SOFTVERSKO INŽENJERSTVO 2
SK 2020/21
PREDMETNI NASTAVNIK: DOC DR LJUBICA KAZI
DATUM: 11.12.2020.
DOPUNJENA VERZIJA NAKON CASA

TEMA 15: OSNOVE OBJEKTNO-ORJENTISANOG PROGRAMIRANJA SA PRIMERIMA U C#

OSNOVE SINTAKSE C# PROGRAMSKOG JEZIKA

Osnove sintakse

- Naredba predstavlja iskaz kojim se zadaje komanda računaru da izvrši neku operaciju. Kraj svake naredbe završava se simbolom tačka-zarez (";").
- Naredbe se mogu pisati jedna za drugom u istom redu, ali najčešće se pišu u posebnim redovima.
- Naredbe se pišu u blokovima oivičenim vitičastim zagradama { }. Blokovi određuju oblast vidljivosti promenljivih, odnosno oblast dejstva nekog koda. Moguće je da postoji više ugnježdenih blokova.
- Programski kod C# jezika razlikuje velika i mala slova ("case sensitive").
- Komentari se koriste da bi se neki deo programskog koda objasnio. Koristimo // za svaki red ili /* */ za blok.

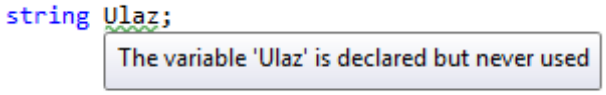
Promenljive, konstante i tipovi podataka

Promenljive predstavljaju nazive koji su dati memorijskim lokacijama u kojima se smeštaju podaci. Podaci u tim promenljivama se u toku izvršavanja programa mogu menjati. Promenljive su određene imenom, memorijskom lokacijom, tipom podatka i vrednošću.

Razlikujemo:

- Deklaracija promenljive – određivanje imena i tipa podatka promenljive.
  Primer: int ukupanBrojStudenata;
- Inicijalizacija promenljive – dodeljivanje početne vrednosti promenljivoj.
- Definicija promenljive – uključuje deklaraciju i inicijalizaciju.

U C# nije obavezna inicijalizacija promenljivih, ali će prilikom kompajliranja biti detektovano ukoliko promenljiva, koja je deklarisana, nije kasnije korišćena.



Možemo razlikovati 3 tipa promenljivih:

- Promenljive tipa vrednosti ("Value type") – promenljiva svojim imenom ukazuje na lokaciju gde se nalaze podaci, prilikom dodele vrednosti podaci se dodeljuju direktno.
- Promenljive tipa reference – promenljiva svojim imenom ukazuje na lokaciju koja sadrži pokazivač koji ukazuje na lokaciju gde se nalaze podaci nekog složenijeg tipa
- Promenljive tipa pokazivača – promenljiva sadrži pokazivač.

Dve osnovne grupe tipova podataka mogu biti:

- Prosti ("Primitivni")

- Složeni (“kompozitni”) – tipovi podataka koji predstavljaju uređene kolekcije drugih podataka, koje programeri definišu kao tipove podataka.

Uobičajeni tipovi podataka u programiranju:
- pokazivač
- mašinski (bit, bajt)
- Boolean
- numerički (celobrojni, realni)
- String, text
- Enumeracija
- slog, skup, klasa/objekat, interfejs
- Stek, red...
- Niz, lista, uredjena lista

*TRANSFORMACIJA TIPOVA PODATAKA – **TYPE CASTING***
Utvrđivanje tipa podatka za neku promenljivu moguće je primenom funkcije: TypeOf
Utvrđivanje veličine memorijske lokacije koju zauzima promenljiva moguće je funkcijom sizeof.

Postoje 2 vrste konverzije tipova podataka:
- Implicitna konverzija – dodela vrednosti promenljivih srodnih tipova, uz uzimanje u obzir principa da tip manjeg opsega može da dodeli svoju vrednost promenljivoj većeg opsega.

```
PRIMER IMPLICITNE KONVERZIJE:
    string Ulaz = txtVrednost.Text;
     int Duzina = Ulaz.Length;
     long LongDuzina = Duzina;
```

- Eksplicitna konverzij**a** – konverzija vrednosti čiji tipovi podataka ne moraju biti srodni, korišćenjem operatora i izraza za konverziju.
  - PRIMER: int i; string s=”123”;
    Prvi način: i=int.Parse(s);
    Drugi način: i= (int)s;
    Treći način: i=123; s = i.ToString();
    Četvrti način: long L=Convert.ToInt64(s);

SLOŽENIJI TIPOVI PODATAKA – STRUKTURE

Niz

Niz čuva sekvencijalnu kolekciju fiksnog broja elemenata istog tipa. Može se smatrati skupom promenljivih koje se čuvaju na susednim memorijskim lokacijama. Svakom element niza pristupa se putem indeksa, tj. rednog broja elementa niza.

Deklarišemo niz u C# koristeći sintaksu:

```
tipPodatka[] nazivNiza;
```

Uređena lista

Array lista predstavlja uređenu kolekciju objekata koji se mogu individualno indeksirati. Za razliku od niza, ovde se mogu dodavati ili uklanjati elementi liste na željenoj poziciji korišćenjem indeksa i array lista će automatski menjati dimenzije dinamički. Takođe omogućava dinamičku alokaciju memorije, dodavanje, pretragu i sortiranje stavki liste. Suština Array List je u tome što su elementi OBJEKTI, dakle može se dodeliti bilokoji tip elementa, a takođe je razlika u odnosu na niz jer ne postoji unapred definisan maksimalan broj elemenata.

```
ArrayList al = new ArrayList();
        al.Add(45);
        al.Add(78);
```

Tipizirana lista

List je sličan Array list, samo što se mora naglasiti tip podatka elementa i svi elementi su tog tipa. Takođe, broj elemenata ove liste nije unapred definisan i fiksan, već se može dinamički menjati.

```
List<int> intList = new List<int>();
intList.Add(10);
List<Student> studentList = new List<Student>();
studentList.Add(new Student());
```

NAZIVI ELEMENATA I PROMENLJIVIH - preporuke

1. Mnemonički nazivi Madjarskom notacijom– preporučljivo je nazivati elemente tako da prva tri slova opisuju tip elementa, a ostali deo naziva se odnosi na semantiku. Npr. lblNaslov – labela Naslov, objOsoba – objekat klase Osoba itd.

Novi standard: SemantikaPunTipObjekta, npr: naslovLabel

2. Velika i mala slova – s obzirom da je C# case sensitive (razlikuje nazive velikim i malim slovima) preporučljivo je da se koristi heurističko uputstvo određenog stila koji zavisi od konkretnog programskog jezika. Uobičajeno je u C# da se koristi PascalCasing i camelCasing stil i to za:

- **Camel Casing** - prvo slovo malo, ostala prva slova reči su velika, PRIMENA: privatni atributi (polja), parametri metoda, promenljive deklarisane unutar metoda;
- **PascalCasing** – Svako prvo slovo reci je veliko, ostala mala, PRIMENA: naziv klase, metode….

Programske strukture

Osnovne programske strukture su sekvenca, selekcija i iteracija.

