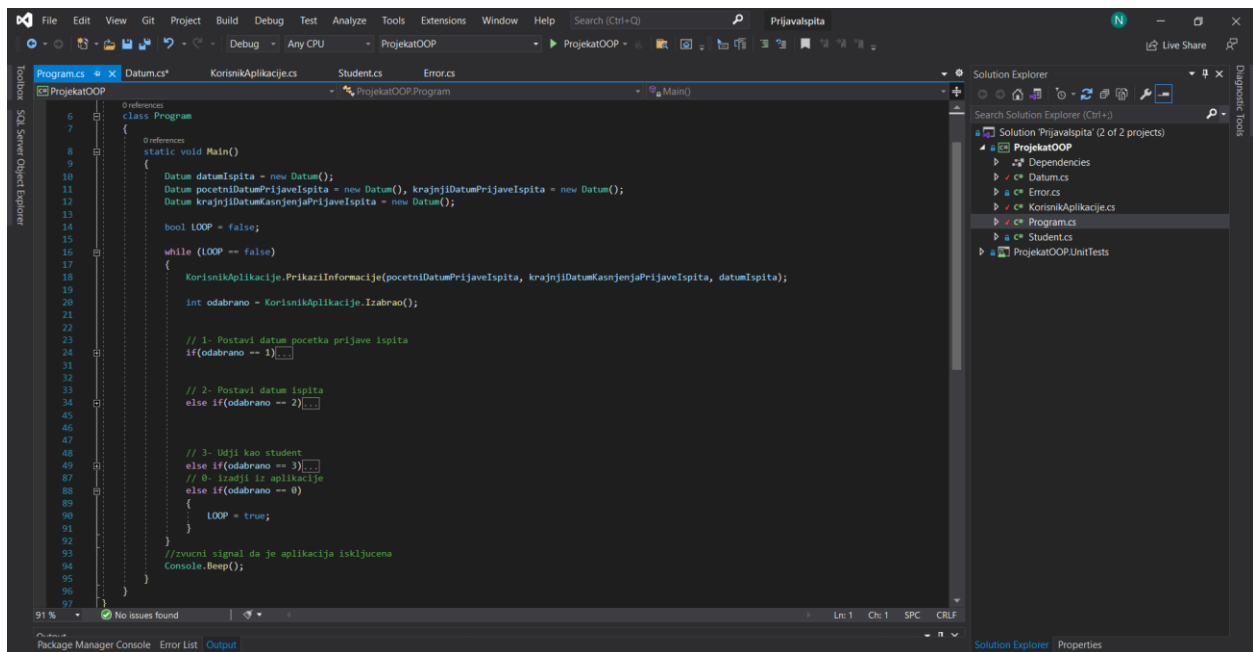


Slika 13. Static metode unutar Datum klase

## Program.cs

Kada smo objasnili funkcionalnost svih klasa, vreme je da objasnimo glavnu klasu i njenu metodu main() koja će koristiti naše klase.

Prvo ćemo objasniti sam kostur aplikacije (slika 14.):

