Fakultet elektrotehnike, računarstva i informacijskih tehnologija
www.ferit.unios.hr

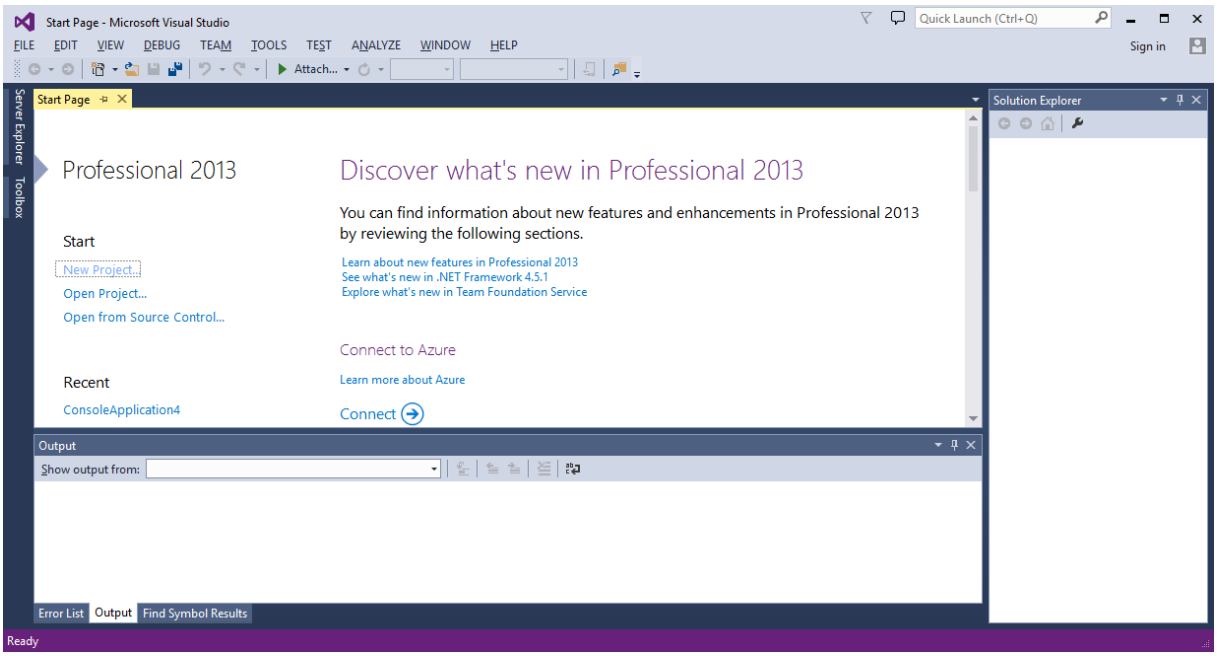
RAZVOJ PROGRAMSKE PODRŠKE OBJEKTNO ORIJENTIRANIM NAČELIMA

Laboratorijska vježba 1

Uvod u C#

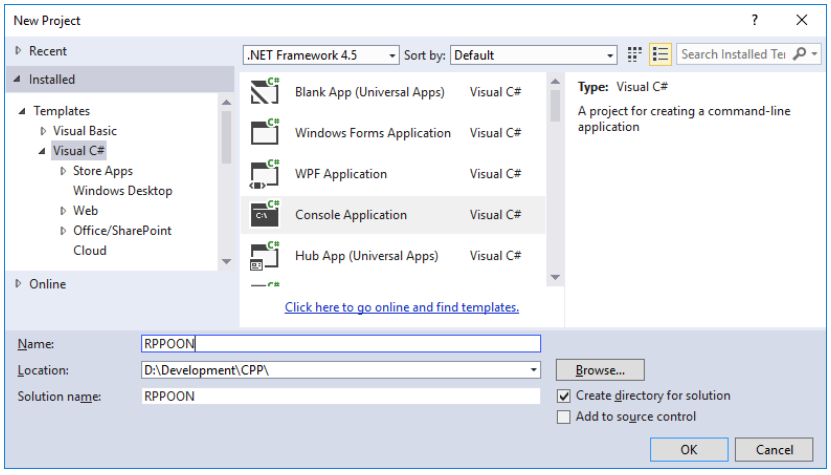
1. Kreiranje i pokretanje projekta

Za kreiranje aplikacija u programskom jeziku C# uobičajeno se koristi Microsoftovo razvojno okruženje (engl. *Integrated Development Environment*, IDE) Visual Studio. Prilikom pokretanja ovog IDE-a moguće je kreirati novi projekt odabirom *New Project...* iz izbornika s lijeve strane, a isto se postiže odabirom *File→New→Project...*



