.NET Okruženje

Vježba 2: Osnove C#

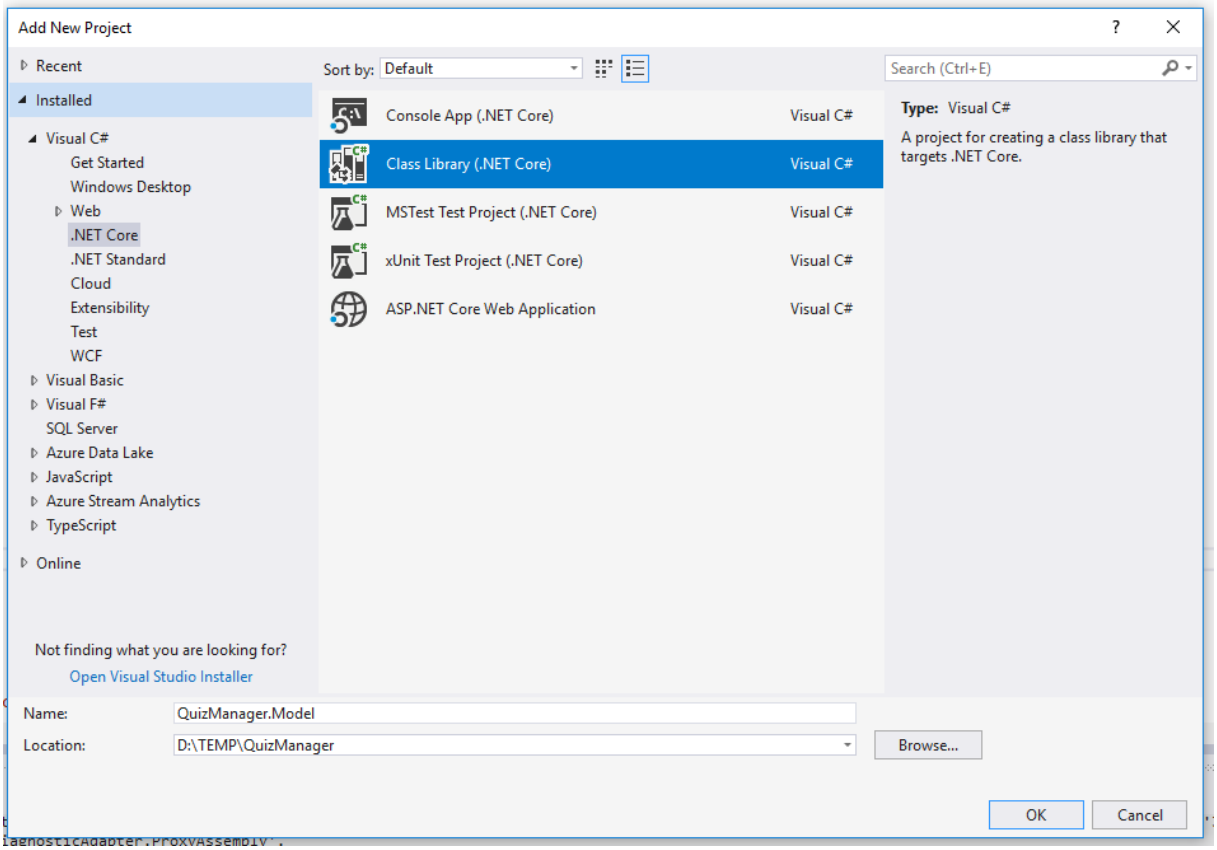
Sadržaj

Osnove C# jezika

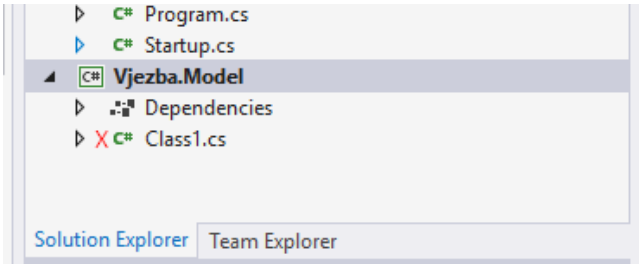
U ovom će se poglavlju prezentirati osnovni pojmovi i koncepti koji će biti korišteni na kolegiju.

