PREDMET: Razvoj Programskih Rješenja	**NASTAVNI ANSAMBL**
AK. GOD. : 2017/2018 **RESURS:** Laboratorijska vježba 1 **DATUM OBJAVE:** 16.10.2017	**PROFESOR:** Dženana Đonko (ddonko@etf.unsa.ba) **ANSAMBL:** Hasić Haris, Hodžić Kerim Đenašević Emir, Hasanbašić Denis, Kiselica Aldin, Mahmutović Mustafa, Ramić Benjamin

Cilj vježbi:

- Upoznavanje sa osnovama Visual Studio 2015 alata
- Upoznavanje sa sintaksom C# programskog jezika
- Upoznavanje sa klasama i dekompozicijom programa
- Upoznavanje sa interfejsima, apstraktnim klasama i nasljeđivanjem u C#
- Upoznavanje sa načinima primjene polimorfizma u C#

Napomena: Prije izrade vježbe je obavezno pročitati predavanja vezana za tematiku vježbe i upoznati se sa osnovnim konceptima jezika C# i objektno orjentisanog programiranja.

LABORATORIJSKA VJEŽBA 1

Upoznavanje sa Visual Studio radnim okruženjem

Visual Studio

Na predmetu se koristi integrisano razvojno okruženje Microsoft Visual Studio 2015. Prilikom razvoja aplikacija, nije neophodno koristiti integrisano razvojno okruženje (*integrated development environment,* IDE) poput Visual Studia. Kod je moguće pisati i u običnom tekstualnom editoru pa program kompajlirati ručno kroz komandnu liniju. Međutim, ovakvi alati su tu da programeru olakšaju rad. Visual Studio, između ostalog, automatizuje proces kompilacije, ima editor teksta koji je prilagođen sintaksi jezika C# što omogućava da predlaže i dopunjava kod, kreira umjesto programera *boilerplate* kod za određene vrstu projekata, te ima alate za *debugging*, od kojih će neki biti prikazani u nastavku.

Kroz Visual Studio je moguće razvijati Windows API, Windows Forms, Windows Presentation Foundation, Universal Windows te Microsoft Silverlight aplikacije. Neki od podržanih programskih jezika su C, C++, C++/CLI, VB.NET, C#, a dodatno je moguće instalirati potrebne servise za M, Python, Ruby.

Microsoft Visual Studio Community Edition je besplatna verzija razvojnog okruženja koja je dovoljna za zahtjeve ovog kursa i moguće ju je pronaći na oficijelnoj stranici https://www.visualstudio.com/downloads/. Također, Visual Studio je moguće preuzeti i koristeći studentski Dreamspark account na https://imagine.microsoft.com/en-us/Catalog.

Uvod u sintaksu jezika C# i Visual Studio kroz primjer

C# je objektno orijentisani jezik opšte namjene koji vuče korijene u jezicima C i C++. Obzirom da su na predmetima Osnove računarstva i Tehnike programiranja savladane osnove jezika C i C++, prelaz na C# studentima ne bi trebao predstavljati problem. Uvod u sintaksu jezika C# je dat u nastavku kroz prikaz rješenja jednostavnog zadatka. Neka to bude zadatak 3 iz zadataka za samostalni rad.

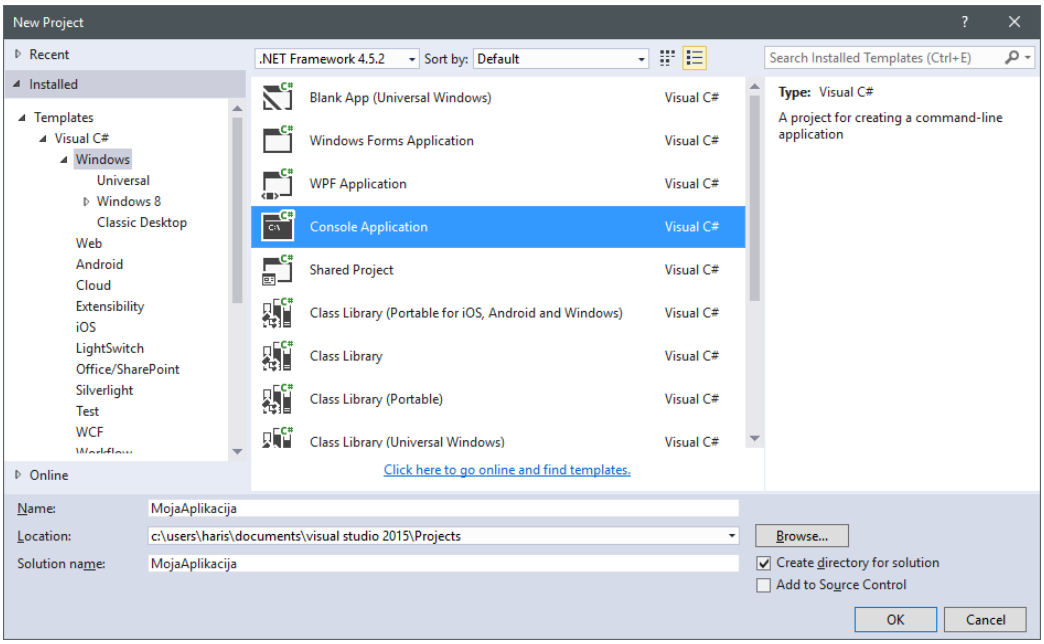
Primjer Napišite konzolni aplikaciju koji traži da se sa tastature unese 6 brojeva, a koji zatim ispisuje da li su svi uneseni brojevi pozitivni i da li među njima ima neparnih brojeva. Za realizaciju programa ne koristiti nizove. Program testirajte na sljedećim karakterističnim primjerima:

Primjer 1: 2 6 10 18 8 6
Primjer 2: 4 –12 10 18 –18 10
Primjer 3: 3 5 4 19 7 12
Primjer 4: –3 9 15 –7 13 11

Kreiranje projekta

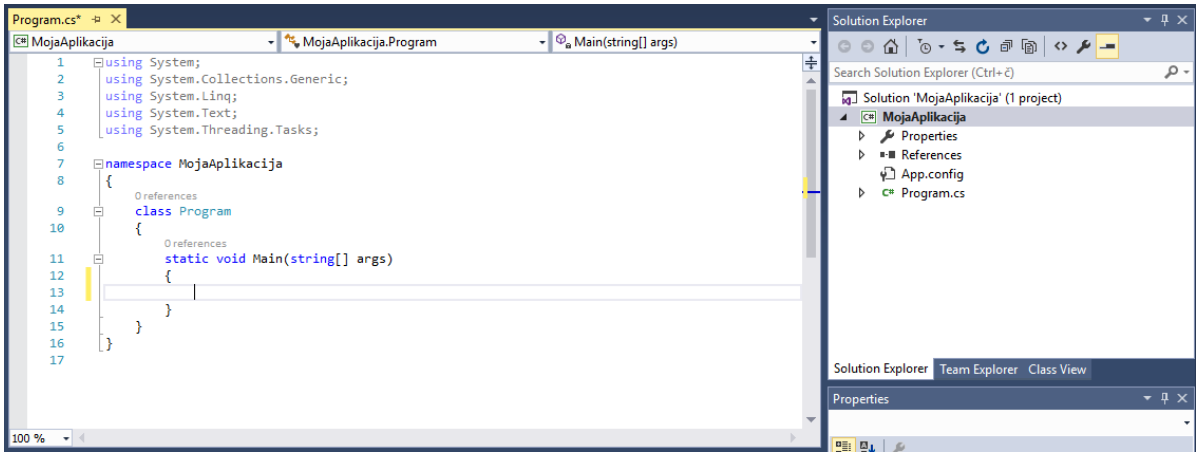
Nakon otvaranja Visual Studia, projekat se kreira sa File –> New –> Project... Otvara se dijalog kao na slici datoj ispod. Potrebno je odabrati Visual C# template, i unutar njega Console Application. Popuniti polja:

- *Name:* Ime projekta
- *Location:* Lokacija projekta
- *Solution Name*: Ime rješenja



Projekat (*Project*) nije isto što i rješenje (*Solution*). *Rješenje* sadrži resurse koji su potrebni za cijelu aplikaciju, te može uključivati jedan ili više projekata, te dodatne datoteke i metapodatke. Definicija rješenja se čuva u .sln i .suo datotekama, gdje je .sln definicija rješenja (projekti koji su sadržani u rješenju, resursi koji nisu vezani za neki konkretan projekat itd.), a .suo čuva preference vezane za IDE. *Projekti* su logičke cjeline unutar rješenja i kao izlaz mogu dati izvršnu datoteku (.exe), dll i slično.

Nakon kreiranja projekta dobijamo prikaz kao na slici ispod. Na desnoj strani je Solution Explorer, gdje se vide svi projekti unutar rješenja, a na lijevoj strani je otvorena datoteka Program.cs u kojoj je izgenerisan kostur klase Program i metoda Main koja je ulazna tačka za izvršavanje programa.



Rješenje zadatka je sljedeće (Dostupno u resursima za RPR LV1. Resurs 1):

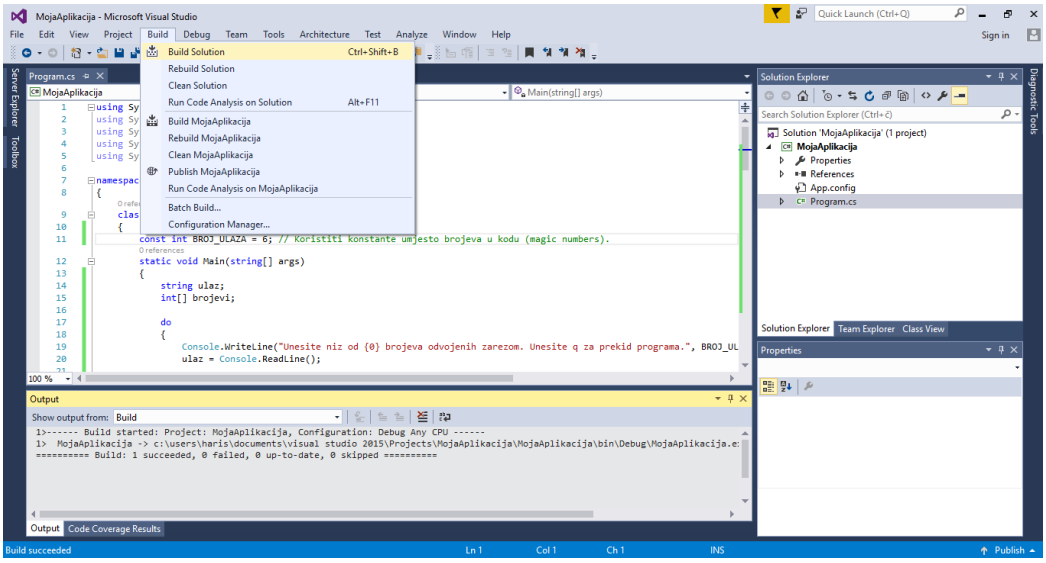
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MojaAplikacija
{
    class Program
    {
        const int BROJ_ULAZA = 6; // Koristiti konstante umjesto brojeva u kodu (magic
numbers).
        static void Main(string[] args)
        {
            string ulaz;
            int[] brojevi;

            do
            {
                Console.WriteLine("Unesite niz od {0} brojeva odvojenih zarezom.
Unesite q za prekid programa.", BROJ_ULAZA);
                ulaz = Console.ReadLine();

                if (ulaz == "q")
                {
                    return;
                }

            } while (!validirajUlaz(ulaz, out brojevi));

            string neparnihBrojeva = imaNeparnih(brojevi) ? "ima" : "nema";
            string brojeviPozitivni = sviPozitivni(brojevi) ? "jesu" : "nisu";