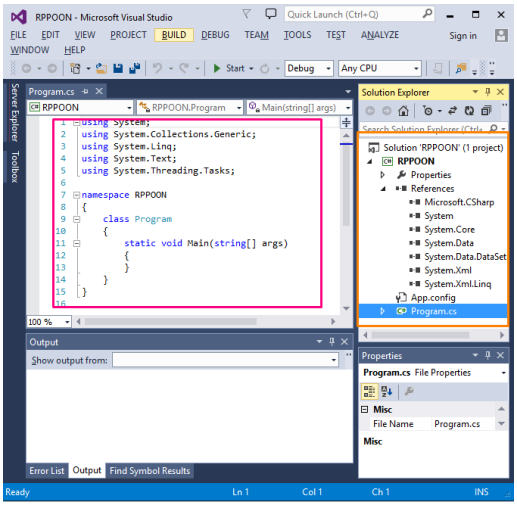
Slika 1.1 Izgled početne stranice Visual Studio 2013 IDE-a

Iz izbornika se odabire Visual C#, dok se kao predložak odabire (za sada) Console Application. U tekstualna polja potrebno je unijeti ime projekta, lokaciju na koju će se projekt smjestiti na lokalnom disku, kao i ime rješenja. Rješenje je zapravo ono što se



Slika 1.2 Izgled početne stranice Visual Studio 2013 IDE-a

kreira, a svako rješenja može sadržavati više projekata. Svaki se projekt tada može izgraditi u izvršnu datoteku ili, primjerice, u dinamički povezivu datoteku (engl. *dynamically linked library*). Na temelju odabranog predloška kreira se projekt s generiranim dijelom koda i uključenim brojnim osnovnim bibliotekama.



Slika 1.3 Izgled početne stranice Visual Studio 2013 IDE-a

Zadatak 1.

Kreirajte novu C# konzolnu aplikaciju i pokrenite ju.

2. Klase

Klase se u C#-u uobičajeno kreiraju u posebnim datotekama koje imaju nastavak .cs. Ime datoteke trebalo bi odgovarati imenu klase. Važno je također osim imena klase povesti računa i o imeniku kojem klasa pripada. Imenici predstavljaju koncept koji omogućuje semantičko grupiranje koda. Tako je moguće imati više različitih klasa istog imena, ali ako pripadaju različitim imenicima – neće biti problema s korištenjem bilo koje od njih. Nove se klase (ili drugi elementi) mogu dodati u projekt desnim klikom miša na ime projekta i zatim na *Add→Class...*, ili kroz *Project→Add class...* izbornik.

Primjer 1.

Kreirajte klasu koja predstavlja studenta s odgovarajućim atributima, poštujući pravila enkapsulacije.

```
class Student
{
    private String name;
    private String surname;

    public string getName() { return this.name; }
    public string getSurame() { return this.Surname; }
    public void setName(string name) { this.name = name; }
    public void setSurname(string surname) { this.surname = surname; }
}
```

Svim se atributima i metodama u klasi mora eksplicitno navesti pravo pristupa, a ako se ne navede, koristi se podrazumijevano pravo pristupa (za članove klase je to *private*, za klase *internal*, za imenike *public* itd.).

Zadatak 2.

Kreirajte klasu koja predstavlja zabilješku. Klasa treba omogućiti bilježenje teksta zabilješke, autora zabilješke te razine važnosti zabilješke. Poštovati pravila enkapsulacije, a omogućiti izmjenu samo teksta i razine važnosti, jednom zadan autor ne smije se izmijeniti.

2.1. Korištenje klase

Klasu koja je napisana u prethodnom primjeru koristi se na način da se kreira njene objekte (ovaj se postupak naziva još i instanciranje klase). Za kreiranje objekata koriste se posebne metode koje se definiraju za svaku klasu, a nazivaju se konstruktorima. Konstruktor za klasu *Student* prikazan je primjerom 2. Moguće je uočiti da se konstruktor zove isto kao klasa te da može postojati više konstruktora – odnosno da ih je moguće preopterećivati, baš kao i uobičajene metode. Kada se kreira objekt neke klase to se ostvaruje uporabom operatora *new* koji rezervira memorijski prostor za objekt na hrpi. Ovaj operator vraća referencu na instancu klase koja je netom kreirana.