Solution i projekti

U pravilu se svaka ozbiljnija aplikacija dijeli na nekoliko slojeva. Ti slojevi su najčešće prezentacijski sloj, model/business/services sloj i sloj za pristup bazi podataka (Data Access Layer ili kraće DAL sloj). Kod razvoja višeslojnih aplikacija, potrebno je uvesti i nezavisnosti između slojeva u „smjeru prema gore“, što znači da prezentacijski sloj može biti ovisan o DAL ili model sloju, ali DAL ili model sloj ne smiju biti ovisni o prezentacijskom sloju. U .NET okruženju se takva raspodjela najelegantnije postiže raspodjelom slojeva na projekte unutar rješenja (solutiona). Po uzoru na tu ideju, napravit ćemo novi projekt i nazvati ga Vjezba.Model. Novi projekt se dodaje odabirom opcije File -> Add -> New Project u izborniku:



Ako pogledamo solution explorer, novi projekt je kreiran i automatski mu je dodana nova klasa Class1, koju možemo slobodno obrisati:



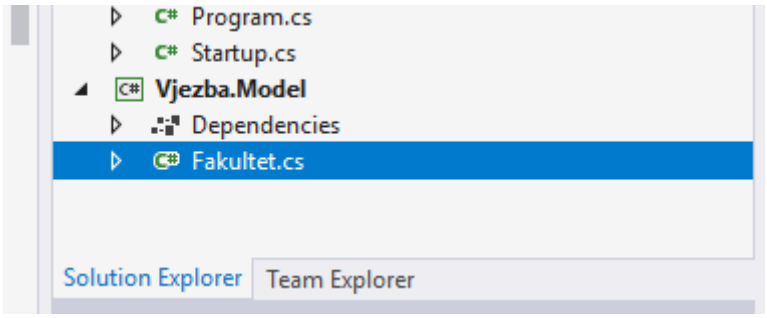
Ovaj projekt je već dodan ukoliko je sa source control servera povučena grana vjezba-02/setup.

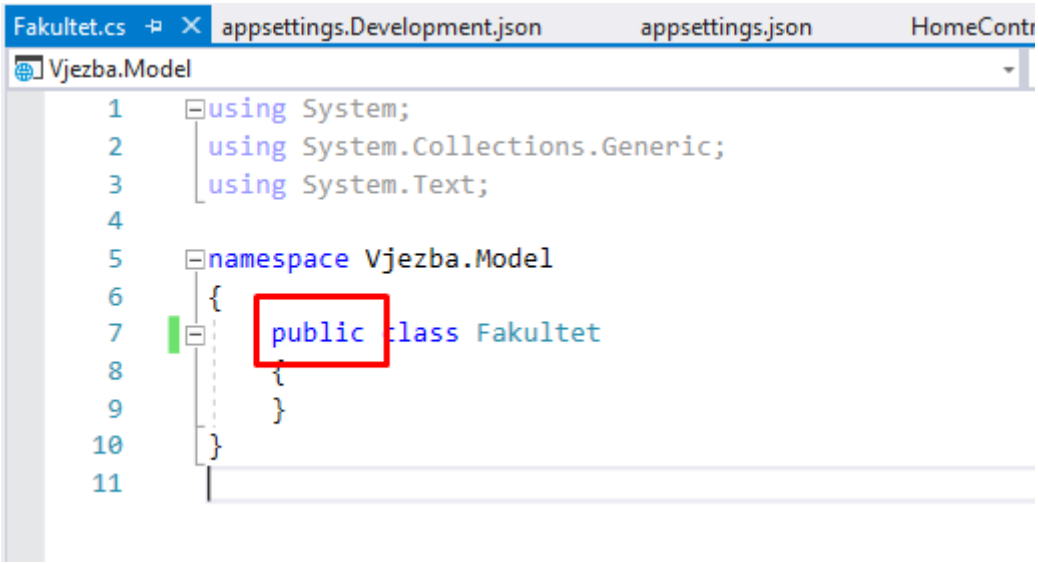
Klase i objekti, svojstva u klasi

Nova klasa u projekt se dodaje tako da u solution explorer prozoru, na željenom projektu, odaberemo desni klik i add new item, te pronađemo odgovarajuću stavku iz izbornika.

Važna napomena: pri dodavanju nove klase njena vidljivost je automatski postavljena na „internal“, te da bi klasa bila vidljiva u drugim projektima potrebno ju je definirati kao „public“.

