UNIVERZITET U NOVOM SADU
TEHNIČKI FAKULTET „MIHAJLO PUPIN“
ZRENJANIN

Doc. dr Ljubica Kazi
Prof. dr Dragica Radosav

OSNOVE OBJEKTNO-ORJENTISANOG PROGRAMIRANJA SA PRIMERIMA U C#

-praktikum za vežbe-

Elektronski udžbenik

ISBN 978-86-7672-307-2

Zrenjanin, 2018. godine

UNIVERZITET U NOVOM SADU
TEHNIČKI FAKULTET „MIHAJLO PUPIN“
ZRENJANIN

Doc. dr Ljubica Kazi
Prof. dr Dragica Radosav

OSNOVE OBJEKTNO-ORJENTISANOG PROGRAMIRANJA SA PRIMERIMA U C#

-praktikum za vežbe-

Zrenjanin, 2018. godine

Autori:
Doc. dr Ljubica Kazi
Prof. dr Dragica Radosav

Recenzenti:
Prof. dr Biljana Radulović
Prof. dr Ivana Berković

Izdavač:
Tehnički fakultet „Mihajlo Pupin" Zrenjanin

Za izdavača:
Prof. dr Dragica Radosav, dekan

Nastavno-naučno veće Tehničkog fakulteta „Mihajlo Pupin" u Zrenjaninu donelo je odluku 18.1.2017. godine da se rukopis ove knjige može koristiti kao udžbenik Fakulteta.

Tehnička obrada:
Doc. dr Ljubica Kazi

Dizajn korice:
Doc. dr Ljubica Kazi

Elektronski udžbenik na CD-u
Biblioteka „Udžbenici", broj 226, školska 2017/2018. godina
ISBN (elektronski): 978-86-7672-307-2

CIP - Каталогизација у публикацији
Библиотека Матице српске, Нови Сад

004.4(075.8)(076)

КАЗИ, Љубица
Osnove objektno-orijentisanog programiranja sa primerima u C# [Elektronski izvor] : praktikum za vežbe / Ljubica Kazi, Dragica Radosav. - Zrenjanin : Tehnički fakultet "Mihajlo Pupin", 2018. - 1 elektronski optički disk (CD-ROM) ; 12 cm. - (Biblioteka "Udžbenici" / Tehnički fakultet "Mihajlo Pupin", Zrenjanin)

Nasl. sa naslovnog ekrana. - Bibliografija.

ISBN 978-86-7672-307-2
1. Радосав, Драгица
а) Софтверско инжењерство - Објектно-оријентисано програмирање - Практикуми
COBISS.SR-ID 320959495

SADRŽAJ:

1. UVOD 3
2. UML MODELOVANJE SOFTVERA 4
2.1. Teorija UML dijagrama i CASE alat Power Designer 4
2.2. Primeri UML dijagrama u dizajnu poslovnog aplikativnog softvera 8
2.2.1. Primeri use case dijagrama 8
2.2.2. Primeri dijagrama komponenti 12
2.2.3. Primer dijagrama razmeštaja 15
2.3. Primer dokumentovanja softverskog rešenja UML dijagramima 18
2.3.1. Zadatak 18
2.3.2. Rešenje 19
2.4. Primer generisanja programskog koda na osnovu modela iz CASE alata 26
2.4.1. Zadatak 26
2.4.2. Rešenje – kreirana biblioteka klasa 27
2.4.3. Analiza generisanog koda klasa i uticaja vrsta veza između klasa 31
3. UPOZNAVANJE SA RAZVOJNIM OKRUŽENJEM VISUAL STUDIO .NET 33
3.1. Windows forms projekat 34
3.2. Class library projekat 39
4. GRAFIČKO OBLIKOVANJE WINDOWS APLIKACIJE I POVEZIVANJE FORMI 40
4.1. Zadatak 40
4.2. Rešenje 40
4.2.1. Forma za prijavljivanje korisnika 40
4.2.2. Forma za glavni meni 42
4.2.3. Forma za rad sa podacima 45
5. SINTAKSA C# PROGRAMSKOG JEZIKA 48
5.1. Osnove sintakse 48
5.2. Promenljive, konstante i tipovi podataka 48
5.3. Operatori i funkcije 52
5.4. Programske strukture 54
5.5. Obrada grešaka 56
5.5.1. Testiranje programa 56
5.5.2. Naredbe za detekciju i obradu grešaka 56
5.5.3. Tipovi standardnih klasa za obradu grešaka 57
5.5.4. Kreiranje sopstvenih klasa za obradu grešaka 58

5.6. Primer primene sintaksnih elemenata programskog jezika C# 60
5.6.1. Zadaci 60
5.6.2. Rešenja 61
6. RAD SA STRUKTURAMA PODATAKA I DATOTEKAMA 71
6.1. Nizovi, liste, tipizirane liste 71
6.2. Rad sa fajlovima i folderima 76
6.3. Rad sa datotekama TXT i XML 79
6.4. Primeri sa strukturama podataka i datotekama 83
6.4.1. Zadaci 83
6.4.2. Rešenja 83
7. OBJEKTNO-ORJENTISANO PROGRAMIRANJE u C# 86
7.1. Teorija 86
7.2. Primeri primene elemenata objektno-orjentisanog programiranja 94
7.2.1. Zadaci 94
7.2.2. Rešenja 94
8. HEURISTIČKA UPUTSTVA I KONVENCIJE ZA OBLIKOVANJE PROGRAMSKOG KODA 111
9. ELEMENTI VIŠESLOJNE SOFTVERSKE ARHITEKTURE 113
10. PROGRAMSKI KOD IMPLEMENTACIJE WINDOWS APLIKACIJE 114
10.1. Memorijske kolekcije za rad sa podacima 114
10.2. Rad sa bazama podataka 114
10.2.1. Standardne klase za rad sa bazom podataka 114
10.2.2. Kreiranje sopstvenih klasa za rad sa bazom podataka 115
10.2.3. Generisanje i korišćenje Entity Framework klasa za rad sa bazom podataka 116
10.3. Rad sa izveštajima 124
10.4. Primer programskog koda windows aplikacije 131
10.4.1. Zadaci 131
10.4.2. Rešenja 132
11. PITANJA ZA TEST 196
12. LITERATURA 198
RECENZIJE 199

1. UVOD

Praktikum za vežbe nastao je na osnovu materijala koji je pripreman i korišćen u okviru vežbi iz nastavnog predmeta Softversko inženjerstvo 1 na Tehničkom fakultetu "Mihajlo Pupin" Zrenjanin, kao i na osnovu više od 20 godina iskustva u nastavi i stručnom radu u oblasti razvoja poslovnih informacionih sistema.

Praktikum obuhvata teme koje se odnose na UML – jezik modelovanja u oblasti objektno-orjentisanog razvoja softvera, sintaksu C# jezika, osnove objektno-orjentisanog programiranja koristeći C#, obradu grešaka, rad sa kolekcijama podataka, rad sa različitim formatima podataka, razvoj desktop aplikacije i rad sa bazama podataka uključujući izradu i korišćenje različitih biblioteka klasa u cilju uvoda u višeslojni razvoj softvera. Praktikum daje i opis načina korišćenja CASE alata za modelovanje i podršku automatskom generisanju delova programskog koda (Power Designer), kao i alata za razvoj aplikacija (Microsoft Visual Studio .NET). Sve teme su predstavljene kroz teorijske osnove, sintaksne elemente, primere i tehnička uputstva.

Praktikum pokriva sve teme predviđene vežbama iz predmeta Softversko inženjerstvo 1 na obrazovnom profilu Informacione tehnologije, ali može biti korišćen i kao priručnik u okviru stručnih kurseva ili drugih srodnih nastavnih predmeta. Praktikum je pripremljen uz konsultovanje standardnog materijala i udžbenika koje je izdala firma Microsoft.

Svako, prirodan nastavak gradiva obuhvaćenog ovim praktikumom odnosi se na napredne tehnike programiranja koje se odnose na kvalitet programskog koda uz smernice koje nude dizajn paterni i radni okviri (framework). Takođe, značajna tema u kontekstu unapređenja kvaliteta programskog koda odnosi se i na refaktorisanje, tj. unapređenje programskog koda koji treba da ima istu funkcionalnost, ali bolju organizaciju, performance itd. Razvoj web aplikacija, distribuiranih i mobilnih aplikacija, uz podršku servisno-orjentisanim arhitekturama svakako predstavlja gradivo koje je prirodan nastavak, ali je obuhvaćeno drugim nastavnim predmetima. Ipak, osnovni elementi troslojne i višeslojne arhitekture dati su u ovom praktikumu, kao uvod u naprednije teme.

Zahvalnost za kritički pregled materijala i sugestije u cilju poboljšanja rukopisa dugujemo recenzentima, redovnim profesorima prof. dr Biljani Radulović i prof. dr Ivani Berković.

Autori

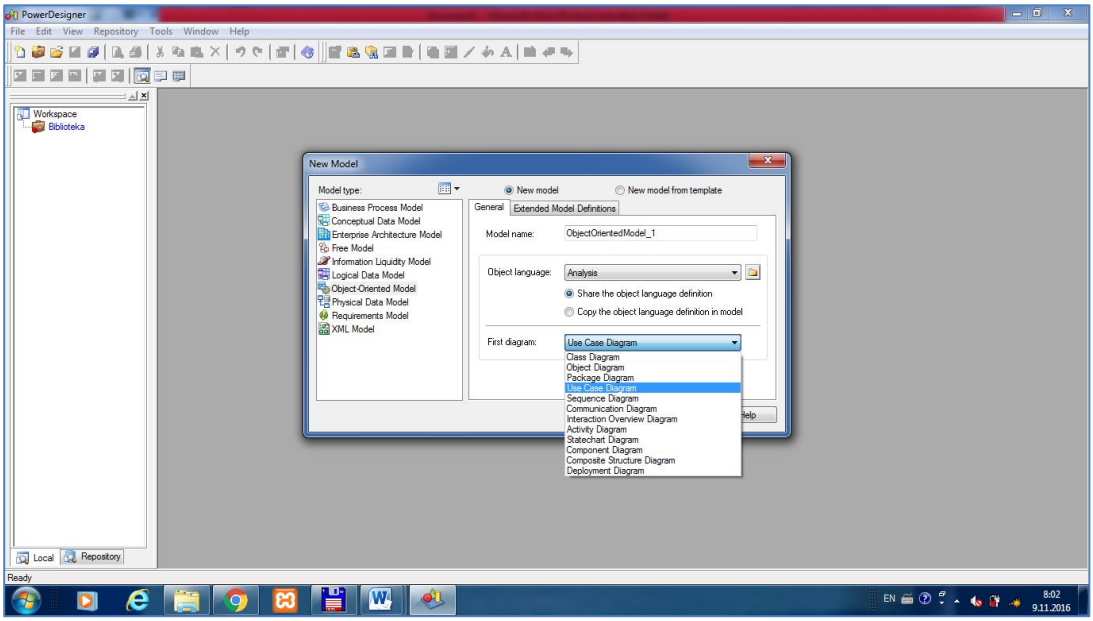
U Zrenjaninu, januara 2017. godine

2. UML MODELOVANJE SOFTVERA

2.1. Teorija UML dijagrama i CASE alat Power Designer

UML (Unified Modeling Language), nastao 1997. godine udruživanjem različitih pristupa objektno-orjentisanom modelovanju, predstavlja prvenstveno grafički jezik modelovanja u svim fazama razvoja softvera, počev od specifikacije zahteva, sistemske analize, dizajna, dokumentovanja itd. Najčešća primena UML je u specifikaciji zahteva korisnika (use case dijagram) i dizajnu (dijagram komponenti, dijagram razmeštaja, dijagram klasa). Prva verzija UML 1.0 uključuje 9 vrsta dijagrama: use case dijagram, class dijagram, object dijagram, sequence dijagram, communication dijagram, activity dijagram, statechart dijagram, dijagram komponenti (component), dijagram razmeštaja (deployment). Druga verzija UML (UML 2.0) iz 2005. godine predstavlja [1] [2] unapređenje u preciznijim pravilima sintakse i semantike i podrške modelovanju kompleksnih sistema. Uključuje dodatne dijagrame: package dijagram, interaction overview, composite structure dijagram, timing dijagram.

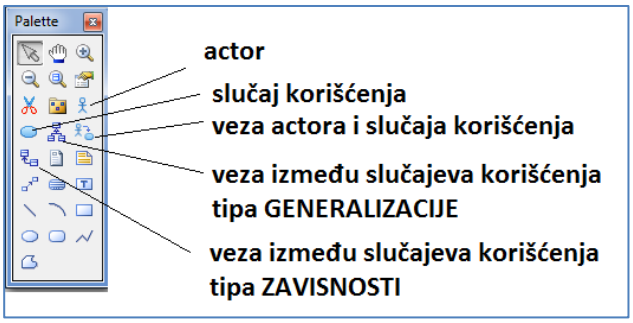
U okviru razvoja softvera radi povezivanja različitih faza razvoja, ubrzavanja i povećanja kvaliteta, koriste se posebni alati – CASE (Computer Aided Software Engineering) alati. Najčešće uključuju više alata koji mogu da razmenjuju podatke rezultata, kako bi naredna faza razvoja mogla efikasnije da se nastavi. Tako se na osnovu jedne vrste modela mogu generisati druge vrste modela ili čak upotrebljivi elementi prototipa rešenja: s jedne strane baze podataka, s druge strane izvorni kod programa. U okviru CASE alata Sybase Power Designer, za realizaciju UML modela koristi se Object Oriented Model. Na početku rada, može se birati univerzalni-apstraktni objektni jezik Analysis ili konkretan programski jezik, koji može biti osnov za automatsko generisanje programskog koda na osnovu modela (npr. C#).



Slika 2.1. Radno okruženje CASE alata Power Designer i izbor UML modela

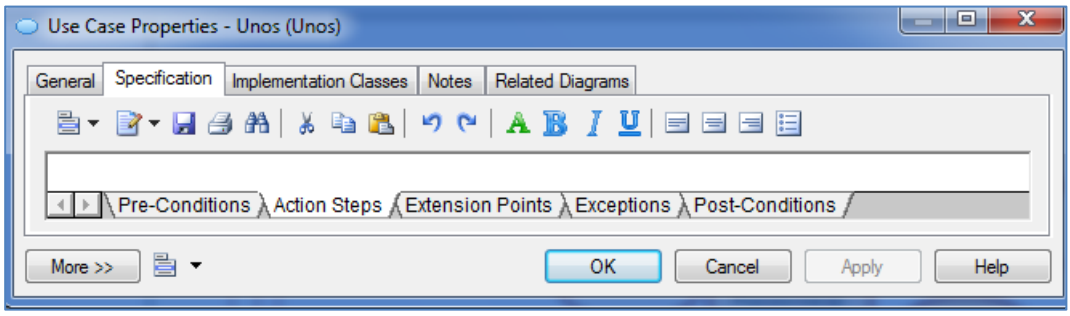
U nastavku je dat opis osnovnih tipova dijagrama koji se najčešće koriste, uz ilustraciju elemenata CASE alata Power designer koji se koriste za kreiranje dijagrama:

- *Use case diagram (Dijagram slučajeva korišćenja)* – Namena: prikaz osnovnih funkcija koju sistem nudi korisniku, sa aspekta vrednosti ili radnih aktivnosti koje sistem rešava. Elementi: actor (profil korisnika), slučaj korišćenja, veze između actora i slučaja korišćenja, veze između samih slučajeva korišćenja.



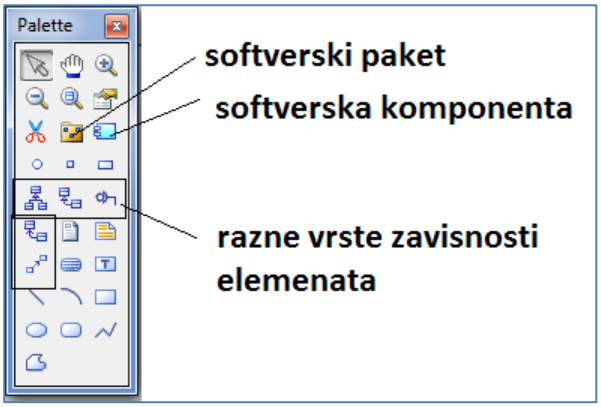
Slika 2.2. Opcije sa palete za dijagram slučaja korišćenja

Pored samog dijagrama, radi detaljnijeg opisa koriste se i specifikacije pojedinih slučajeva korišćenja koje obuhvataju: preduslove (potrebno stanje sistema pre početka rada slučaja korišćenja), postuslove (rezultate rada), action steps (uključuje pseudokod opisa interakcije korisnika i sistema uz različite scenarije), exceptions (obrada grešaka), extension points (tačke proširenja na druge slučajeve korišćenja).



Slika 2.3. Elementi specifikacije slučaja korišćenja

- *Component diagram (Dijagram komponenti)* – Namena: predstavlja osnovne elemente softverskog sistema i njihove međusobne zavisnosti, Elementi: paketi, komponente, fajlovi, veze zavisnosti.

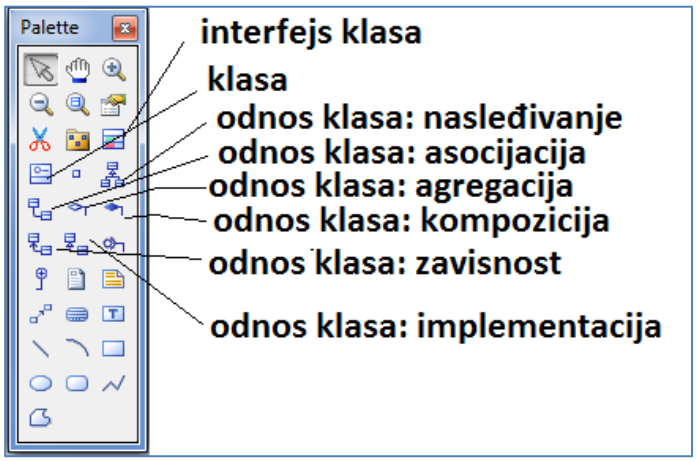


Slika 2.4. Opcije sa palete za dijagram komponenti

- *Deployment diagram (Dijagram razmeštaja)* – Namena: predstavlja raspored softverskih komponenti po čvorovima (serverima ili klijentskim računarima) u računarskoj mreži. Elementi: čvorovi, softverske komponente, veze zavisnosti.

Slika 2.5. Opcije sa palete za dijagram razmeštaja

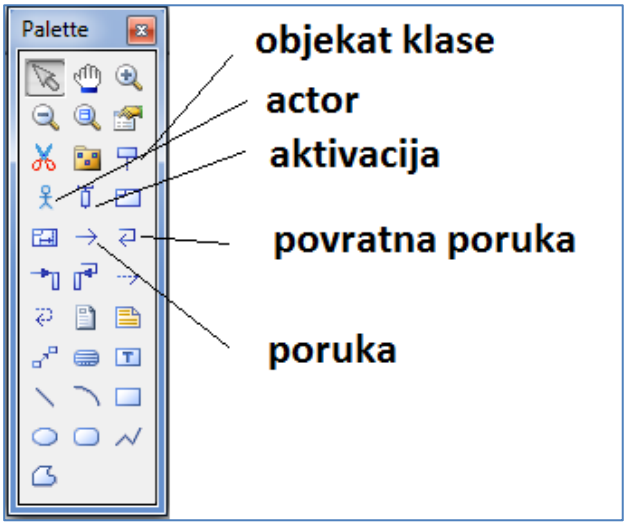
- *Class diagram (Dijagram klasa)* – Namena: predstavlja klase softverskog rešenja i njihove zavisnosti, Elementi: Klase (elementi klase su: atributi, metode), veze između klasa (asocijacija, agregacija, kompozicija, nasleđivanje).
    - Veza tipa asocijacije predstavlja strukturnu vezu gde se pri prevođenju ove veze u programski kod kao atribut jedne klase nalazi objekat druge klase koja je povezana vezom tipa asocijacije. Bitno svojstvo na dijagramu je navigable, koje određuje objekat koje klase će "migrirati".
    - Veza tipa nasleđivanja omogućava nasleđivanje (korišćenje kao da je navedeno u strukturi) atributa i metoda u klasi naslednika u odnosu na pretka, odnosno nadklasu.
    - Agregacija predstavlja vezu izme đu celine i njenih delova. Agregacijom se opisuje relacija pripadnosti.
    - Ukoliko celina upravlja životnim vekom delova onda se takva veza naziva kompozicija. Kompozicija odgovara (slična je) vezi između jakog i slabog objekta u modelu objekti veze. Kompozicija kazuje da instanca može pripadati samo jednom vlasniku.



Slika 2.6. Opcije sa palete za dijagram klasa

- Sequence diagram (Dijagram sekvenci) – Namena: predstavlja vremenski sled komunikacije objekata klasa, putem razmene poruka, u realizaciji jednog slučaja

korišćenja (odnosno jednog scenarija u okviru jednog slučaja korišćenja). Elementi: actor (može biti čovek ili drugi automatizovani-softverski sistem), objekti klasa, poruke (grafički su predstavljene kao strelice, a predstavljaju pozive metoda objekata klasa sa vrednostima argumenata) ili povratne poruke (kada objekat poziva svoju metodu), aktivacija (vertikalni pravougaonik kojim se predstavlja dužina trajanja "života" jednog objekta u memoriji).



Slika 2.7. Opcije sa palete za dijagram sekvenci

Ostale vrste dijagrama:

- Object diagram – predstavlja objekte klasa sa konkretnim vrednostima atributa radi ilustracije i objašnjenja značenja klasa
- State chart dijagram – predstavlja različita stanja koje objekat jedne klase može da ima.
- Communication dijagram – slično dijagramu sekvenci, predstavlja objekte klasa i razmenu poruka, bez prikaza vremenske dimenzije toka razmene poruka (prikazuje se samo redni broj poruke).
- Activity dijagram - može se koristiti u fazi specifikacije toka poslovnih procesa (sistemska analiza), predstavljanju action steps kod specifikacije jednog slučaja korišćenja ili predstavljanju načina realizacije neke metode jedne klase.

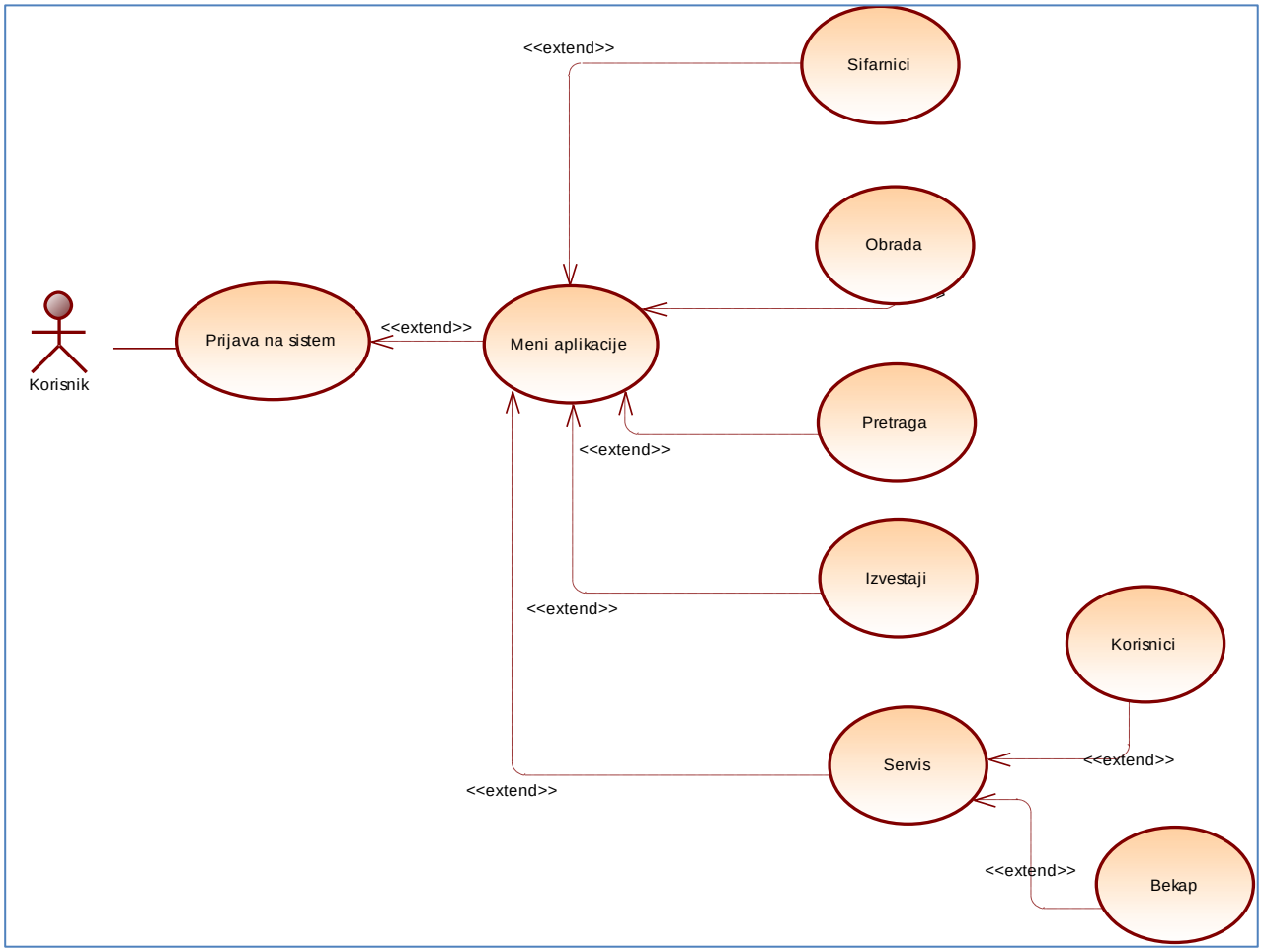
2.2. Primeri UML dijagrama u dizajnu poslovnog aplikativnog softvera

2.2.1. Primeri use case dijagrama

Uobičajena primena use case dijagrama prema tvorcima UML je u predstavljanju poslovnih funkcija koje su podržane (ili će biti podržane) novim sistemom, ali se u opisima često kombinuju i nazivi samih softverskih funkcija.
Heurističko uputstvo: preporučuje se preciznije nazivanje slučajeva korišćenja u 2 odvojena dijagrama i konteksta sa različitim načinom izražavanja – u kontekstu poslovnih procesa koji su podržani sistemom (primena use case dijagrama u funkciji poslovnog modeloranja) i u kontekstu softverskih funkcija koje su podržane sistemom (u funkciji specifikacije zahteva i dizajna). U ovom drugom kontekstu, use case dijagram se može koristiti za prikaz: strukture menija aplikacije ili skupa softverskih funkcija koje su na raspolaganju nekom profilu korisnika i njihovih međusobnih veza (za svaki profil korisnika najčešće se kreira poseban dijagram slučaja korišćenja). U oba slučaja, predstavlja osnov za dalju implementaciju softvera.