SEKVENCA

a) jedna naredba

naredba;

b) blok naredbi
{blok naredbi}

NAPOMENA: promenljiva koja je deklarisana u okviru jednog bloka naredbi, vidljiva je samo u okviru tog bloka ili njemu podređenog (ugnježdenog) bloka naredbi, a spolja nije vidljiva, tj. ne može se koristiti. Navođenje imena promenljive koja nije vidljiva u nekom bloku rezultuje greškom.

SELEKCIJA

a) ispitivanje uslova i izvršavanje jedne naredbe, napisano u jednom redu.
if (uslov) {naredba1} else {naredba2};

b) ispitivanje uslova i izvršavanje bloka naredbi.
```
if (uslov)
{blok naredbi}
else
{blok naredbi};
```

c) više mogućih alternativa.
Vrednost izraza se poredi sa vrednostima konstanti koje su ponuđene iza Case .
```
switch (izraz)
{
Case konstanta1:
        Blok naredbi ili naredba1;
        break;
Case konstanta2:
        Blok naredbi ili naredba2;
        break;
Default : Blok naredbi ili naredba3 koja se izvršava ako izraz ne zadovolji nijednu od ponuđenih vrednosti;
}
```

ITERACIJA

a) ciklus sa preduslovom
```
While (uslov)
{blok naredbi

};
```

b) ciklus sa postuslovom
```
Do
{
Blok naredbi
} while (uslov);
```

c) ciklus sa fiksnim brojem iteracija
```
for (int i=0; i<100, i++)
{
```

```
Blok naredbi
};
```

d) ciklus sa brojem iteracija koje zavise od broja članova neke strukture

```
foreach (x in struktura)
{
Blok naredbi
};
```

OBRADA GREŠAKA

Osnovni oblik bloka naredbi kojom se realizuje detekcija i obrada grešaka:

```
Try
{
DEO KODA KOJI JE OSETLJIV I MOZE DA NAPRAVI GRESKU
}
Catch (ExceptionTip1 promenljivaGreska1)
{
OBRADA GRESKE TIPA 1
MessageBox.Show (promenljivaGreska1.Message);
}
Catch (ExceptionTip2 promenljivaGreska2)
{
OBRADA GRESKE TIPA 2
MessageBox.Show (promenljivaGreska2.Message);
}
Catch (ExceptionTipN promenljivaGreskaN)
{
OBRADA GRESKE TIPA N
MessageBox.Show (promenljivaGreskaN.Message);
}

Finally
{
BLOK KOJI SE SVAKAKO IZVRSAVA, AKO NASTUPI GRESKA ILI NE
}
```

Greška (Exception) je problem koji se javlja u okviru izvršavanja programa. Exception u C# je odgovor na okolnosti koje su dovele do greške. Exception u C# obezbeđuju način transfera kontrole od jednog dela programa na drugi. Osnovne ključne reči u obradi grešaka u C# su:

- Try - try blok identifikuje blok naredbi gde određeni tipovi grešaka mogu nastati I aktivirati neki od catch blokova koji slede.
- catch: greška se detektuje u try bloku, a u catch bloku se opisuje kako se određeni tip greške obrađuje.
- finally: deo koda koji se svakako mora izvršiti, bilo da je nastupila greška ili ne.
- **throw:** program generiše grešku određenog tipa putem naredbe throw.

DOPUNA: Razlikovati:

- throw Exception – kod za iniciranje da se desi greška
- throws exception- opis metode, naglašavanje da može da generiše grešku, da je osetljiv kod.

OBJEKTNO-ORJENTISANO PROGRAMIRANJE

PREDNOSTI

Neke od glavnih prednosti objektno-orjentisanog programiranja su:

- modularizacija – podela programskog koda na klase, gde svaka klasa ima svoje zaduženje
- mogućnost dorade programskog koda bez izmena osnovnog dela softvera koji dobro funkcioniše – omogućeno je nasleđivanjem i mogućnošću redefinisanja nasleđenih metoda
- navedene prednosti omogućuju timski rad, ponovnu iskoristivost programskog koda, olakšavaju održavanje, olakšavaju proširenja funkcionalnosti i sl.

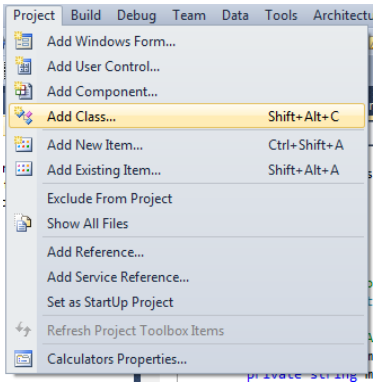
DEFINICIJA KLASE

Klasa je osnovni gradivni element objektno-orjentisanog programa. Opisuje se svojstvima (atributima) i ponašanjem (metodama). Objekat je konkretna pojava neke klase. Objekat se može smatrati promenljivom čiji je tip podataka zapravo klasa. Da bi se kreirao objekat u memoriji, potrebno je da se prvo deklariše da je određenog tipa (da pripada klasi, tj. da je objekat te klase), a zatim instancira. Instanciranje se realizuje eksplicitnim pozivom konstruktora klase (naredba new). Konstruktor priprema prostor u memoriji za smeštaj atributa klase, a obično sadrži i programski kod za inicijalizaciju svih atributa na početnu vrednost. Stanje objekta opisano je vrednostima njegovih atributa u svakom trenutku.

Objekat može da menja stanje. Preporučljivo je da se objektu menja stanje na osnovu poziva odgovarajućih metoda, a ne direktnim pristupom atributima. Zapravo, suštinski atributi objekta (nazivaju se i polja klase) su privatni, a može im se direktno pristupati samo ako imaju odgovarajućeg „predstavnika za javnost", tj. property, koji je definisan imenom i set-get metodama. Set metoda omogućuje dodeljivanje vrednosti atributu, a get metoda omogućava čitanje vrednosti iz atributa.

BIBLIOTEKE KLASA

Klase možemo imati u okviru programskog koda aplikacije, ali i organizovati u okviru posebne biblioteke klasa. Ekranske forme korisničkog interfejsa su takođe klase.



Kada se projekat tipa class library kompajlira, dobija se DLL fajl (Dynamic Link Library).

Da bismo mogli da koristimo neku gotovu ili naknadno dodatu biblioteku klasa, moramo:

1. dodati odgovarajuci DLL u References
2. pisati u gornjem delu programskog koda na mestu koriscenja (formi, drugoj klasi):

```
Using  NazivBibliotekeKlasa;
```

MODIFIKATORI PRISTUPA

Za klasu, attribute i metode moguće je dodeliti prava pristupa putem oznaka koje se ispred njihovih deklaracija postavljaju. Te oznake se zovu modifikatori pristupa.

- Private – vidljivost samo na nivou klase
- Public – vidljivost od strane korisnika biblioteke klasa spolja
- Protected – vidljivost samo od strane naslednika u okviru nasledjivanja klasa
- Internal – vidljivost u okviru istog projekta, tj. namespace.