Primjerice, ukoliko dodamo klasu **Fakultet** u projekt **Vjezba.Model**, moramo dodati ključnu riječ „public“:





Odnos klasa

Klase mogu implementirati nasljeđivanje, što znači da izvedena klasa nasljeđuje svojstva i funkcije iz bazne klase. Klase također mogu biti u odnosu u kojem neka klasa sadrži niz referenci na objekte druge klase, najčešće u obliku liste. U nastavku je dan primjer, gdje klasa A nasljeđuje klasu B (B je bazna klasa), dok klasa A također ima u sebi listu referenci na objekte klase C (tzv. odnos 1-N).

```
public class B
{

}

public class A : B
{
    private List<C> _objektiKlaseC;
}

public class C
{

}
```

Svojstva i polja, konstruktori i metode

Kada govorimo o informacijama koje su sadržane u klasama, najčešće koristimo termine kao što su svojstva (**properties**) i rijeđe polja (**fields**). Polje (član) je doslovno varijabla koja je sadržana unutar neke klase, i kad se kreira instanca neke klase, moguće je koristiti to polje kao i svaku drugu varijablu. Iz gornjeg isječka koda, član **_objektiKlaseC** je privatna varijabla, koja je tipa List<C>. U pravilu, članovi bi uvijek trebali biti privatni, što znači da se tim članovima ne može pristupiti nigdje izvan klase u kojoj se nalaze[1]. Svojstvo, s druge strane, možemo promatrati kao getter i setter funkciju, samo napisanu malo drukčijom sintaksom nego što je to u C++ ili Java jeziku:

```
47 references | Ivan Cesar, Less than 5 minutes ago | 1 auth
public class Osoba
{
    private string _adresa;
    0 references | 0 changes | 0 authors, 0 changes
    public string Adresa
    {
        get
        {
            return this._adresa;
        }
        set
        {
            this._adresa = value;
        }
    }
```

```
47 references | Ivan Cesar, 5 minutes ago | 1 author, 1 change
public class Osoba
{
    private string _adresa;

    0 references | 0 changes | 0 authors, 0 changes
    public string GetAdresa()
    {
        return this._adresa;
    }

    0 references | 0 changes | 0 authors, 0 changes
    public void SetAdresa(string value)
    {
        this._adresa = value;
    }
```

U gornjoj klasi, svojstvo je **Adresa**, a funkcije koje dohvaćaju ili postavljaju vrijednost tog svojstva su definirane ključnim riječima **get** i **set**. Vrijednost svojstva se u pravilu čuva u privanom članu. Što se tiče prava pristupa, vrijede ista pravila kao i za polja – međutim kod svojstava se najčešće koristi pravo pristupa **public** ili **protected**. Postoje i automatska svojstva, koja se mogu zapisati ovako[2]:

[1] Više o pravima pristupa može se naći na ovom linku: https://msdn.microsoft.com/en-us/library/wxh6fsc7.aspx

[2] Više o svojstvima uz primjer nasljeđivanja: https://msdn.microsoft.com/en-us/library/w86s7x04.aspx

```
        public int SvojstvoX { get; set; }
    }
```

Dobro je pitanje, zašto bi uopće koristili automatska svojstva a ne polja? Postoje razlike kako C# doživljava varijable i svojstva, te se neće detaljnije ulaziti u to područje[3]. Međutim, držat ćemo se sljedećih pravila:

- Klasa ne smije sadržavati javna polja
- Klasa bi trebala sadržavati svojstva (javna ili protected, u rijetkim slučajevima private)
- Koristiti automatsko svojstvo gdje je moguće

Pri kreiranju objekta, prva funkcija koja se poziva unutar klase je konstruktor. Da bi klasa funkcionirala, nije potrebno napraviti konstruktor – osnovni (default) konstruktor će biti dodan automatski. Međutim, u nekim slučajevima je dobro dodati konstruktor za osiguravanje incijalizacije, kao što je to primjer u sljedećem kodu:

```
public class A : B
{
    public List<C> ObjektiKlaseC { get; set; }

    public A()
    {
        ObjektiKlaseC = new List<C>();
    }
}
```

U gornjem slučaju je dobro koristiti konstruktor jer tada možemo inicijalizirati na ispravan način kompleksni objekt – kreirati mu novu instancu. Kada konstruktora ne bi bilo, onaj tko koristi tu klasu mora samostalno inicijalizirati listu, te ukoliko ju pokuša koristiti bez inicijalizacije, može dobiti **null reference** iznimku.