            Console.WriteLine("Brojevi u nizu " + brojeviPozitivni + " svi
pozitivni.");
            Console.WriteLine("U nizu " + neparnihBrojeva + " neparnih brojeva.");
            // Ne zatvarati konzolu odmah po zavrsetku programa.
            Console.ReadLine();
```

```
        }

        static bool validirajUlaz(string ulaz, out int[] brojevi)
        {
            brojevi = new int[BROJ_ULAZA];
            string[] ulazi = ulaz.Split(',');

            if (ulazi.Length != BROJ_ULAZA) // Korisnicki ulaz nije ispravan.
            {
                Console.WriteLine("Ulaz nije ispravan. Potrebno je unijeti 6 brojeva.
Pokusajte ponovo.\n");
                return false;
            }

            for (int i = 0; i < BROJ_ULAZA; i++)
            {
                if (!Int32.TryParse(ulazi[i], out brojevi[i])) {
                    // Ulaz nije broj.
                    Console.WriteLine("Ulaz {0} nije ispravan broj.", ulazi[i]);
                    return false;
                }
            }

            return true;
        }

        static bool sviPozitivni(int[] brojevi)
        {
            foreach (int broj in brojevi)
            {
                if (broj <= 0)
                {
                    return false;
                }
            }
            return true;
        }