DOPUNA: Ako je modifikator pristupa izostavljen, u C# se za klasu podrazumeva da je internal (spolja nevidljiv, tj, ponasa se kao private). U PHP, ako se izostavi modifikator pristupa za klasu, podrazumeva se public. U javi – ako se izostavi, vidljivost na nivou paketa (default tj. isto sto internal).

ELEMENTI KLASE

Klasa se opisuje kljucnom reci CLASS iza koje sledi naziv klase i blok naredbi koje pripadaju klasi.

```
Class NazivKlase {
}
```

Većina klasa (osim statičkih) da bi mogla da se kreira u memoriji, tj. obezbedi objekat klase instanciranjem, mora imati konstruktor. Konstruktor je eksplicitan kada se opisuje imenom, parametrima i blokom naredbi,a može postojati i implicitan (kada se ne vidi u programskom kodu, a podrazumeva se da postoji default konstruktor uvek koji barem priprema objekat klase u memoriji). Konstruktor počinje ključnom reči public nakon koje odmah sledi naziv klase čiji je to konstruktor. Konstruktor klase ekranske forme je automatski kreiran i vidljiv (sadrži naredbu: InitializeComponent() kojom se pripremaju grafičke kontrole za prikaz). Za klase koje programeri pišu preporučljivo je da postave konstruktor i da u bloku konstruktora inicijalizuju promenljive.

```
//klasa
Class NazivKlase {

// konstruktor
public NazivKlase () {
}
}
```

Konstruktor se u programskom kodu poziva naredbom NEW.
Primer: clsNazivKlase objNazivKlase = new clsNazivKlase ();

Često konstruktor ima parametre, kojima se klasi dostavljaju, u trenutku kreiranja, vrednosti koje su potrebne za rad. Tada se u telu konstruktora dodeljuju te vrednosti atributima klase.

U strukturi klase se najčešće nalaze atributi (polja) i metode (procedure, funkcije) koje se realizuju nad atributima i-ili parametrima metoda. Modifikator pristupa polja je uvek private i u nazivu, prema konvenciji, ima prefiks p ili m_. Polja ili atributi suštinski predstavljaju globalne promenljive na nivou klase i njih možemo pozivati i koristiti u bilokojoj metodi. Oni čuvaju vrednosti i opisuju stanje objekta klase u svakom trenutku. Izmenom atributa menja se stanje objekta.

```
Class NazivKlase {

// privatni atribut, tj. polje
private string m_prezime;

}
```

Da bi se atributu moglo pristupati, svaki atribut ima svog javnog predstavnika – **property (svojstvo).** Property se realizuju putem set i get metode i ne moraju uvek biti oba zastupljena. Ako je svojstvo readonly ima samo get, a ako je writeonly samo set. Public property je vezan za odgovarajući private atribut, tj. polje.

```
Class NazivKlase {

// privatni atribut, tj. polje
private string m_prezime;

// javni atribut, tj. svojstvo
public string Prezime {
get {
    return m_prezime;
}
      set {
            m_prezime=value;
            }
}
```

Konstruktor najviše služi za inicijalizaciju objekta klase, ali i za inicijalizaciju privatnih atributa, tj. postavljanje na početnu vrednost.

```
Class NazivKlase {

// privatni atribut, tj. polje
private string m_prezime;

// konstruktor
public NazivKlase () {
}
```

```
m_prezime="";
}
```

Kreiranje objekta klase u memoriji realizuje se inicijalizacijom, koja se kod objekata zove instanciranje. Naredni kod prikazuje deklaracija + instanciranje (poziv konstruktora naredbom new ukljucuje i inicijalizaciju promenljivih, tj. privatnih atributa, polja jer je tako navedeno u bloku naredbi konstruktora).

```
NazivKlase objNovaKlasa = new NazivKlase ();
```

Uništavanje objekta realizuje se naredbom DISPOSE, čime se poziva destruktor:

```
objNovaKlasa.Dispose();
```

Destruktor se najčešće ne navodi eksplicitno u kodu, ali može da ima sledeći izgled:

```
~NovaKlasa(){
//naredbe za izvršavanje prilikom uništavanja objekta
}
```

Dodeljivanje vrednosti atributu zapravo je poziv set metode koja pripada property:

```
objNovaKlasa.Prezime = "Markovic";
```

Metode predstavljaju aktivnosti ili operacije koje klasa može da sprovodi sa podacima koje dobije kroz parametre ili kroz interne podatke klase, tj. attribute. Public property je zasnovan na metodama set i get. U nastavku je dat primer za svojstvo I implementaciju putem set-get metoda.

```
Class NazivKlase {

// privatni atribut, tj. polje
Private string m_prezime;

// javni atribut, tj. svojstvo
Public string Prezime {
get {
    return m_prezime;
}
     Set {
          m_prezime=value;
          }
}
```

Drugi oblik metode je sa konkretnim operacijama, kao na primeru:

```
public string DajPrezimeIme() {
return this.m_prezime + " " + this.m_ime;
}
```

Metoda koja je definisana se koristi:

- Ako je poziv public property, poziva se set ili get metoda prilikom čitanja ili dodele vrednosti property.
- Pozivanje drugih tipova metoda, pri čemu se obraća pažnja da li metoda vraća vrednost, da bi se pripremila promenljiva adekvatnog tipa:

```
String PrezimeIme = objNovaKlasa.DajPrezimeIme();
```

Konstruktor može da ima parametre, koji predstavljaju ulazne vrednosti koje se mogu preuzeti i koristiti kao inicijalne vrednosti atributa.

```
// konstruktor
public NazivKlase (string pocetnoPrezime, string pocetnoIme) {
}
m_prezime=pocetnoPrezime;
m_ime=pocetnoIme;
}
```

Procedure i funkcije u klasama se zovu metode. Metode mogu imati ulazne, izlazne ili ulazno-izlazne parametre poziva, odnosno argumente. Parametri mogu biti prosleđeni po vrednosti (in – ulazni, out – izlazni) ili po referenci (ref – ulazno-izlazni).

PRIMER:

```
Private string NazivProcedure (ref int Prom1, in int prom2, out int3 prom3)
{

}
```

Kada se unutar klase u nekoj metodi obraćamo elementima iste klase, koristimo naredbu THIS.

PRIMER- pozivanje sopstvenih privatnih atributa:

```
String PunoIme = this.m_prezime + " " +  this.m_ime;
```

Objekat klase možemo poništiti naredbom Dispose.

```
Primer objNazivKlase.Dispose();
```

UOBIČAJENA STRUKTURA PROGRAMSKOG KODA KLASE

S obzirom da većina klasa pripada standardnim klasama, uobičajeno je da se prilikom pisanja programskog koda klasa poštuje konvencija o redosledu navođenja blokova programskog koda klasa:

```
MODIFIKATOR PRISTUPA   Naziv Klase

{

// atributi



// property



// konstruktor



// privatne metode
```

```
// javne metode

}
```

OSNOVNI PRINCIPI OBJEKTNO-ORJENTISANOG PROGRAMIRANJA

Osnovni principi objektno-orjentisanog programiranja su ENKAPSULACIJA, NASLEĐIVANJE i POLIMORFIZAM. Takođe, javljaju se i mogućnosti preklapanja metoda (OVERLOADING, u okviru jedne klase metode istog naziva) i redefinisanja metoda (OVERRIDING, istoimene metode u okviru osnovne i izvedene-klase naslednika).