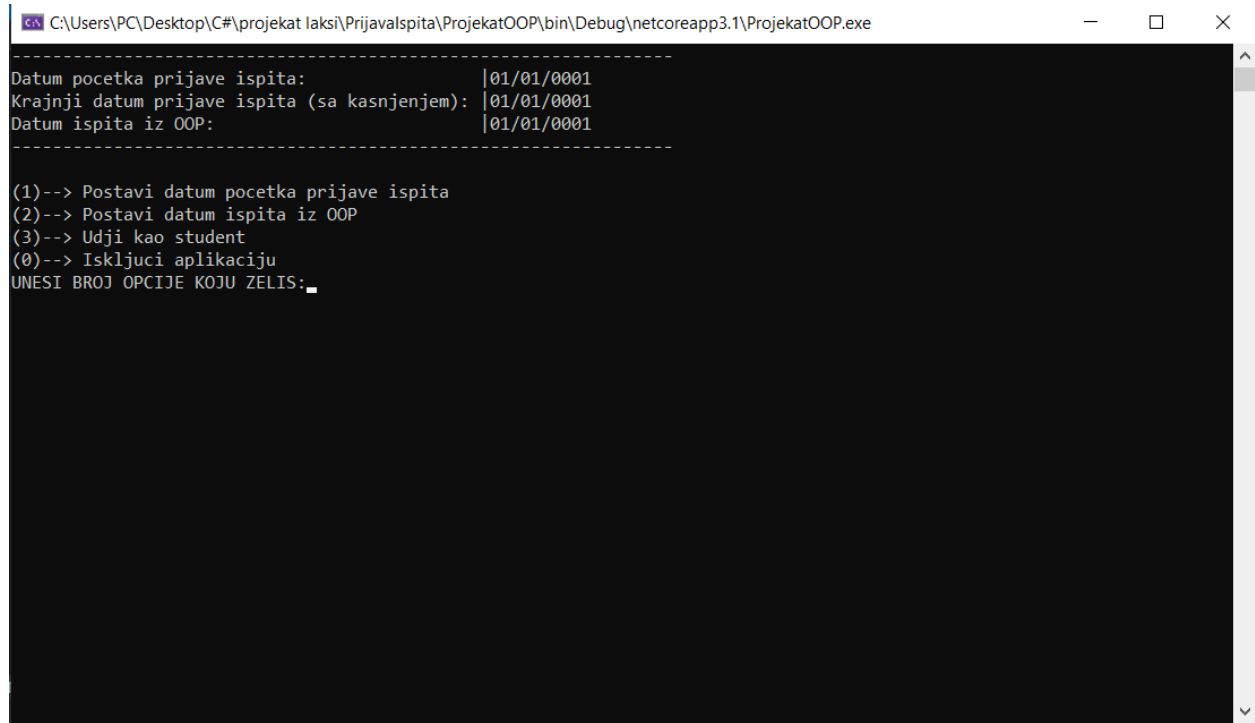


Slika 14. Main()

Prvo što aplikacija radi je deklarisanje default promenljivih objekata (Datuma) koje ćemo menjati u toku rada aplikacije.

Postavljamo LOOP koji će omogućiti da se aplikacija ne zaustavi dok mi ne budemo to želeli.

Sada počinje naša aplikacija sa prikazom i unosima. Čitajući prvu liniju koda u while bloku `KorisnikAplikacije.PrikaziInformacije(...)`, možemo da pretpostavimo šta će se desiti, prikazaće se glavni interfejs korisniku (slika 15.).



```
C:\Users\PC\Desktop\C#\projekat laksi\Prijavalspita\ProjekatOOP\bin\Debug\netcoreapp3.1\ProjekatOOP.exe

-----
Datum pocetka prijave ispita:      |01/01/0001
Krajnji datum prijave ispita (sa kasnjenjem): |01/01/0001
Datum ispita iz OOP:              |01/01/0001
-----

(1)--> Postavi datum pocetka prijave ispita
(2)--> Postavi datum ispita iz OOP
(3)--> Udji kao student
(0)--> Iskljuci aplikaciju
UNESI BROJ OPCIJE KOJU ZELIS: _
```

Slika 15. Prikaz glavnog interfejsa

Sledeća linija koda uzima podatke od korisnika, tj. korisnik bira koju funkciju želi i u zavisnosti od izbora biće usmeren na sledeći deo ili error poruku da je pogrešio sa unosom (treba napomenuti da je aplikacija u potpunosti otporna na greške (ako nađete grešku bili bi smo srećni da nam javite), tj. aplikacija se neće zaustaviti sa radom ako dođe do greške). Izbore smo definisali sa if i elsif blokovima. (slika 16.)

```
while (LOOP == false)
{
    KorisnikAplikacije.PrikaziInformacije(pocetniDatumPrijavaIspita, krajnjiDatumKasnjenjaPrijavaIspita, datumIspita);

    int odabrano = KorisnikAplikacije.Izabrao();

    // 1- Postavi datum pocetka prijave ispita
    if(odabrano == 1){...}

    // 2- Postavi datum ispita
    else if(odabrano == 2){...}

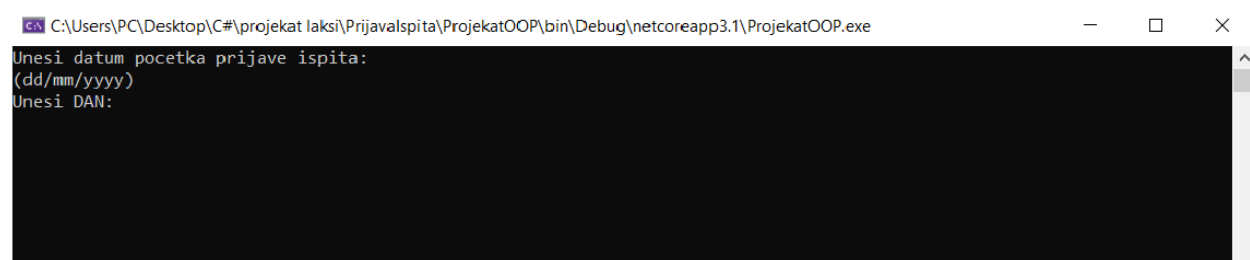
    // 3- Udji kao student
    else if(odabrano == 3){...}
    // 0- izađji iz aplikacije
    else if(odabrano == 0)
    {
        LOOP = true;
    }
}
```