Primjer 2.

Kreirajte konstruktore za klasu student. Iskoristite konstruktore za stvaranje nekoliko objekata ove klase.

```
        // ... code missing
        public Student() {
            this.name = "Anonymous";
            this.surname = "Duck";
        }

        public Student(string name, string surname) {
            this.name = name;
            this.surname = surname;
        }
```

```
/*****************************  Program.cs ****************************************/
        static void Main(string[] args)
        {
            Student student; // This is just the declaration!
            student = new Student(); // This is object creation!
            Console.WriteLine(student.getName());
            student = new Student("Donald", "Duck");
            Console.WriteLine(student.getName());
            Console.WriteLine("Enter a name: ");
            string name = Console.ReadLine();
            student.setName(name);
            Console.WriteLine(student.getName());
            // Notice the lack of the delete operator!
        }
```

Zadatak 3.

Kreirajte konstruktore za zabilješke (barem tri konstruktora). Napravite tri zabilješke koristeći različite konstruktore. Ispišite autore i tekst zabilješki.

2.2. Svojstva

Kako bi se izbjeglo pisanje brojnih metoda za postavljanje i dohvaćanje vrijednosti, C# nudi mogućnost korištenja svojstava (engl. *properties*) koja predstavljaju metode koje je moguće rabiti nalik atributima.

Primjer 3.

Dodajte svojstva za sve atribute klase Student i iskoristite ih.

```
        // ... code missing
        public string Name {
            get { return this.name; }
            set { this.name = value; }
        }

        public string Surname
        {
            get { return this.surname; }
            set { this.surname = value; }
        }
```

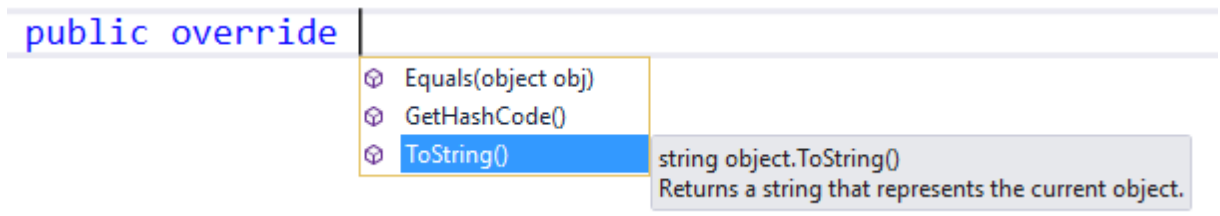
```
        /*****************************************/
        static void Main(string[] args)
        {
            Student student; // This is just the declaration!
            student = new Student();
            Console.WriteLine(student.Name);
            student = new Student("Donald", "Duck");
            Console.WriteLine(student.Name);
            Console.WriteLine("Enter a name: ");
            string name = Console.ReadLine();
            student.Name = name;
            Console.WriteLine(student.Name);
        }
```

Zadatak 4.

Izmijenite vlastitu klasu tako da uz pristupne metode za postavljanje i dohvaćanje sadrži i svojstva. Pripazite na prethodne zadatke i prava pristupa!

3. Nasljeđivanje i polimorfizam

U programskom jeziku C# moguće je naslijediti samo i isključivo jednu klasu. Moguće je, doduše, imati hijerarhiju nasljeđivanja gdje se preko nadređene klase nasljeđuju njene osnovne klase. Osnovna klasa svim klasama u C#-u je klasa *Object*. Ona sadrži neke korisne metode koje su zajedničke svim klasama, a koje je korisno prepisati (engl. *override*) u vlastitim klasama kako bi se postigla željena funkcionalnost specifična za našu klasu. Primjer ovakve metode je *toString()* metoda koja vraća string reprezentaciju objekta.



Slika 3.1 Izgled početne stranice Visual Studio 2013 IDE-a

Primjer 4.

Najprije pokušajte pozvati metodu *toString()* na objektu klase Student.