Enkapsulacija

Enkapsulacija je skrivanje implementacije. Realizuje se korišćenjem modifikatora pristupa. Modifikator *private* omogućuje da se atribut ili metoda može koristiti samo na nivou klase, dok izvan klase nije vidljiv. Modifikator *protected* omogućava da atributi ili metode mogu biti vidljivi na nivou te klase i svih klasa naslednika.
Preporučljivo je da se atributi uvek deklarišu kao privatni i da se omogući minimalni pristup atributima putem public property. Preporučljivo je da se vrednost atributa može menjati kroz izvršavanje metoda.

Preopterećene (overloading) metode jedne klase

U okviru jedne klase može biti više metoda istog naziva, ali moraju imati različite parametre. Tada govorimo o preopterećenim (preklapajuće – OVERLOADING) metodama. Sve metode mogu biti preopterećene, pa i konstruktor – možemo imati više varijanti konstruktora – sa i bez parametara, sa različitim parametrima itd.

PRIMER:

```
public string DajPrezimeIme()
{
return this.m_prezime + " " + this.m_ime;
}

public string DajPrezimeIme(bool prvoPrezime)
{
        string izraz="";
        if (prvoPrezime)
        {
        Izraz = this.m_prezime + " " + this.m_ime;
        }
        Else
        {
        Izraz = this.m_ime + " " +  this.m_prezime;

        }
        return izraz;
}
```

Nasleđivanje klasa

Nasleđivanje klasa je mogućnost da jedna klasa preuzme atribute i funkcionalnost druge klase, uz mogućnost dopuna i izmena. Govorimo o odnosu nadređene (osnovne, bazne, opštije, tj. „roditelj") klase i podređene (izvedene, preciznije, tj. „dete") klase. Nasleđivanje se obeležava sa :

```
Public class IzvedenaKlasa: BaznaKlasa
{
}
```

U okviru konstruktora izvedene klase može se pozvati konstruktor osnovne klase.

```
Public IzvedenaKlasa (): base ()
{
}
```

Pomoću reči „base" pozivamo se na baznu klasu, nakon čega može slediti navođenje svojstava ili metoda bazne klase. Bazna klasa za sve klase je Object. Objekat klase naslednika može da vidi i da pristupa svim public i protected svojstvima i metodama.

Nadjačavanje (redefinisanje - override) svojstava i metoda u izvedenoj klasi

Nadjačavanje (OVERRIDE) svojstava i metoda u izvedenoj klasi daje mogućnost da se nasleđena svojstva i metode sa istim imenom ponovo navedu, ali se njihova implementacija izmeni, tj. redefiniše. Na ovaj način nova implementacija nadjačava nasleđenu, pa se prilikom poziva metode sa istim nazivom poziva redefinisana, a ne nasleđena metoda. Da bi nadjačavanje bilo moguće, u osnovnoj klasi je potrebno za takva svostva I metode postaviti ključnu reč VIRTUAL, a u izvedenoj klasi OVERRIDE.

```
U osnovnoj klasi:    Virtual public string ID
U izvedenoj klasi:   Override public string ID
```

Specifične vrste klasa

- SEALED klasa – zapečaćena, ne može biti nasleđena
- ABSTRACT klasa – mora biti nasleđena, ne može imati objekte
- **STATIC** klasa – ne može imati objekte, sadrži statičke vrednosti atributa (konstante) i statičke metode, poziva se nazivom klase, nakon čega sledi naziv atributa ili metoda.
- INTERFEJS klasa – predstavnik jedne ili više klasa, predstavlja ugovor o tome šta će biti implementirano. Sakriva ko i kako implementira svojstva i metode. Ne sadrži implementaciju, već samo okvir. Konkretna klasa nasleđuje klasu interfejsa i implementira njene okvirne elemente.

```
Public Interface imeinterfejsa {

Svojstvo:

Int Imesvojstva {
Get;
Set;
}

Metod
```

```
Void NazivMetode (parameter…);

}
```

Klasa koja implementira interfejs:

```
Public class KonkretnaKlasaInterfejsa: KlasaInterfejsa
{
}
```

Polimorfizam

DOPUNA – ovaj deo je bolje objašnjen
Definicija - Polimorfizam omogućuje da se objekti klasa ponašaju (izvršavaju metode) različito u odnosu na kontekst, tj. tip objekta.
Objašnjenje - Objekti se ponašaju u odnosu na svoja nadjačana svojstva ili metode, drugačije nego što je nasleđeno od bazne klase. Moguće je objektu osnovne klase dodeliti referencu na izvedenu klasu, međutim takav objekat treba da primeni metode izvedene klase.

PRIMER gde se može ilustrovati polimorfizam je tipizirana lista objekata gde su u definiciji inicijalno postavljeni kao tip elementa - objekti bazne klase. Dodeljujemo kao vrednosti zapravo reference na objekte bazne I izvedene klase. Pozivamo metodu istog naziva iz bazne I izvedene klase za sve elemente liste. Kada se realizuje metoda gde je elemenat liste zapravo objekat bazne klase, poziva se metoda bazne klase, a kada je element liste objekat izvedene klase, poziva se istoimena metoda izvedene klase.

NEKI VAŽNI POJMOVI I PROCESI

- **Garbage collection** – proces upravljanja memorijom, gde se memorija čisti od zaostalih instanci objekata koji se ne koriste i-ili promenljivih. Ovo je mehanizam koji programmer ne kontroliše, već se realizuje automatski. Svaka promenljiva ima oblast važenja u svom bloku naredbi i nakon završetka bloka ili završetka procedure, sve promenljive se brišu iz memorije automatski.
- **Serijalizacija** – snimanje vrednosti atributa objekata klasa u određen tip fajlova, npr. XML. Zato postoji osobina "serializable", koja naglašava da li je objekte neke klase moguće sačuvati trajno u okviru neke datoteke.

Primeri primene elemenata objektno-orjentisanog programiranja

Zadatak
Na kartici za štampu dodati RichTextBox ispod ReportViewer kontrole. Na osnovu sadržaja NastavnoOsoblje.XML ispisati podatke o nastavnom osoblju u RichTextBox-u, ali ispis realizovati pozivom polimorfnih metoda objekata klasa iz liste objekata nastavnog osoblja. Ukoliko je nastavnik, ispisati zvanje, a ako nije, samo prezime i ime.