Slika 16. Konstrukcija if i if else

Prve dve funkcije su vezane za unos datuma. U prvoj funkciji unosimo pocetni datum prijave ispita i automatski će nam se generisati i kranji datum prijave ispita (3 dana od pocetka prijave ispita) i krajnji datum kasnjenja prijave ispita (9 dana od krajnjeg datuma prijave ispita) i setovaće datum ispita na default da ne bi imali mogućnost da prevarimo program. U drugoj funkciji imamo definisanje datuma samog ispita, koji ne može da se podesi ako prethodno nismo definisali početni datum prijave ispita (slika 17.).

```
// 1- Postavi datum pocetka prijave ispita
if(odabrano == 1)
{
    pocetniDatumPrijaveIspita = KorisnikAplikacije.UnosiDatum("pocetka prijave ispita");
    krajnjiDatumPrijaveIspita = pocetniDatumPrijaveIspita.Add(3);
    krajnjiDatumKasnjenjaPrijaveIspita = krajnjiDatumPrijaveIspita.Add(9);
    datumIspita = new Datum(); //Ako podesimo 1 pa 2, pa onda opet udjemo u 1 a datum ispita nam bude manji od pocetnog
}

// 2- Postavi datum ispita
else if(odabrano == 2)
{
    if(pocetniDatumPrijaveIspita.Godina != 0001)
    {
        datumIspita = Datum.PodesiDatumIspita(krajnjiDatumKasnjenjaPrijaveIspita);
    }
    else
    {
        Error.Poruka("Prvo moras da postavis pocetak prijave ispita!");
    }
}
```



Slika 17. Prfa dva if bloka sa sličnim displejom



Kada smo podesili datume sada ulazimo kao student na servis (blok koda 3-if else). Prvo sto radimo to je simulacija ulaska studenta tako što unosimo datum(ovo može i sa DateTime.Now, ali zbog bolje simulacije stavljeno je ovako, da tačno možemo da odaberemo obseg). Sledeća linija koda je da upišemo broj indeksa studenta i da naša aplikacija pronadje tog studenta ako postoji (objašnjeno ranije za klasu student), i u zavisnosti od toga da li indeks postoji da udje u sledeći blok koda. (slika 18.)



```
// 3- Udji kao student
else if(odabrano == 3)
{
    Console.WriteLine("SIMULACIJA ULASKA STUDENTA");
    Datum datumLogovanjaStudentaNaServis = KorisnikAplikacije.UnosiDatum("ulaska studenta na servis"); // Ovak korak moze // da se zanmari u produkciji
    Student student = Student.PronadjiStudenta();

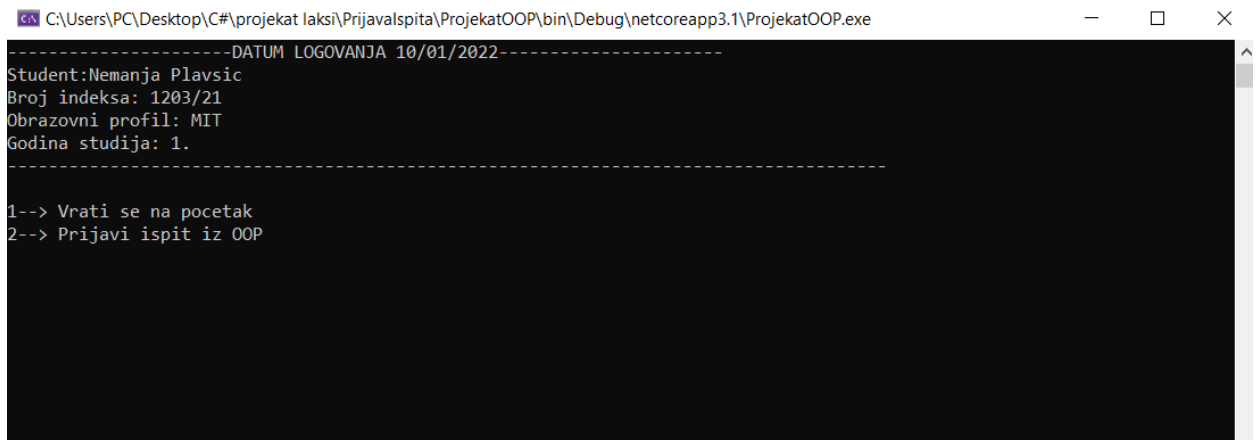
    if(student.BrIndeksa != null){...}
}
```