Uz konstruktore i svojstva, gotovo svaka klasa sadrži niz metoda – funkcija koje predstavljaju implementaciju ponašanja neke klase. Funkcija također može imati razine pristupa **private**, **protected** i **public**, a može biti zavisna od postojanja objekta ili nezavisna (**static**). Primjer jednostavne funkcije je dan u nastavku:

```
public int PrebrojiKolikoJeUListi()
{
    int rezultat = 0;
    foreach (var x in ObjektiKlaseC)
        rezultat++;
    return rezultat;
}
```

Funkcija je definirana:

- Tipom povratne vrijednosti
- Imenom (predlaže se CamelCase)
- Parametrima koje prima (tip parametra i koliko ih ima)

[3] Više o tome može se pročitati primjerice ovdje: http://blog.codinghorror.com/properties-vs-public-variables/

Iznimke

Kao i u drugim programskim jezicima, C# ima razvijen mehanizam rukovanja iznimkama. Iznimka se može dogoditi uslijed nepredviđenih ulaznih parametara, pogreške u kodu i sl. Iznimke se mogu baciti i uloviti, ali svaka iznimka koju možemo baciti ili uloviti nasljeđuje iz bazne klase Exception. U samom .NET radnom okviru ima niz već postojećih iznimki koje se mogu iskorititi u nizu situacija (kao u zadatku 2.1), ali moguće je i kreirati vlastitu iznimku – to je ni manje ni više nego nova klasa koja nasljeđuje iz klase Exception ili iz bilo koje njoj izvedene klase. Za bacanje iznimki koristimo ključnu riječ **throw**, dok za hvatanje iznimki koristimo kombinaciju **try-catch-(finally)** [4].

Zadatak 2.1

Dodati u projekt Vjezba.Model klase **Osoba**, **Profesor**, **Student** i **Fakultet**. Ispravno implementirati nasljeđivanje te odnose između tih klasa. Za spremanje 1-N veze koristiti List<> objekt. Gdje god je moguće, koristiti automatska getter i setter svojstva. Dodatno, potrebno je ispoštovati sljedeća poslovna pravila

1. Osoba treba sadržavati informacije: **Ime**, **Prezime**, **OIB**, **JMBG**. OIB ima 11 znakova, i sve moraju biti znamenke, dok JMBG ima točno 13 znakova, također sve moraju biti znamenke. Ukoliko prilikom postavljanja svojstva OIB ili JMBG gornje pravilo nije ispoštivano, potrebno je baciti iznimku **InvalidOperationException**.
    a. Iz JMBG broja je moguće izvući datum rođenja, te osoba također treba imati getter svojstvo tipa DateTime i naziva **DatumRodjenja**.
2. Profesor treba uz ime, prezime, oib i jmbg, imati i podatak o nazivu odjela (**Odjel**), zvanju (enumeracija[5] **Zvanje**) i datumu izbora u zvanje (**DatumIzbora**). Enumeracija zvanje se sastoji od vrijednosti: Asistent, Predavac, VisiPredavac, ProfVisokeSkole.
    a. Potrebno je napraviti funkciju **KolikoDoReizbora** u kojoj treba uzeti u obzir trenutno zvanje i datum izbora u zvanje, te trenutni datum. Povratna vrijednost je int broj koji označava koliko godina je do idućeg izbora u zvanje. Izbor u zvanje se za asistente održava svake 4 godine, dok za ostale svakih 5 godina.
3. Student, uz ime, prezime, oib i jmbg, treba imati i **JMBAG** podatak. JMBAG je broj od točno 10 znamenki, te ukoliko to pravilo nije ispoštovano kod postavljanja JMBAG podatka potrebno je isto baciti iznimku kao i kod OIB ili JMBG broja. Uz JMBAG, potrebno je i omogućiti evidenciju prosjeka (decimal **Prosjek**), broja položenih predmeta (int **BrPolozeno**) i ukupno osvojeno ECTS bodova (int **ECTS**).
4. U klasi Fakultet može biti najviše jedno svojstvo tipa List, ali Fakultet mora čuvati podatke o svim Profesorima i svim Studentima koji su tamo. Prilikom kreiranja instance klase Fakultet lista mora biti inacijalizirana.
    a. U klasi fakultet potrebno je napraviti funkcije **KolikoProfesora** i **KolikoStudenata** koja vraća int vrijednost koliko ima profesora a koliko studenata na fakultetu.