Postupak:

- Kreirati NastavnoOsoblje.XML fajl sa podacima o nastavnom osoblju (Prezime, Ime, DatumRodjenja, SifraZvanja). Ako u polju za zvanje ne pise nista, rec je o asistentu.
- Primenom prethodno navedene metode ucitati XML u dataset.
- Kreirati projekat tipa class library i biblioteku klasa KlasePodataka.
- Kreirati klasu clsNastavnoOsoblje koje ima kao atribute i svojstva Ime, Prezime, DatumRodjenja, a kao metode set-get zbog property i metodu DajPodatke.
- Kreirati klasu clsNastavnik koja nasledjuje klasu clsNastavnoOsoblje i ima dodatni atribut i svojstvo Zvanje(**to svojstvo je objekat klase clsZvanje**). Redefinisati nasledjenu metodu DajPodatke tako da pored ostalih podataka koje daje, ukljuci i podatke o nazivu zvanja.
- Kreirati tipiziranu listu nastavnog osoblja i napuniti je objektima klase nastavno osoblje (za svaki element liste se cita dataset, instancira objekat clsNastavnoOsoblje i pune atributi vrednostima iz dataseta, a kada se naidje na zapis iz dataseta koji ima zvanje, instancira se objekat klase clsNastavnik ion tada dodeljuje kao element liste). Prilikom citanja i kreiranja objekata kreirati i metodu DajNazivZvanja (sifra zvanja) koja se nalazi u korisnickom interfejsu (inace bi trebala da bude u posebnoj klasi).
- Ispis vrednosti o podacima nastavnog osoblja u RichTextBox putem poziva metode DajPodatke redom, za svaki clan tipizirane liste.

Obrazlozenje

U ovom zadatku ilustrujemo osnovne koncepte objektno-orjentisanog programiranja, u kombinaciji sa tipiziranim listama.

- Enkapsulacija – implementacija klasa sa privatnim atributima i svojstvima putem get/set metoda.
- Nasleđivanje – klasa Nastavnik nasleđuje klasu NastavnoOsoblje.
- Odnos klasa tipa asocijacije – klasa Nastavnik sadrži objekat klase Zvanje.
- Tipizirana lista sa elementima koji su zapravo objekti tipa reference, tj. objekti klasa.
- Polimorfizam – tipizirana lista objekata NastavnoOsoblje sadrži objekte tog tipa i tipa naslednika, tj. klase Nastavnik. Pozivima metoda DajPodatke izvršiće se odgovarajuća varijanta ove metode, uz uvažavanje redefinisanja (override) koje je realizovano u okviru klase naslednika.

DOPUNA - Napomena: U navedenom primeru postoji mala nepreciznost – Izvor podataka bi trebao Zaposleni.XML, a takodje i bazna klasa Zaposleni. Cilj je prikazati pojednostavljenu realizaciju spiska svih zaposlenih, pri čemu neki imaju nastavno-naučno zvanje (to su nastavnici I saradnici), dok drugi zaposleni nemaju (npr. nenastavno osoblje). U ovom primeru je dat još jednostavniji kontekst koji nije zastupljen u realnosti – da postoje clanovi nastavnog osoblja koji nemaju naastavno-naucno zvanje, a to bi bili saradnici I asistenti. Dakle, da bi ovaj primer bio realniji, trebalo bi da se XML I bazna klasa zovu Zaposleni ili Osoblje.

REŠENJE:

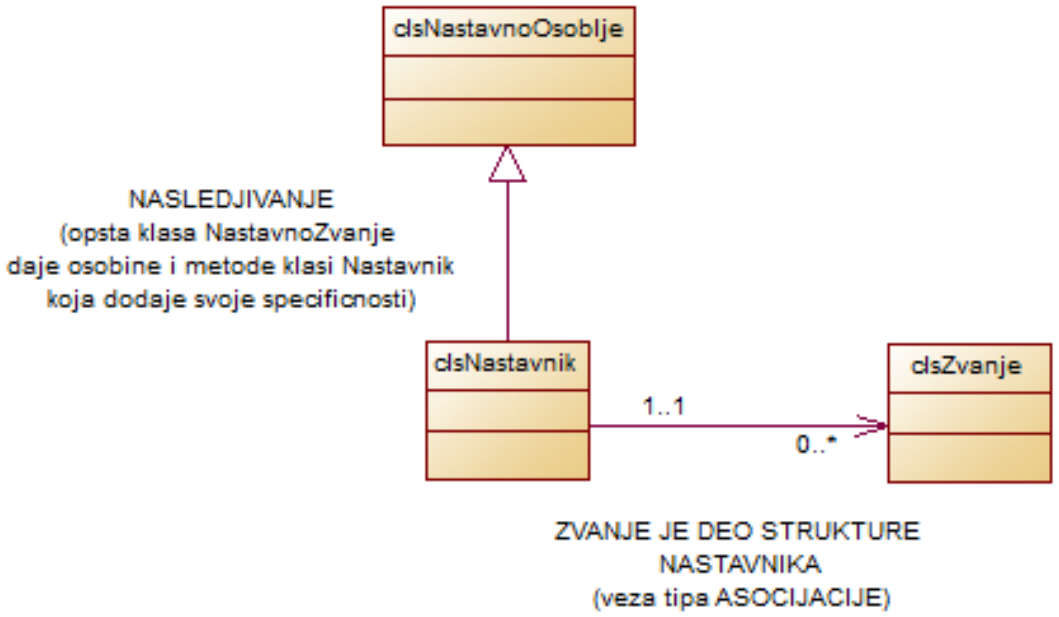
NastavnoOsoblje.XML

Obratićemo pažnju na format datuma GODINA/MESEC/DAN, da bi ga mogla prepoznati kasnije naredba DateTime.Parse (stringDatuma) i pretvoriti u datumsku vrednost.

```
<?xml version="1.0" standalone="yes"?>
<SpisakNastavnoOsoblje>
<NastavnoOsoblje>
<Prezime>Markovic</Prezime>
<Ime>Marko</Ime>
<DatumRodjenja>1967/06/12</DatumRodjenja>
<Zvanje></Zvanje>
</NastavnoOsoblje>
<NastavnoOsoblje>
<Prezime>Jovanovic</Prezime>
<Ime>Jovan</Ime>
<DatumRodjenja>1970/02/23</DatumRodjenja>
<Zvanje>1</Zvanje>
</NastavnoOsoblje>
<NastavnoOsoblje>
<Prezime>Ivanovic</Prezime>
<Ime>Ivan</Ime>
<DatumRodjenja>1950/08/14</DatumRodjenja>
<Zvanje>2</Zvanje>
</NastavnoOsoblje>
<NastavnoOsoblje>
<Prezime>Ivanic</Prezime>
<Ime>Ivana</Ime>
<DatumRodjenja>1949/12/12</DatumRodjenja>
<Zvanje></Zvanje>
</NastavnoOsoblje>
</SpisakNastavnoOsoblje>
```

BIBLIOTEKA KLASA „KlasePodataka.dll“

U ovoj biblioteci klasa imamo 3 klase, čiji su odnosi predstavljeni sledećom slikom:



Klasa clsZvanje

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

namespace KlasePodataka
{
Public class clsZvanje
    {
// atributi
privateint pSifra;
privatestring pNaziv;

// public property
publicint Sifra
        {
get
            {
return pSifra;
            }
set
            {
if (this.pSifra != value)
this.pSifra = value;
            }
        }

publicstring Naziv
        {
get
            {
return pNaziv;
            }
set
            {
if (this.pNaziv != value)
this.pNaziv = value;
            }
        }

// konstruktor
public clsZvanje()
        {
pSifra = 0;
pNaziv = "";
        }


// privatne metode

// javne metode
```

```
publicstring DajPodatke()
        {
returnthis.pSifra + " " + this.pNaziv;
        }