        static bool imaNeparnih(int[] brojevi)
        {
            foreach (int broj in brojevi)
            {
                if (broj % 2 != 0)
                {
                    return true;
                }
            }
            return false;
        }
    }
}
```

String.Split(): http://msdn.microsoft.com/en-us/library/system.string.split(v=vs.110).aspx
Int32.TryParse(): http://msdn.microsoft.com/en-us/library/f02979c7(v=vs.110).aspx

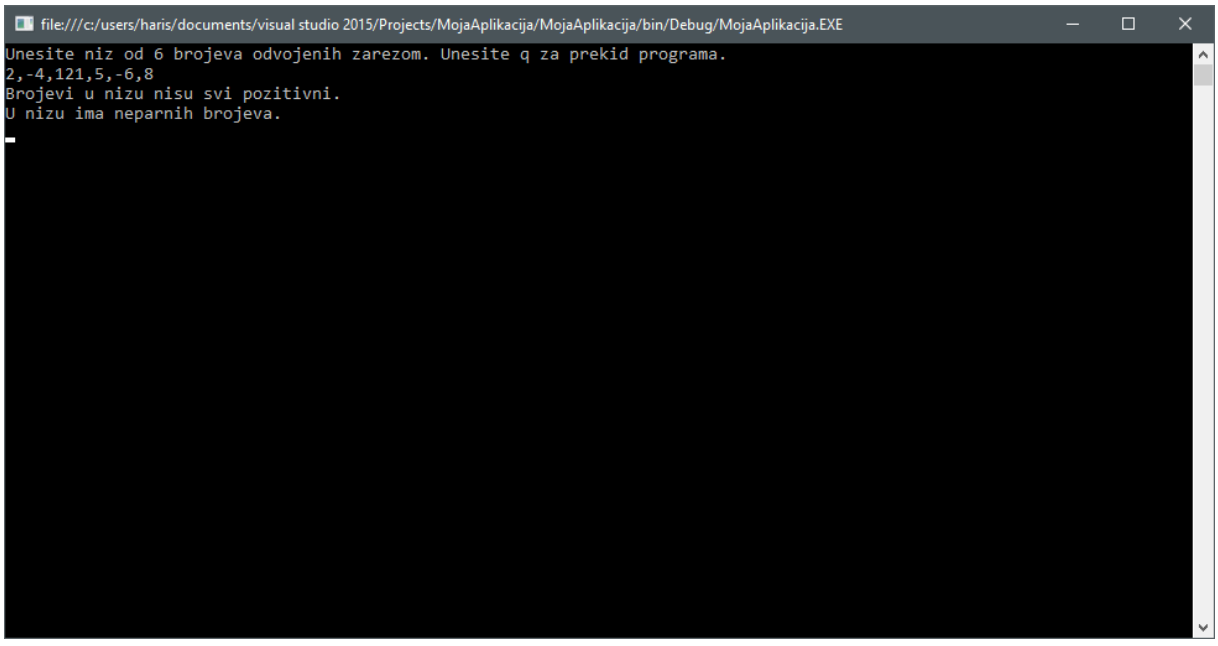
Build projekta

Nakon što je kod završen, potrebno je uraditi *build*, putem Build –> Build Ime Rješenja Primjer (skraćenica Shift+F6), a moguće je uraditi i *build* cijelog rješenja putem Build –> Build Solution (skraćenica F6). Ukoliko je proces uspješan, u donjem lijevom uglu piše „*Build succeeded*“, kao na slici ispod.



Izvršavanje

Program pokrećemo na Debug –> Start debugging (ili pomoću tastature, F5).



Debugging i korištenje prekidnih tačaka

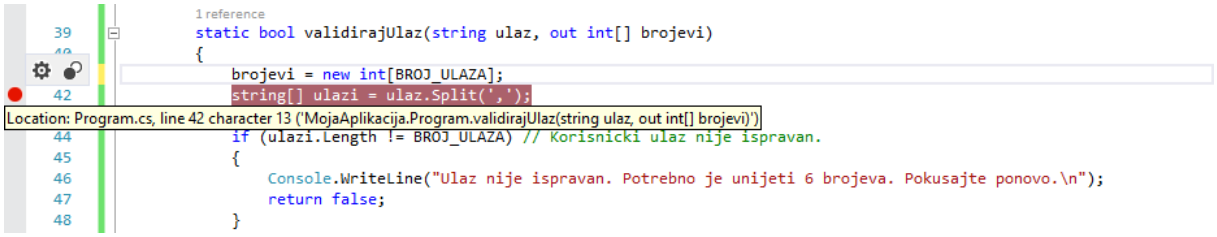
Osim pokretanja programa opcijom *Start Without Debugging*, program se može pokrenuti i u načinu rada za ispravljanje grešaka (*Debug mode*). Program se pokreće u načinu za ispravljanje grešaka odabirom opcije *Start* iz izbornika *Debug* ili pritiskom na tipku F5. Ukoliko je program pokrenut u načinu za ispravljanje grešaka te ukoliko negdje unutar

programa postoji prekidna tačka (*breakpoint*) program će na tom mjestu privremeno zaustaviti svoje izvršavanje i čekati na našu sljedeću akciju.

Postavljenje prekidne tačke vrši se pozicioniranjem na liniju u kojoj želimo postaviti prekidnu tačku i jednim od sljedeća tri načina:

- Desni klik miša uzrokuje pojavljivanje 'pop-up' izbornika na kojem odaberemo opciju Insert Breakpoint
- Klikom na zasivljeni dio interfejsa lijevo od teksta
- Odabirom opcije *New Breakpoint* iz izbornika *Debug*

Jedan primjer Breakpoint-a je prikazan na slici ispod.



Dodatne inormacije o debagiranju mogu se naći na sljedećem linku:

https://msdn.microsoft.com/en-us/library/dn986851.aspx

Upoznavanje sa konceptima objektno orjentisanog programiranja u C#

Naslijeđivanje i polimorfizam - hijerarhija klâsa

Kroz prateće predavanje upoznali ste se sa osnovnim konceptima vezanim uz naslijeđivanje i polimorfizam. U okviru ove vježbe posvetiti ćemo se njihovim praktičnim aspektima. Nakon što ste definirali vaše korisničke tipove podataka (klase) tako što ste za svaku klasu odredili njen naziv, polja i atribute koji opisuju svaki objekat datog tipa, kao i metode kroz koje se ostvaruje interakcija sa tim objektom. Još jednom je potrebno ukazati na razliku između klase (što predstavlja definiciju tipa - koncept koji kazuje da tip ima npr. polje Ime i da je to ime moguće mijenjati) i objekta kao instance klase (konkretni konstrukt napravljen po definiciji).

Time ste se ujedno upoznali sa dva osnovna principa u objektno orijentiranom dizajnu i programiranju: sakrivanje informacija od korisnika klase i enkapsulacija. Prvi se odnosi na činjenicu da unutrašnja struktura klase (privatna polja, atributi i metode) nije vidljiva krajnjem korisniku. Na taj način je krajnji korisnik odvojen od konkretne implementacije date klase i olakšane su kasnije izmjene te klase.

Primjerice, vremenski tipovi mogu svoje stanje čuvati bilo kao odvojena polja za sate i minute ili kao jedno polje za broj "tickova" procesora, ili na bilo koji treći način koji tehnički odgovara developeru. Dokle god postoje atributi kroz koje korisnik pristupa detaljima vremena krajnjeg korisnika ne interesuje kako je klasa interno implementirana. Samim tim je krajnji korisnik (npr. neki drugi developer) spriječen u naumu da se eventualno dublje veže sa vašom klasom na način da se počne oslanjati na mogućnosti koje su podložne promjenama iz bilo kojih razloga.

Princip enkapsulacije podrazumijeva da je mijenjanje stanja objekta omogućeno samo na kontrolisan način - tačno onako kako je developer to zamislio. To predstavlja značajnu razliku (i poboljšanje) u odnosu na rad sa strukturama gdje se sa svim poljima može bez ikakvih ograničenja raditi direktno.

Primjerice, ako je polje dan privatno, jedini način da se postavi dan u objektu "vrijeme" je kroz atribut ili neku metodu. Ako je developer ispravno implementira, onemogućiti će postavljanje nevalidne vrijednosti za tekući dan u bilo kojem scenariju. S druge strane, kod struktura ne postoji ništa što bi korisnika spriječilo da direktno pristupi cjelobrojnom polju "dan" i postavi mu bilo koju vrijednost iz opsega cjelobrojnog tipa.

Opisana dva principa osiguravaju disciplinu, konzistentnost i sigurnost objektno-orijentisanog dizajna, međutim sami za sebe ne rješavaju probleme koji nastaju u radu sa složenim programskim rješenjima. Konkretno, kôd će i dalje biti usko vezan uz problemski domen i njegova ponovna iskoristivost zavisi isključivo od discipline i dalekovidnosti developera. Još jedan čest problem su izmjenjivi zahtijevi koji se postavljaju pred developere: dodavanje novih funkcionalnosti i tipova podataka često povlači za sobom velike zahvate u postojećem kôdu da bi ga se prilagodilo novim okolnostima. Da bi se navedeno minimiziralo i napisani kôd učinio lako proširivim, izmjenjivim i ponovno iskoristivim koriste se principi naslijeđivanja, polimorfizma i programiranja temeljem interfejsa. Navedeno se najbolje pokazuje na primjeru.

Dodatne inormacije o objektno orjentisanom programiranju sa primjerima u C# programskom jeziku možete naći na sljedećem linku:

https://msdn.microsoft.com/en-us/library/mt656686.aspx

Primjer 1

Potrebno je osmisliti tipove podataka za programsko rješenje za neku banku. Banka trenutno omogućava fizičkim licima da otvore tekuće račune.

Prvi isječak kôda odgovara klasama pisanim bez upotrebe naprednih koncepata objektno-orijentisanog dizajna. Kôd je funkcionalan u smislu da je moguće kroz glavni program instancirati objekat tipa Bank, dodavati korisnike i otvarati im račune, te kasnije pronaći korisnika po nekom identifikacionom broju i raditi sa nekim od njegovih računa.