C:\Users\PC\Desktop\C#\projekat laksi\Prijavalspita\ProjekatOOP\bin\Debug\netcoreapp3.1\ProjekatOOP.exe

Unesi broj indeksa (u formi 123/21):

Slika 18. Udji kao student i prikaz ulaska

Kada tačno unesemo broj indeksa studenta ulazimo u sledeći loop koji će nam predstaviti glavni interfejs za studenta.(slika 19.)



```
-----DATUM LOGOVANJA 10/01/2022-----
Student:Nemanja Plavsic
Broj indeksa: 1203/21
Obrazovni profil: MIT
Godina studija: 1.
-----

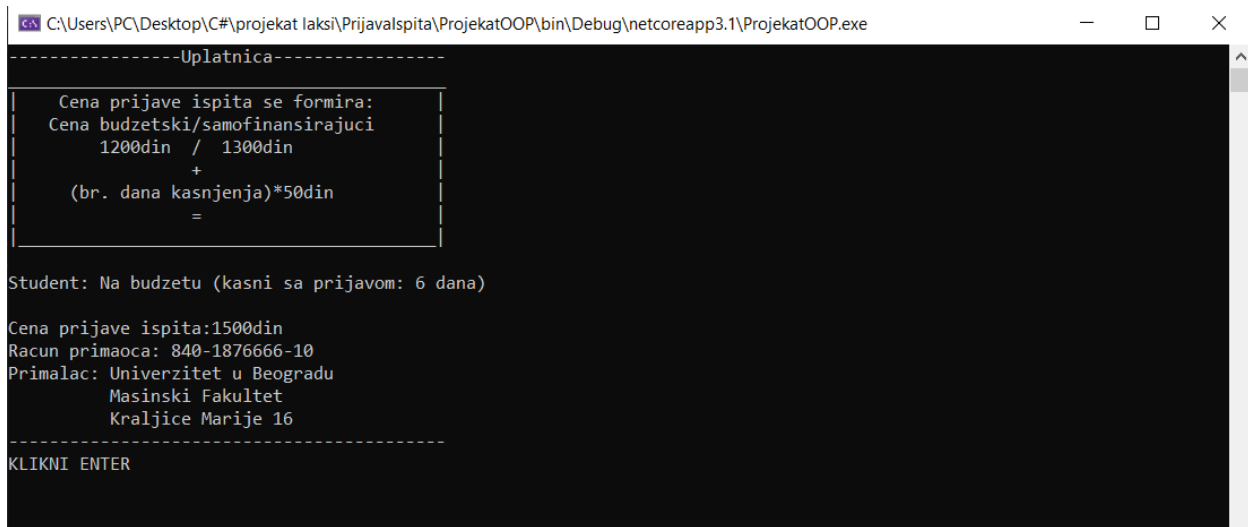
1--> Vрати se na pocetak
2--> Prijavi ispit iz OOP
```

Slika 19. Glavni interfejs za studenta

Student će imati samo dve opcije koje može da izabere:

1. Da se vrati na početak
2. U zavisnosti kada je ušao na servis(onaj datum logovanja koji smo simulirali) biće mu prikazana opicija da li može da prijavi ispit, da li je prošla prijava, da li je poranio ili da li želi da pirjavi na dan ispita

Ako student može da prijavi ispit, to znači da se ulogovao u opsegu početnog datuma prijave ispita i krajnjeg datuma kašnjenja prijave ispita. I odabirom na funkciju 2. ulazi u uplatnicu na kojoj piše kako mu je obračunata cena za ispit sa datom uplatnicom koju mora da popuni (slika 20.).



```
C:\Users\PC\Desktop\C#\projekat laksi\PrijavaIspita\ProjekatOOP\bin\Debug\netcoreapp3.1\ProjekatOOP.exe
-----Uplatnica-----
|
|  Cena prijave ispita se formira:
|  Cena budzetski/samofinansirajuci
|    1200din / 1300din
|      +
|    (br. dana kasnjenja)*50din
|      =
|
|-----|
Student: Na budzetu (kasni sa prijavom: 6 dana)
Cena prijave ispita:1500din
Racun primaoca: 840-1876666-10
Primalac: Univerzitet u Beogradu
          Masinski Fakultet
          Kraljice Marije 16
-----
KLIKNI ENTER
```