2.2.1.1. PRIMER STRUKTURE MENIJA POSLOVNE APLIKACIJE

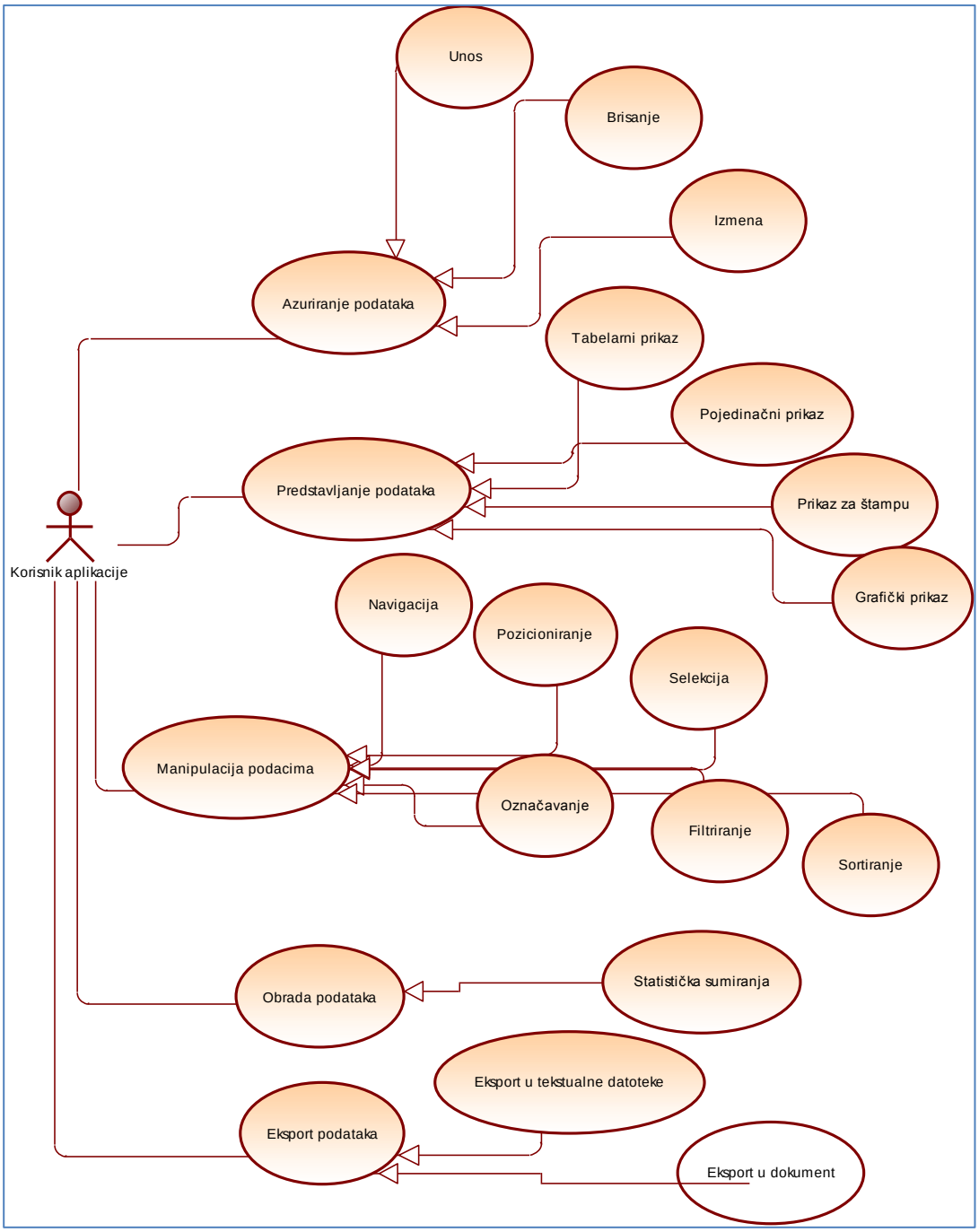
Uobičajena struktura poslovne softverske aplikacije sastoji se iz odeljka za rad sa šifarnicima (opšti podaci), podrške poslovnim procesima, rad sa pretragom izveštajima, kao i servisne opcije.



Slika 2.8. Dijagram slučaja korišćenja za primer strukture menija uobičajene poslovne aplikacije

2.2.1.2. PRIMER SOFTVERSKIH FUNKCIJA POSLOVNE APLIKACIJE

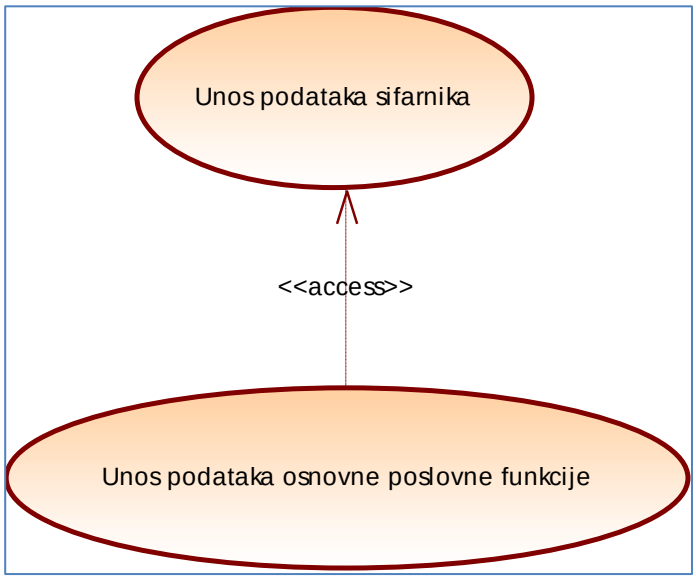
Uobičajene softverske funkcije poslovne aplikacije obuhvataju: CRUD operacije (unos (C - create), čitanje podataka (R- read), izmena (U – update), brisanje (D – delete); Predstavljanje učitanih podataka (tabelarni prikaz, pojedinačni prikaz, prikaz za štampu, grafički prikaz (grafikon)); Manipulaciju podacima (navigacija, pozicioniranje, selekcija, označavanje, filtriranje, sortiranje); Obrada podataka (statistike, razna sumiranja); Eksport podataka u različite formate tekstualnih datoteka ili dokumente.



Slika 2.9. Dijagram slučaja korišćenja sa uobičajenim softverskim funkcijama poslovne aplikacije

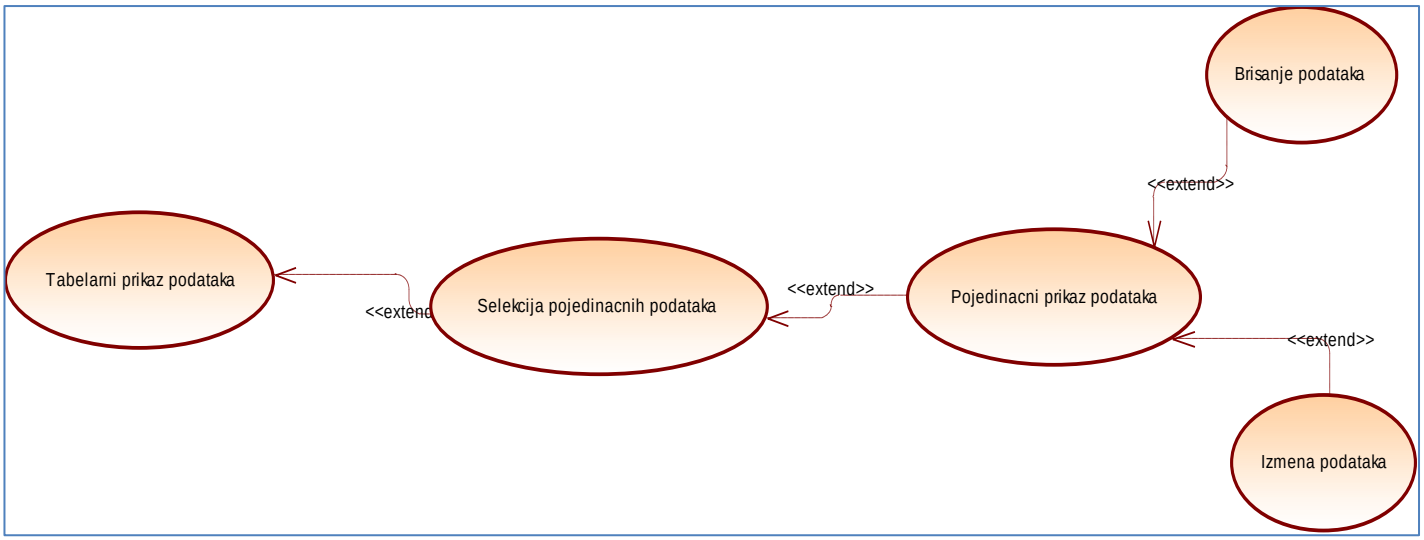
Slučajevi korišćenja, odnosno softverske funkcije su najčešće međusobno povezane i zavisne u okviru funkcionisanja. Njhova uobičajena zavisnost se može predstaviti narednim dijagramima.

Unos podataka osnovne poslovne funkcije pristupa podacima koje je obezbedio unos podataka sifarnika, pa je odnos tipa “access”.



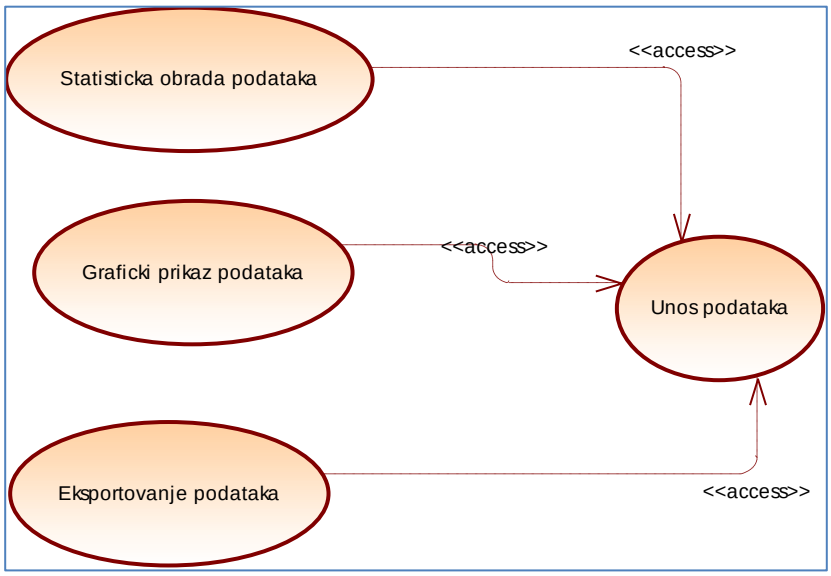
Slika 2.10. Odnos “access” između unosa podataka osnovne poslovne funkcije i unosa podataka šifarnika koji obezbeđuje podatke za poslovnu funkciju

Da bismo obrisali ili izmenili podatke, moramo im pristupiti iz tabelarnog prikaza selektovanjem i prikazom tih pojedinačnih podataka. Tabelarni prikaz se proširuje (“extend”) dodatnim mogućnostima selekcije i prikazivanja pojedinačnih podataka, ali korisnik ne mora da izabere te opcije (zato je extend, proširuje ali je neobavezno korišćenje takvih opcija za korisnika).



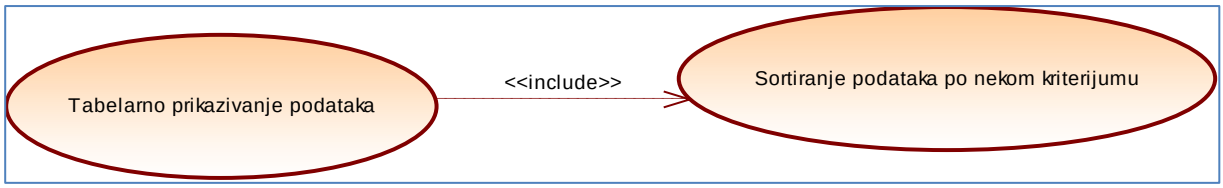
Slika 2.11. Odnos brisanja i izmene podataka prema selekciji zapisa iz tabelarnog prikaza

Statistička obrada, grafički prikaz i eksport podataka pristupaju podacima koje obezbeđuje unos podataka, tako da je odnos “access”.



Slika 2.12. Odnos "access" kao pristup podacima koje obezbeđuje unos podataka za statističku obradu, grafički prikaz i eksportovanje podataka

Sortiranje podataka je sastavni deo tabelarnog prikaza, bez sortiranja ne može se tabelarni prikaz izvršiti. i kada se ne naglasi, podaci su sortirani prema nekom kriterijumu, pa makar i po redosledu snimanja u bazu podataka. Zato tabelarni prikaz uključuje ("include") sortiranje podataka, bez kog ne može da se izvrši.



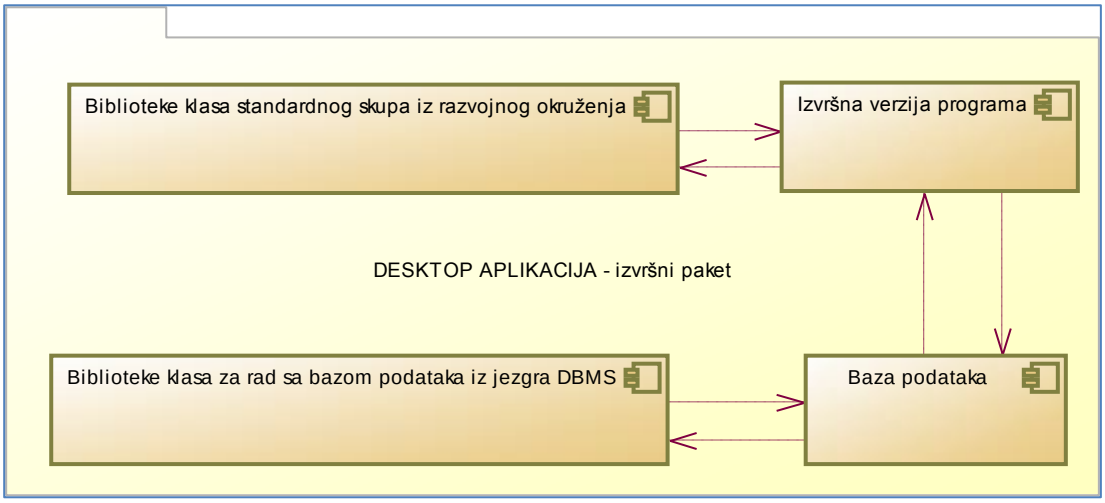
Slika 2.13. Odnos "include" gde je sortiranje obavezno uključeno u tabelarni prikaz podataka

2.2.2. Primeri dijagrama komponenti

Komponente softverske aplikacije se razlikuju prema tipu aplikacije. Možemo razlikovati komponente aplikacije u dizajn režimu (u toku razvoja) i u izvršnom režimu, kao i skup elementa potrebnih za instalaciju. U nastavku će biti prikazani primeri poslovnih aplikacija koje obavezno sadrže i komponente za rad sa bazom podataka.

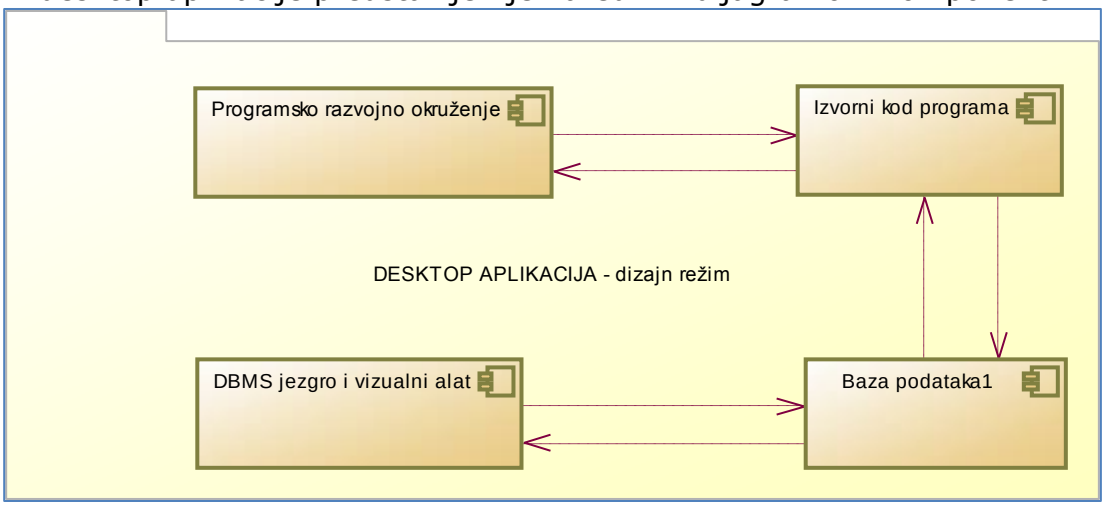
PRIMER DESKTOP APLIKACIJE

Primer desktop aplikacije u izvršnom režimu sastoji se od izvršne verzije programa i pratećih standardnih biblioteka klasa, kao i baze podataka i odgovarajućih biblioteka klasa jezgra sistema za upravljanje bazom podataka (DBMS).



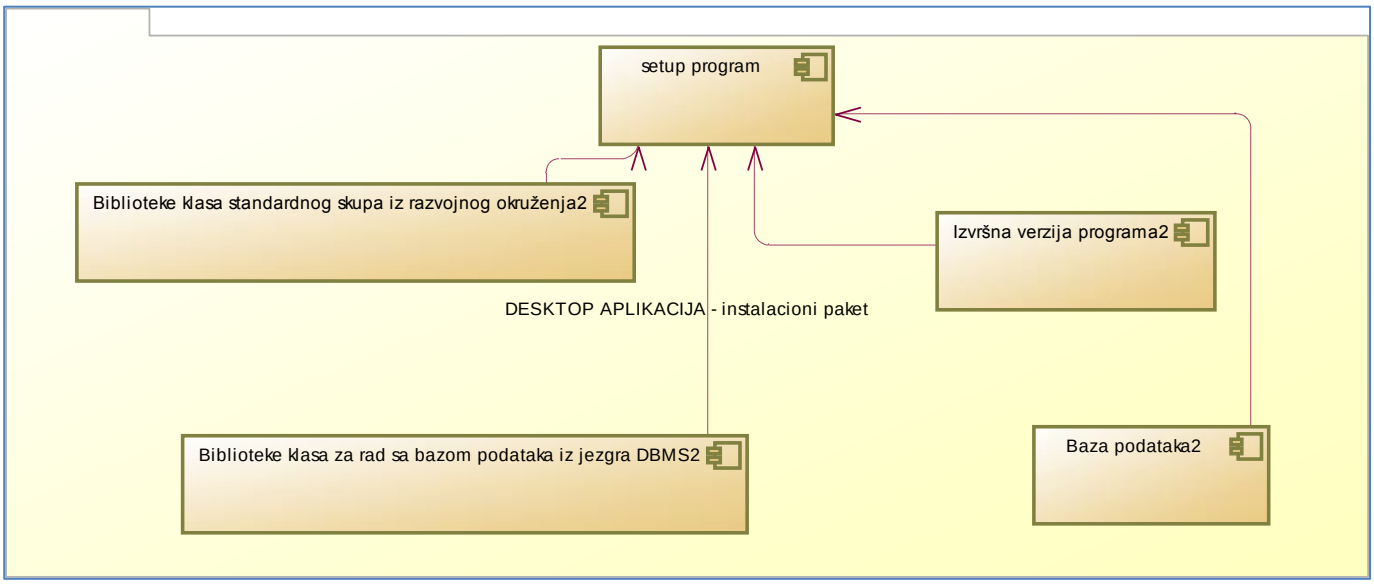
Slika 2.14. Izvršni paket komponenti desktop aplikacije

Dizajn režim desktop aplikacije predstavljen je narednim dijagramom komponenti:



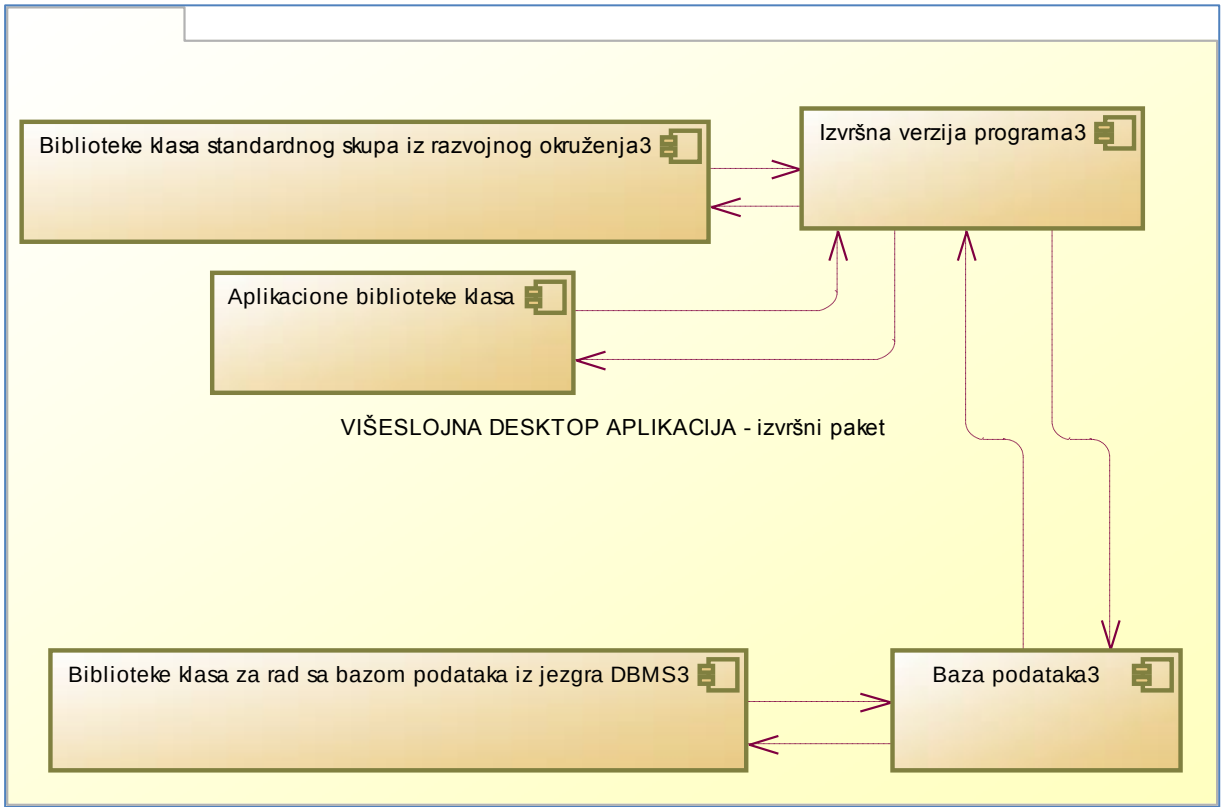
Slika 2.15. Dizajn režim - paket komponenti desktop aplikacije

Konačno, instalaciona verzija programa sastoji se od fajlova izvršne verzije koja uključuje i setup program koji objedinjuje sve fajlove izvršnog paketa.



Slika 2.16. Instalacioni paket komponenti desktop aplikacije

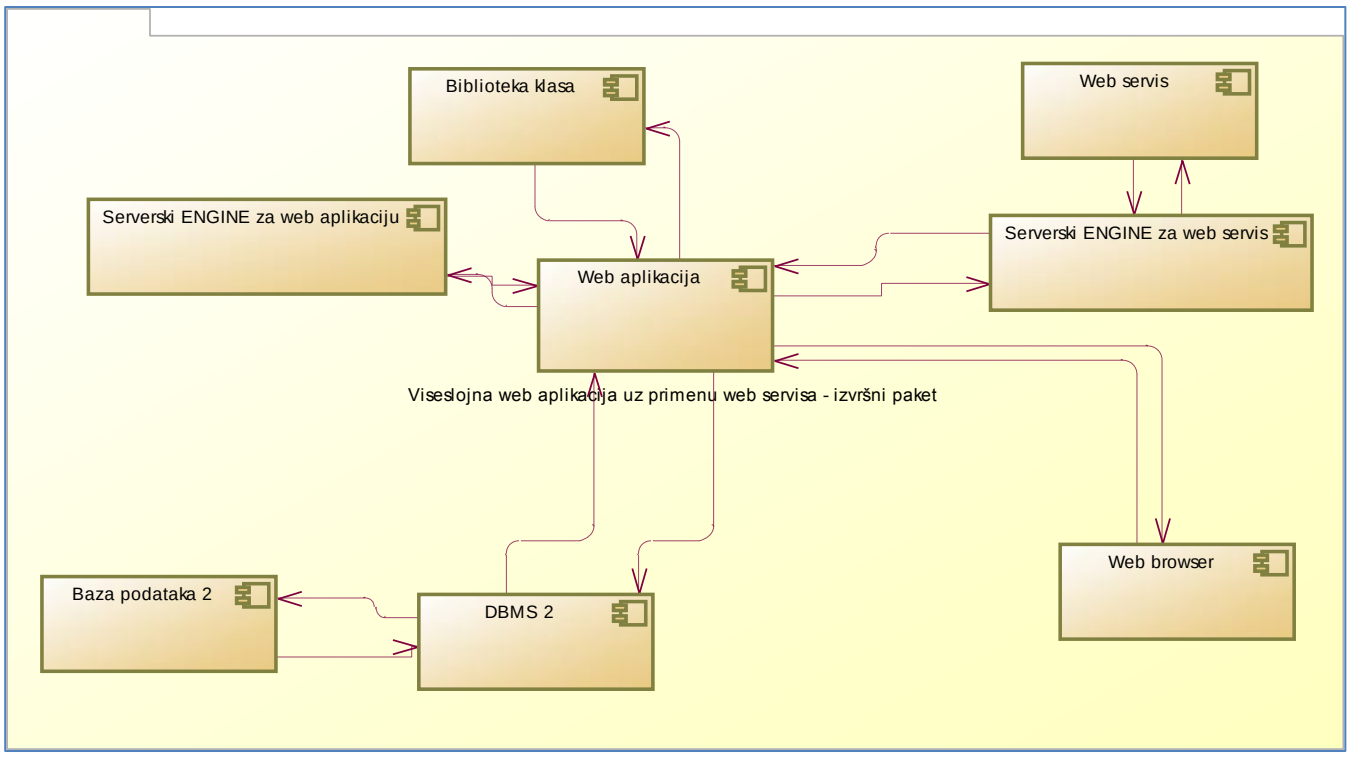
Izvršni režim desktop aplikacije može da sadrži i zasebnu biblioteku klasa ili više biblioteka klasa koje omogućavaju da se deo aplikativne logike podeli po slojevima. Tada govorimo o višeslojnoj desktop aplikaciji.



Slika 2.17. Izvršni paket komponenti višeslojne desktop aplikacije

PRIMER VIŠESLOJNE WEB APLIKACIJE UZ PRIMENU WEB SERVISA

Danas je aktuelan razvoj web aplikacija, posebno zbog mogućnosti da se izvršavaju i na mobilnim uređajima. Svakako, u porastu je i razvoj zasebnih mobilnih aplikacija. Naredni dijagram komponenti prikazuje tipičan skup komponenti savremene web aplikacije.



Slika 2.18. Izvršni paket komponenti web aplikacije uz primenu web servisa

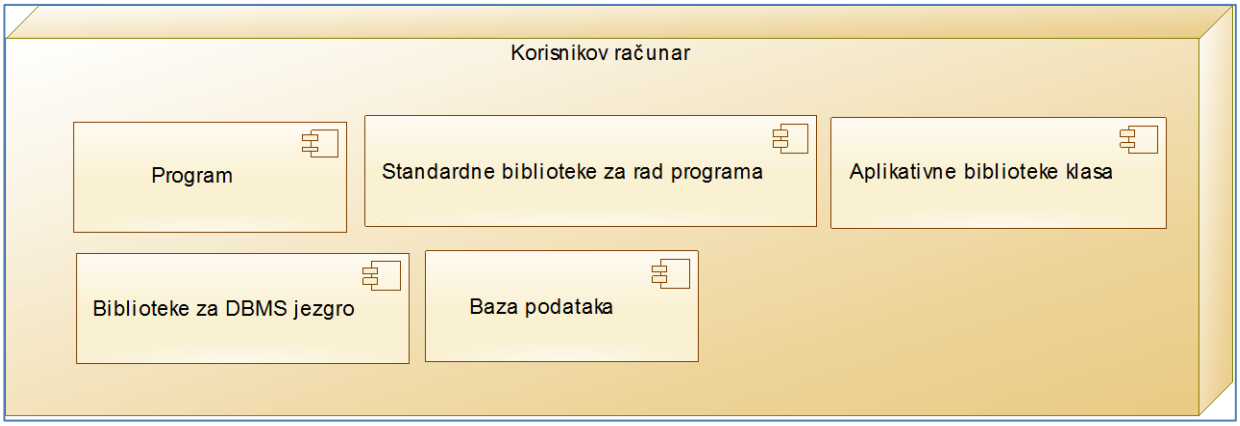
Web browser klijentskog računara obraća se web aplikaciji koja se izvršava na osnovu posebnog programa "web servera" koji omogućava tumačenje i izvršavanje naredbi u web aplikaciji ("Serverski ENGINE"). Radi omogućavanja lakšeg održavanja,web aplikacije se razvijaju kao višeslojne, gde se pojedini delovi programskog koda smeštaju u okviru posebnih biblioteka klasa. Biblioteke klasa se kompajliraju i u izvršnom obliku uključuju u web aplikaciju, kako bi omogućile podršku pojedinim delovima funkcionalnosti. Same biblioteke klasa se nalaze najčešće na istom računaru kao i sama web aplikacija ili na posebnim računarima tzv. Aplikacionim serverima. Jedan deo funkcija se može koristiti od strane drugih proizvođača softvera uslužno, putem povezivanja sa udaljenim web servisima (ne mora se uvek "cela" aplikacija nalaziti na jednom mestu, već je neki deo funkcionalnosti uradila druga softverska kompanija i ponudila na javno korišćenje, kao servis koji je besplatan ili se plaća u određenom ugovornom periodu). Web servisi predstavljaju on-line biblioteke klasa kojima se pristupa putem URL-a. Svakako, poslovne web aplikacije zasnivaju svoj rad najčešće na bazama podataka i da bi se mogle koristiti, uključeni su sistemi za upravljanje bazom podataka, kao posebne komponente.