```
namespace Polymorphism
{
    class Program
    {
        static void Main(string[] args)
        {
            NoPoly.Banka b = new NoPoly.Banka();
            b.dodajKlijenta(new NoPoly.Osoba() { Id = 1, Ime = "Haris" });

            NoPoly.Osoba p = b.pronadjiKlijenta(1);
            b.otvoriRacunZaOsobu(p);

            p.Racuni[0].poloziNovac(550);

            Console.WriteLine("Stanje: {0}", b.pronadjiKlijenta(1).Racuni[0].Stanje);
        }
    }
}
```

```
namespace Polymorphism.NoPoly
{
    public class Osoba
    {
        public Int32 Identitet { get; set; }
        public String Ime { get; set; }
        public List<Racun> Racuni { get; set; }

        public Osoba()
        {
            Racuni = new List<Racun>();
        }
    }

    public class Racun
    {
        private static Decimal limitZaPodizanje = 1000;

        public Int32 RacunID { get; set; }

        private Decimal _stanje;

        public Racun(Decimal pocetnoStanje)
        {
            _stanje = pocetnoStanje;
        }

        public Decimal Stanje
        {
            get
            {
                return _stanje;
            }
        }

        public void poloziNovac(Decimal kol)
        {
            if (kol < 0)
            {
                throw new Exception("Količina novca koju polažete na račun mora biti
pozitivna.");
            }
            _stanje += kol;
        }

        public void povuciNovac(Decimal kol)
        {
            if (kol < 0)
            {
                throw new Exception("Količina novca koju podižete sa računa mora biti
pozitivna.");
            }
            else if (kol > limitZaPodizanje)
            {
                throw new Exception("Količina novca koju podižete sa računa mora biti manja
od limita za podizanje.");
            }
            _stanje -= kol;
        }
    }

    public class Banka
    {
        private List<Osoba> Klijenti { get; set; }
        private List<Racun> Racuni { get; set; }
```

```
        public Banka()
        {
            Klijenti = new List<Osoba>();
            Racuni = new List<Racun>();
        }

        public void dodajKlijenta(Osoba p)
        {
            Klijenti.Add(p);
        }

        public void otvoriRacunZaOsobu(Osoba p)
        {
            Racun acc = new Racun(0) {
                RacunID= Racuni.Count
            };

            Racuni.Add(acc);
            p.Racuni.Add(acc);
        }

        public Osoba pronadjiKlijenta(Int32 identitet)
        {
            return Klijenti.Single(osoba => osoba.Identitet == identitet);
        }

        public void prebaciNovac(Racun acc, Decimal kolicina)
        {
            acc.poloziNovac(kolicina);
        }
    }
}
```

Drugi isječak kôda odgovara klasama pisanim uz upotrebu naprednih koncepata objektno-orijentiranog dizajna. Kôd je također funkcionalan ali je moguće primijetiti dosta na prvi pogled nepotrebnih komplikacija (tzv. *overengineering* rješenja).

```
namespace Polymorphism
{
    class Program
    {
        static void Main(string[] args)
        {
            Poly.Banka b = new Poly.Banka();
            b.dodajKlijenta(new Poly.Osoba(1, "Haris"));

            Poly.Osoba p = b.pronadjiKlijenta(1) as Poly.Osoba;
            b.otvoriTekuciRacunZaOsobu(p);

            p.Racuni[0].poloziNovac(550);
            Console.WriteLine("Stanje: {0}", b.pronadjiKlijenta(1).Racuni[0].Stanje);

            Console.ReadLine();
        }
    }
}
```

```
namespace Polymorphism.Poly
{
    abstract public class Klijent
    {
        public Int32 Id { get; set; }
        public List<Racun> Racuni { get; set; }
```

```
        protected Klijent(Int32 id)
        {
            Id = id;
            Racuni = new List<Racun>();
        }
    }

    public class Osoba : Klijent
    {
        public String Ime { get; set; }

        public Osoba(Int32 id, String ime)
            : base(id)
        {
            this.Ime = ime;
        }
    }

    abstract public class Racun
    {
        public Int32 RacunID { get; set; }

        protected Decimal _stanje;

        protected Racun(Int32 id, Decimal pocetnoStanje)
        {
            RacunID = id;
            _stanje = pocetnoStanje;
        }

        public Decimal Stanje
        {
            get
            {
                return _stanje;
            }
        }

        public virtual void poloziNovac(Decimal kol)
        {
            if (kol < 0)
            {
                throw new Exception("Količina koju podižete sa računa mora biti
pozitivna.");
            }
            _stanje += kol;
        }

        public virtual void povuciNovac(Decimal kol)
        {
            if (kol < 0)
            {
                throw new Exception("Količina koju povlačite mora biti pozitivna.");
            }
            _stanje -= kol;
        }
    }

    public class TekuciRacun : Racun
    {
        private static Decimal limitZaPodizanje = 1000;

        public TekuciRacun(Int32 racunID, Decimal pocetnoStanje)
            : base(racunID, pocetnoStanje)
        {
```