Nakon toga

```
        public override string ToString()
        {
            return this.Surname + ", " + this.Name;
        }
```

```
        /************************ Program.cs ********************************/
        static void Main(string[] args)
        {
            Student student; // This is just the declaration!
            student = new Student();
            Console.WriteLine(student.ToString());
            Console.WriteLine(student);
        }
```

Zadatak 5.

Prepišite postojeću metodu i definirajte vlastitu *ToString()* metodu.

Ako se nasljeđuje vlastita klasa, onda se to postiže tako da se navede dvotočka i nakon nje ime klase koja se nasljeđuje. Izvedena klasa sadrži tako sve metode atribute i sve metode osnovne klase.

Primjer 5.

Kreirajte klasu koja predstavlja studenta FERIT-a, koja nasljeđuje osnovnu klasu student. Kreirajte objekt izvedene klase i pozovite metodu *ToString()*. Koja će metoda biti pozvana? Koje metode su dostupne?

```
    class FeritStudent : Student
    {
        private string programmingLanguage;

        public FeritStudent() {
            programmingLanguage = "N/A";
        }

        public FeritStudent(string name, string surname, string programmingLanguage)
        : base(name, surname) {
            this.programmingLanguage = programmingLanguage;
        }

        public string ProgrammingLanguage {
            get { return this.programmingLanguage;  }
            set { this.programmingLanguage = value; }
        }

        public override string ToString()
        {
            return base.ToString() + " likes: " + ProgrammingLanguage;
        }
    }
```

```
        /************************ Program.cs ********************************/
        static void Main(string[] args)
        {
            Student student = new Student();
            Console.WriteLine(student.ToString());
            FeritStudent feritStudent = new FeritStudent();
            Console.WriteLine(feritStudent.ToString());
            Student studentRef = new FeritStudent(); // Polymorphism
            Console.WriteLine(studentRef.ToString());
        }
```

Zadatak 6.

Kreirajte vlastitu klasu koja predstavlja bilješku s dodanim vremenom, a koja nasljeđuje osnovnu klasu koja predstavlja bilješku. Za pohranu vremena koristiti objekt DateTime klase, podrazumijevano vrijeme za kreiranje objekta je trenutno vrijeme, a omogućiti i postavljanje vremena na novom objektu i promjenu vremena na postojećem objektu. Preopteretite metodu *ToString()* tako da zna prikazati i vrijeme bilješke.

4. Kolekcije

Kolekcije su specijalizirane klase koje se koriste za pohranu podataka. Uobičajeno predstavljaju neke od dobro poznatih podatkovnih struktura poput liste, reda ili rječnika. Funkcionalnosti poput dodavanja i uklanjanja elemenata ostvarene su implementacijom različitih sučelja tako da nude funkcionalnosti poput dodavanja ili uklanjanja elemenata, sortiranja i slično. Riječ je o generičkim klasama, a ovo u konačnici znači da konkretna lista može sadržavati objekte različitih klasa. Koje će objekte lista sadržavati određuje se prilikom deklaracije, eksplicitnim navođenjem tipa podatka u listi. Primjerom 6 prikazano je na koji je način moguće kreirati listu nasumično generiranih cijelih brojeva i sortirati ju.

Primjer 6.

Napišite program koji omogućuje popunjavanje liste proizvoljne duljine pseudo slučajnim cijelim brojevima. Nakon popunjavanja prikažite generirane brojeve ispisom na konzolu, sortirajte listu, a na kraju ispišite sortirane brojeve.

```
/************************ Program.cs ********************************/
        static void Main(string[] args)
        {
            int count = 10;
            const int UPPER_LIMIT = 10;

            List<int> randomNumbers = generateRandomNaturalNumbers(count, UPPER_LIMIT);
            Console.WriteLine(numbersToString(randomNumbers));
            randomNumbers.Sort();
            Console.WriteLine(numbersToString(randomNumbers));
        }

        private static List<int> generateRandomNaturalNumbers(int size, int upperLimit)
        {
            Random generator = new Random(); // Careful!
            List<int> randomNumbers = new List<int> ();
            for (int i = 0; i < size; i++)
            {
                randomNumbers.Add(generator.Next(1, upperLimit));
            }
            return randomNumbers;
        }

        public static String numbersToString(List<int> numbers)
        {
            StringBuilder stringBuilder = new StringBuilder();
            foreach (int number in numbers)
            {
                stringBuilder.Append(number).Append(", ");
            }
            return stringBuilder.ToString();
        }
```

Primjerom 7 prikazano kako je moguće kreirati listu studenata FERIT-a i rabiti ju za dodavanje i uklanjanje studenata.