U okviru vežbi iz ovog predmeta neće se razvijati web aplikacije i ovaj deo je dat samo zbog ilustracije i upoznavanja studenata sa aktuelnim tehnologijama razvoja softvera.

2.2.3. Primer dijagrama razmeštaja

PRIMER VIŠESLOJNE DESKTOP APLIKACIJE

Kod višeslojne desktop aplikacije, sve komponente se nalaze na jednom čvoru, tj. računaru.



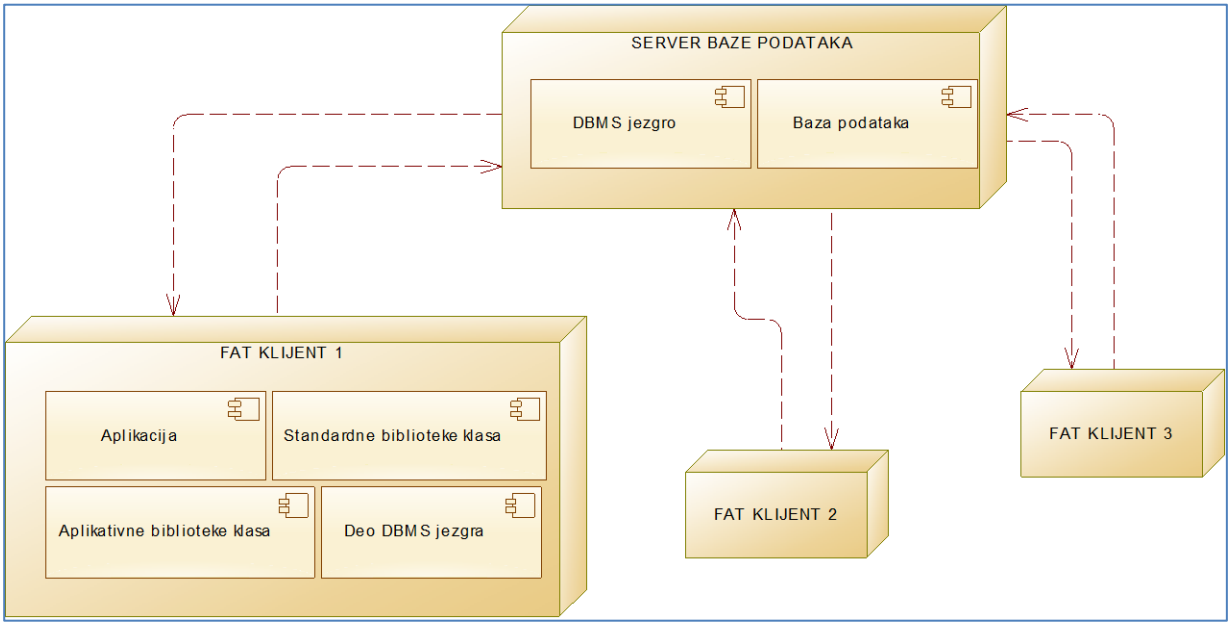
Slika 2.19. Izvršni paket komponenti web aplikacije uz primenu web servisa

Primer:

- Program: Ambulanta.exe
- Standardne biblioteke za rad programa: .NET dll biblioteke
- Aplikativne biblioteke klasa: KlasePodataka.dll
- Biblioteke za DBMS jezgro: biblioteke koje omogucuju rad sa MS SQL Server
- Baza podataka: Pacijenti.mdf

PRIMER KLIJENT-SERVER APLIKACIJE sa FAT KLIJENTOM

Fat/rich/heavy/rich client je varijanta klijent-server arhitekture gde se aplikacija sa svim bibliotekama nalazi na klijentskom računaru, dok se baza podataka može nalaziti na drugom računaru. Ovaj concept je prisutan u lokalnim računarskim mrežama tipa banaka, šaltera na autobuskim stanicama i slično.

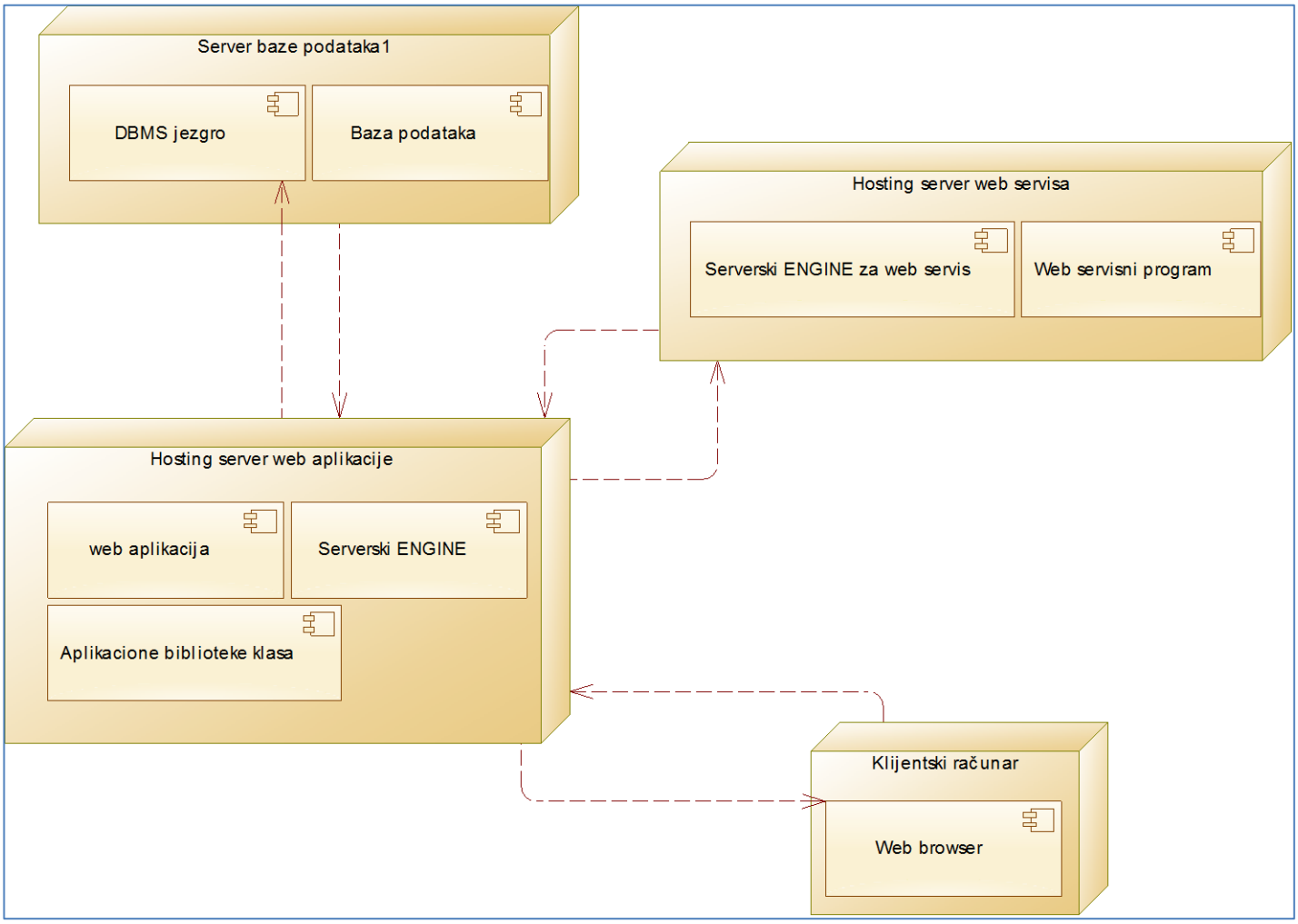


Slika 2.20. Izvršni paket komponenti klijent-server aplikacije sa FAT klijentom

PRIMER KLIJENT-SERVER APLIKACIJE sa THIN KLIJENTOM

Thin client je varijanta klijent-server arhitekture kada se na klijentskom računaru nalazi samo manji aplikativni deo, a cela programska logika i baza podataka se nalazi na drugim računarima - serverima. Tipična situacija je kada klijentski računar ima samo web browser, kojim se pristupa web serverskom računaru gde se izvršava web aplikacija.

Savremen pristup razvoju web aplikacija uključuje primenu Java Script-a koji omogućuje da se deo funkcionalnosti, posebno u smislu opsluživanja korisničkog interfejsa, ipak izvršava na klijentskom računaru. Savremeni web browseri imaju podršku za Java Script, tako da se u ovoj varijanti, korisnički web browser nije thin klijent u najstrožijem smislu.

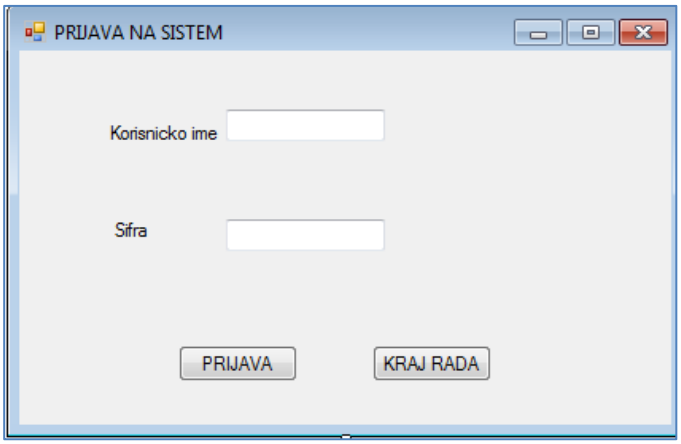


Slika 2.21. Izvršni paket komponenti klijent-server aplikacije sa THIN klijentom

2.3. Primer dokumentovanja softverskog rešenja UML dijagramima

2.3.1. Zadatak

Zadatak je da se na osnovu date ekranske forme – grafički deo i deo programskog koda, dokumentuje deo programa primenom UML dijagrama.
U nastavku je dat material za analizu kao jednostavan primer ekranske forme Windows Forms aplikacije za prijavu na sistem.



Slika 2.22. Izgled ekrana za prijavljivanje korisnika

Programski kod kojim se implementira grafički dizajn dat je u nastavku:

```
   private System.Windows.Forms.Label label1;
      private System.Windows.Forms.Label label2;
      private System.Windows.Forms.TextBox txbKorisnickoIme;
      private System.Windows.Forms.TextBox txbSifra;
      private System.Windows.Forms.Button btnPrijava;
      private System.Windows.Forms.Button btnKrajRada;
```

Programski kod koji predstavlja implementaciju ove forme dat je u nastavku:

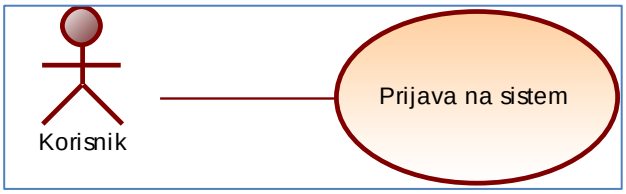
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
//
using SqlDBUtils;
using System.Data;

namespace KorisnickiInterfejs
{
   public partial class frmLogin : Form
   {
     // globalne promenljive, tj. atributi ove klase
```

```
        clsSqlKonekcija objKonekcija;


        public frmLogin()
        {
            InitializeComponent();
        }

        private void btnKrajRada_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void btnPrijava_Click(object sender, EventArgs e)
        {
            clsSqlTabela objTabela = new clsSqlTabela(objKonekcija, "Korisnik");
            DataSet dsProvera = objTabela.DajPodatke("select * from Korisnik where
KorisnickoIme='" + txbKorisnickoIme.Text + "' and Sifra='" + txbSifra.Text + "'");
            if (dsProvera.Tables[0].Rows.Count > 0)
            {
                frmGlavniMeni formaGlavniMeni = new frmGlavniMeni();
                formaGlavniMeni.ShowDialog();
            }
            else
            {
                MessageBox.Show ("GRESKA, nepostojeci korisnik!");
                txbKorisnickoIme.Focus();
            }