```
        }

        // Preklapanje (Override) za specifično ponašanje kada se povlači novac
        override public void povuciNovac(Decimal kol)
        {
            if (kol < 0)
            {
                throw new Exception("Količina mora biti pozitivna da bi se povukao
novac.");
            }
            else if (kol > limitZaPodizanje)
            {
                throw new Exception("Količina mora biti manja od limita za podizanje.");
            }
            _stanje -= kol;
        }
    }

    public class Banka
    {
        private List<Klijent> Klijenti { get; set; }
        private List<Racun> Racuni { get; set; }

        public Banka()
        {
            Klijenti = new List<Klijent>();
            Racuni = new List<Racun>();
        }

        public void dodajKlijenta(Osoba p)
        {
            Klijenti.Add(p);
        }

        public void otvoriTekuciRacunZaOsobu(Osoba p)
        {
            TekuciRacun acc = new TekuciRacun(Racuni.Count, 0);
            Racuni.Add(acc);
            p.Racuni.Add(acc);
        }

        public Klijent pronadjiKlijenta(Int32 id)
        {
            return Klijenti.Single(osoba => osoba.Id == id);
        }

        public void poloziNovac(TekuciRacun acc, Decimal kol)
        {
            acc.poloziNovac(kol);
        }

        private String klijentUString(Klijent c)
        {
            if (c is Osoba)
            {
                return (c as Osoba).Ime;
            }
            return "";
        }
    }
}
```

Prodiskutirajte dva predložena rješenja sa asistentima. Koje od njih je prikladnije za postavku problema i zašto? Koje od njih smatrate da predstavlja bolji zalog za budućnost i zašto? Imate

li vi neku drugačiju ideju? Zašto je uopće korištena klasa Bank? Da li je dobro da Klijent "vidi" Račun ali ne i obratno? Zašto Banka vidi i Klijente i Račune?

Primjer 2

Klase mogu da imaju konceptualno zajedničke stvari, ali ih drugačije tehnički implementiraju. Neka je dato auto i biciklo. Auto i biciklo mogu da se kreću naprijed, nazad lijevo i desno pri čemu se ta krentnja obavlja na različite načine u odnosu na vozilo (auto mora biti upaljeno i imati dovoljno goriva) pri čemu auto još mora da ima mogućnost manipulacije motorom i mogućnost punjenja rezervoara.

```
interface IVozilo
{
    void naprijed();
    void nazad();
    void lijevo();
    void desno();
}

interface IMotorno
{
    void upaliMotor();
    void ugasiMotor();
    void napuniRezervar();
}



class Bicikl : IVozilo
{
    public void naprijed() { }
    public void nazad() { }
    public void lijevo() { }
    public void desno() { }
}

class Auto : IVozilo, IMotorno
{
    public void naprijed() { }
    public void nazad() { }
    public void lijevo() { }
    public void desno() { }
    public void upaliMotor() { }
    public void ugasiMotor() { }
    public void napuniRezervar() { }
}
```

Ponekad je potrebno istovremeno koristiti različite implementacije (rješenja) nekog problema. Neka je data aplikacija kod koje korisnik poslati odjednom sliku preko svih mogućih plugina koje posjeduje.

```
// Definišemo interfejs za plugin koji će svi autori konkretnih plugina morati
// implementirati kako bi aplikacija znala komunicirati sa njima
interface IUploadPlugin
{
    public string NazivPlugina { get; }
    public string NazivAutora { get; }
```

```
        public void uploadSliku(byte[] slika, int height, int width);
        public string URL { get; }
    }
    // Konkretan plugin za upload na pokit image hosting
    class PokitUploadPlugin : IUploadPlugin
    {
        public string NazivPlugina { get { return "Pokit upload plugin"; } }
        public string NazivAutora { get { return "Dinko Dujic"; } }
        public void uploadSliku(byte[] slika, int height, int width)
        {
            WebClient.UploadFile("pokit.org/upload"...);
        }
        public string URL { get { return "http://pokit.org/img/" + token } }
    }
    // Konkretan plugin za upload na FTP server
    class FTPUploadPlugin : IUploadPlugin
    {
        public string NazivPlugina { get { return "FTP upload plugin"; } }
        public string NazivAutora { get { return "Djuro Djuric"; } }
        public void uploadSliku(byte[] slika, int height, int width)
        {
            FtpWebRequest ftpClient = (FtpWebRequest)FtpWebRequest.Create(...);
        }
        public string URL { get { return "ftp://" + domena + putanja + "/" + token } }
    }
    // Konkretan plugin za upload na pokit image hosting
    class PicassaUploadPlugin : IUploadPlugin
    {
        public string NazivPlugina { get { return "Google picassa plugin"; } }
        public string NazivAutora { get { return "Larry Page"; } }
        public void uploadSliku(byte[] slika, int height, int width)
        {
            GoogleClinet.Picassa.UploadFile("picassa.com"...);
        }
        public string URL { get { return "http://pokit.org/img/" + token } }
    }
    // Nakon što su plugini naslijedili interfejs možemo na uniforman način
    // komunicirati sa njima te izmjene u kodu za dodavanjem novog plugina minimalne:
    foreach (IUploadPlugin p in plugins) {
        p.uploadSliku(slika,slika.Hight,slika.Width);
        linkovi.Add(p.URL);
```

Zadaci za samostalan rad i vježbu – Uvod u C#

Zadatak 1

Napišite program koji traži da se sa tastature unese brzina broda u čvorovima koja se zadaje isključivo kao cijeli broj (obavezno koristiti varijablu tipa “int”), a zatim izračunava i ispisuje brzinu broda u km/h kao decimalan broj. Koristite činjenicu da je čvor morska milja na sat, a da je jedna morska milja 1852 m (ovaj podatak obavezno definirati u programu kao konstantu). Na primjer, ukoliko se kao brzina broda unese broj 20, program treba da ispiše rezultat 37.04 jer je 20 čvorova = 37.04 km/h.

Zadatak 2

Napišite program koji traži da se sa tastature unese cijeli broj n, a zatim iscrtava na ekranu jednakostranični trougao sastavljen od zvjezdica čija je osnovica horizontalna a vrh usmjeren nagore. Na primjer, ukoliko se unese n = 4, ispis na ekranu treba da izgleda kao