[4] Primjeri iznimki se mogu vidjeti ovdje: https://msdn.microsoft.com/en-US/library/ms173163%28v=vs.80%29.aspx
[5] Primjer enumeracije: https://msdn.microsoft.com/en-us/library/sbbt4032.aspx

b. U klasi fakultet napraviti funkciju koja dohvaća studenta po JMBAG broju. Funkcija se treba zvati **DohvatiStudenta** i primati jedan parametar – JMBAG broj. Ukoliko student ne postoji, vratiti null vrijednost.

Sučelja (interface)

Sučelje je skup pravila kojim definiramo kako izgleda ili koja ponašanja sadrži neka klasa koja ga implementira. Primjerice, klasa koja implementira sučelje IDisposable sigurno sadrži metodu Dispose. Ovo sučelje također indicira da klasa koristi resurse koje bi trebalo osloboditi nakon što smo završili s korištenjem te klase. Prešutni dogovor u C# je da sučelja počinju slovom „I“ (od Interface), te ih je na taj način lako uočiti i razlikovati od „običnih“ klasa[6].

Kolekcije (generics)

Gotovo svaka poslovna aplikacija će morati voditi evidenciju (pamtiti) informacije o nizu različitih podataka - primjerice, niz računa, narudžbenica, studenata, brodova, ili kao u prošlom zadatku – profesora i studenata. U tu svrhu koriste se razne kolekcije koje nam omogućavaju niz funkcionalnosti za rukovanje ovisno o konkretnoj potrebi i kontekstu. U C# najčešće se koriste strogo tipizirane kolekcije (generics) - to su kolekcije koje mogu pamtiti samo određene tipove podataka, tj. objekte neke specifične klase. Evo nekoliko primjera kolekcija i tipiziranih kolekcija uz pojašnjenje:

- **List** - kolekcija u koju se može spremiti bilo kakav objekt. Ovo je primjer netipizirane kolekcije. Elementima liste (tipiziranim i netipiziranim) se može pristupati preko uglatih zagrada i indexa: **`mojaLista[5]`** -> vraća **6. element** liste koja se zove **mojaLista**. Povratna vrijednost je tipa **object**.
- **List<int>** - lista cjelobrojnih vrijednosti - ne možemo dodati ništa osim cjelobrojne (int) vrijednosti. Možemo napisati primjerice **`int x = mojaIntLista[2];`**
- **List<ContactData>** - lista objekata tipa ContactData
- **Dictionary<int, string>** - primjer kolekcije u kojoj su vrijednosti spremljene pod nekim ključem. Primjerice, u ovoj kolekciji ključ je cjelobrojna vrijednost, a vrijednost je string. Ovoj kolekciji pristupamo na način da kao index koristimo ključ - primjerice, **`mojDict[2] = "Dva"`**

Ipak, najčešće korištena klasa je upravo List<XX> klasa, gdje je XX naravno odgovarajući tip[7].

Petlje i enumeriranje kolekcije

Da bi pregledali elemente neke kolekcije, moramo koristiti neku vrstu petlje. U C# najčešće se koristi foreach petlja koja prolazi po svim elementima neke kolekcije. Primjer takve petlje nalazi se na stranici 6 kod implementacije metode „PrebrojiKolikoJeUListi“[8].

[6] Nešto više o sučeljima: https://msdn.microsoft.com/en-us/library/ms173156.aspx
[7] Popis funkcija u List klasi. Klikom na svaku funkciju je moguće detaljnije pogledati primjere: https://msdn.microsoft.com/en-us/library/6sh2ey19%28v=vs.110%29.aspx
[8] Više o petljama: https://msdn.microsoft.com/en-us/library/f0e10e56%28v=vs.90%29.aspx

Zadatak 2.2

U klasi Fakultet potrebno je dodati funkciju **DohvatiProfesore** koja vraća popis svih profesora (povratni tip je IEnumerable<Profesor>), poredanih po datumu izbora u zvanje počevši od najstarijeg datuma.