    }
}
```

Klasa clsNastavnoOsoblje

– omogućava redefinisanje metode DajPodatke prilikom nasleđivanja, jer sadrži reč **virtual**

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

namespace KlasePodataka
{
Publicclass clsNastavnoOsoblje
    {
// atributi
privatestring pPrezime;
privatestring pIme;
privateDateTime pDatumVremeRodjenja;

// public property

publicstring Prezime
        {
get
            {
return pPrezime;
            }
set
            {
if (this.pPrezime != value)
this.pPrezime = value;
            }
        }

publicstring Ime
        {
get
            {
return pIme;
            }
set
            {
if (this.pIme != value)
this.pIme = value;
            }
        }
```

```
publicDateTime  DatumVremeRodjenja
        {
get
            {
return pDatumVremeRodjenja;
            }
set
            {
if (this.pDatumVremeRodjenja != value)
this.pDatumVremeRodjenja = value;
            }
        }

// konstruktor
public clsNastavnoOsoblje()
        {
            pPrezime = "";
            pIme = "";
            pDatumVremeRodjenja = DateTime.Now;
        }

// privatne metode

privatestring DajDatum()
        {
string Datum="";
            Datum = this.pDatumVremeRodjenja.Day.ToString() + "." +
this.pDatumVremeRodjenja.Month.ToString() + "." +
this.pDatumVremeRodjenja.Year.ToString() + ".";

return Datum;
        }

// public metode

Public virtual string DajPodatke()
        {
returnthis.pPrezime + " " + this.pIme + " " + this.DajDatum();
        }


    }
}
```

Klasa clsNastavnik

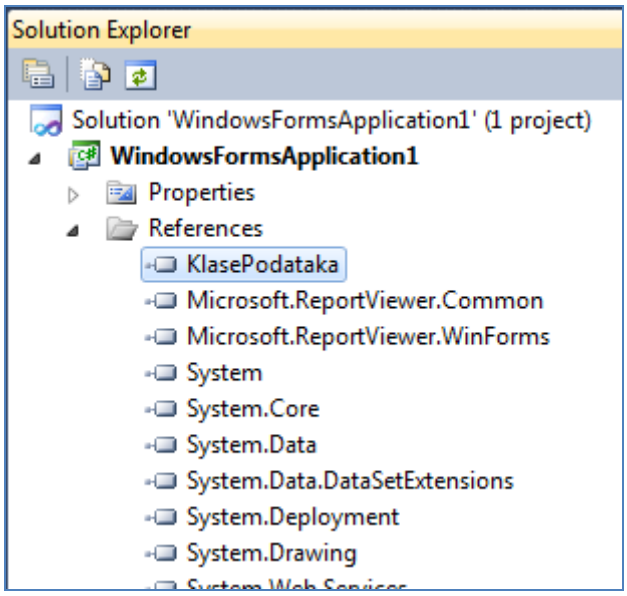
nasleđuje klasu clsNastavnoOsoblje i redefiniše metodu DajPodatke naredbom override.

DOPUNA: sadrži objekat klase clsZvanje, zato u konstruktoru treba da se instancira taj objekat.

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

```
namespace KlasePodataka
{
publicclassclsNastavnik: clsNastavnoOsoblje
    {
// atributi
privateclsZvanje pZvanje;

// public property
publicclsZvanje Zvanje
        {
get
            {
return pZvanje;
            }
set
            {
if (this.pZvanje != value)
this.pZvanje = value;
            }
        }

      // konstruktor
public clsNastavnik(): base ()
        {
// ******** konstruktor inicijalizuje sve promenljive
//pozivom base () pokrece se konstruktor bazne klase
//
// 1. Nacin - ovo nije dobro jer nepotrebno pravimo jos jedan objekat:
//clsZvanje objZvanje = new clsZvanje();
//pZvanje = objZvanje;

// ovo je dodatak u odnosu na ono sto se desava u konstruktoru bazne klase
      // 2. Nacin – ovo je dovoljno
            pZvanje = new clsZvanje();
        }

// privatne metode

// javne metode

Public override string DajPodatke()
        {
returnbase.DajPodatke () + " " +  pZvanje.DajPodatke ();
        }

    }
}
```

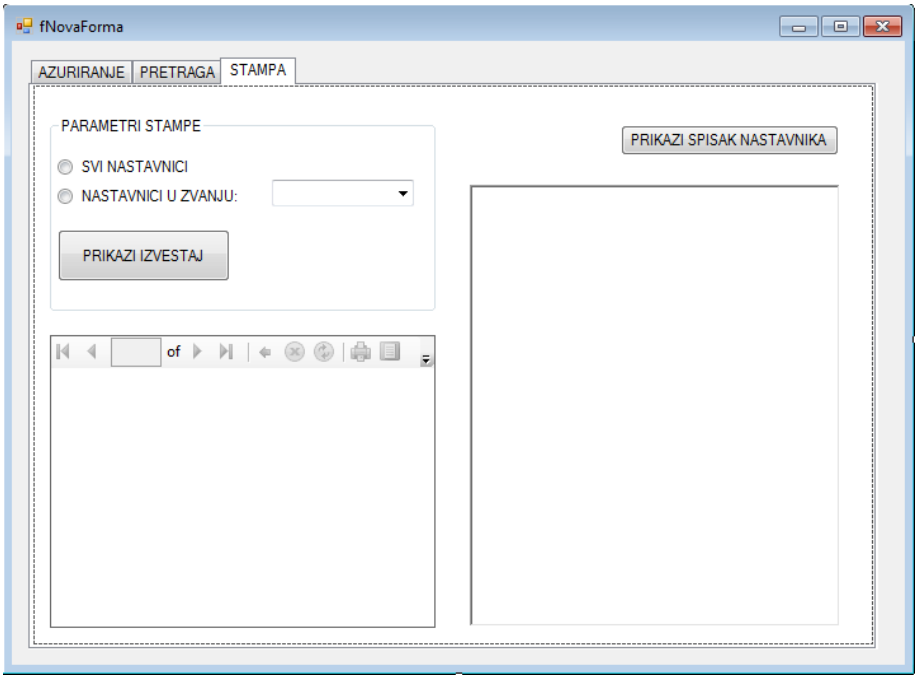
Kreiramo KlasePodataka.dll fajl putem Build komande. Povezujemo KlasePodataka.dll za korisnički interfejs Windows aplikacije.