        }

        private void frmLogin_Load(object sender, EventArgs e)
        {
            objKonekcija = new clsSqlKonekcija("BUBILIS", "", "SE1login");
            objKonekcija.OtvoriKonekciju();
        }
    }
}
```

2.3.2. Rešenje

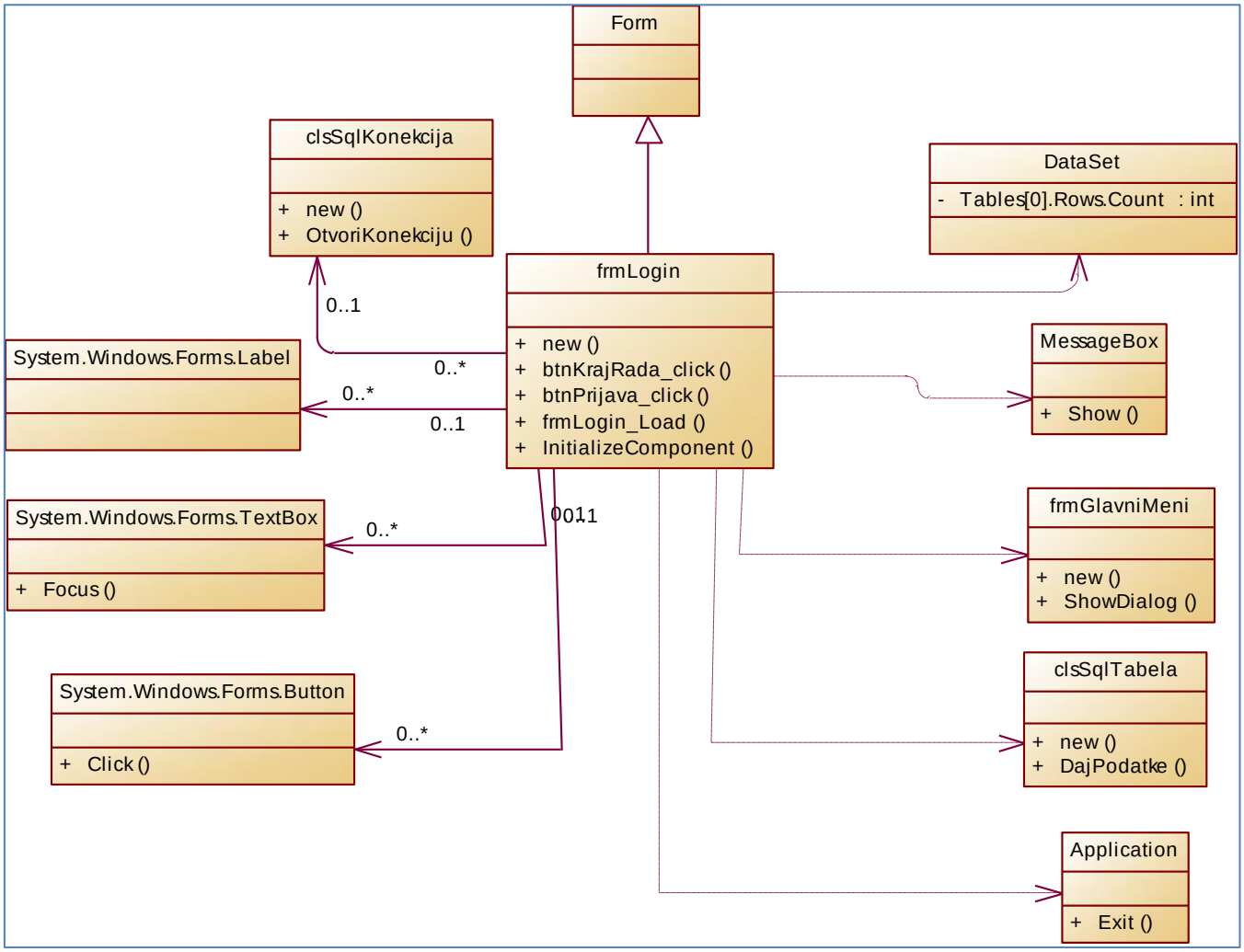
U nastavku će biti prikazani rezultati dokumentovanje softvera primenom UML dijagrama. Dijagram komponenti i razmeštaja neće biti prikazan, jer je ovo primer windows forms aplikacije, dakle desktop aplikacija, pri čemu je baza podataka na istom računaru kao i aplikacija.

Dijagram slučaja korišćenja za ovaj jednostavan primer dat je dijagramom na sledećoj slici:



Slika 2.23. Jednostavan dijagram slučaja korišćenja za primer prijave na sistem

Dijagram klasa predstavlja sve klase u realizaciji navedenog primera. U programskom kodu se vidi da je i sama ekranska forma klasa, koja nasleđuje sve osobine opšte klase Form. Takođe, koriste se i dodatne klase. Naredni dijagram klasa prikazuje samo attribute i metode klasa koje su korišćene u primeru programskog koda. U opštem slučaju, većina klasa ima attribute i metode, a od metoda svakako se podrazumeva metoda new (konstruktor). Razlikujemo glavnu formu frmLogin, klasu Form koju nasleđuje (linija sa trouglom na vrhu je simbol nasleđivanja), strukturne klase (klase koje svojim objektima ulaze u strukturu forme – grafički ili kao objekti iz dela programskog koda – prikazane strelicom uobičajenog vrha) i klase od kojih zavisi jer se spominju u implementaciji metoda (isprekidane strelice – dependency).



Slika 2.24. Dijagram klasa za primer softverske implementacije prijave na sistem

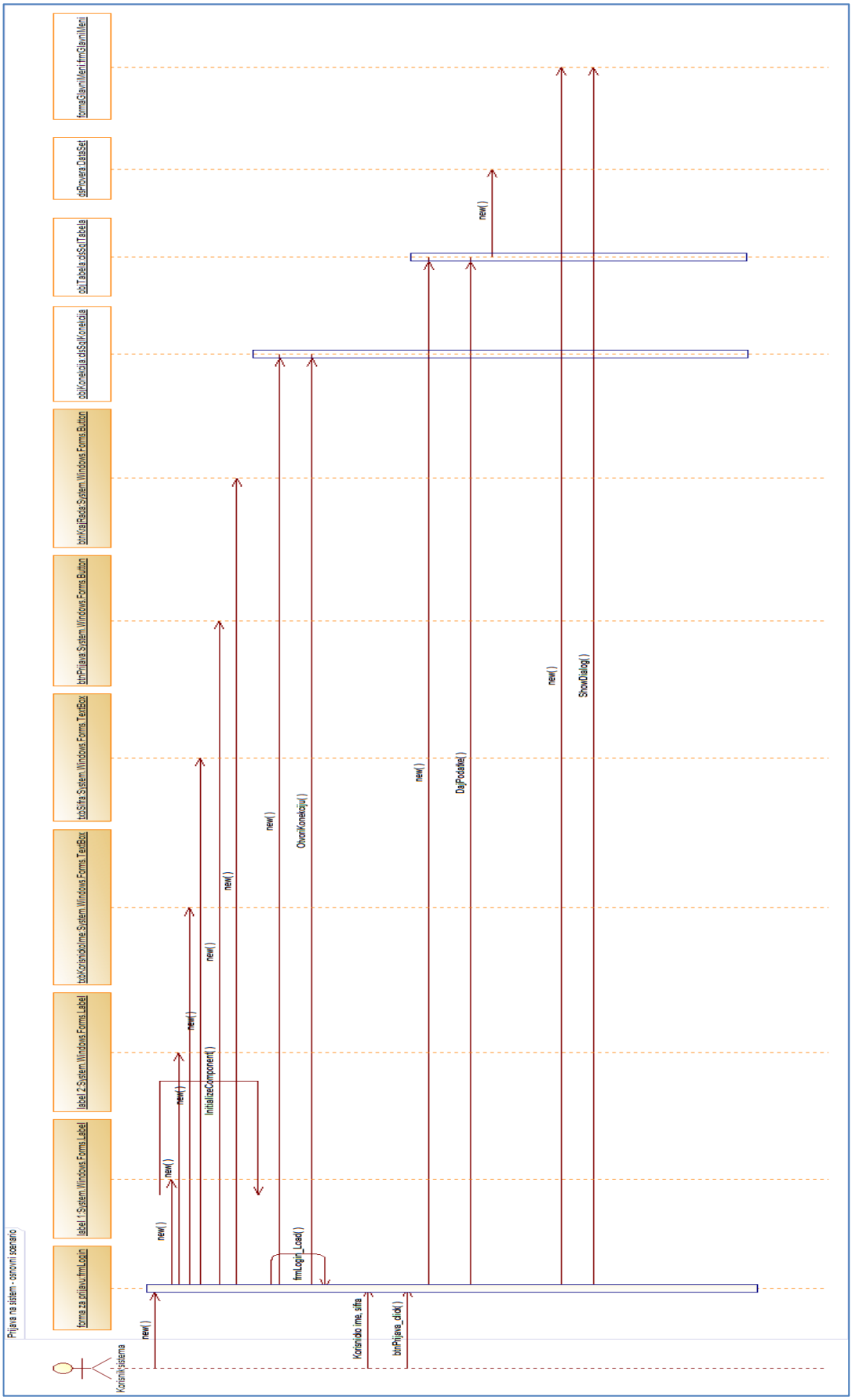
Dijagram sekvenci prikazuje sled izvršavanja programskog koda ove forme – automatskog dela i dela u interakciji korisnika i aplikacije. U nastavku je dat dijagram sekvenci za osnovni scenario upotrebe, a to znači za situaciju kada se korisnik uspešno prijavi na sistem i učita se ekranska forma glavnog menija.

Nakon startovanja forme za prijavljivanje, automatski se izvršava konstruktor gde se instanciraju objekti grafičkih klasa (labela, text box, button), a nakon toga se automatski izvršava form_load događaj, gde se instancira objekat za konekciju sa bazom podataka. Nakon učitavanja forme, korisnik može da vidi formu i unosi podatke u odgovarajuće text boxove, odnosno pokreće odgovarajuće akcije korišćenjem softverskih tastera (pokretanjem Button click događaja). Sistem vraća korisniku poruku o uspehu realizacije akcija ili podatke koji predstavljaju rezultat izvršavanja akcija.

OPŠTI REDOSLED AKTIVNOSTI NA DIJAGRAMU SEKVENCI ZA SVE EKRANSKE FORME UOBIČAJENE WINDOWS (DESKTOP) APLIKACIJE:
- instanciranje objekta forme
- pokreće se automatski konstruktor forme (self message) gde se automatski inicijalizuju i iscrtavaju sve grafičke kontrole, a mogu se u okviru tog koda inicijalizovati i promenljive i instancirati objekti klasa koje se koriste na formi
- pokreće se automatski form load događaj (form load je kod windows aplikacija, a kod web formi je page load) gde se mogu izvršavati takođe inicijalizacije i pozvati početne procedure
- grafička forma je prikazana i sada su moguće akcije korisnika. Tipične aktivnosti i rad sistema u interakciji korisnika i sistema odnosi se na:

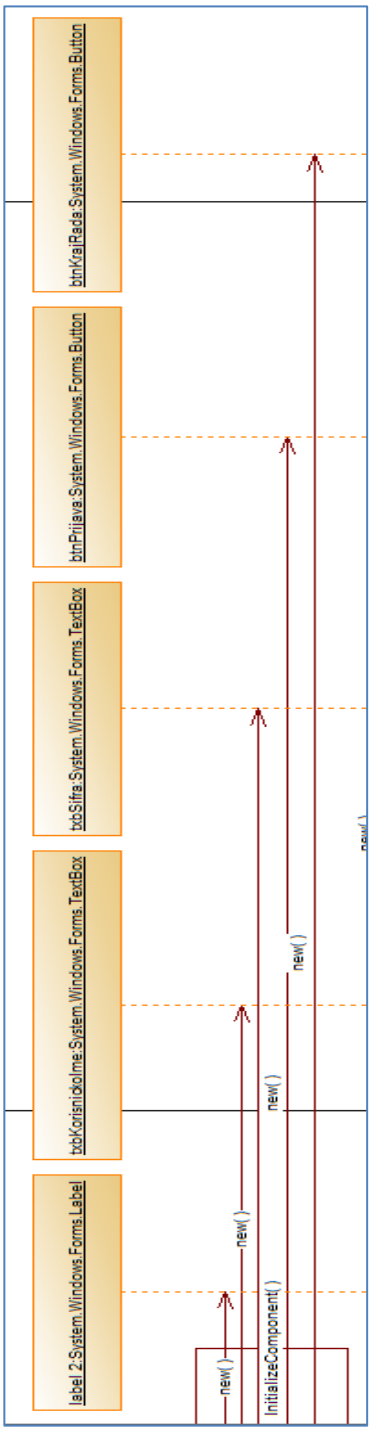
- Unos – korisnik bira opciju UNOS, pokreće se priprema forme za unos brisanjem sadržaja kontrola i postavljanjem fokusa na prvu kontrolu. Korisnik unosi vrednosti. Korisnik bira opciju SNIMI, nakon čega sistem konsultuje klase koje treba da snime podatke u bazu podataka, uz najčešću prethodnu proveru podataka.
- Tabelarni – korisnik bira opciju SVI, gde se učitavaju podaci (konsultovanjem klasa koje rade sa bazom podataka) i prikazuju i grid kontroli.
- Filter – korisnik unosi kriterijum filtriranja i bira taster FILTRIRAJ, nakon čega sistem puni grid kontrolu filtriranim podacima (konsultovanjem klasa koje rade sa bazom podataka).

Primer celovitog dijagrama sekvenci za redosled instanciranja objekata i razmenu poruka između objekata klasa na primeru LOGIN softverske funkcije prikazan je u nastavku:



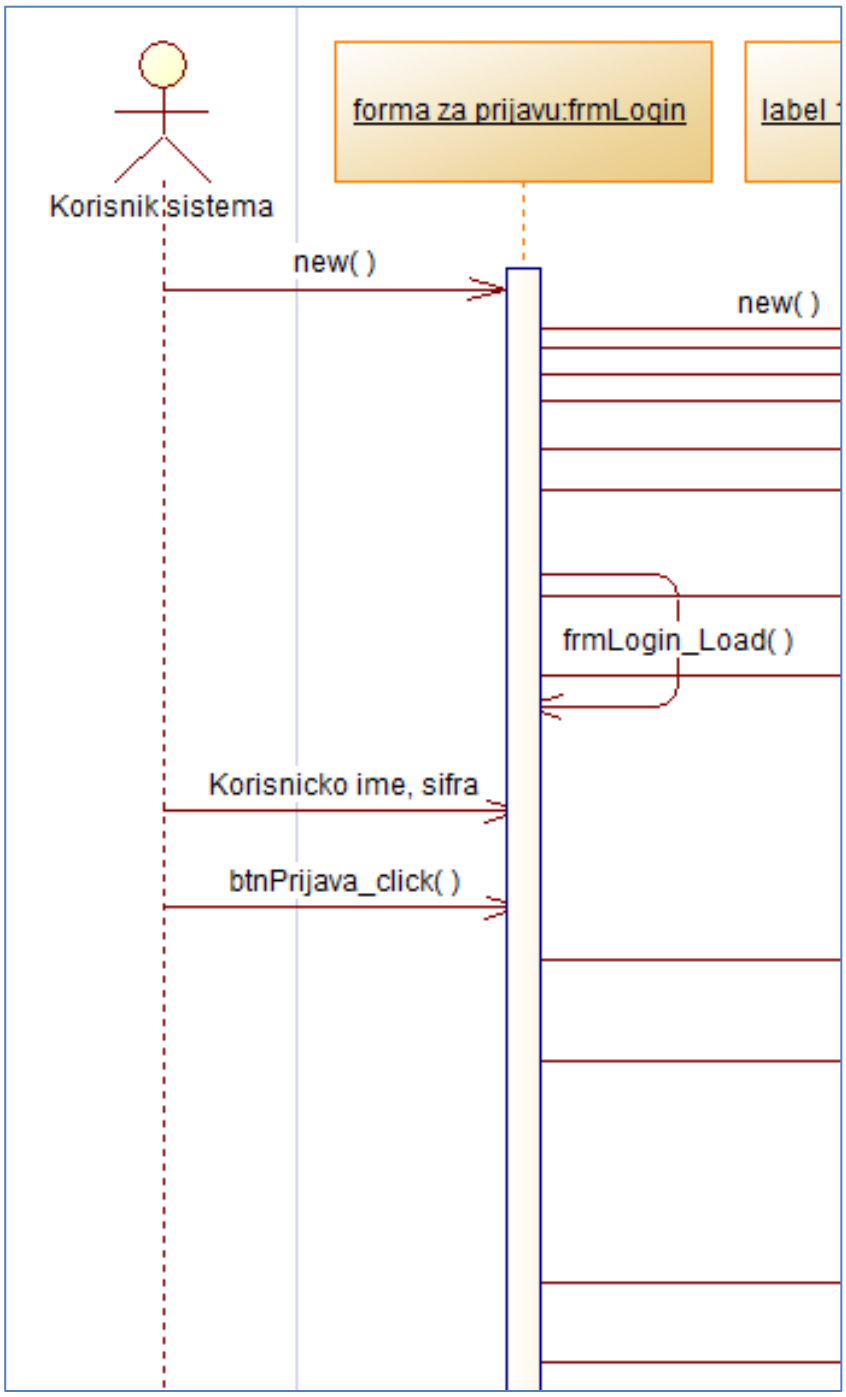
Slika 2.25. Celokupan dijagram sekvenci za primer prijave korisnika na sistem

Prvi deo prethodno prikazanog dijagrama sekvenci odnosi se na inicijalizaciju grafičkih komponenti:



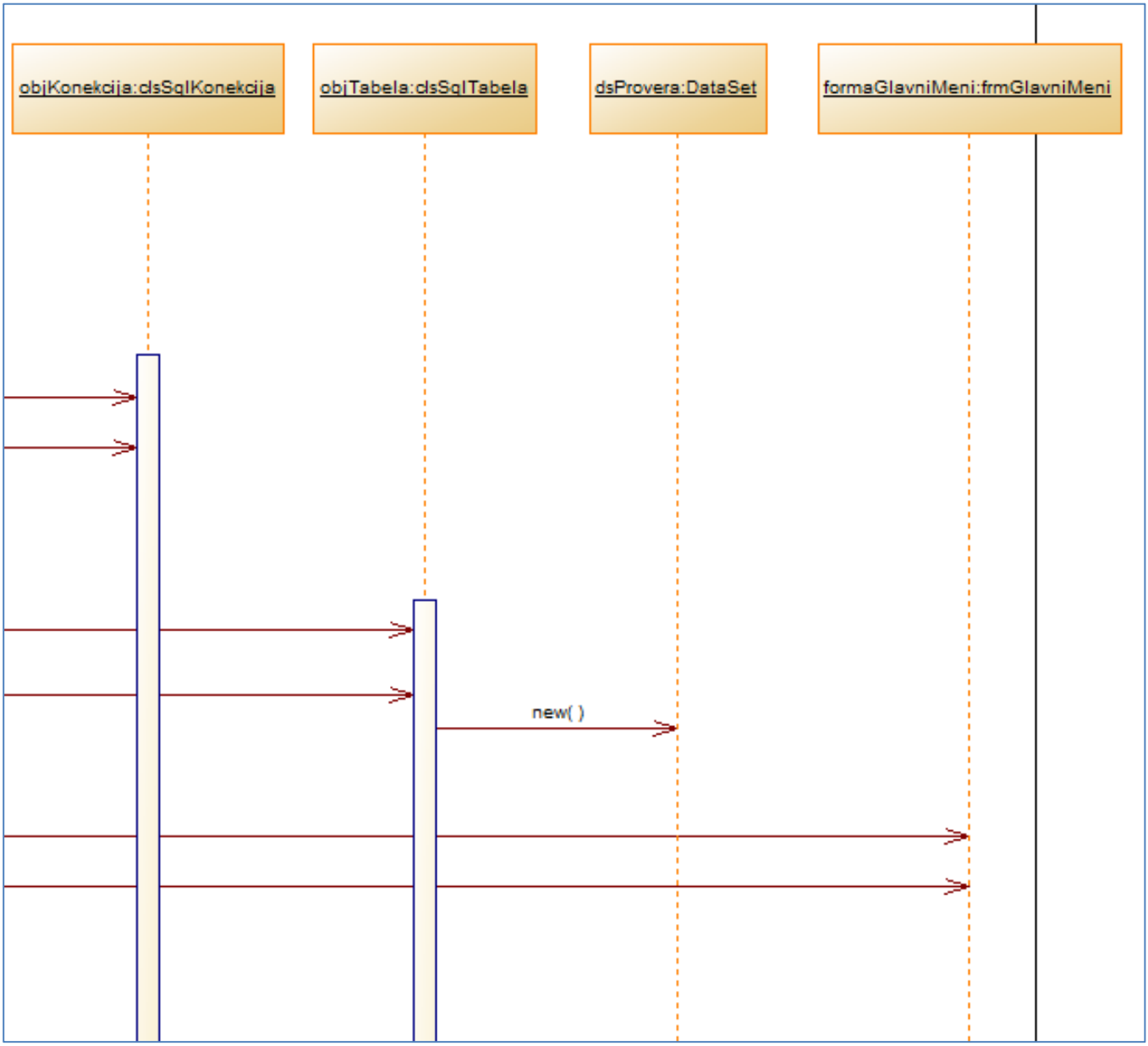
Slika 2.26. Prvi deo dijagrama sekvenci – inicijalizacija grafičkih komponenti forme

Hronološki posmatrano, drugi (donji) deo prethodnog dijagrama se odnosi na sled poruka u interakciji sa korisnikom.



Slika 2.27. Hronoloski drugi (donji) deo dijagrama sekvenci – interakcija sa korisnikom

Treći deo dijagrama predstavlja nastavak prethodnog dijagrama (desni deo u odnosu na prethodno prikazani deo) dat je u nastavku:



Slika 2.28. Desni deo dijagrama sekvenci - rad sa drugim klasama i formom Glavni meni

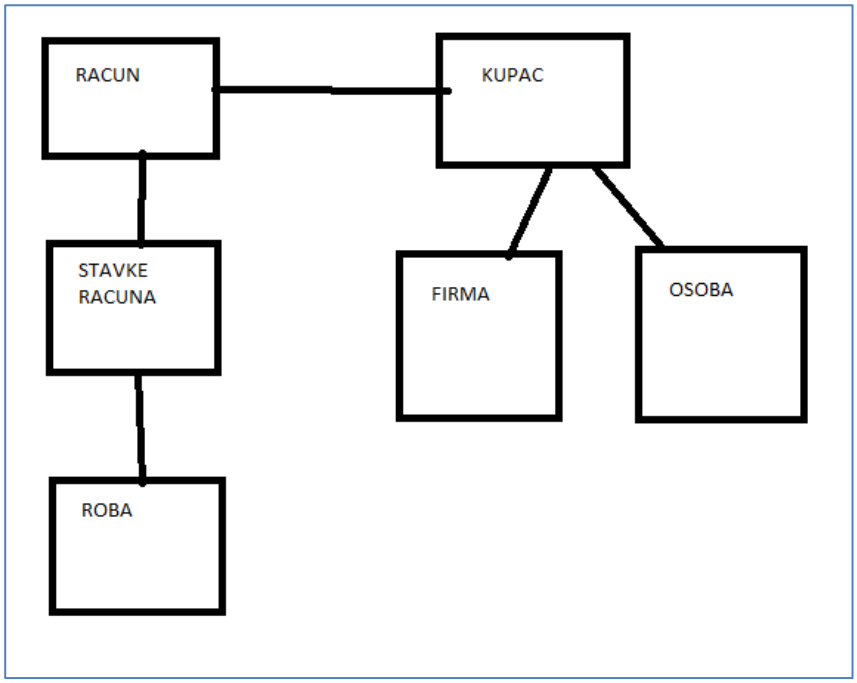
Na prethodno prikazanom dijagramu za LOGIN se ne pojavljuje klasa MessageBox, jer je prikazan samo osnovni scenario, dakle kada se korisnik uspešno prijavi na sistem, što rezultuje učitavanjem forme frmGlavniMeni. Klasa MessageBox bi se instancirala u alternativnom scenariju kada korisničko ime ili šifra korisnika nisu ispravne i korisnik se obaveštava o grešci.

2.4. Primer generisanja programskog koda na osnovu modela iz CASE alata

Na osnovu dijagrama klasa UML-a može se kreirati programski kod klasa. U nastavku je dat primer i rešenje.

2.4.1. Zadatak

1. Kreirati CDM (Conceptual data model) na osnovu date sheme i objašnjenja.



Slika 2.29. Odnos entiteta na CDM modelu – primer Racuna

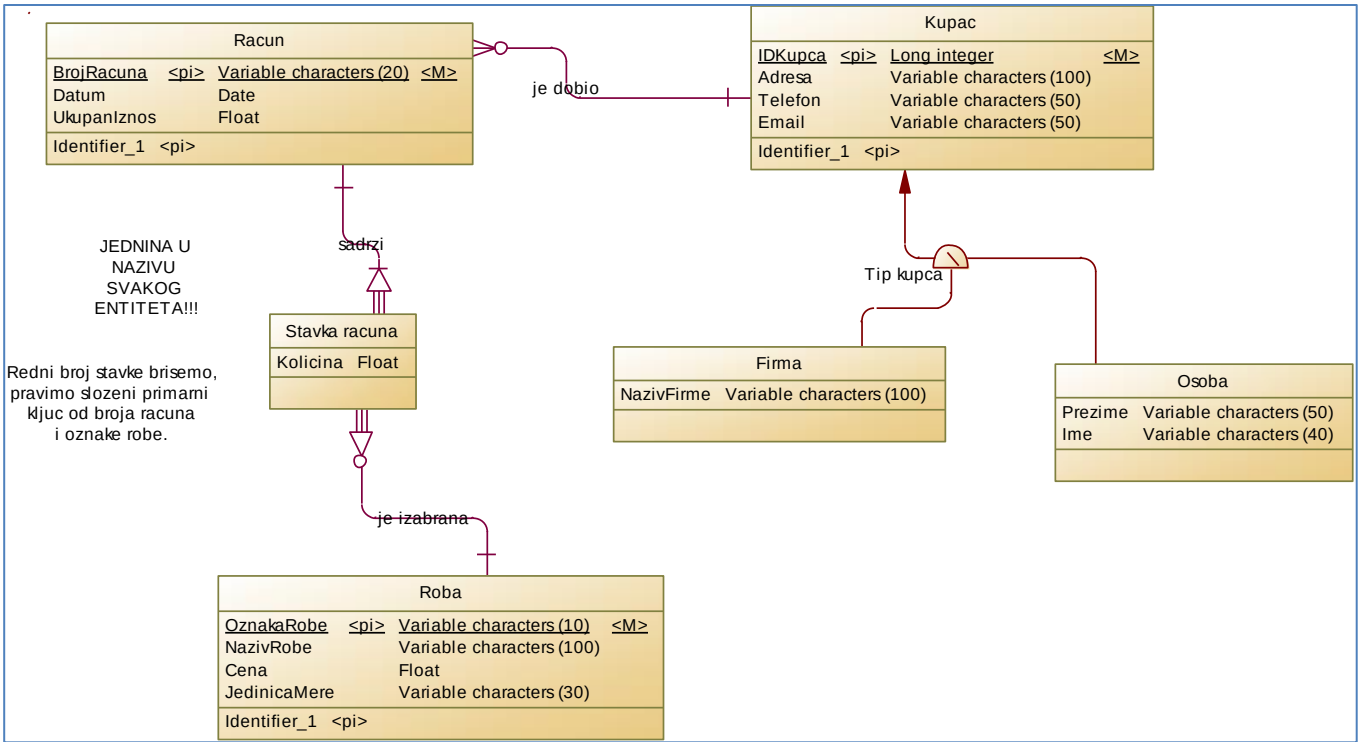
Racun (1:M dependent) POVEZAN SA stavke računa POVEZAN SA Roba
Racun POVEZAN SA kupac
Kupac IS A – DVE VRSTE: firma, osoba.

2. Generisati dijagram klasa iz CDM modela.
3.Podesiti karakteristike relacija:
- asocijacija (obicna relacija)
- nasledjivanje (od is-a)
- kompozicija (celina - deo)

4. Dodati set/get metode.
5. Iz dijagrama klasa generisati C# kod biblioteke klasa.
6. Dodati zajednički namespace (naziv projekta) za svaku klasu, da bi bile vidljive spolja i jedne između drugih.
7. Generisati dll (Dynamic Link Library).
8. Kreirati projekat tipa Windows forms i povezati korisnicki interfejs sa kreiranom bibliotekom DLL.

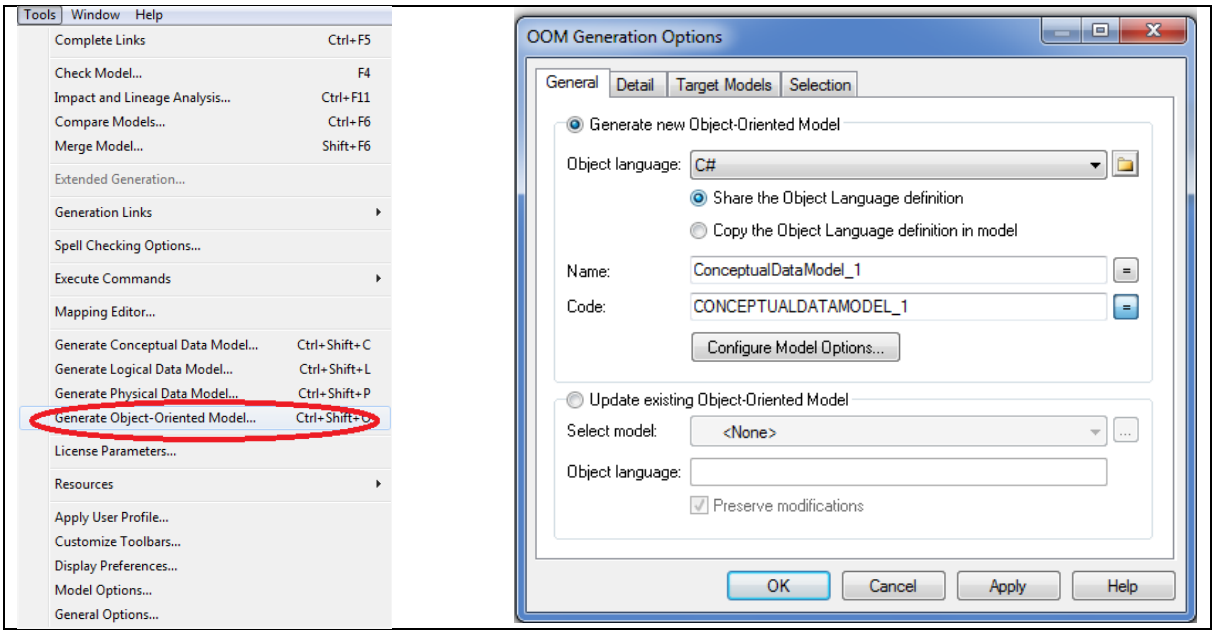
2.4.2. Rešenje – kreirana biblioteka klasa

1. Kreiran CDM model.



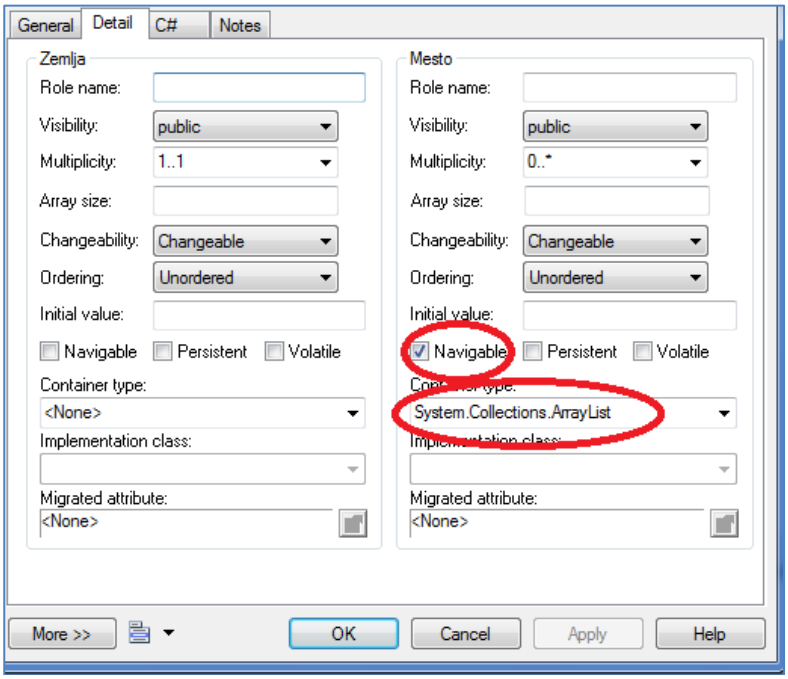
Slika 2.30. Dijagram CDM kreiran na osnovu prethodne slike

2. Generisanje OOM (object-oriented model) izborom opcije Tools – generate object oriented model, biramo jezik C#.



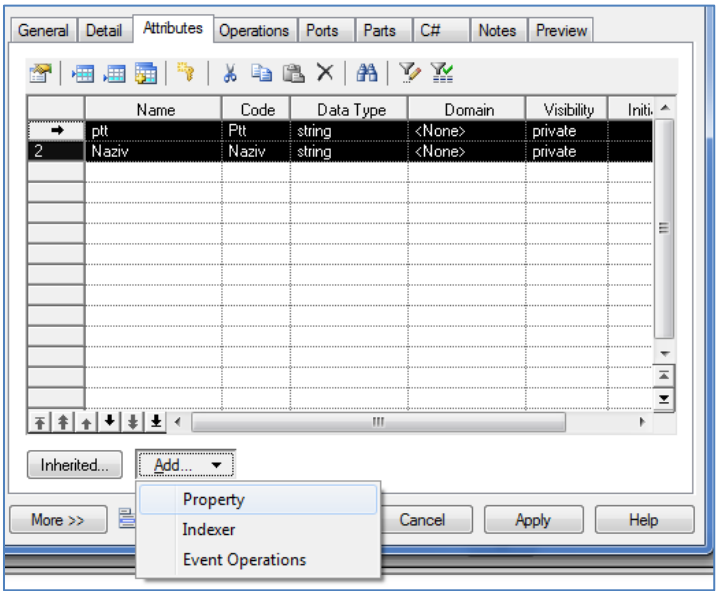
Slika 2.31. Stavka menija i podešavanje za generisanje dijagrama klasa OOM

3. Generisani model OOM, tj. dijagram klasa nema dobro podešene relacije, pa treba reorganizovati. Posebno se obraća pažnja na NAVIGABLE (migriranje objekata jedne klase u drugu klasu) i Container type (ako je ArrayList, uređena lista objekata jedne klase će migrirati i pojaviti se kao atribut druge klase).



Slika 2.32. Podešavanje svojstava Navigable i Container Type na relaciji između klasa

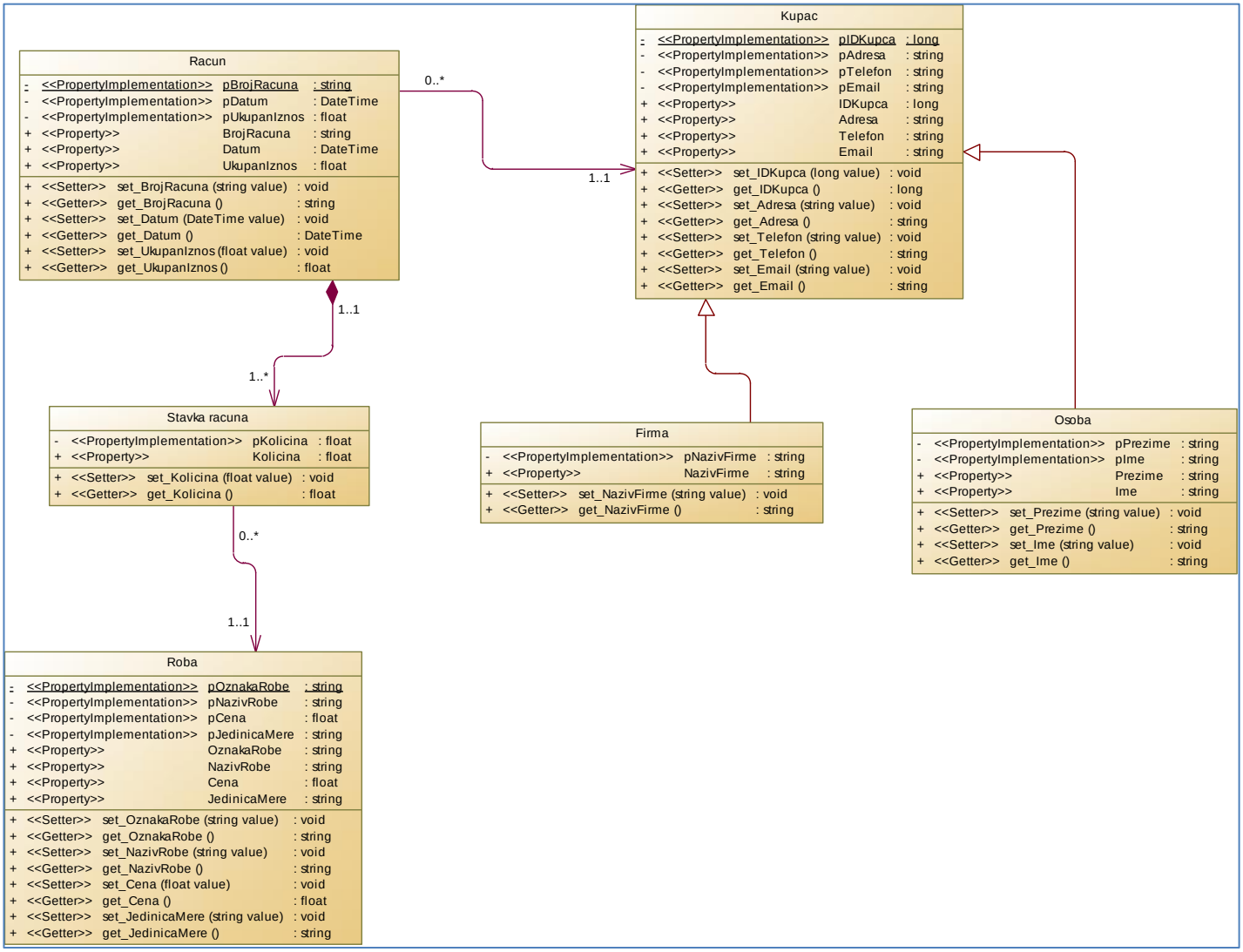
4. Podešavamo attribute da budu privatni, a dodajemo set-get metode selektovanjem atributa i izborom opcije add property.



Slika 2.33. Selektovanje atributa i izbor opcije Add Property za automatsko kreiranje public atributa sa get-set operacijama

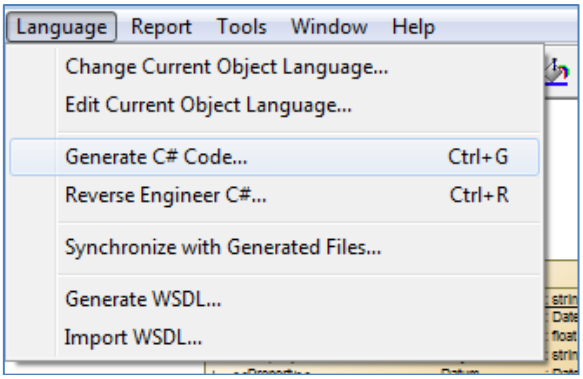
Na ovaj način za privatne (prave) attribute dodajemo mogućnost javnog pristupa za postavljanje vrednosti (set) ili čitanje (get).

Dobijeni rezultat:



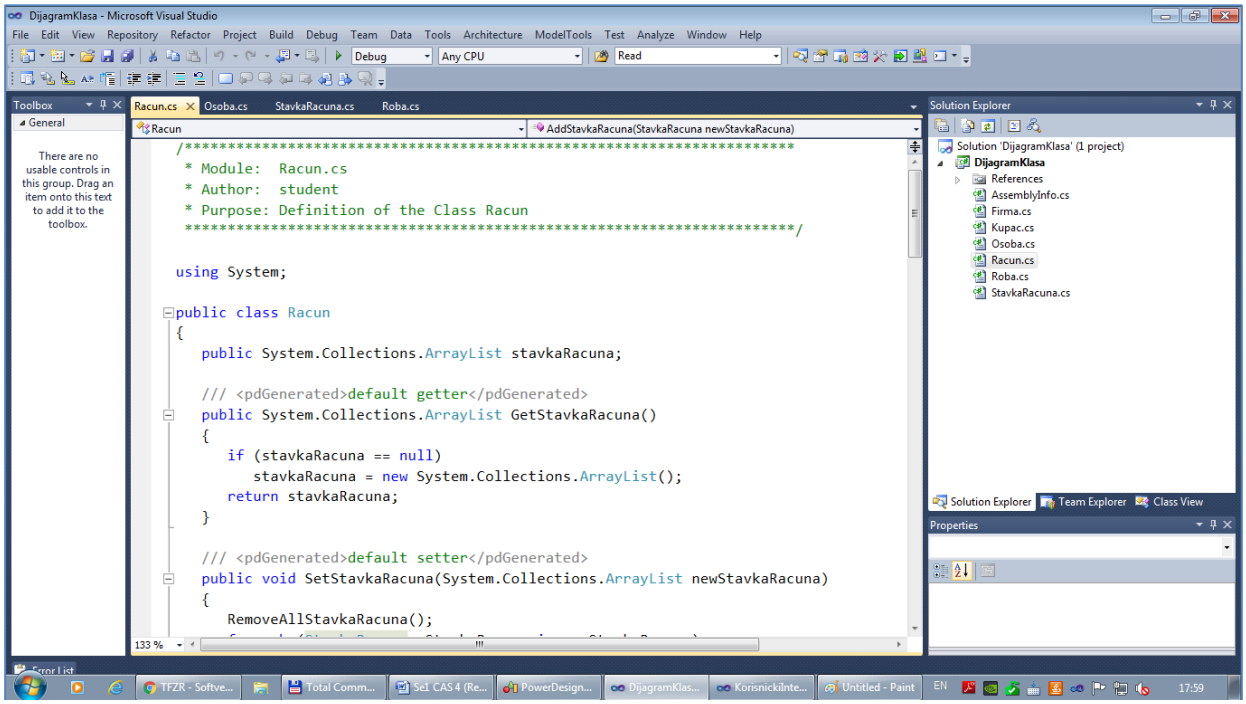
Slika 2.34. Konačan izgled dijagrama klasa nakon korekcija relacija i dodavanja get-set operacija

5. Generisanje C# koda klasa – opcija Language, generate C# code.



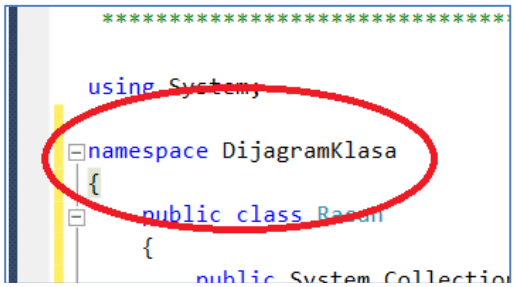
Slika 2.35. Izbor opcije za automatsko kreiranje C# koda klasa na osnovu dijagrama klasa

6. Dobijamo projekat tipa class library.



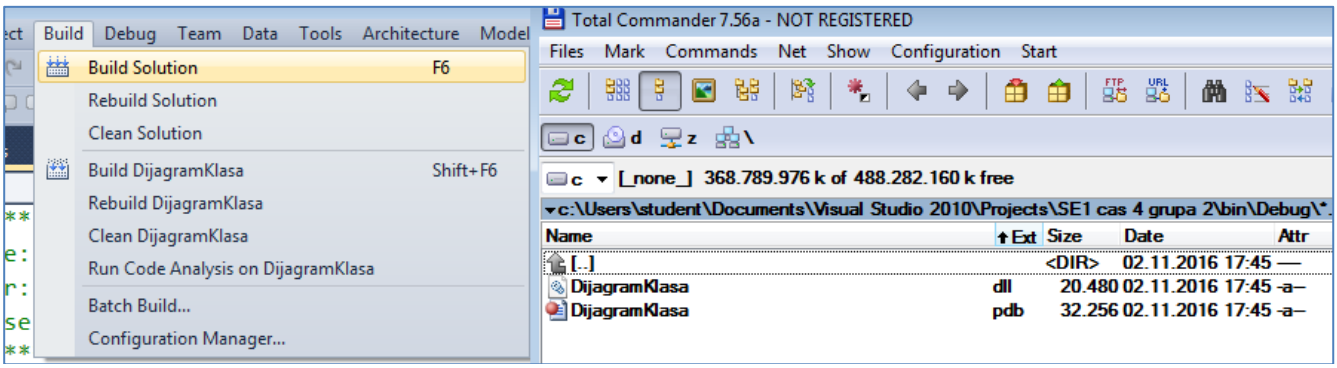
Slika 2.36. Razvojno okruženje Visual Studio .NET nakon otvaranja projekta tipa class library, koji je generisan iz CASE alata na osnovu dijagrama klasa

S obzirom da CASE alat nema podršku za novije verzije .NET, dodajemo zajednički ***namespace*** – to je naziv projekta kome pripadaju sve klase. Bez zajedničkog namespace, klase se međusobno ne vide i ne mogu se referencirati međusobno. Mogle bi jedino ako bi bile u istom fajlu sve klase, unutar jednog zajedničkog namespace.



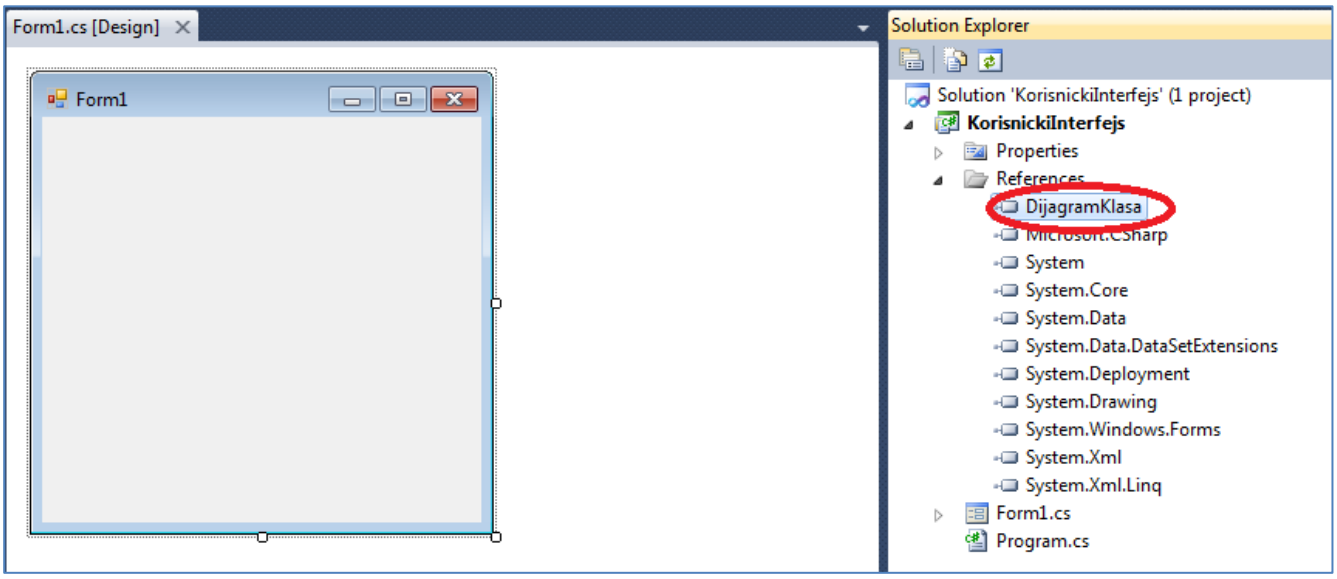
Slika 2.37. Dodavanje namespace kako bi sve klase pripadale zajedničkom projektu

Kreiramo izvršni oblik biblioteke klasa Dynamic Link Library (dll) fajl pokretanjem BUILD opcije iz projekta.



Slika 2.38. Izbor opcije za kreiranje DLL

7. Kreiramo windows forms projekat i dodajemo reference na dll. U solution exploreru, references, add reference i dodajemo kreirani dll. Pojavljuje se na spisku.



Slika 2.39. Rezultat dodavanja reference u Windows forms projekat

2.4.3. Analiza generisanog koda klasa i uticaja vrsta veza između klasa

Automatskim generisanjem programskog koda dobijena je jednostavna klasa koja ima samo set-get metode, kao i složenije klase koje sadrže elemente koji preslikavaju različite tipove veza na dijagramu klasa.

OBICNA KLASA, GET I SET ZA REALIZACIJU PROPERTY