```
   *
  ***
 *****
*******
```

Zadatak 3

Napišite program koji traži da se sa tastature unese broj ***n***, a nakon toga ***n*** elemenata niza tipa ***int***. Potrebno je iz glavnog programa pozvati funkciju:

private static void bubbleSortNiz(int[] arr)

koja sortira i ispisuje u glavnom programu uneseni niz cijelih brojeva. Nakon poziva ove funkcije pozvati funkciju:

private static void izbaciElementNiza(int[] arr, int element

koja iz originalnog niza izbacuje elementproslijeđenim kao parametar, ispisati niz bez elementa. Nakon poziva ove funkcije ispisati originalni niz.

Referenca: Pseudo kôd za Bubble Sort

Zadatak 4

Napišite program koji traži da se sa tastature unese prirodan broj n, a nakon toga se unose elementi kvadratne matrice formata n×n. Program nakon toga treba da ispiše redni broj kolone sa najvećom sumom elemenata, redni broj reda sa najmanjom sumom elemenata, kao i sumu elemenata na dijagonali. Koristiti jagged ili multidimenzionalni niz.

Referenca (Jagged) : http://msdn.microsoft.com/en-us/library/vstudio/2s05feca.aspx
Referenca (Multi) : http://msdn.microsoft.com/en-us/library/vstudio/2yd9wwz4.aspx

Zadatak 5

Napišite funkciju “DaLiJePalindrom” koja za string koji joj je proslijeđen kao parametar ispituje da li predstavlja palindrom ili nije, i kao rezultat vraća odgovarajuću logičku vrijednost “true” ili “false”.

Napomena: Pod palindromima smatramo riječi ili rečenice koje se isto čitaju sa obe strane (“kapak”). Prilikom ispitivanja treba ignorirati razmake, interpunkcijske znake i razliku između velikih i malih slova, tako da rečenica “Ana voli Milovana” treba da bude prepoznata kao palindrom, iako bukvalno pročitana sa suprotnog kraja glasi “anavoliM ilov anA”.

Zadatak 6

a) Napišite funkciju "IzbrisiPodstring" sa dva parametra tipa "string". Funkcija treba da kreira novi string u koji će prepisati sadržaj prvog stringa iz kojeg su izbačena sva eventualna pojavljivanja stringa koji je zadan drugim parametrom kao njegovog podstringa (ukoliko takvih pojavljivanja nema, string se prepisuje neizmijenjen), i da vrati tako kreirani string kao rezultat. Na primjer, nakon poziva funkcije:

s = IzbrisiPodstring("abcxyzslkgxyzalj", "xyz");

u stringu "s" (uz pretpostavku da je propisno deklariran) trebao bi se nalaziti niz znakova "abcslkgalj" funkciju napisati pomoću metoda klase String.

b) Napišite funkciju "NadjiPodstring" koja pronalazi da li se jedan string nalazi unutar drugog, i ukoliko se nalazi, gdje se nalazi. Funkcija ima dva parametra tipa "string". Funkcija treba da ispita da li se string predstavljen drugim parametrom nalazi unutar stringa predstavljen prvim parametrom kao njegov dio (na primjer, string "je lijep" se nalazi kao dio stringa "danas je lijep dan". Ukoliko je odgovor potvrdan, funkcija treba da kao rezultat vrati redni broj pozicije na kojoj se drugi string nalazi unutar prvog, pri čemu brojanje počinje od nule (tako da za prethodni primjer rezultat treba biti 6). Ukoliko je odgovor odrečan, funkcija treba da vrati –1 kao rezultat. Funkciju napisati pomoću metoda klase String.

Zadatak 7

Napišite program koji traži od korisnika da unese spisak riječi (broj riječi se prethodno unosi sa tastature), a zatim ispisuje na ekran prvu i posljednju riječ iz spiska po abecednom poretku, kao i popis svih unesenih riječi, ali bez ispisivanja duplikata (tj. bez ispisivanja riječi koje su se već jednom ispisale). Program realizirajte korištenjem niza stringova, odnosno niza čiji su elementi tipa "string".

Zadaci za samostalan rad i vježbu – OOP u C#

Zadatak 1

Banka iz Primjera 1 je proširila svoj biznis i sada omogućava otvaranje i štednih računa za korisnike (pored tekućih računa). Štedni račun sadrži dodatnu informaciju o valuti. Tu nije kraj promjenama - predstavnici banke očekuju da će se uskoro omogućiti i pravnim licima da koriste sve usluge banke kao i fizička lica. Pravno lice opisano je nazivom i adresom. Vaš zadatak je preuzeti do sada urađeno programsko rješenje (solution) sa stranice predmeta i prilagoditi klase u oba imenska prostora tako da podržite sve nove zahtijeve. Diskutirajte rješenja sa asistentima.

Zadatak 2

Razviti konzolno programsko rješenje koje će omogućiti za agenciju Stan 1) evidenciju stanova 2) mjesečni obračun najma stana. Stanovi mogu biti namješteni i nenamješteni. Za sve stanove se navodi površina (broj kvadrata) stana, lokacija stana (gradsko ili prigradsko područje) i da li postoji Internet konekcija. U slučaju da je stan namješten navodi se ukupna vrijednost namještaja i broj kućanskih aparata u stanu.