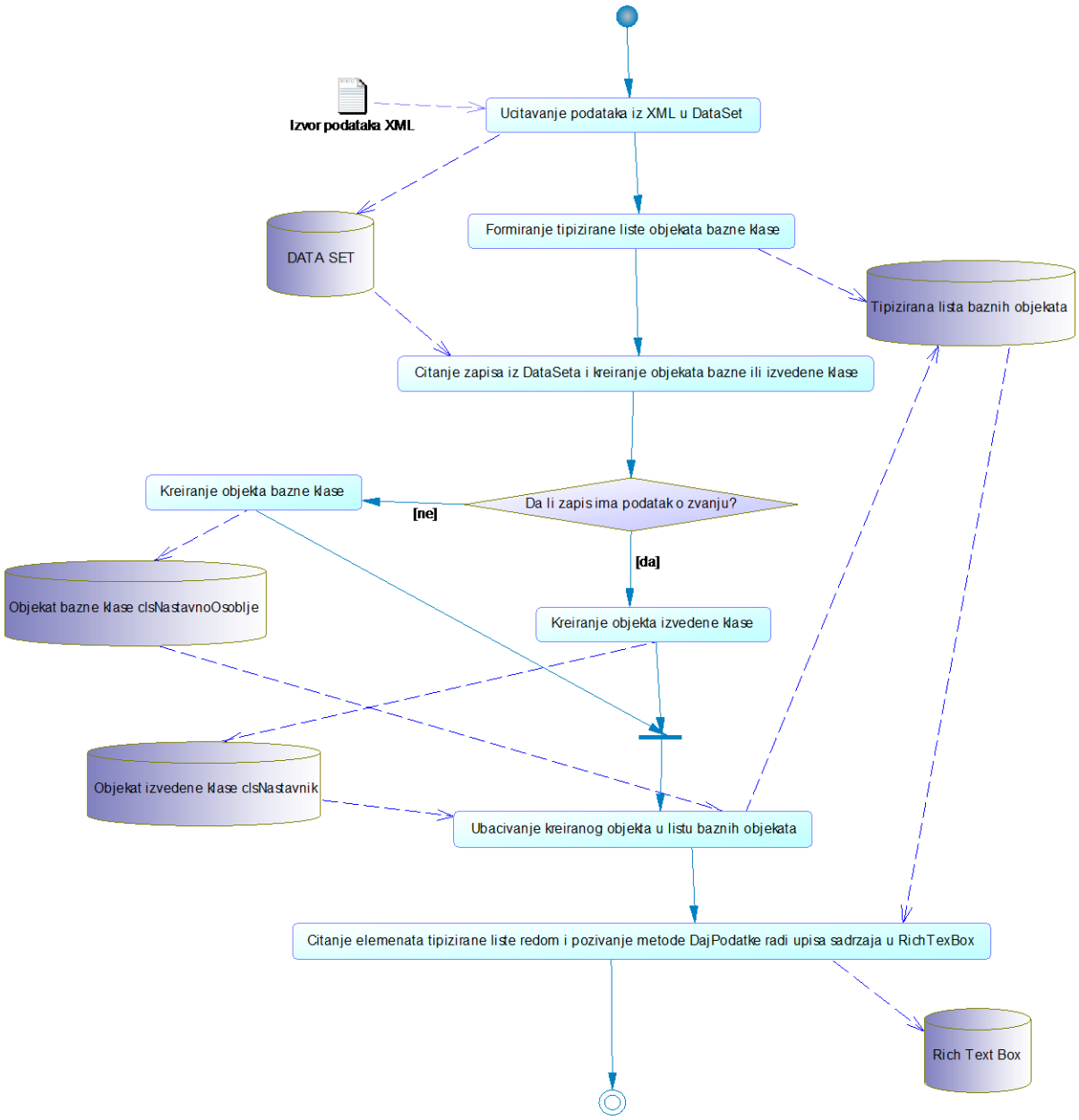
Ubacujemo using na početku forme:

```
using KlasePodataka;
```

Ubacujemo Rich text box i taster u desni deo ekranske kartice za stampu.



Redosled izvršavanja programskog koda kojim se puni RichTextBox podacima iz XML-a je dat u nastavku:



DataSet nakon ucitavanje XML fajla sadrzi:

RB KOLONE →	0	1	2	3
RB ZAPISA	PREZIME	IME	DATUM RODJENJA	ZVANJE
0	Markovic	Marko	1967/06/12	
1	Jovanovic	Jovan	1970/02/23	1
2	Ivanovic	Ivan	1950/08/14	2
3	Ivanic	Ivana	1949/12/12	

Za 0. i 3. Zapis sifra zvanja nije data i za te zapise ce se kreirati objekat bazne klase clsNastavnoOsoblje koji ima samo kao atribute: Prezime, Ime I DatumRodjenja. Za zapis 1 I 2 ce se kreirati objekat izvedene (nasledjene) klase clsNastavnik.

Tipizirana lista objekata bazne klase može da sadži kao elemente objekte bazne klase ili izvedene (nasleđene klase). U ovom slučaju, lista objekata će sadržati ovakve elemente:

Element tipizirane liste objekata bazne klase clsNastavnoOsoblje Pozicija [0]	Element tipizirane liste objekata bazne klase clsNastavnoOsoblje Pozicija [1]	Element tipizirane liste objekata bazne klase clsNastavnoOsoblje Pozicija [2]	Element tipizirane liste objekata bazne klase clsNastavnoOsoblje Pozicija [3]
|	|	|	|
OBJEKAT BAZNE KLASE	**OBJEKAT IZVEDENE KLASE**	**OBJEKAT IZVEDENE KLASE**	OBJEKAT BAZNE KLASE

Kada se pristupa elementima ove liste, za svaki element se poziva metoda sa istim nazivom DajPodatke. Kada se pristupi elementu bazne klase, pokrenuće se njegova verzija ove metode (prikazuje samo prezime, ime I datum rodjenja), a kada se pristupi elementu izvedene klase, pokrenuće se redefinisana verzija istoimene metode, tako da će se prikazati još I naziv zvanja. Na ovaj način smo ilustrovali **polimorfizam**.

Programski kod za prikaz spiska nastavnika:

```
privatevoid btnPrikaziSpisakNastavnika_Click(object sender, EventArgs e)
        {
// PROMENLJIVE
List<clsNastavnoOsoblje> objListaNastavnogOsoblja =
newList<clsNastavnoOsoblje>();
clsNastavnoOsoblje objNastavnoOsoblje;
clsNastavnik objNastavnik;
clsZvanje objZvanje;

string pomPrezime = "";
string pomIme = "";
DateTime pomDatumVremeRodjenja = DateTime.Now; // inicijalna vrednost
string pomDatumRodjenjaString = "";
int pomSifraZvanja = 0;
string pomNazivZvanja = "";

// ----------------------------------------
// preuzimanje podataka iz XML u DataSet
DataSet dsPodaciNastavnoOsoblje =
PreuzmiPodatkeIzXML(Application.StartupPath, "NastavnoOsoblje.XML");
```