```
public class Roba
{
  private string POznakaRobe;
  private string PNazivRobe;
  private float PCena;
  private string PJedinicaMere;

  public string OznakaRobe
  {
    get
    {
      return POznakaRobe;
    }
    set
    {
      if (this.POznakaRobe != value)
        this.POznakaRobe = value;
    }
  }
```

Objasnjenje za sva 3 najvaznija tipa veze sa dijagrama klasa – KAKO SE TO REALIZUJE U KODU:

NASLEDJIVANJE – klasa Osoba nasleđuje klasu Kupac. Opšti pojam Kupca ima zajedničke karakteristike za sve kupce, bilo oni pojedinci ili firme. Specijalizacijom, Osoba koja je kupac ima jos ime i prezime.

```
public class Osoba: Kupac
```

ASOCIJACIJA – objekat jedne klase sadrži kao svoj atribut objekat druge klase. U ovom primeru klasa StavkaRačuna sadrži atribut "roba" koji je tipa podatka "Roba", tj. "roba" je objekat klase "Roba".

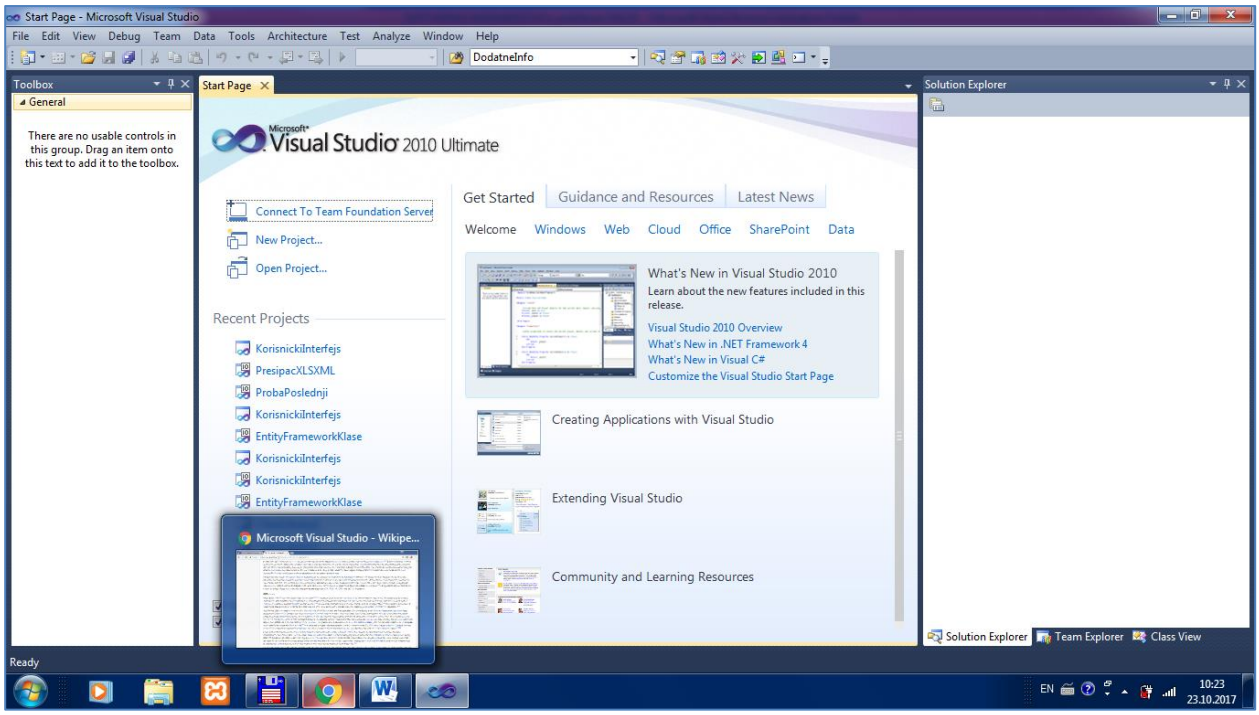
```
public class StavkaRacuna
{
  public Roba roba;

   private float PKolicina;
```

KOMPOZICIJA – objekat klase celine sadrži skup objekata (najčešće u formi liste) druge klase. U ovom primeru klasa Račun je klasa celine koja sadrži ArrayList objekata klase StavkaRačuna. Bolja varijanta je da se koriste tipizirane liste gde se naglašava tip podatka, tj. tip objekta koji je element liste, ali u nastavku je naveden ArrayList.

```
   public class Racun
    {
       public System.Collections.ArrayList stavkaRacuna;
public void AddStavkaRacuna(StavkaRacuna newStavkaRacuna)
       {
..
public void RemoveStavkaRacuna(StavkaRacuna oldStavkaRacuna)
       {
```

3. UPOZNAVANJE SA RAZVOJNIM OKRUŽENJEM VISUAL STUDIO .NET

Razvojno okruženje Visual studio .NET razlikuje se u odnosu na verzije alata. Verzija 2010 takođe se razlikuje u odnosu na namenu, čime se razlikuje obuhvat raspoloživih funkcija. U ovom odeljku će biti prikazani osnovni elementi verzije Visual Studio .NET 2010 Ultimate.



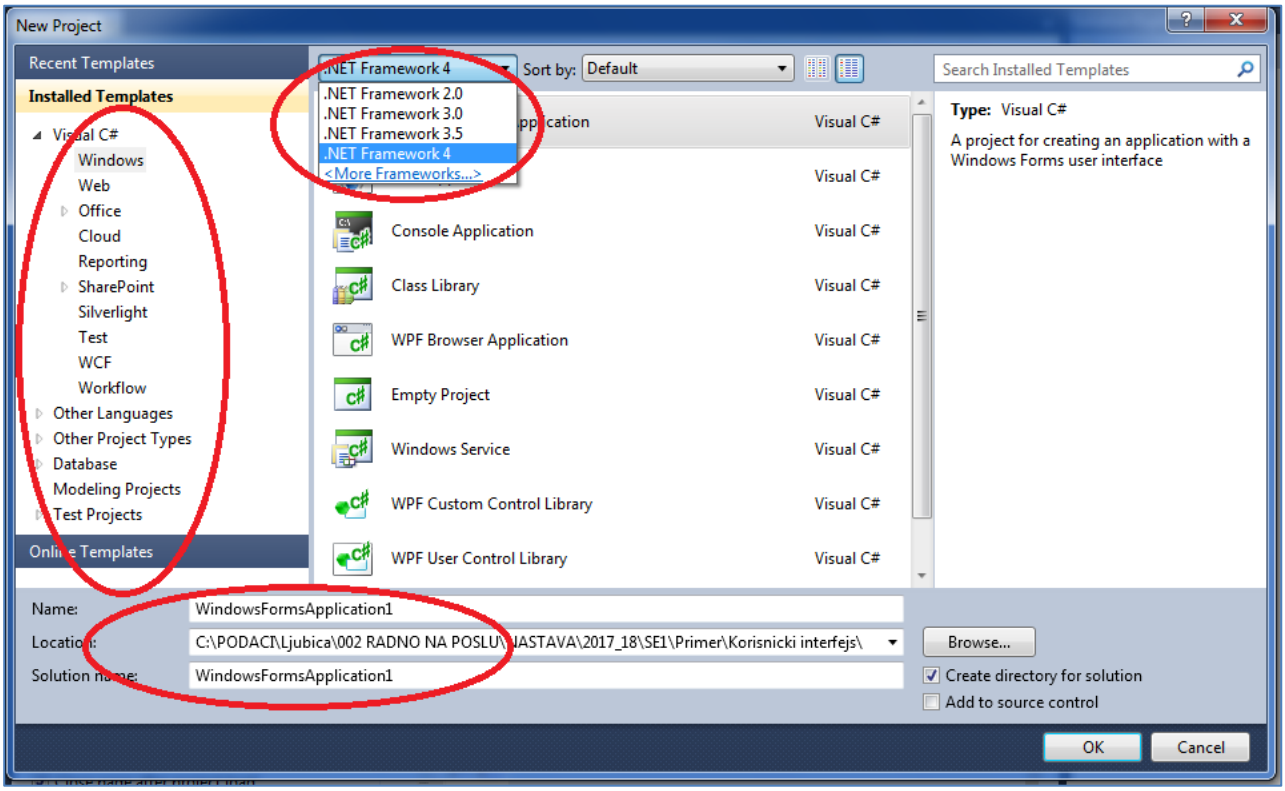
Slika 3.1. VS NET 2010 Ultimate razvojni alat, nakon pokretanja

U okviru uvodnog ekrana može se pokrenuti ranije realizovan projekat, tj. nastaviti sa radom ("Recent projects") ili izabrati kreiranje novog projekta ("New project").

U okviru opcije New project mozemo birati tip projekta, kao i verziju .NET frameworka, čime se bira opseg mogućnosti navedenih tipova projekata. Takođe, biramo naziv i lokaciju snimanja projekta.

Najčešći tipovi projekata su:

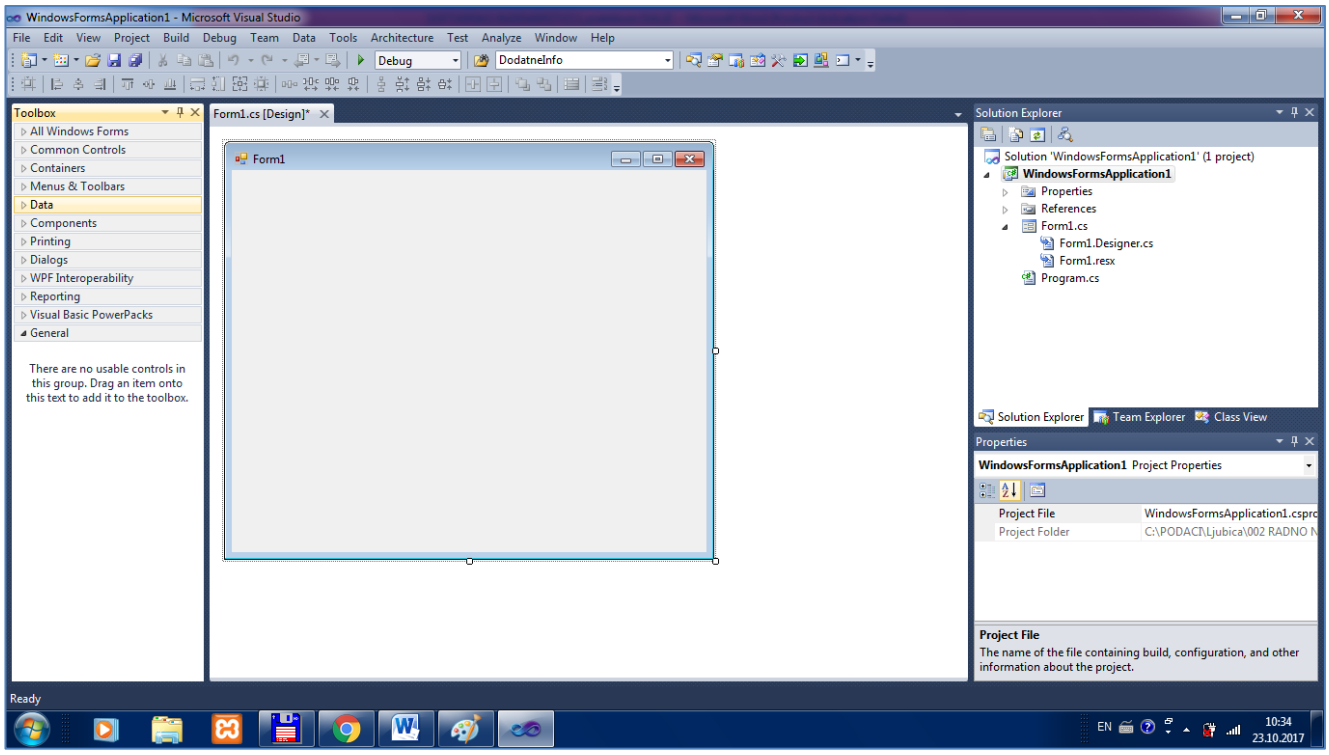
- Windows – windows forms projekat, class library projekat
- Web – ASP NET web application, ASP NET MVC application, ASP NET Web service application (NET framework 3.5)
- Reporting – crystal reports application
- Test Projects – Test documents project
- Database – SQL Server Project.



Slika 3.2. Izbor .NET framework, tipa projekta, naziva projekta i putanje gde će projekat biti sačuvan

3.1. Windows forms projekat

Kada kreiramo projekat tipa Windows forms, dobijamo radno okruženje, kao na sledećoj slici.



Slika 3.3. Radno okruženje Windows forms tipa projekta, nakon kreiranja

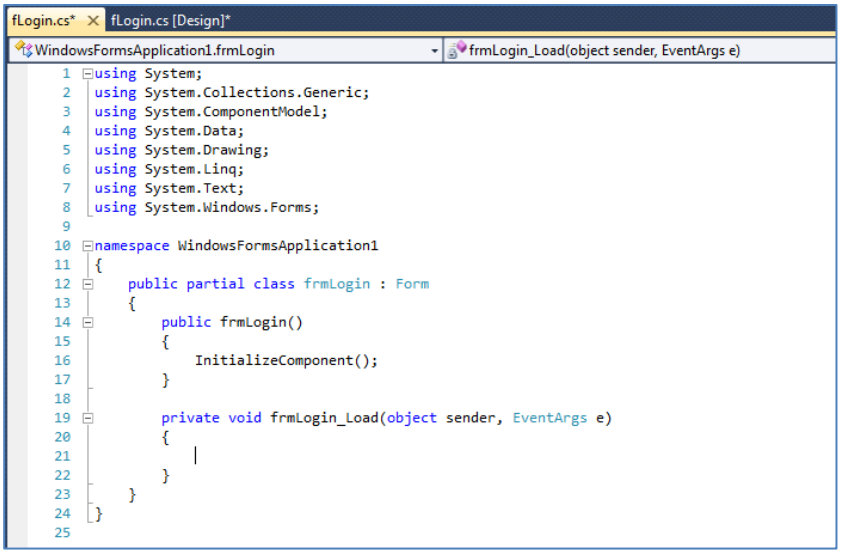
Osnovni gradivni element Windows aplikacije je forma. Celina svih fajlova koji čini aplikaciju udružena je u projekat, a više projekata može biti objedinjeno u Solution.

S leve strane vidimo Toolbox sa grupama vizualnih elemenata (kontrola) koje možemo postaviti na formu. Desno vidimo Solution explorer gde se nalazi spisak fajlova koji čini projekat i solution. Dole desno je prikaz osobina (Properties) grafičkog elementa – same forme ili kontrola koje sun a formi. Elementi mogu imati osobine (Properties – prikazani alfabetski ili po grupama) ili događaje (Events) povezane sa odgovarajućim elementom (symbol munje).

Najčešće kontrole koje se postavljaju na formu su:

- Common controls – button, check box, combo box, datetimepicker, label, listbox, radio button, text box, rich text box
- Containers – tab control, group box
- Menus and toolbars – menu strip, status strip
- Data – data grid view, binding source
- Reporting – report viewer

Nakon postavljanja kontrola, sledi pisanje programskog koda. Delu za pisanje koda može se pristupiti na više načina: klikom na prazan prostor same grafičke forme, klikom na neku kontrolu ili izborom nekog događaja sa properties prozora-events. Prostor za pisanje koda predstavljen je sledećom slikom. Sastoji se iz using dela (referenciranje na biblioteke klasa koje se koriste), namespace (naziv celog projekta gde pripada i ova forma), naziv forme (svaka forma je zapravo klasa koja nasleđuje opštiju klasu Form), deo koda forme (oivičen vitičastim zagradama), koji automatski uključuje konstruktor (pokreće se automatski i vrši grafičku pripremu forme putem procedure InitializeComponent) i Form_Load događaj (pokreće se nakon konstruktora, ali ovde programer može da definiše koji kod će se izvršiti).



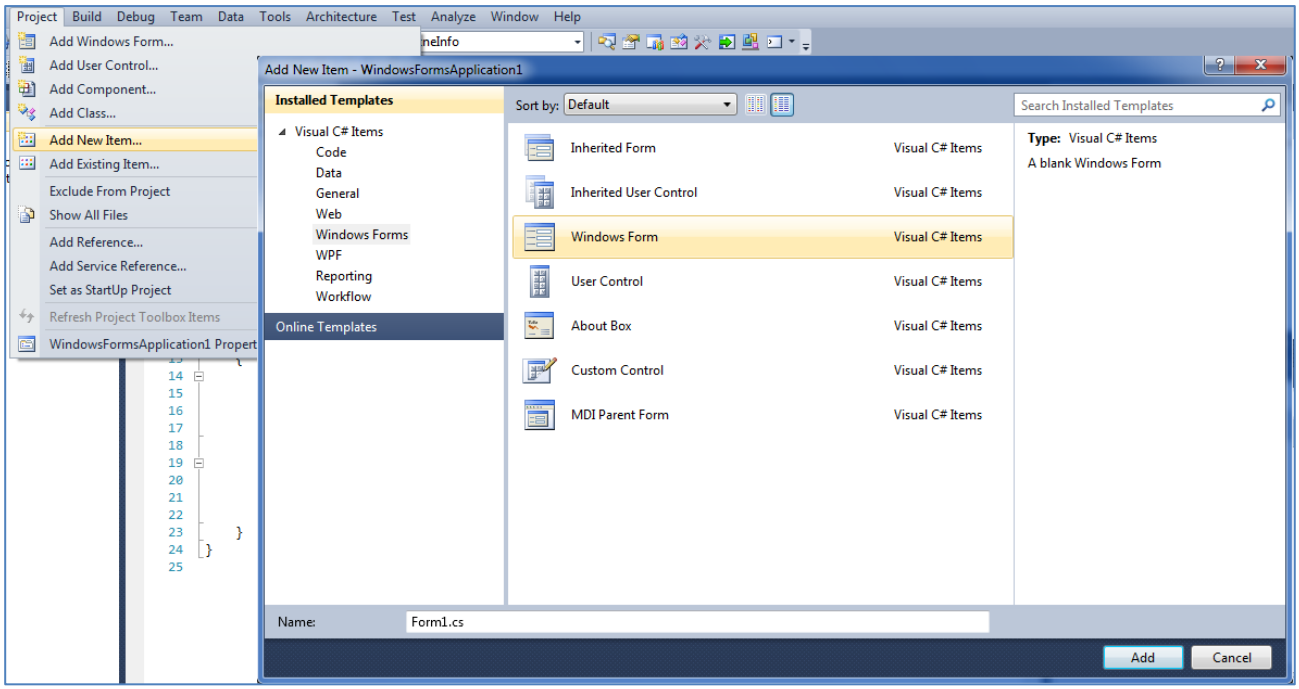
Slika 3.4. Prostor za pisanje programskog koda aplikacije

Projektu možemo dodavati nove fajlove različitog tipa, izborom opcije sa menija Project –Add new item. Tipovi fajlova su podeljeni po grupama. Najčešće korišćeni elementi koji se dodaju:

- Code – class

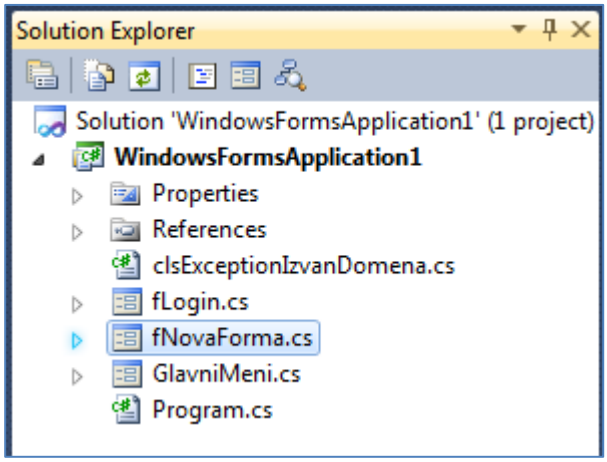
- Data – dataset, local database, service-based database, XML
- General – Bitmap file, icon
- Windows forms – Windows form, MDI parent form
- Reporting – report, crystal report.

Na slici koja sledi prikazano je kako dodati novu formu na projekat.



Slika 3.5. Dodavanje nove ekranske forme u projekat

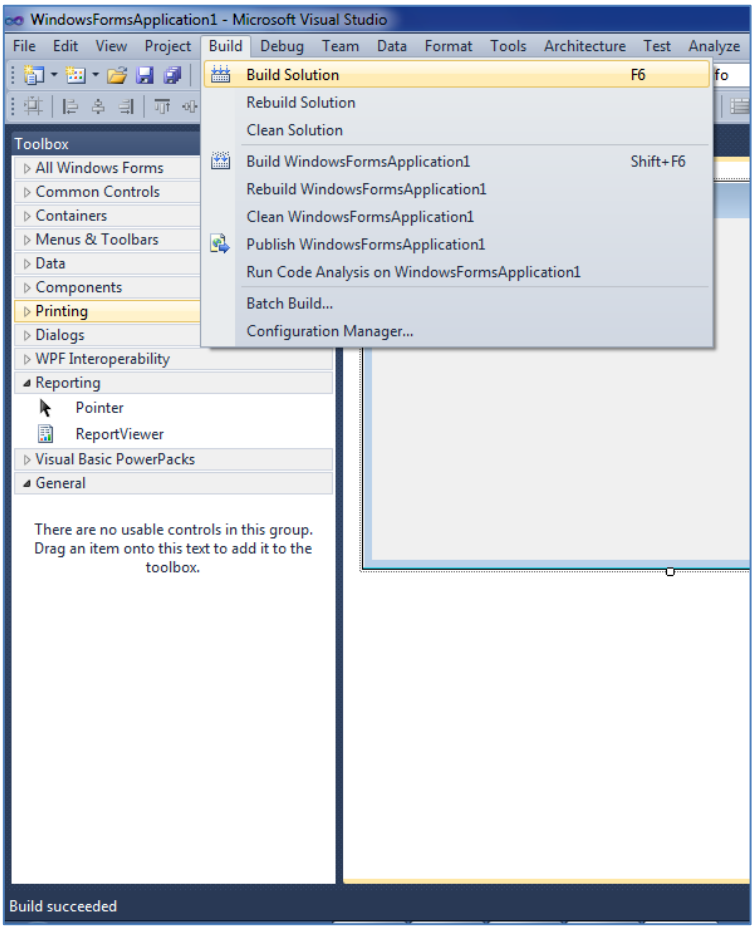
Nakon dodavanja formi, one se prikazuju u okviru Solution Explorera.



Slika 3.6. Prikaz dodate forme u okviru Solution Explorera

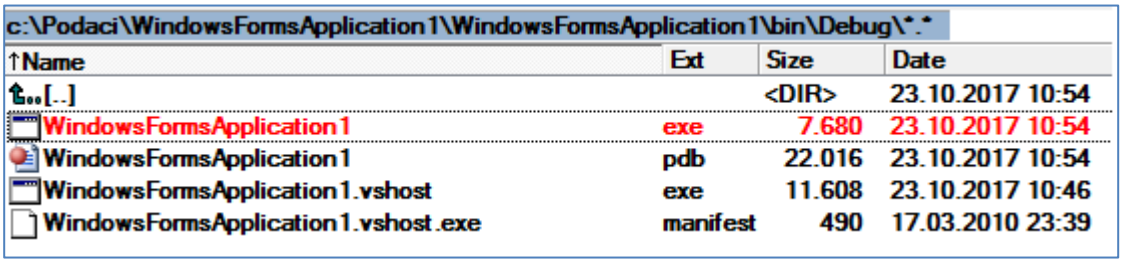
Treba razlikovati Solution i projekat. Jedan solution može sadržavati više različitih projekata.

Povremeno se može proveriti korektnost programskog koda ili kreirati izvršna verzija (EXE fajl) pokretanjem BUILD opcije – Build Solution.



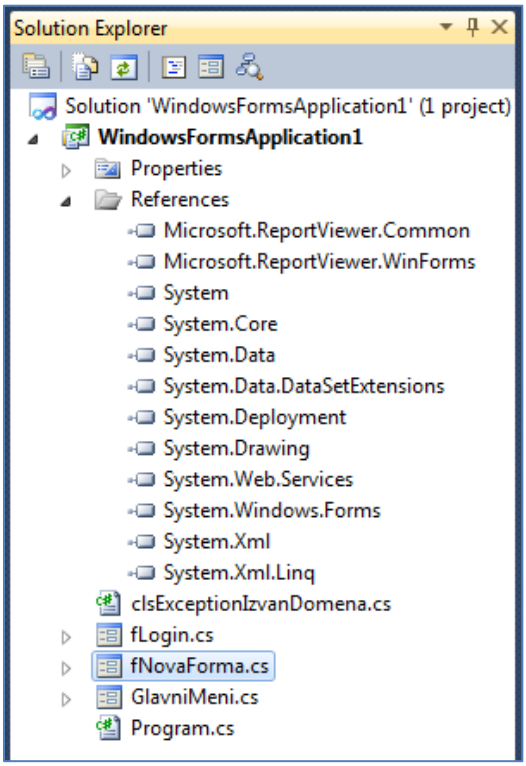
Slika 3.7. Pokretanje opcije Build solution

Nakon što je uspešno kreirana izvršna verzija, Windows aplikacija u izvršnoj formi (fajl sa ekstenzijom EXE) se može naći u okviru bin-debug foldera aplikacije.



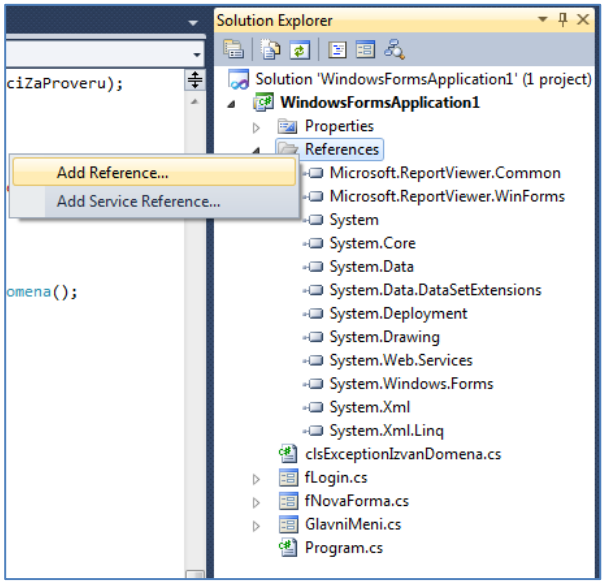
Slika 3.8. Kreirani EXE fajl Windows forms aplikacije

Projektu se može dodati nova biblioteka klasa. Kada se kreira projekat tipa Windows forms, u okviru Solution Explorera može se videti spisak automatski dodatih biblioteka klasa – References spisak.



Slika 3.9. Prikaz references spiska automatski priključenih standardnih biblioteka klasa

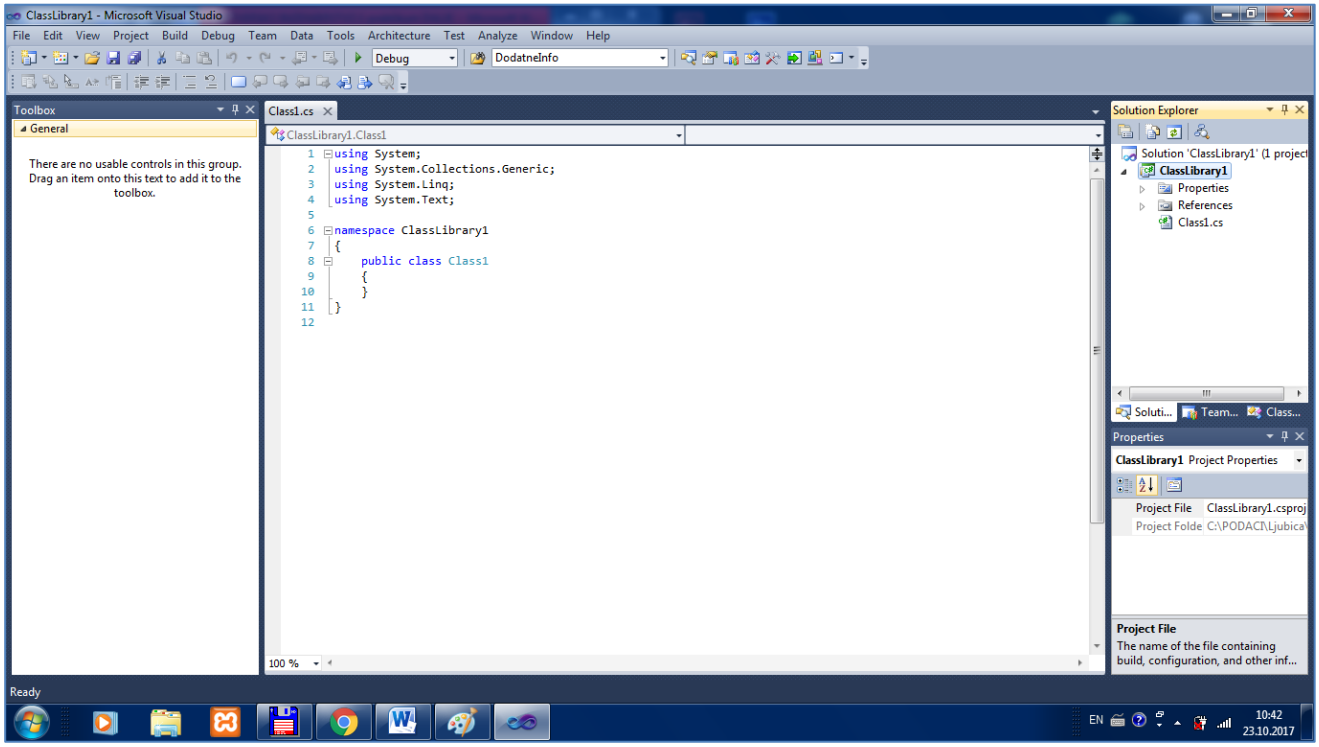
Možemo kreirati svoje biblioteke klasa ili pribaviti neki gotovi DLL. Možemo takve DLL biblioteke dodati na spisak references, opcijom ADD reference (desni taster miša kliknemo na References i dobija se pop-up meni sa opcijom Add reference).



Slika 3.10. Dodavanje DLL biblioteke pomoću opcije Add reference

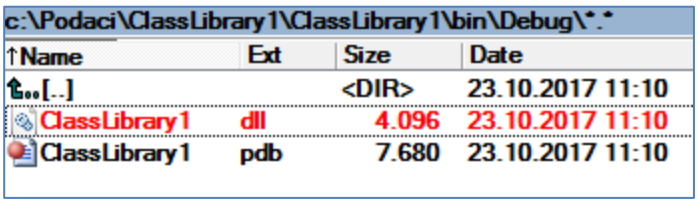
3.2. Class library projekat

Kada se pokrene kreiranje projekta tipa Class Library, dobijamo radno okruženje za pisanje programskog koda klasa, naravno, bez grafičkih alata za postavljanje kontrola, kao što je prikazano na slici.



Slika 3.11. Radno okruženje projekta tipa Class Library, nakon kreiranja

Nakon pokretanja Build opcije, dobijamo u bin-debug folderu kompajliranu verziju biblioteke klasa sa ekstenzijom DLL (Dynamic Link Library).



Slika 3.12. Kreirana biblioteka klasa u izvršnom obliku (DLL)

4. GRAFIČKO OBLIKOVANJE WINDOWS APLIKACIJE I POVEZIVANJE FORMI

4.1. Zadatak

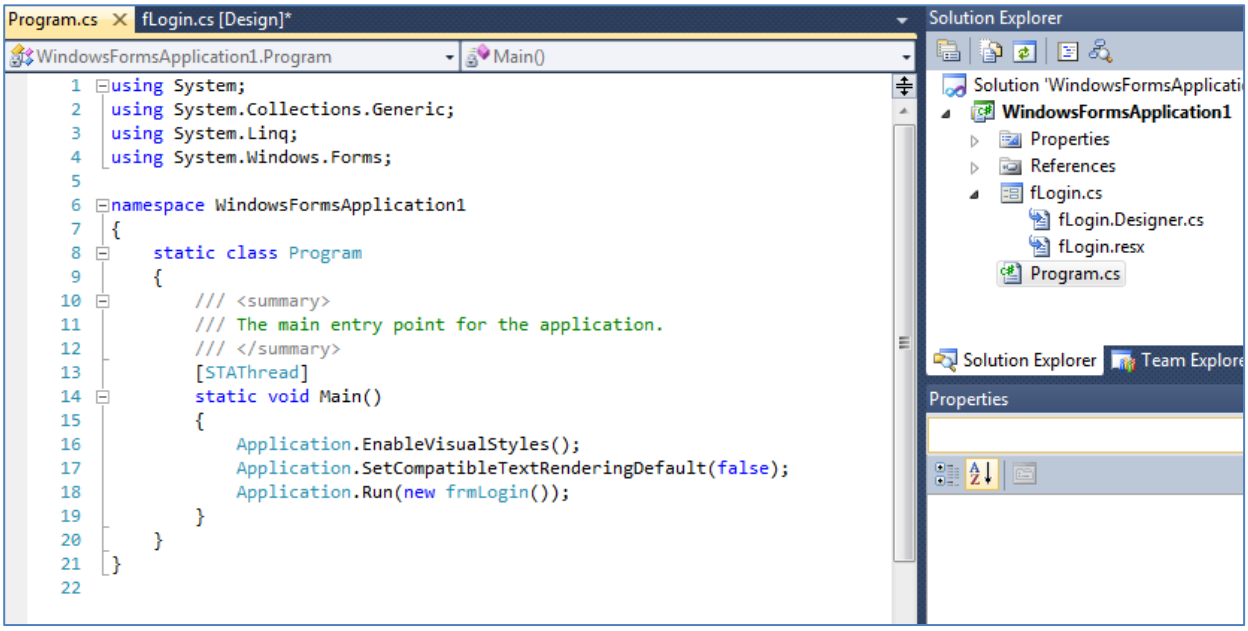
Uobičajene funkcije poslovne aplikacije prikazane su u poglavlju koje se odnosi na UML i dijagram slučajeva korišćenja. Zadatak je grafički oblikovati forme windows aplikacije tako da obuhvataju osnovne operacije sa podacima jedne poslovne aplikacije.

4.2. Rešenje

U nastavku će biti prikazano grafičko oblikovanje windows aplikacije za tipične ekrane aplikacije, koristeći raspoložive alate sa palate grafičkih kontrola.

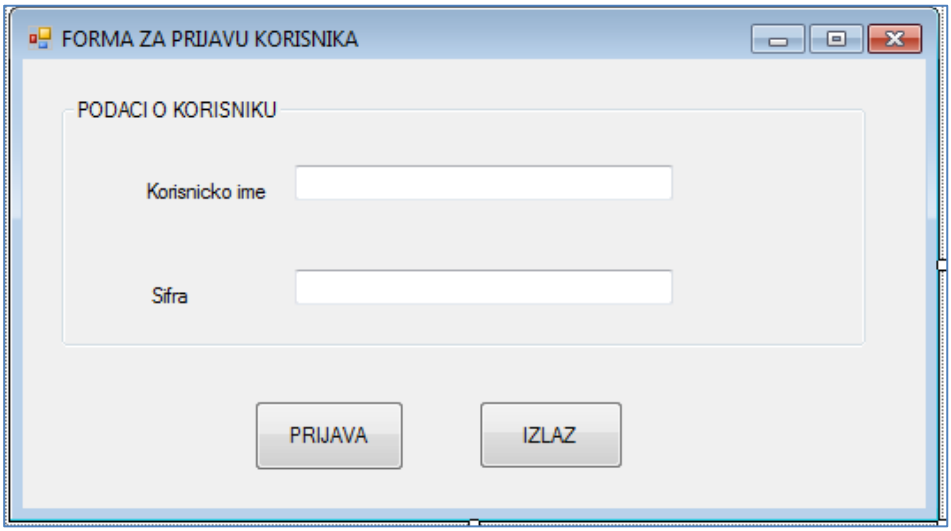
4.2.1. Forma za prijavljivanje korisnika

Forma za prijavljivanje korisnika je prva forma koja se pokreće, što se može podesiti u fajlu Program.cs.



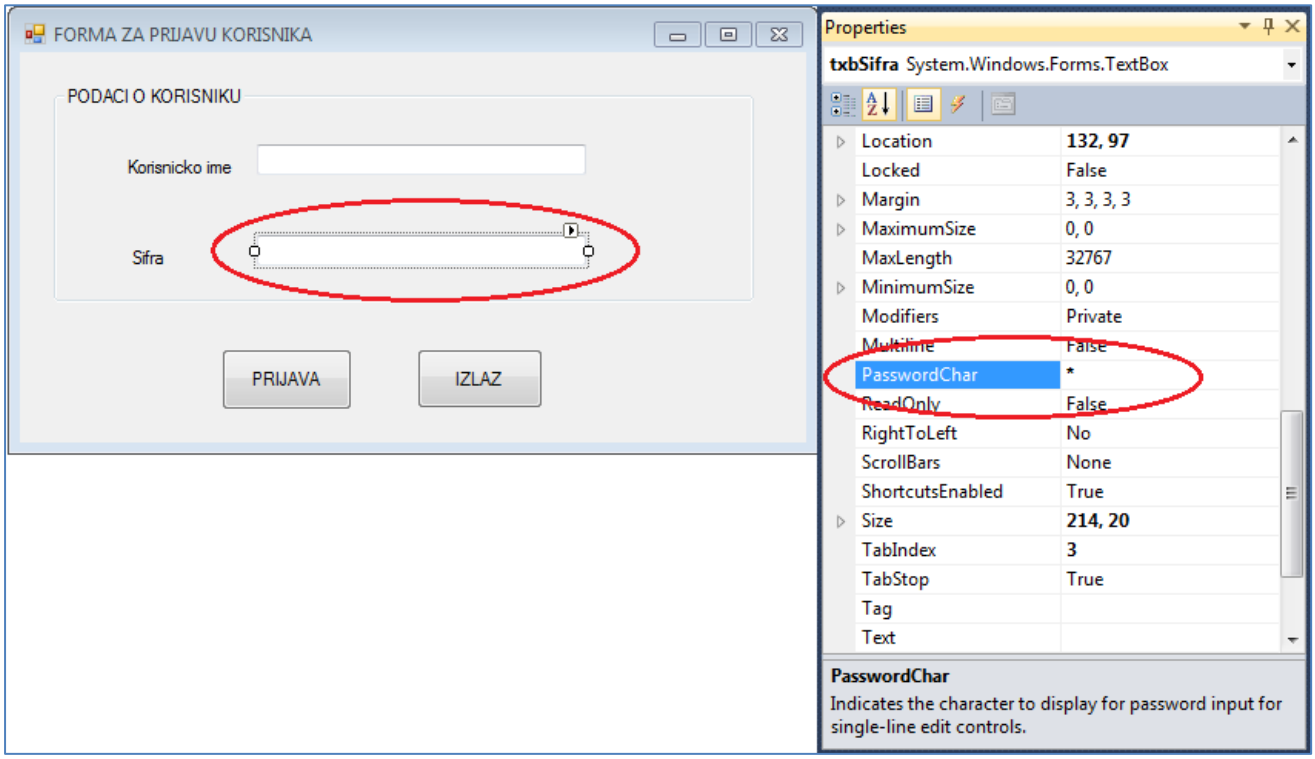
Slika 4.1. Podešavanje statne ekranske forme u okviru fajla Program.cs

Dizajn forme za prijavu korisnika prikazan je na sledećoj slici. Korišćen je group box za objedinjavanje podataka o korisniku, labela, text box i button. Prilikom dodavanja kontrola, svaka kontrola kojoj će se programski pristupati je dobila mnemonički naziv (Name svojstvo u Properties), npr. btnPrijava, txbKorisničkoIme...dok je ono što je napisano na samoj kontroli podešeno u Text svojstvu u Properties, npr. PRIJAVA, IZLAZ.



Slika 4.2. Izgled uobičajene forme za prijavljivanje korisnika

Podesićemo za text box gde se unose podaci o šifri da se ne prikazuju unete vrednosti, već simbol kojim se uneti karakteri maskiraju (sakrivaju). Najčešće je to simbol zvezdice.



Slika 4.3. Podešavanje Password Char karaktera koji će se prikazati u text boxu za šifru

Programski kod za prebacivanje fokusa sa jedne na drugu kontrolu primenom ENTER tastera na tastaturi (očitavanjem koda karaktera) je predstavljen u nastavku:

```
private void txbKorisnikoIme_KeyPress(object sender, KeyPressEventArgs e)
{
   //KOD ZA ENTER

   if (Convert.ToInt32(e.KeyChar) == 13)
```

```
        {
            txbSifra.Focus();
        }
    }

    private void txbSifra_KeyPress(object sender, KeyPressEventArgs e)
    {
        //KOD ZA ENTER

        if (Convert.ToInt32(e.KeyChar) == 13)
        {
            btnPrijava.Focus();

        }
    }
```

Pokretanje forme za glavni meni:

```
private void btnPrijava_Click(object sender, EventArgs e)
    {
        GlavniMeni  objGlavniMeni = new GlavniMeni();
        objGlavniMeni.ShowDialog();
    }
```

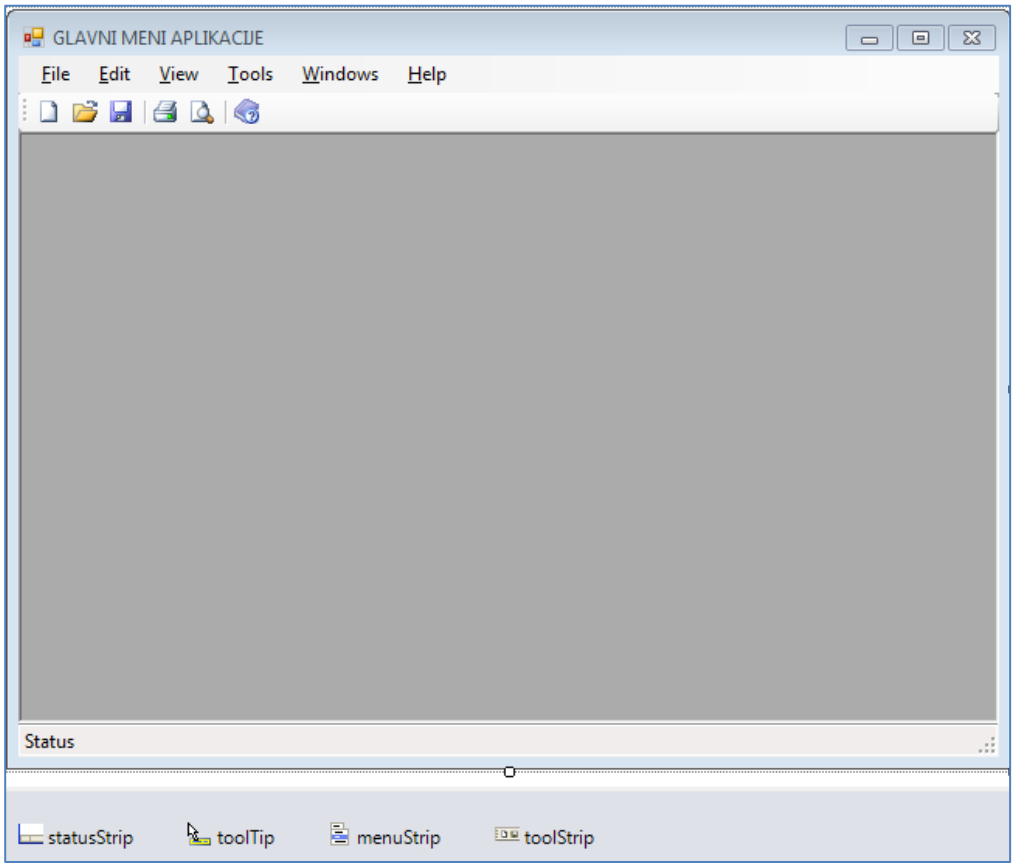
Izlaz iz aplikacije:

```
private void btnIzlaz_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }
```

4.2.2. Forma za glavni meni

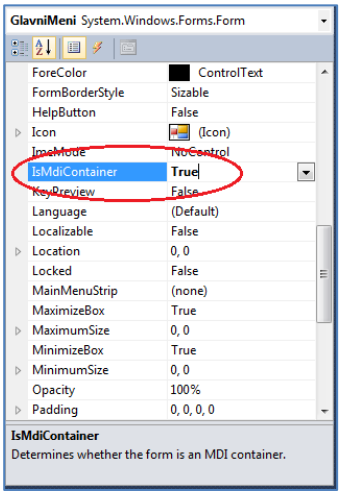
Uobičajeno je da je forma za glavni meni zapravo MDI (Multiple Document Interface) parent ili container forma. MDI parent form je tip forme koja objedinjuje sve ostale forme i sadrži glavni meni i statusnu liniju cele aplikacije. Ako minimizujemo MDI formu, automatski se i sve ostale forme koje njoj pripadaju takođe minimizuju. Ako se zatvori MDI forma, zatvaraju se automatski i sve forme unutar nje.

Nakon dodavanja MDI parent forme na projekat, dobijamo, nakon izmene naziva fajla i naslovne linije:



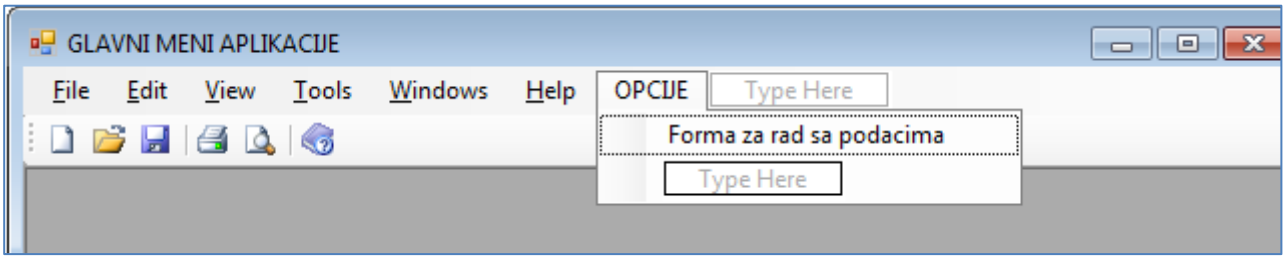
Slika 4.4. Forma sa glavnim menijem, tipa MDI

MDI forma ima u Properties za svojstvo IsMDIcontainer podešeno – true, dok ostale, obične forme imaju – false.



Slika 4.5. Podešavanje osobine forme da je MDI tipa

Na ovoj formi koja predstavlja glavni meni treba promeniti stavke menija i povezati sa konkretnim stranicama da se pokreću kada se klikne na stavku menija. Dodaćemo stavke menija u dizajn režimu.

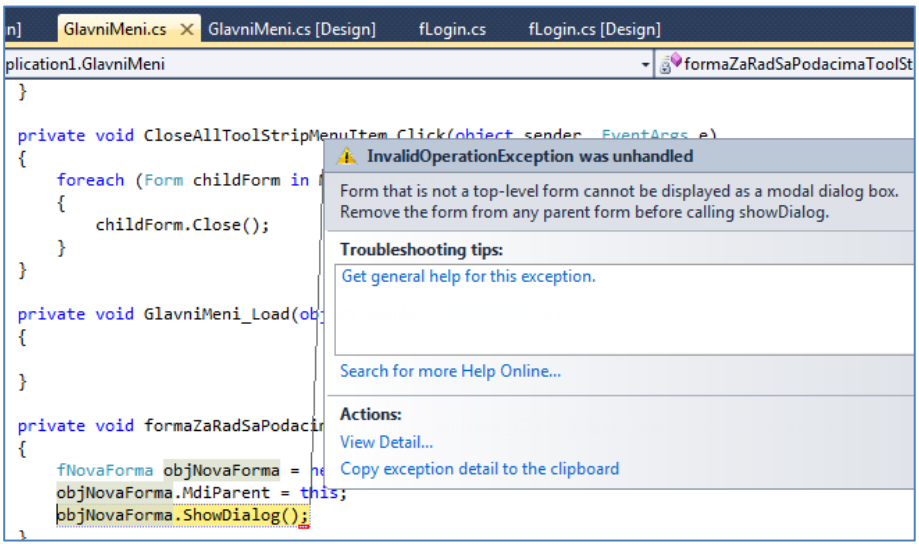


Slika 4.6. Dodata stavka i podstavka menija

Pokretanje forme za rad sa podacima:

```
private void formaZaRadSaPodacimaToolStripMenuItem_Click(object sender, EventArgs e)
      {
         fNovaForma objNovaForma = new fNovaForma();
         objNovaForma.MdiParent = this;
         objNovaForma.ShowDialog();
      }
```

Ovde se javio problem, jer je korišćena metoda **ShowDialog** koju je moguće koristiti samo kod MODALNOG prikazivanja formi. Modalne forme su forme sa kojima korisnik jedino može raditi u nekom trenutku, bez mogućnosti da paralelno koristi neke druge forme. U ovom slučaju, nova forma koja se otvara iz glavnog menija je MDI CHILD forma i pripada MDI formi kao većoj celini, što znači da korisnik može istovremeno da radi i sa širom, MDI formom u svakom trenutku. Dakle, MDI child forma nije modalna i ne može se pozvati ShowDialog, već samo **Show** naredbom.

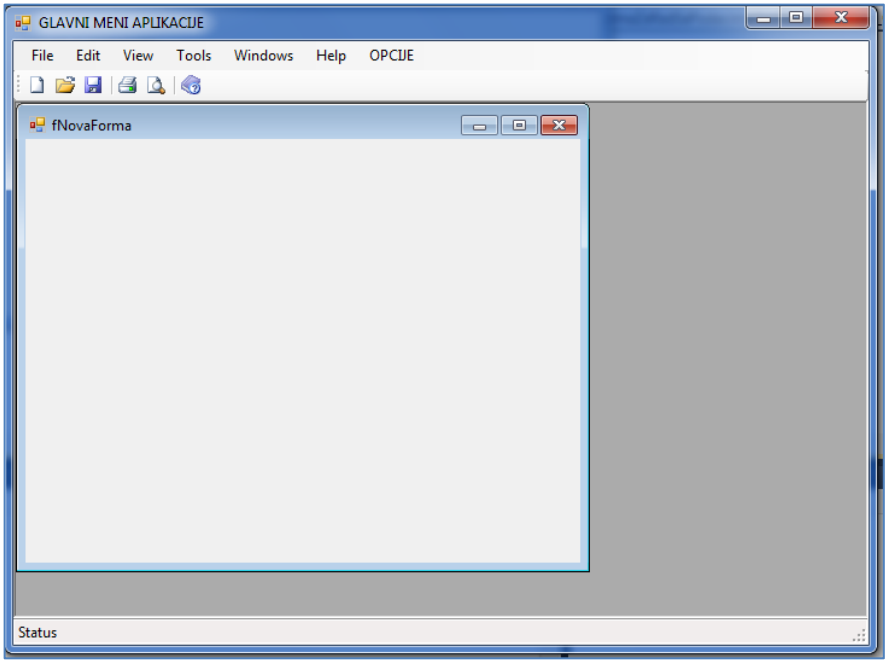


Slika 4.7. Prijavljena greška kada MDI child forma koristi naredbu ShowDialog

Dakle, ispravan kod je:

```
private void formaZaRadSaPodacimaToolStripMenuItem_Click(object sender, EventArgs e)
      {
         fNovaForma objNovaForma = new fNovaForma();
         objNovaForma.MdiParent = this;
         objNovaForma.Show();
      }
```

Kada se učita MDI child forma, ona se prikazuje unutar MDI parent forme. Kada se MDI parent forma minimizuje, sve MDI child forme se minimizuju.

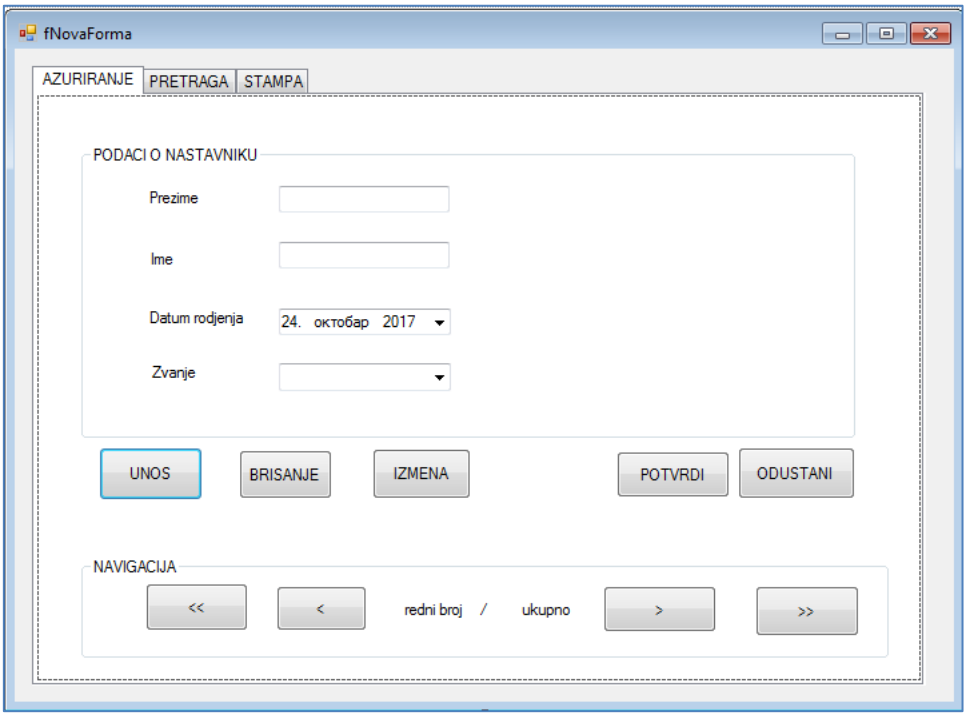


Slika 4.8. Forma tipa MDI sa učitanom MDI child formom

4.2.3. Forma za rad sa podacima

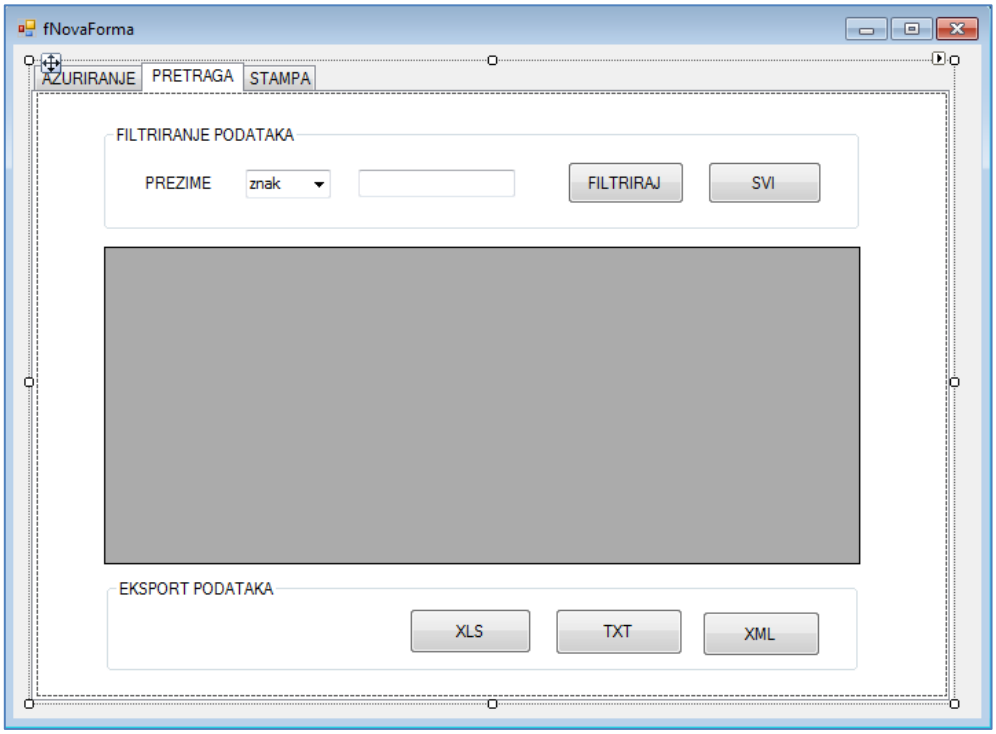
Forma za rad sa podacima ima uobičajene opcije za ažuriranje podataka, tabelarni prikaz sa filtriranjem i eksportom, kao i štampu. Sve operacije vezane za jednu tabelu baze podataka ili manji skup tabela koje su potrebne za jedan poslovni proces mogu se organizovati na jednoj formi, koristeći tab kontrolu sa tri kartice – azuriranje, pretraga, stampa.

Naredna slika prikazuje grafičko uređivanje dela stranice za ažuriranje sa opcijama za unos, brisanje, izmenu i navigaciju (prvi,prethodni, sledeći, poslednji). Na kartici za azuriranje koriscene su kontrole: labela, text box, date time picker, combo box, button i group box.



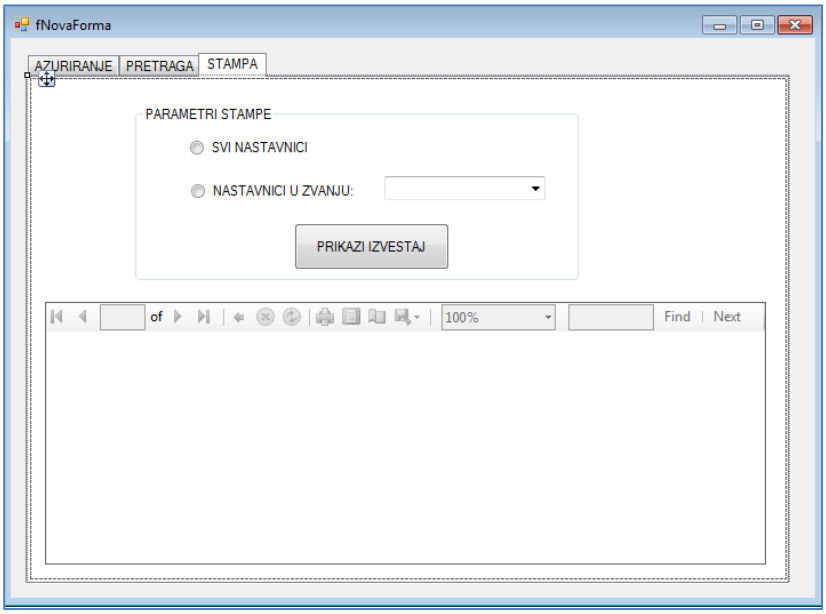
Slika 4.9. Forma za rad sa podacima – prva ekranska kartica za navigaciju i ažuriranje

Opcije za tabelarni prikaz, filter i eksport nalaze se na drugoj kartici. Korišćene kontrole: labela, combo box, text box, button, group box, data grid view.



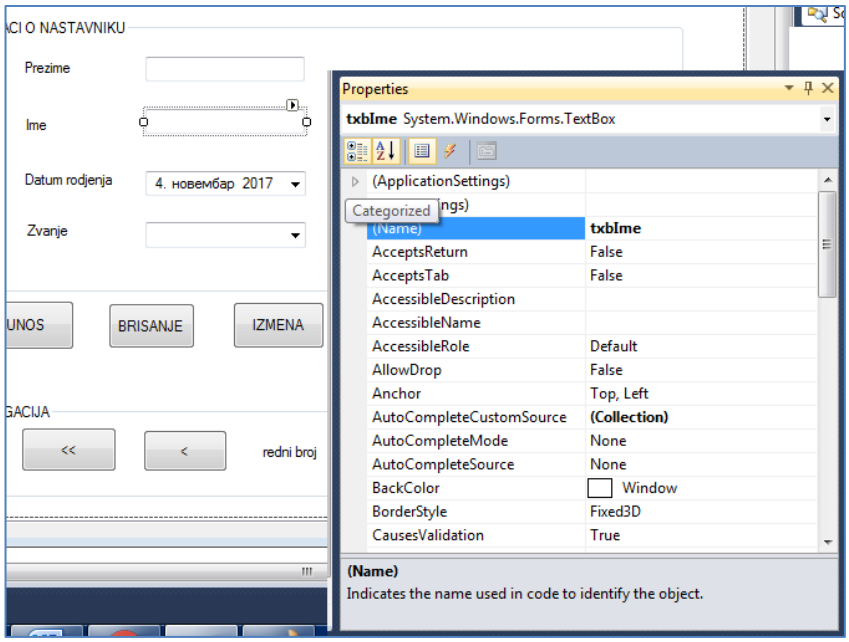
Slika 4.10. Forma za rad sa podacima – druga ekranska kartica za tabelarni prikaz, filtriranje i eksport podataka

Opcije za izbor vrste izveštaja (svi podaci i filtrirani podaci, tj. parametarski izveštaj) kao i kontrola za prikaz izveštaja pre štampe, prikazane su na 3. Kartici. Korišćene kontrole: radio button, combo box, button, Report viewer.



Slika 4.11. Forma za rad sa podacima – treća ekranska kartica za štampu

Nakon postavljanja grafičkih kontrola, one se mogu urediti u okviru Properties prozora. Potrebno je odrediti im mnemoničko ime, gde će prva 3 slova označavati skraćen tip kontrole, a ostali deo naziv u skladu sa kontekstom primene. Npr. text box za unos imena će imati naziv txbIme. Taj naziv će se kasnije koristiti u programskom kodu.



Slika 4.12. Podešavanje osobina (Properties) grafičkih kontrola

5. SINTAKSA C# PROGRAMSKOG JEZIKA

Osnove sintakse programskog jezika čine pravila za pisanje osnovnih elemenata programa. U nastavku će biti prikazan kratak pregled elemenata sintakse [3].

5.1. 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.

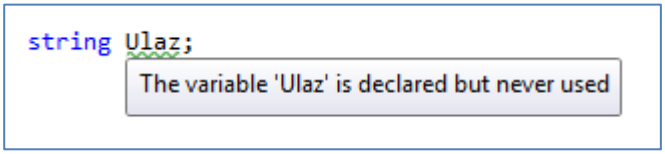
5.2. Promenljive, konstante i tipovi podataka

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.

Tipovi 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

Nazivi promenljivih u C#

- Prvi znak mora biti slovo ili _ ili @
- Ostali znaci mogu biti slova, brojevi ili _
- Kao nazivi promenljivih ne smeju se koristiti službene reči, tj. nazivi naredbi, npr. „using"...

Promenljive tipa vrednosti u C#

Sadrže vrednosti. Izvedene su od klase System.ValueType. Primeri za C# 2010:

Type	Represents	Range	Default Value
bool	Boolean value	True or False	False
byte	8-bit unsigned integer	0 to 255	0
char	16-bit Unicode character	U +0000 to U +ffff	'\0'
decimal	128-bit precise decimal values with 28-29 significant digits	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0 \text{ to } 28}$	0.0M
double	64-bit double-precision floating point type	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	0.0D
float	32-bit single-precision floating point type	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F
int	32-bit signed integer type	-2,147,483,648 to 2,147,483,647	0

long	64-bit signed integer type	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	8-bit signed integer type	-128 to 127	0
short	16-bit signed integer type	-32,768 to 32,767	0
uint	32-bit unsigned integer type	0 to 4,294,967,295	0
ulong	64-bit unsigned integer type	0 to 18,446,744,073,709,551,615	0
ushort	16-bit unsigned integer type	0 to 65,535	0

u- znači "unasigned", tj. nema negativnih brojeva.

Promenljive tipa reference u C#

Promenljive tipa reference ukazuju na memorijsku lokaciju gde se nalaze vrednosti promenljivih, a ne sadrže same vrednosti tih promenljivih. Tu spadaju:

- Standardni reference tipovi:
  - Object – bazna klasa za sve tipove podataka, to je System.Object klasa.
  - Dynamic – opšti tip podatka, koji se može konvertovati u konkretan.
  - String – niz karaktera, to je System.String klasa.
- Korisnički definisani reference tipovi:
  - Class
  - Interface
  - Delegate

Promenljive tipa reference su i tipa niz, lista, tipizirana lista.

Pokazivači u C#

Promenljive tipa pokazivača čuvaju memorijsku adresu promenljive drugog tipa. Primeri:

```
char* cptr;
int* iptr;
```

Razlika string i String

Kada deklarišemo promenljivu tipa string, radimo sa njom kao sa prostom promenljivom vrednosnog tipa. Kada deklarišemo promenljivu tipa String, tada zapravo instanciramo objekat klase System.String i radimo sa takvim objektom korišćenjem atributa i metoda.

Enumeracija

Tip podatka koji se definiše navođenjem skupa elemenata (konstanti) koji može da ima promenljiva tog tipa.

Primer:

```
Enum Temperatura
{
TempSmrzavanja=0,
TempKljucanja=100
}
```

Koriscenje:

```
(int) Temperatura.TempSmrzavanja
```

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 konverzija** – 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);

KONSTANTA

Konstanta je slična promenljivoj, s tim da ne može da joj se menja vrednost u toku izvršavanja programa.

DEFINICIJA: Const int nazivkonstante=vrednost;
PRIMER: public const int X = 1, Y = 2, Z = 3;

5.3. Operatori i funkcije

Redosled navođenja operatora u ovom odeljku određuje i stepen prioriteta. Svaka naredna grupa ima niži stepen prioriteta izvešavanja nego operatori prethodne grupe, ali viši prioritet u odnosu na narednu grupu operatora.

Primarni operatori
f(x) – poziv funkcije
a[x] – pristup članu indeksirane strukture.
x++ – postfix increment. Vraća vrednost x, a nakon toga menja vrednost memorijske lokacije označene sa x tako što je uvećava za 1.
x-- – postfix decrement. Vraća vrednost x i nakon toga na lokaciji x umanjuje vrednost x za 1.
New – instanciranje objekta klase.
Typeof () – vraća System.Type objekat koji je argument ove funkcije.
default(T) – vraća default inicijalizovanu vrednost za tip podatka T, a to je: null za promenljive tipa reference, nulu za numeričke tipove,a za strukturne tipove vraća promenljivu sa elementima koji imaju null ili nulu.
Sizeof () – vraća veličinu u bajtima za objekat koji je argument ove funkcije.

Unarni operatori
-x – numerička negacija.
!x – logička negacija.
++x – prefix increment. Izvršava uvećavanje vrednosti x za 1 i vraća vrednost x nakon što je promenjena, tj. uvećana za 1.
--x – prefix decrement. Izvršava umanjenje za 1 vrednosti x i vraća tu umanjenu vrednost.
(T)x – type casting.
&x – address of – memorijska adresa promenljive x.

Multiplikativni operatori
x * y – množenje.
x / y – deljenje. Ako su operandi tipa integer, rezultat je integer idecimalni deo se ne prikazuje.
x % y – modul. Modul je ostatak pri deljenju broja x i y, ako su x i y tipa integer.

Aditivni operatori
- x + y – sabiranje.
- x – y – oduzimanje.
- Spajanje stringova

primer:
```
string NoviString = prviString + " " + drugiString;
```

Shift operatori
x << y – pomeraju bitove levo i popunjavaju nulom sa desne strane.
x >> y – pomeraju bitove desno. Ako je levi operand int ili long, levi bitovi se popunjavaju bitom znaka, a ako su uint ili ulong, levi bitovi se popunjavaju nulom.

Relacionalni i Type-testing operatori
x < y – vrednost ovog izraza je tačna, ako je x manje od y.
x > y – vrednost ovog izraza je tačna, ako je x veće od y.
x <= y – vrednost ovog izraza je tačna, ako je x manje ili jednako sa y.

x >= y – vrednost ovog izraza je tačna, ako je x veće ili jednako sa y.
Is – type compatibility. Vraća vrednost TAČNO ako vrednost levog operanda može da se menja po tipu (cast) u tip podatka specificiran sa desne strane operanda.
As – type conversion. Vraća izmenjen (type casting) vrednost levog operanda u tip naveden kao desni operand, ali vraća null ako takva konverzija nije moguća.

Operatori jednakosti
x == y – jednakost. Vraća vrednost TAČNO ako su vrednosti promenljivih identične. Za promenljive tipa reference, osim stringa, vraća da li su reference na memorijsku lokaciju iste, a ne da li su same vrednosti iste. Ipak, za proveru reference je bolje koristiti ReferenceEquals metod nad objektima.
x != y – nije jednako. Slično kao i za jednakost.

Logički operatori
- logički ili na nivou bita. Korišćenje sa integer tipovima enum tipovima je takođe dozvoljeno.

x & y –AND.
x ^ y – XOR.
x | y –OR.
x && y – uslovno logičko AND. Ako je prvi operand NETAČNO, drugi se ni ne proverava.
x || y – uslovno logičko OR. Ako je prvi operand TAČNO, drugi se ni ne proverava.

Null-coalescing operator
x ?? y – vraća x ako nije null vrednost, a inače vraća y.

Kondicionalni operator
t ? x : y – Ako je t tačno, proveri i vrati vrednost x, inače proveri i vrati vrednost y.

Operatori dodele ili lambda operatori
x = y – dodela vrednosti. Kod promenljivih tipa reference, pod imenom x i y se nakon ovog trenutka očitava zajednička memorijska lokacija.
x += y – increment. Dodaje vrednost y na vrednost x i smešta na lokaciju promenljive x i vraća novu vrednost x.
x -= y – decrement. Oduzima vrednost y na vrednost x i smešta na lokaciju promenljive x i vraća novu vrednost x.
x *= y – multiplication assignment. Množi vrednost y sa vrednosti x i smešta na lokaciju promenljive x i vraća novu vrednost x.
x /= y – division assignment. Deli vrednost x sa vrednosti y i smešta na lokaciju promenljive x i vraća novu vrednost x.
x %= y – modulus assignment. Deli vrednost x sa vrednosti y i smešta ostatak na lokaciju promenljive x i vraća novu vrednost x.
x &= y – AND assignment. Primenjuje logičko AND vrednosti y sa vrednosti x, smešta u x i vraća novu vrednost x.
x |= y – OR assignment. Primenjuje logičko OR vrednosti y sa vrednosti x, smešta u x i vraća novu vrednost x.
x ^= y – XOR assignment. Primenjuje logičko XOR vrednosti y sa vrednosti x, smešta u x i vraća novu vrednost x.
x <<= y – left-shift assignment. Pomera vrednost x levo za y mesta, smešta rezultat u x i vraća novu vrednost x.
x >>= y – right-shift assignment. Pomera vrednost x desno za y mesta, smešta rezultat u x i vraća novu vrednost x.

5.4. 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
};
```

PREKIDI TOKA PROGRAMA ILI FUNKCIJE

goto, break, continue, return

GOTO naredba – skok na obeleženo mesto. Nije preporučljivo da se koristi.

```
LABELA:
…
GOTO LABELA;
```

PROCEDURE I FUNKCIJE

Atomarne grupe naredbi možemo pozivati zajedničkim imenom, čime kreiramo procedure ili funkcije. Procedura ima zaglavlje (signaturu) i telo (blok naredbi).

SIGNATURA – opšti oblik

(Modifikator pristupa) (vrednost koju vraća) NAZIV (parametri definisani kao skup: ulazno-izlazni tip tippodatka/klasa naziv parametra)

- Modifikatori pristupa: private, public, protected
- Vrednost koju vraća: void (ne vraća vrednost – procedura) ili neki tip podatka (prosti ili složeni)
- Ulazno-izlazni tip podatka: ref (parameter tipa reference tj. ulazno-izlazni), in, out

PRIMER:

```
SIGNATURA: Private void DajPrezime (out string Prezime)
POZIV:  DajPrezime(Prezime);
Ili
SIGNATURA: Private string DajPrezime()
POZIV:  string Prezime = DajPrezime();
```

Ukoliko procedura vraća vrednost, može se smatrati funkcijom. Funkcija vraća vrednost svojim pozivom. Procedura može vratiti vrednost i ulazno-izlaznim parametrom. Ukoliko je u pitanju

funkcija, ne pišemo **void** već tip vrednosti koju vraća, a kao poslednju naredbu u bloku moramo navesti naredbu:

```
return promenljivakojavraćavrednost;
```

5.5. Obrada grešaka

5.5.1. Testiranje programa

Provera grešaka u izvršavanju samog programa može da se vrši testiranjem programa. Osnovne metode testiranja su metode crne kutije (testiranje bez ulaženja u programski kod, testira se samo ponašanje izvršne verzije programa u odnosu na specifikaciju funkcionalnosti ili u odnosu na nefunkcionalne zahteve) i metode bele kutije (testiranje uz poznavanje programskog koda, testiraju se pojedini delovi koda). Jedna od tehnika, koja omogućava testiranje metodama bele kutije je obeležavanje tačke prekida programa nakon čega će se izvršavanje programa nastaviti, ali uz praćenje vrednosti promenljivih i ponašanje programa.

```
111         private void formaZaRadSaPodacimaToolStripMenuItem_Click(object sender, EventArgs e)
112         {
113             fNovaForma objNovaForma = new fNovaForma();
114             objNovaForma.MdiParent = this;
115             objNovaForma.Show();
116         }
117     }
118 }
119
```

5.5.2. Naredbe za detekciju i obradu grešaka

Kako bi se izbegle situacije da program prekida sa radom, jer je u stanju kada ne može da odreaguje na neku situaciju, takve situacije treba predvideti i obezbediti odgovarajuće naredbe. Ipak, programmer ne može da predvidi sve situacije i zato postoje posebne naredbe za obradu grešaka, ukoliko nastupe. Postoje posebne vrste-tipovi grešaka. Opšta klasa za rad sa greškama je System.Exception, odnosno biblioteka System.ApplicationException.

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.

5.5.3. Tipovi standardnih klasa za obradu grešaka

Osnovna klasa za rad sa greškama je System.Exception i ostale klase su izvedene iz nje:

- System.ApplicationException – obrada grešaka aplikativnog programa. Greške koje programeri definišu treba da su izvedene iz ove klase.
- System.SystemException – bazna klasa za sve predefinisane sistemske greške.

Primeri izvedenih grešaka na osnovu SystemException:

Exception Class	Description
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from deferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

PRIMER KONZOLNE APLIKACIJE:

```
using System;
namespace ErrorHandlingApplication
{
  class DivNumbers
  {
    int result;
    DivNumbers()
    {
      result = 0;
    }
    public void division(int num1, int num2)
    {
      try
      {
        result = num1 / num2;
      }
      catch (DivideByZeroException e)
      {
        Console.WriteLine("Exception caught: {0}", e);
      }
      finally
      {
        Console.WriteLine("Result: {0}", result);
      }
    }
    static void Main(string[] args)
    {
      DivNumbers d = new DivNumbers();
      d.division(25, 0);
      Console.ReadKey();
    }
  }
}
```

REZULTAT IZVRŠAVANJA:

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
at ...
Result: 0
```

5.5.4. Kreiranje sopstvenih klasa za obradu grešaka

Korisnički-definisane klase za obradu grešaka se izvode iz Exception class. Nakon što je zadovoljen uslov, pomoću naredbe throw pokreće se izvršavanje programskog koda koji je definisan za odgovarajući tip greške koji je programmer dodatno osmislio.

PRIMER:

```
using System;
namespace UserDefinedException
{
```

```
  class TestTemperature
  {
    static void Main(string[] args)
    {
      Temperature temp = new Temperature();
      try
      {
        temp.showTemp();
      }
      catch(TempIsZeroException e)
      {
        Console.WriteLine("TempIsZeroException: {0}", e.Message);
      }
      Console.ReadKey();
    }
  }
}
public class TempIsZeroException: Exception
{
  public TempIsZeroException(string message): base(message)
  {
  }
}
public class Temperature
{
  int temperature = 0;
  public void showTemp()
  {
    if(temperature == 0)
    {
      throw (new TempIsZeroException("Zero Temperature found"));
    }
    else
    {
      Console.WriteLine("Temperature: {0}", temperature);
    }
  }
}
```

REZULTAT IZVRŠAVANJA:

```
TempIsZeroException: Zero Temperature found
```

5.6. Primer primene sintaksnih elemenata programskog jezika C#

5.6.1. Zadaci

Za prethodno grafički uređene stranice poslovne aplikacije, realizovati delove programskog koda kojima se ilustruju osnovne programske strukture. Svi programski kodovi ovog zadatka realizuju se u okviru osnovne forme za rad sa podacima.

KOMENTARI I PLANIRANJE ALGORITMA

1. U okviru tastera "POTVRDI", "FILTRIRAJ", "SVI", "PRIKAZI IZVESTAJ" komentarima pripremiti odeljke programskog koda kojima bi se realizovala odgovarajuća funkcionalnost.

SEKVENCA
2. Na tasteru "ODUSTANI" na prvoj ekranskoj kartici realizovati programski kod kojim se briše sadržaj svih kontrola.
3. Napisati proceduru, koja bi se pozvala u okviru click događaja na ovom tasteru, za realizaciju brisanja sadržaja svih kontrola.

SELEKCIJA
4. Na tasteru "POTVRDI" na prvoj ekranskoj kartici osnovne forme za rad sa podacima realizovati programski kod kojim se proverava da li su sve kontrole popunjene. Ako neka nije, postaviti focus na prvu kontrolu koja nije popunjena.

5. Na tasteru "FILTRIRAJ" na drugoj ekranskoj kartici u zavisnosti od vrednosti iz combo boxa ("znak"), formirati izraz za filtriranje, koji bi se koristio u SQL upitu tipa select.

ITERACIJA
6. Na ekranskoj kartici za štampu, napuniti Combo box nazivima zvanja nastavnog osoblja. Podaci o zvanjima su dati u okviru eksternog XML fajla. Kreirati proceduru koja preuzima podatke iz XML i proceduru koja puni combo, pa pozvati na Form_Load.

SELEKCIJA I OBRADA GREŠAKA
7. Na ekranskoj kartici za štampu na tasteru "PRIKAZI IZVESTAJ" proveriti:
- Da li je bar jedan radio button izabran
- Ako je izabran 2. Radio button, da li je izabrana vrednost combo boxa ili je prazan

8. Napuniti combo sa vrednostima zvanja i na kartici za stampu pozivom istih procedura kao i za combo na 1. Kartici, u okviru form load događaja. Ako je izabran 2. Radio button i combo box je napunjen, proveriti da li ima vrednost iz skupa dozvoljenih vrednosti. Kreirati odgovarajuću proceduru koja proverava da li combo box ima vrednost iz skupa dozvoljenih vrednosti.

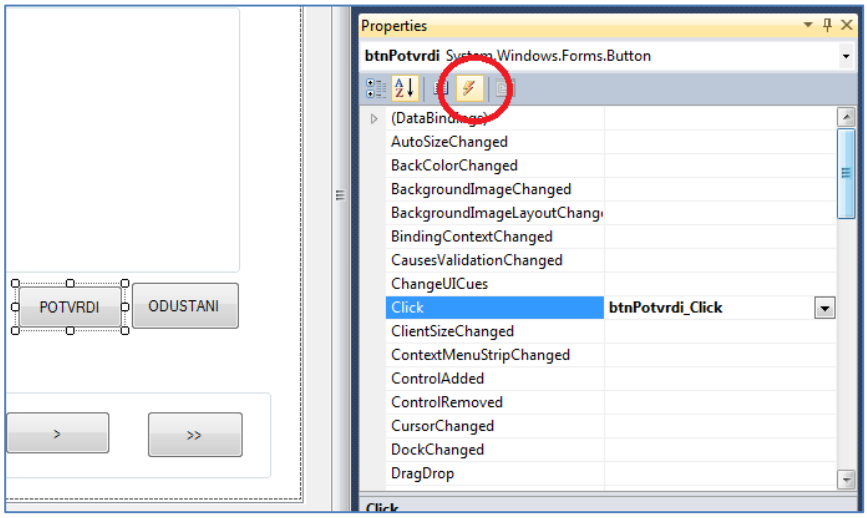
5.6.2. Rešenja

1. zadatak

Kontrole osim osobina imaju i prateće događaje. Kreiranje zaglavlja događaja, npr.