Obračun najma se vrši tako što zainteresovana osoba (klijent agencije Stan) navede rang vrijednosti za površinu stana koji želi iznajmiti i lokaciju stana. Nakon toga se prikažu svi stanovi koji odgovaraju navedenim osobinama, sa dodatnim podacima o tome da li je stan namješten i da li ima Internet konekciju. Nakon toga klijent bira stan i vrši se obračun najma za 1 mjesec. Bez obzira da li stan nenamješten ili namješten ako pripada gradskom području osnovna cijena najma je 200 KM mjesečno, a ako pripada prigradskom području cijena najma je 150 KM. Cijena kvadrata stana za sve stanove je 1KM. Na osnovnu cijenu najma se dodaje broj kvadrata pomnožen sa jediničnom cijenom kvadrata stana. Za nenamješten stan se nakon toga cijena najma povećava za 2% ukoliko stan ima Internet konekciju, a za namješten za 1%. U slučaju da je stan namješten izračunata cijena najma se nadalje povećava tako što se doda 1% od ukupne vrijednosti namještaja ukoliko je broj kućanskih aparata manji od 3, a ako je veći ili jednak 3 dodaje se 2% od ukupne vrijednosti namještaja. Obavijestiti korisnika o iznosu potrebnom za najam.

Scenarij testiranja: Površina Lokacija Namješten Internet Vrijednost namještaja Broj aparata

1. 50 gradsko nenamješten – Da
2. 80 prigradsko nemamješten – Da
3. 40 prigradsko namješten – Da 2000 2
4. 80 gradsko namješten – Ne 3000 6

Klijent agencije Stan unosi da želi stan čija površina je u rangu 60-90, prikazuju se stan 2 i 4. Cijena najma za stanove je respektivno 234,6 i 640.

Zadatak 3

Proširite rješenje za agenciju Stan tako da se može iznajmiti i luksuzni apartman koji na osnovnu cijenu od 1500 km dodaje još cijenu osoblja koje radi na imanju. Osoblje može biti batler, kuhar i vrtlar. Svaki član osoblja ima svoje ime, prezime, datum uposlenja i mjesečnu platu. Kuhar ima listu jela (string) koje je u stanju da napravi. Vrtlar sadrži i stan u kojim se stanuje, s obzirom da kompanija ima ugovor sa vrtlarima da uz posao dobiju i nenamješten stan. Batler ima pored osnovnih još atribut godine iskustva. Za potrebe pregleda imovine kompanije potrebno je moguće ispisati sve stanove i osoblje koje kompanija posjeduje. Pri izradi rješenja primijeniti polimorfizam.

Zadatak 4

Razviti konzolno programsko rješenje koje će omoćiti evidenciju i upravljanje gasova. Postoje četiri tipa gasova G1, G2, G3 i G4. Svaki tip gasa ima boju (crvena,žuta,plava,zelena), okus, miris, temperaturu i pritisak. Gasovi tipova G1 i G3 su nestabilni i potrebno je omogućiti provjeru njihove stabilnosti i korekciju temperature. G1 je stabilan ako je temperatura manja od 100 C (Celzijus) stepeni a G3 je stabilan ako nije crvene boje i ako je temperature manja od 150 C stepeni. Korekcija temperature za G1 tipove se vrši tako što se temperature smanji za 30 C stepeni a G3 za 50. Tipovi G1, G2 I G4 imaju određenu dozu toksičnog djelovanja koje se mjeri pomoću toksičnosti (T). Potrebno je omogućiti provjeru toksičnog dijelovanja tako što za svaki od toksičnih gasova mjeri da li je u granicama normale (G1 između 10T – 50T, G2 između 30T-80T i G4 između 20-50T). Potrebno je omogućiti

odabir svih gasova, toksičnih gasova ili nestabilnih gasova. U slučaju odabira svih gasova ispisuju se osnovni podaci o svim gasovima. Odabirom toksičnih ispisuju se samo toksični i pored osnovnih ispisuje se i informacija o njihovoj toksičnosti a odabirom nestabilnih ispisuju se samo nestabilni i pored osnovnih ispisuje se I informacija o stanju stabilonsti pri čemu se daje opcija korisniku da odabere gas čija će se korekcija obaviti ili da se upiše slovo "b" čime će se vratiti na prvi odabir gasova. Izvršiti dekompoziciju klasa i primijentiti polimorfizam. Napisati main klasu koja demonstrira razvijeno riješenje pri čemu prvo omogućava i unos gasova.

Zadatak 5

Razviti konzolno programsko rješenje "MiniSah" koje na ploču 8x8 raspoređuje figure pijun, top i kralj (kraljica, lovac I konj su zamjenjeni figurom pijun (figure postoje za samo jednog igrača). Program treba da ispiše u svakom koraku trenutno stanje ploče (pijun - P, top - T, kralj – K i prazno mjesto O). Korisnik može da odabere opciju 1 i 2. Opcija 1 omogućava da za unsene koordinate figure i koordinate polja program treba ispiše da li je moguće da se obavi dati potez I ako je moguće da se obavi potez (promijeni stanje ploče). Opcija 2 ponudi sve moguće poteze koje je u trenutnom stanju moguće obaviti. Izvršiti dekompoziciju klasa i primijentiti polimorfizam. Napisati main klasu koja demonstrira razvijeno riješenje.