Primjer 7.

Za klasu koja predstavlja studente dodajte još jedan parametar koji predstavlja prosjek ocjena. Kreirajte listu studenata FERIT-a, iz liste uklonite sve studente koji imaju prosjek ocjena manji od 4.0., sortirajte listu i ispišite preostale studente.

- Dodati atribut u klasu student
- Dodati svojstvo u klasu student
- Proširiti konstruktore klasa student i FERIT student
- Proširiti *ToString()* metodu klase student
- Implementirati sučelje IComparable (i dodati *CompareTo()* metodu propisanu sučeljem)

```
/************************ Program.cs ********************************/
        static void Main(string[] args)
        {
            int size = 10;
            double minAverageGrade = 4.0;
            Random generator = new Random();
            FeritStudentGenerator feritGenerator = new FeritStudentGenerator(generator);
            List<FeritStudent> students = feritGenerator.GenerateFeritStudents(size);
            Console.WriteLine("All Students: ");
            PrintStudents(students);
            SelectExcellentStudents(students, minAverageGrade);
            Console.WriteLine("Best Students: ");
            PrintStudents(students);
            students.Sort();
            Console.WriteLine("Sorted best Students: ");
            PrintStudents(students);
        }

        private static void SelectExcellentStudents(List<FeritStudent> students,
              double minAverageGrade)
        {
            for (int i = students.Count-1; i >= 0; i--)
            {
                if (students[i].GradeAverage < minAverageGrade)
                {
                    students.RemoveAt(i);
                }
            }
        }

        private static void PrintStudents(List<FeritStudent> students)
        {
            foreach (FeritStudent student in students)
            {
                Console.WriteLine(student);
            }
        }
```

```
    class FeritStudent : Student, IComparable
    {
        // add to FeritStudent:
        public int CompareTo(FeritStudent otherStudent)
        {
            return this.GradeAverage.CompareTo(otherStudent.GradeAverage);
        }
```

```
class FeritStudentGenerator
{
    private Random generator;
    private String prependName = "Anon_";
    private String prependSurname = "Anonymous_";
    private int maxStudentNumber = 10000;

    public FeritStudentGenerator(Random generator)
    {
        this.generator = generator;
    }

    public FeritStudent GenerateFeritStudent()
    {
        String name = GenerateRandomName();
        String surname = GenerateRandomSurname();
        String programmingLanguage = PickProgrammingLanguage();
        double gradeAverage = GenerateRandomGradeAverage();
        return new FeritStudent(name, surname, gradeAverage, programmingLanguage);
    }


    public List<FeritStudent> GenerateFeritStudents(int size)
    {
        List<FeritStudent> students = new List<FeritStudent>(size);
        for (int i = 0; i < size; i++)
        {
            students.Add(this.GenerateFeritStudent());
        }
        return students;
    }

    private String GenerateRandomName()
    {
        return "Anon_" + this.generator.Next(10000);
    }

    private string GenerateRandomSurname()
    {
        return "Anonymous_" + this.generator.Next(10000);
    }

    private string PickProgrammingLanguage()
    {
        return generator.NextDouble() > 0.5 ? "C++" : "C#";
    }

    private double GenerateRandomGradeAverage()
    {
        double minAverage = 5.0;
        double maxAverage = 1.0;
        return this.generator.NextDouble() * (maxAverage - minAverage) + minAverage;
    }
}
```

Zadatak 7.

Kreirajte klasu koja predstavlja ToDo listu koja omogućuje praćenje zadataka (zabilješki). Klasa mora omogućiti dodavanje i uklanjanje (obavljanje) zadataka preko javnog sučelja, dohvaćanje pojedinih bilješki te ispis cijele ToDo liste (ne vezati uz konzolu!). Kreirajte objekt navedene klase i napunite svoju ToDo listu zabilješkama čije ćete vrijednosti unijeti s konzole (barem tri). Nakon toga ispišite sadržaj ToDo liste na ekran, obavite sve zadatke s najvišom razinom prvenstva i ponovno ispišite sadržaj ToDo liste.

Paziti na enkapsulaciju, ne vezati izlaz uz konzolu, voditi računa o imenovanju!