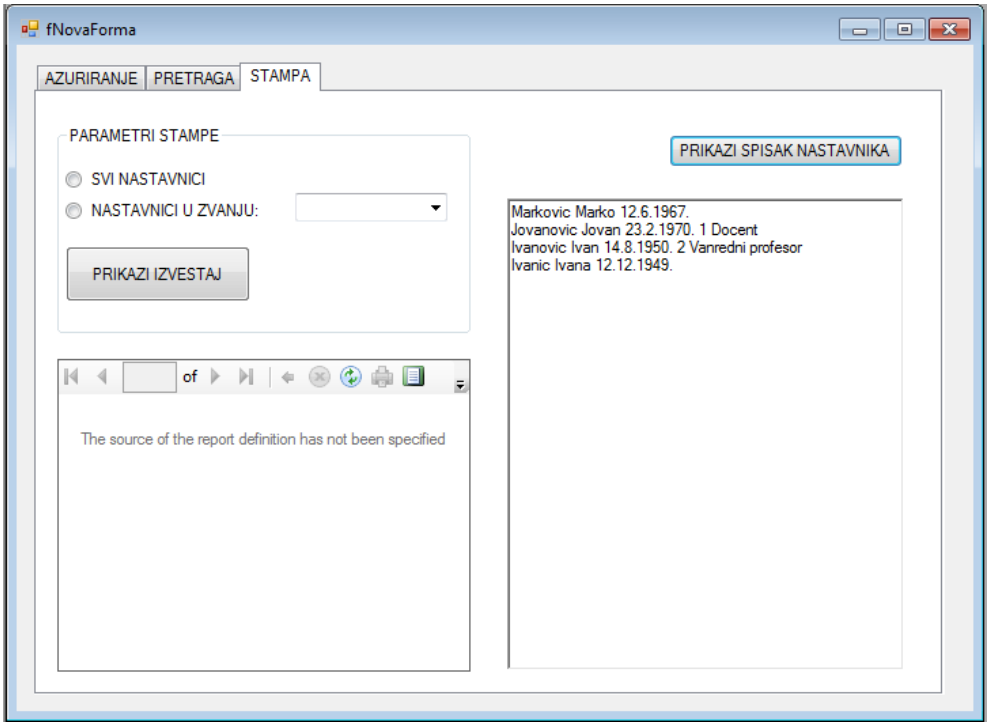
```
DataSet dsPodaciZvanja =PreuzmiPodatkeIzXML(Application.StartupPath,
"Zvanja.XML");
// ----------------------------------------
// formiranje liste objekata
int maxBrojZapisa = dsPodaciNastavnoOsoblje.Tables[0].Rows.Count;

for (int brojac = 0; brojac < maxBrojZapisa; brojac++)
            {
// preuzimanje podataka iz DataSeta
                pomPrezime =
dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[0].ToString();
                pomIme =
dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[1].ToString();
                pomDatumRodjenjaString =
dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[2].ToString();
                pomDatumVremeRodjenja =
DateTime.Parse(pomDatumRodjenjaString);

if
(dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[3].ToString().Equal
s(""))
                {
                    objNastavnoOsoblje = newclsNastavnoOsoblje();
                    objNastavnoOsoblje.Prezime = pomPrezime;
                    objNastavnoOsoblje.Ime = pomIme;
                    objNastavnoOsoblje.DatumVremeRodjenja =
pomDatumVremeRodjenja;
                    objListaNastavnogOsoblja.Add(objNastavnoOsoblje);
                }
else// popunjeno je polje sa sifrom zvanja
                {
// preuzimamo vrednost iz data seta za sifru zvanja
                    pomSifraZvanja =
int.Parse(dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[3].ToStrin
g());

// konvertujemo sifru zvanja u naziv zvanja
                    pomNazivZvanja = DajNazivZvanja(dsPodaciZvanja,
                    pomSifraZvanja.ToString());
// pripremamo objekat klase Zvanje
                    objZvanje = newclsZvanje();
                    objZvanje.Sifra = pomSifraZvanja;
                    objZvanje.Naziv = pomNazivZvanja;

// instanciramo objekat tipa nastavnika
                    objNastavnik = newclsNastavnik();
                    objNastavnik.Prezime = pomPrezime;
                    objNastavnik.Ime = pomIme;
                    objNastavnik.DatumVremeRodjenja = pomDatumVremeRodjenja;
                    objNastavnik.Zvanje = objZvanje;

                    objListaNastavnogOsoblja.Add(objNastavnik);
                }
            };
// ----------------------------------------
// prikaz podataka u Rich Text Box
```

```
// JEDNOSTAVNIJI NACIN
for (int brojac = 0; brojac < maxBrojZapisa; brojac++)
            {

rtbSpisakNastavnika.AppendText(objListaNastavnogOsoblja[brojac].DajPodatke()
+ System.Environment.NewLine);
            }

// BOLJI NACIN
//foreach (var element in objListaNastavnogOsoblja)
//{
//    rtbSpisakNastavnika.AppendText(element.DajPodatke() +
System.Environment.NewLine);
//}
        }
```

U ovom kodu se poziva metoda DajNazivZvanja koja iz datog dataseta u parametru poziva trazi sifru zvanja iz drugog parametra I vraca naziv zvanja, sto je potrebno prilikom prikazivanja u RichTextBox jer se u XML-u cuva ID zvanja, ali je korisniku bolje da vidi naziv.

```
private string DajNazivZvanja(DataSet dsPodaciZvanja, string SifraZvanja)
     {
        string pomNazivZvanja = "";
        int maxBroj = dsPodaciZvanja.Tables[0].Rows.Count;
        // provera vrednosti
        for (int brojac = 0; brojac < maxBroj; brojac++)
        {
           if
(dsPodaciZvanja.Tables[0].Rows[brojac].ItemArray[0].ToString().Equals(SifraZvanj
a))
           {
              pomNazivZvanja =
dsPodaciZvanja.Tables[0].Rows[brojac].ItemArray[1].ToString();
              break;
           }
        }


        return pomNazivZvanja;
     }
```

REZULTAT:



LITERATURA

1. Kazi Lj, Radosav D, Osnove objektno-orjentisanog programiranja sa primerima u C#, TF M Pupin, 2018.

TEST PITANJA

OSNOVE C# PROGRAMIRANJA

1. Koja je razlika između deklaracije, inicijalizacije i definicije promenljive?
2. Dve osnovne grupe tipova podataka, objasniti i navesti primere za C#.
3. Objasniti typecasting.
4. Razlika izmedju implicitnog i eksplicitnog type castinga.
5. Razlika izmedju niza, uredjene liste i tipizirane liste u C#.
6. Dva nacina pisanja naziva promenljivih – mnemonicki naziv madjarskom notacijom i novi standard.
7. Koja je razlika između Pascal Casing i camelCasing?
8. Osnovne programske strukture.
9. Razlika for I foreach.
10. Osnovna obrada gresaka u C#.
11. Objasniti Exception I throw exception.

OBJEKTNO-ORJENTISANO PROGRAMIRANJE

12. Prednosti OOP.
13. Definicija klase.
14. Biblioteke klasa.
15. Objasnjenje DLL.
16. Kako koristimo biblioteke klasa u drugim projektima?
17. Modifikatori pistupa – navesti, objasniti.
18. Struktura i elementi uobicajene klase.
19. Namena konstrukora.
20. Koja je razlika između property i atributa klase?
21. Vrste parametara procedura i funkcija.
22. Znacenje this u OOP.
23. Navesti osnovna 3 principa objektno-orjentisanog programiranja.
24. Kako se realizuje enkapsulacija?
25. Šta je nasleđivanje?

26. Kako se označava nasleđivanje u C#?
27. Šta klasa dete nasleđuje od klase roditelj?
28. Šta je polimorfizam?
29. Koja je razlika između method overriding i method overloading?
30. Specifične vrste klasa, navesti i objasniti.
31. Objasniti statičku klasu.
32. Kako se koristi statička klasa?
33. Objasniti garbage collection u programiranju.
34. Šta znači serijalizacija?