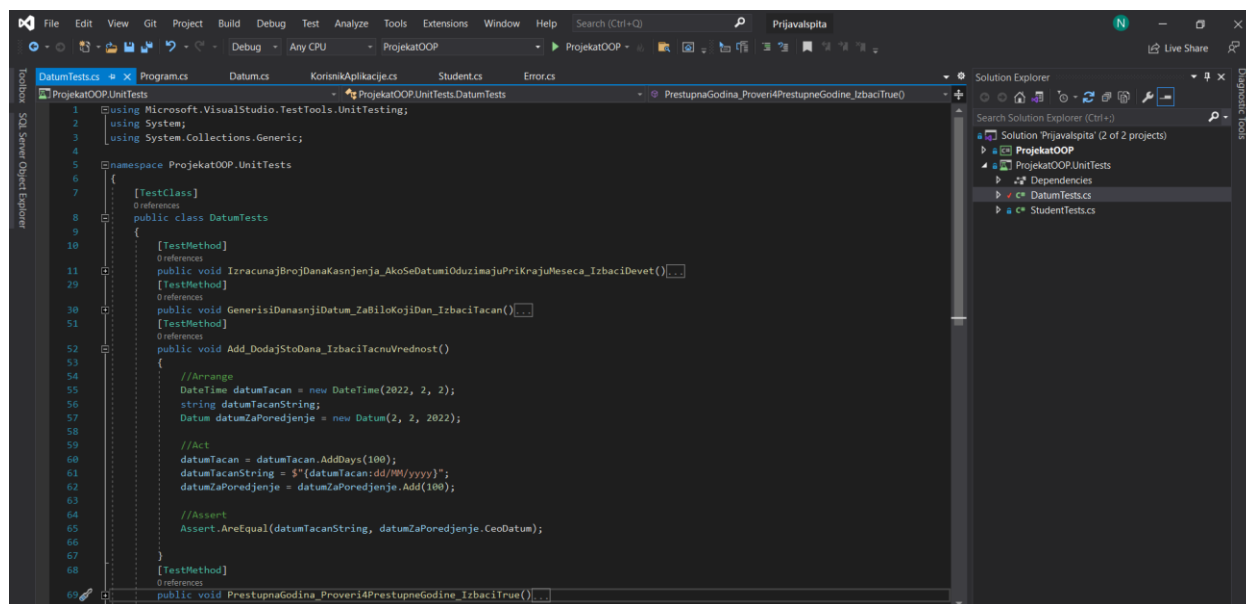
Slika 20. Uplatnica

Klikom enter vraćamo se na početni interfejs.

Izborom funkcije 0 program se gasi sa zvukom.

## Testovi

U našoj aplikaciji koristili smo „MS Tests”, sa ovom vrstom projekta koji nam omogućava Visual Studio možemo da testiramo kritične metode u našoj aplikaciji. Postoje 3 vrste testova, a to su unit testovi (testovi malih segmenata), integration testovi (testira aplikaciju sa externim promenljivima) i End-to-end testovi (pokretanje cele aplikacije kroz svoj interfejs). Svaki kod bi trebao da bude testiran, jer nekada naš kod može da dovede do greške u nekom odgovornom sistemu i da dovede do smrti nekog živog bića (na primer programiranje kočnice na kolima, ako tu naš kod zakaže čovek može da pogine). Testovi se pišu u klasi koja nosi naziv klase u kojoj se nalazi naša metoda koju želimo da testiramo i prefiks Tests (ImeKlase.Tests). Ta klasa će u sebi imati metode koje se testiraju i sve te metode su void. Metode se po konvenciji nazivaju ImeMetode\_Scenario\_OcekivanPonašanje. Kada testiramo metodu, nju delimo u 3 dela, Arrange, Act i Assert (AAA), u prvom delu Arrange pripremamo promenljive, u drugoj delu delujemo i u trećoj proveravamo da li smo dobili očekivanu vrednost. (slika 21.)



Slika 21. Testovi

## **Zaključak**

U našem projektu nismo koristili 2 stuba OOP-a, a to su abstrakcija i nasleđivanje, jer nam nisu ni bili potrebni. Možda bi ih uključili da smo problem rešavali na drugačiji način. Svaki problem ima više načina da se reši, usavršavanjem i praksom treba da dosegnemo nivo da bar najpribližnije znamo da je naš način dobar i da završava posao, a i da možemo lako nadograđuje aplikaciju i da se lako popravljaju bagovi ako do njih dodje.