```
private void btnPotvrdi_Click(object sender, EventArgs e)
{
```

Može se realizovati duplim klikom na button gde se automatski kreira zaglavlje procedure i prazan blok procedure u programskom kodu forme, a takođe se registruje i u okviru properties prozora. Ovaj ili bilokoji događaj se može inicijalno kreirati direktno iz properties prozora, u okviru dela za Events (symbol "munjice").



Slika 5.1. Izbor tastera (simbol munje) za prikaz spiska događaja koji se odnosi na izabranu kontrolu

```
private void btnPotvrdi_Click(object sender, EventArgs e)
      {
         // preuzimanje vrednosti sa korisnickog interfejsa

         // provera potpunosti, jedinstvenosti i ispravnosti podataka

         // snimanje podataka u bazu podataka

         // obavestavanje korisnika o uspesnosti snimanja
      }

private void btnFiltriraj_Click(object sender, EventArgs e)
      {
         // preuzimanje vrednosti za filtriranje

         // formiranje izraza za filtriranje

         // primena filtriranja nad preuzimanjem podataka

         // prikaz filtriranih vrednosti u gridu
      }
```

```
private void btnSvi_Click(object sender, EventArgs e)
      {
         // preuzimanje svih podataka iz baze podataka

         // prikazivanje podataka u gridu
      }

private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
      {
         // preuzimanje parametara stampe

         // provera podataka parametara stampe

         // selekcija tipa izvestaja

         // prikazivanje selektovanog izvestaja u report vieweru
      }
```

2. zadatak

```
      private void btnOdustani_Click(object sender, EventArgs e)
      {
         // brisanje sadrzaja kontrola
         txbPrezime.Text = "";
         txbIme.Text = "";
         cmbZvanje.Text = "";
         // s obzirom da ne moze da se isprazni, stavlja se na inicijalnu vrednost
         dtpDatumRodjenja.Value = DateTime.Now;
      }
```

3. zadatak

Prvo treba urediti programski kod forme, koja predstavlja klasu i ima uobičajene odeljke: atributi, konstruktor, nase metode, dogadjaji. Dodajemo komentare da bismo odvojili delove.

```
public partial class fNovaForma : Form
   {
      // globalne promenljive - atributi klase

      // konstruktor
      public fNovaForma()
      {
         InitializeComponent();
      }

      // nase procedure

      // dogadjaji
      private void fNovaForma_Load(object sender, EventArgs e)
      { . . .
```

U odeljku nase procedure dodajemo proceduru za brisanje sadrzaja kontrola:

```
private void IsprazniKontrole()
      {
         // brisanje sadrzaja kontrola
         txbPrezime.Text = "";
         txbIme.Text = "";
         cmbZvanje.Text = "";
         // s obzirom da ne moze da se isprazni, stavlja se na inicijalnu vrednost
         dtpDatumRodjenja.Value = DateTime.Now;
      }
```

Na mestu poziva na dogadjaju btnOdustani_Click postavljamo kod:

```
private void btnOdustani_Click(object sender, EventArgs e)
      {
         IsprazniKontrole();
      }
```

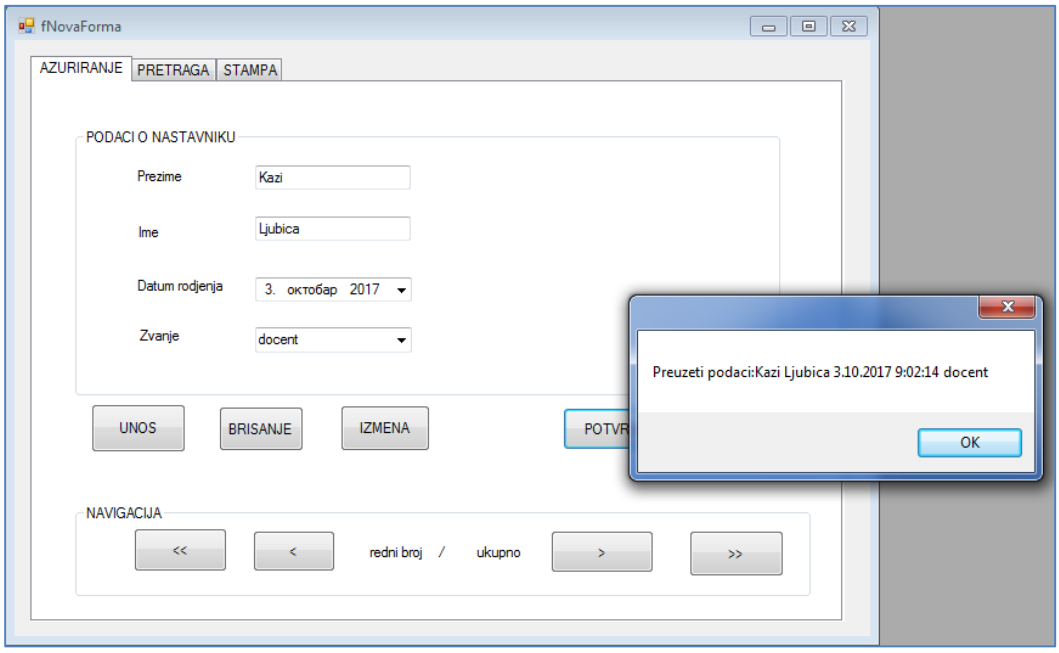
4. zadatak

```
private void btnPotvrdi_Click(object sender, EventArgs e)
      {
         // provera potpunosti podataka
         if ((txbPrezime.Text.Equals("")) || (txbPrezime.Text.Length == 0) || (txbPrezime.Text
== null))
         {
            txbPrezime.Focus();
            return;
         }

         if ((txbIme.Text.Equals("")) || (txbIme.Text.Length == 0) || (txbIme.Text == null))
         {
            txbIme.Focus();
            return;
         }

         if ((cmbZvanje.Text.Equals("")) || (cmbZvanje.Text.Length == 0) || (cmbZvanje.Text
== null))
         {
            cmbZvanje.Focus();
            return;
         }

         // preuzimanje vrednosti sa korisnickog interfejsa
         string prezime = txbPrezime.Text;
         string ime = txbIme.Text;
         DateTime datum = dtpDatumRodjenja.Value;
         string zvanje = cmbZvanje.Text;

         MessageBox.Show("Preuzeti podaci:" + prezime + " " + ime + " " + datum.ToString()
+ " " + zvanje);
```

```
        // provera jedinstvenosti podataka

        // provera ispravnosti podataka

        // snimanje podataka u bazu podataka

        // obavestavanje korisnika o uspesnosti snimanja
    }
```

Rešenje

Nakon uspešnog unosa svih podataka, prikazuje se message box sa unetim podacima, kao na sledećoj slici.

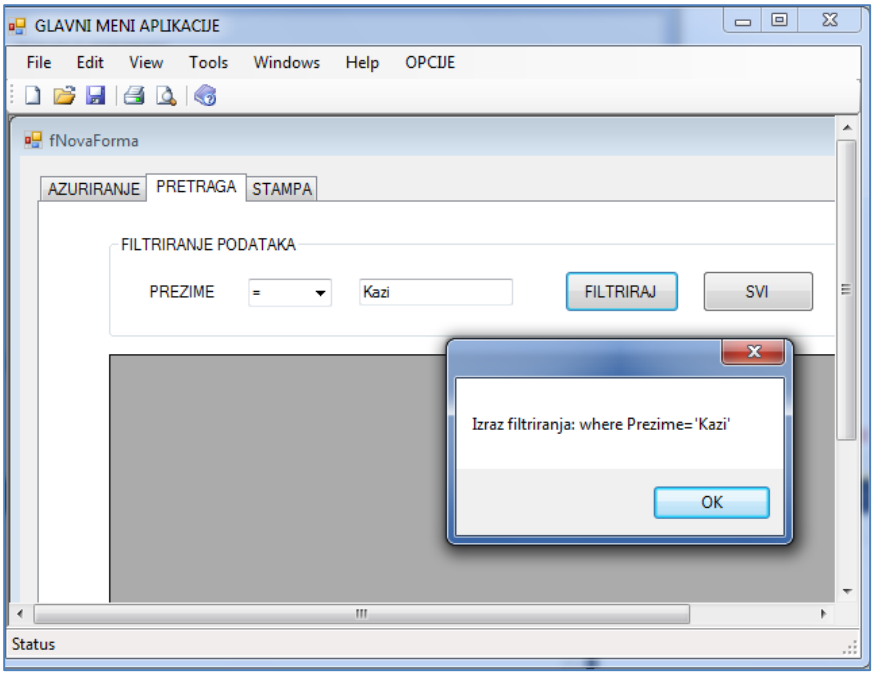


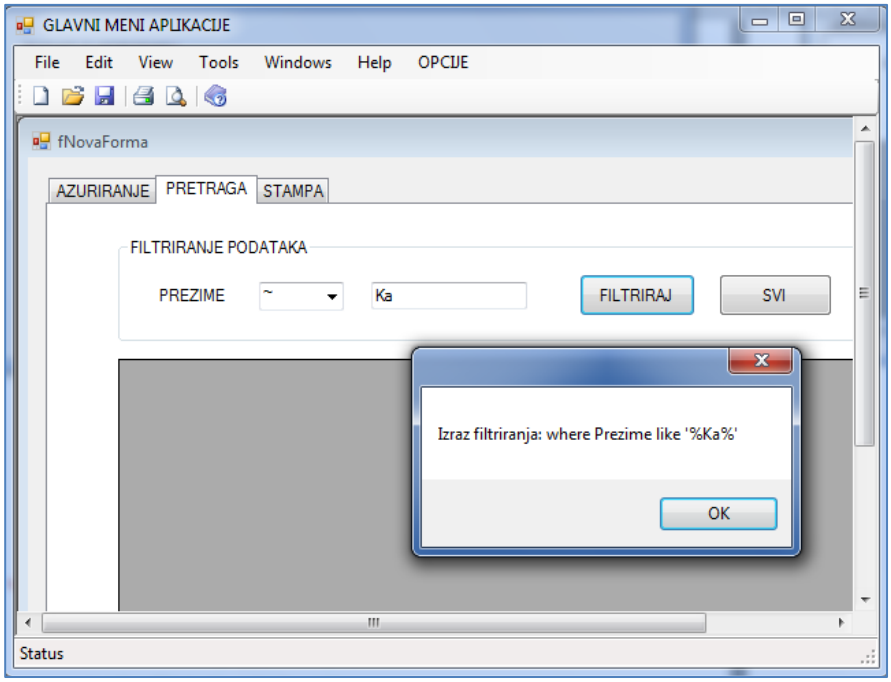
Slika 5.2. Rezultat izvršavanja u okviru 4. zadatka

5. zadatak

```
    private void btnFiltriraj_Click(object sender, EventArgs e)
    {
        // preuzimanje vrednosti za filtriranje
        string znakFiltriranja = cmbZnak.Text;
        string vrednostFiltriranja = txbFilter.Text;
        string izrazFiltriranja=" where ";

        // formiranje izraza za filtriranje
        switch (znakFiltriranja)
        {
            case "=":
                izrazFiltriranja = izrazFiltriranja  + "Prezime='" + vrednostFiltriranja + "'";
                break;
            case "~":
                izrazFiltriranja = izrazFiltriranja + "Prezime like '%" + vrednostFiltriranja + "%'";
                break;
```

```
    }
    MessageBox.Show("Izraz filtriranja:" + izrazFiltriranja);

    // primena filtriranja nad preuzimanjem podataka

    // prikaz filtriranih vrednosti u gridu
}
```





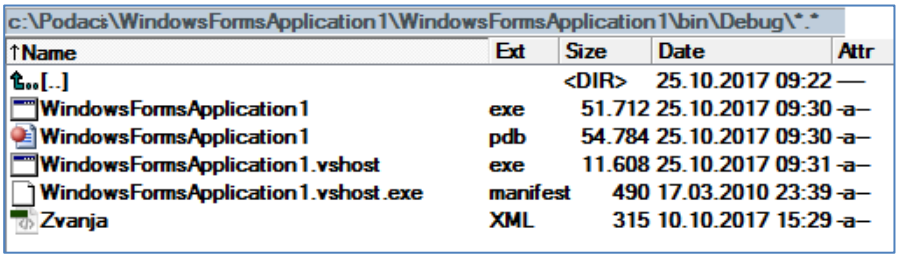
Slika 5.3. Rezultat izvršavanja u okviru 5. zadatka

6. zadatak

Kreiramo procedure za preuzimanje podataka iz XML i punjenje combo, u odeljku Nase procedure. Procedura PreuzmiPodatke iz XML vraća vrednost tipa DataSet, koji se kao klasa definiše u okviru gornjeg dela forme u using sekciji:

```
using System.Data;
```

Procedura za učitavanje podataka iz XML ima 2 parametra. Pravimo je da bude univerzalna, jer nekada može XML fajl biti i na nekoj konkretnoj putanji. Sada smeštamo XML fajl u okviru bin-debug foldera zajedno sa EXE fajlom aplikacije.



Slika 5.4. Snimanje XML fajla u okviru iste putanje gde je i EXE fajl

```
private DataSet PreuzmiPodatkeIzXML(string putanjaXMLFajla, string NazivXMLFajla)
      {
         DataSet dsPodaci = new DataSet() ;
         dsPodaci.ReadXml(putanjaXMLFajla + NazivXMLFajla);
         return dsPodaci;
      }

private void NapuniCombo(DataSet dsPodaciZaCombo)
      {
         int maxBroj = dsPodaciZaCombo.Tables[0].Rows.Count;
         for (int brojac=0; brojac<maxBroj; brojac++)
         {

cmbZvanje.Items.Add(dsPodaciZaCombo.Tables[0].Rows[brojac].ItemArray[1].ToString());
         };

      }
```

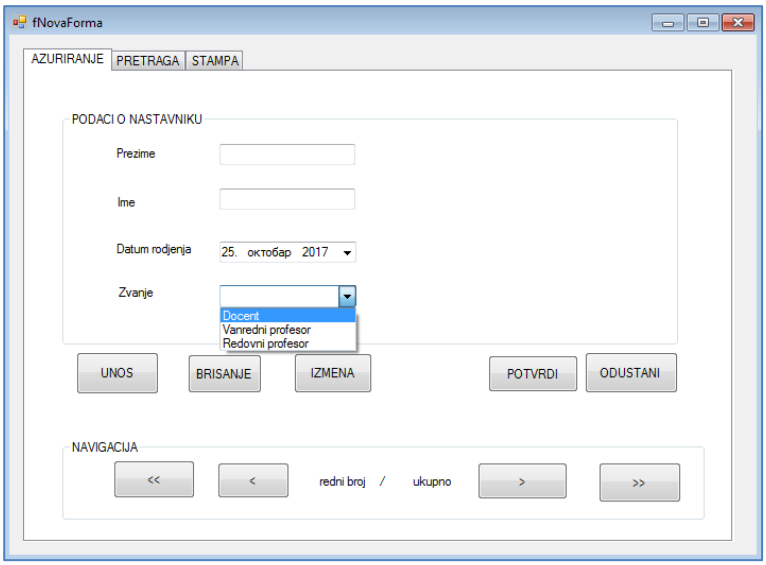
Pozivamo procedure ugnježdeno, u okviru form load događaja.

```
    private void fNovaForma_Load(object sender, EventArgs e)
      {
         // automatski postavljeno kada se postavi ReportViewer kontrola
         this.reportViewer1.RefreshReport();

         // ucitavanje podataka u combo
         NapuniCombo(PreuzmiPodatkeIzXML("", "Zvanja.XML"));
      }
```

Drugo rešenje – postepeno izvršavanje procedura:

```
    private void fNovaForma_Load(object sender, EventArgs e)
     {
        // automatski postavljeno kada se postavi ReportViewer kontrola
        this.reportViewer1.RefreshReport();

        // ucitavanje podataka u combo
        DataSet  dsPodaci = PreuzmiPodatkeIzXML("", "Zvanja.XML");
        NapuniCombo(dsPodaci);

     }
```

Rešenje



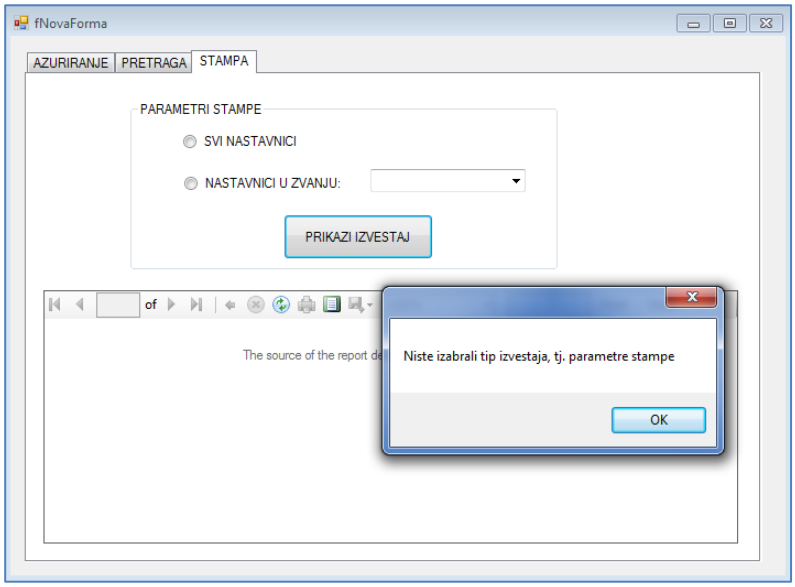
Slika 5.5. Prikaz napunjenog combo boxa, u izvršnom režimu rada

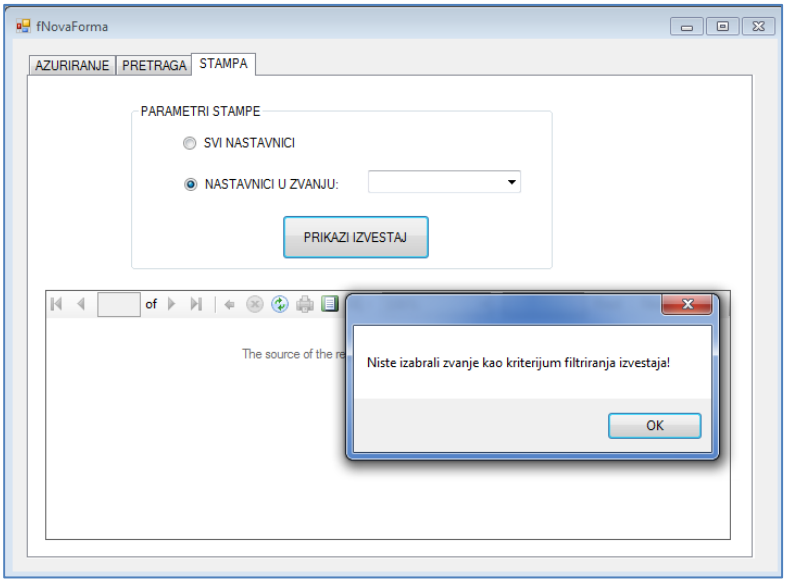
7. zadatak

```
    private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
    {

        // provera podataka parametara stampe
        if ((!rdbSviNastavnici.Checked) && (!rdbNastavniciUZvanju.Checked))
        {
            MessageBox.Show ("Niste izabrali tip izvestaja, tj. parametre stampe");
            return;
        }

        if ((rdbNastavniciUZvanju.Checked) & ((cmbZvanjaFilter.Text.Length == 0) ||
(cmbZvanjaFilter.Text.Equals("")) || (cmbZvanjaFilter.Text == null)))
        {
            MessageBox.Show("Niste izabrali zvanje kao kriterijum filtriranja izvestaja!");
            return;
        }

        // preuzimanje parametara stampe
```

```
        // selekcija tipa izvestaja

        // prikazivanje selektovanog izvestaja u report vieweru
    }
```

Rešenje





Slika 5.6. Rezultat izvršavanja 7. zadatka

8. zadatak

Dopuna combo boxa za zvanja na kartici za stampu za filtriranje, dodajemo u okviru napuni combo.

```
private void NapuniCombo(DataSet dsPodaciZaCombo)
    {
        int maxBroj = dsPodaciZaCombo.Tables[0].Rows.Count;
```

```
        for (int brojac=0; brojac<maxBroj; brojac++)
        {

cmbZvanje.Items.Add(dsPodaciZaCombo.Tables[0].Rows[brojac].ItemArray[1].ToString());

cmbZvanjaFilter.Items.Add(dsPodaciZaCombo.Tables[0].Rows[brojac].ItemArray[1].ToString()
);
        };

    }
```

Procedura za proveru vrednosti zvanja:

```
private bool ProveraVrednostiZvanja(string unetaVrednost, DataSet dsPodaci)
    {
        // deklaracija i inicijalizacija lokalnih promenljivih
        bool podaciIzDomena = false;
        int maxBroj = dsPodaci.Tables[0].Rows.Count;

        // provera vrednosti
        for (int brojac = 0; brojac < maxBroj; brojac++)
        {
            podaciIzDomena=unetaVrednost.Equals
(dsPodaci.Tables[0].Rows[brojac].ItemArray[1].ToString());
            if (podaciIzDomena) break;
        };

        return podaciIzDomena;
    }
```

Poziv procedure za proveru vrednosti zvanja u okviru dogadjaja click:

```
private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
    {

        // provera podataka parametara stampe
        if ((!rdbSviNastavnici.Checked) && (!rdbNastavniciUZvanju.Checked))
        {
            MessageBox.Show ("Niste izabrali tip izvestaja, tj. parametre stampe");
            return;
        }

        if ((rdbNastavniciUZvanju.Checked) & ((cmbZvanjaFilter.Text.Length == 0) ||
(cmbZvanjaFilter.Text.Equals("")) || (cmbZvanjaFilter.Text == null)))
        {
            MessageBox.Show("Niste izabrali zvanje kao kriterijum filtriranja izvestaja!");
            return;
        }
        if ((rdbNastavniciUZvanju.Checked) & (cmbZvanjaFilter.Text.Length > 0))
        {
            DataSet dsPodaciZaProveru = PreuzmiPodatkeIzXML("", "Zvanja.XML");
```

```
            bool podaciIzDomena = ProveraVrednostiZvanja(cmbZvanjaFilter.Text,
dsPodaciZaProveru);
             // if (!podaciIzDomena) -ovo nije dobro - bolje je u uslov
           //staviti POZITIVNO PITANJE
           // cesto se pogresi u programiranju kada se za uslov stavi
           //negacija!!!
             if (podaciIzDomena)
             {
                MessageBox.Show("Podatak za filtriranje JESTE iz dozvoljenog skupa
vrednosti!");
             }
             else
             {
                MessageBox.Show("Podatak za filtriranje NIJE iz dozvoljenog skupa vrednosti!");
             }
        }

        // preuzimanje parametara stampe

        // selekcija tipa izvestaja

        // prikazivanje selektovanog izvestaja u report vieweru
      }
```

6. RAD SA STRUKTURAMA PODATAKA I DATOTEKAMA

6.1. Nizovi, liste, tipizirane liste

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;
```

gde se tip podatka odnosi na tip podatka svakog elementa niza.

Inicijalizacija niza je potrebna kako bi se mogle dodeljivati vrednosti elementima niza. Promenljiva tipa niza pripada promenljivima tipa reference, pa je potrebno koristiti "new" ključnu reč kako bi se kreirala instanca niza. Nakon inicijalizacije, C# kompajler automatski postavlja sve elemente niza na podrazumevanu (default) vrednost u odnosu na tip podatka (npr. za integer je 0).

```
Primer: double[] balance = new double[10];
```

Dodela vrednosti element niza je navođenjem indeksa elementa niza:

```
balance[0] = 4500.0;
```

ili u toku deklaracije navođenjem vrednosti:

```
double[] balance = { 2340.0, 4523.69, 3421.0};
int [] marks = new int[5]  { 99,  98, 92, 97, 95};
int [] marks = new int[]  { 99,  98, 92, 97, 95};
```

Moguće je kopirati sadržaj jedne promenljive tipa niza u drugi, ali će nakon toga obe promenljive ukazivati na istu memorijsku lokaciju i sve vrednosti jedne promenljive automatski se odnose na iste vrednosti druge promenljive.

```
int[] score = marks;
```

Pristup element niza realizuje se navođenjem naziva niza i indeksa u okviru uglastih zagrada.

```
double salary = balance[9];
```

Postavljanje i čitanje vrednosti niza može da se realizuje primenom for ciklusa. Takođe, može da se koristi i foreach ciklus.

```
PRIMER:
        int []  n = new int[10]; /* n is an array of 10 integers */
        /* initialize elements of array n */
        for ( int i = 0; i < 10; i++ )
        {
          n[i] = i + 100;
        }
        /* output each array element's value */
        foreach (int j in n )
        {
```

```
        int i = j-100;
        Console.WriteLine("Element[{0}] = {1}", i, j);
    }
```

ARRAY LIST

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.

U nastavku je dat tabelarni prikaz najčešće korišćenih osobina (property) i metoda.

Property	Description
Capacity	Gets or sets the number of elements that the ArrayList can contain.
Count	Gets the number of elements actually contained in the ArrayList.
IsFixedSize	Gets a value indicating whether the ArrayList has a fixed size.
IsReadOnly	Gets a value indicating whether the ArrayList is read-only.
Item	Gets or sets the element at the specified index.

Methods
public virtual int Add(object value); Adds an object to the end of the ArrayList.
public virtual void AddRange(ICollection c); Adds the elements of an ICollection to the end of the ArrayList.
public virtual void Clear(); Removes all elements from the ArrayList.
public virtual bool Contains(object item); Determines whether an element is in the ArrayList.
public virtual ArrayList GetRange(int index, int count); Returns an ArrayList which represents a subset of the elements in the source ArrayList.
public virtual int IndexOf(object); Returns the zero-based index of the first occurrence of a value in the ArrayList

or in a portion of it.
public virtual void Insert(int index, object value); Inserts an element into the ArrayList at the specified index.
public virtual void InsertRange(int index, ICollection c); Inserts the elements of a collection into the ArrayList at the specified index.
public virtual void Remove(object obj); Removes the first occurrence of a specific object from the ArrayList.
public virtual void RemoveAt(int index); Removes the element at the specified index of the ArrayList.
public virtual void RemoveRange(int index, int count); Removes a range of elements from the ArrayList.
public virtual void Reverse(); Reverses the order of the elements in the ArrayList.
public virtual void SetRange(int index, ICollection c); Copies the elements of a collection over a range of elements in the ArrayList.
public virtual void Sort(); Sorts the elements in the ArrayList.
public virtual void TrimToSize(); Sets the capacity to the actual number of elements in the ArrayList.

PRIMER:

```
ArrayList al = new ArrayList();
al.Add(45);
al.Add(78);
al.Add(33);
al.Add(56);
al.Add(12);
al.Add(23);
al.Add(9);

Console.WriteLine("Capacity: {0} ", al.Capacity);
Console.WriteLine("Count: {0}", al.Count);

Console.Write("Content: ");
foreach (int i in al)
{
  Console.Write(i + " ");
}

Console.WriteLine();
Console.Write("Sorted Content: ");
al.Sort();
```

```
        foreach (int i in al)
        {
          Console.Write(i + " ");
        }
        Console.WriteLine();
        Console.ReadKey();
      }
    }
}
```

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.

Inicijalizacija promenljive tipa list

```
List<int> intList = new List<int>();
ili
IList<int> intList = new List<int>();
```

U prethodnom primeru List<int> je konkretna implementacija IList<T> interfejsa. List<T> implicitno može da primeni "type cast" nad IList<T>. Preporučuje se kreiranje promenljivih tipa interfejsa, kao što je IList<int>.

U nastavku je dat spisak najčešće korišćenih atributa i metoda.

PROPERTY/METHOD	**USAGE DESCRIPTION**
Items	Gets or sets the element at the specified index
Count	Returns the total number of elements exists in the List<T>
Method	Usage
Add	Adds an element at the end of a List<T>.
AddRange	Adds elements of the specified collection at the end of a List<T>. NAPOMENA: Ova metoda funkcioniše samo sa List<T>, a nije na raspolaganju za IList
BinarySearch	Search the element and returns an index of the element.
Clear	Removes all the elements from a List<T>.
Contains	Checks whether the speciied element exists or not in a List<T>.
Find	Finds the first element based on the specified predicate function.
Foreach	Iterates through a List<T>.
Insert	Inserts an element at the specified index in a List<T>.
InsertRange	Inserts elements of another collection at the specified index.
Remove	Removes the first occurence of the specified element.

PROPERTY/METHOD	USAGE DESCRIPTION
RemoveAt	Removes the element at the specified index.
RemoveRange	Removes all the elements that match with the supplied predicate function.
Sort	Sorts all the elements.
TrimExcess	Sets the capacity to the actual number of elements.
TrueForAll	Determines whether every element in theÂ List<T> matches the conditions defined by the specified predicate.

PRIMERI

Kreiranje

```
IList<int> intList = new List<int>();
intList.Add(10);
intList.Add(20);
intList.Add(30);
intList.Add(40);
IList<string> strList = new List<string>();
strList.Add("one");
strList.Add("two");
strList.Add("three");
strList.Add("four");
strList.Add("four");
strList.Add(null);
strList.Add(null);
IList<Student> studentList = new List<Student>();
studentList.Add(new Student());
studentList.Add(new Student());
studentList.Add(new Student());
IList<int> intList = new List<int>(){ 10, 20, 30, 40 };
IList<Student> studentList = new List<Student>() {
            new Student(){ StudentID=1, StudentName="Bill"},
            new Student(){ StudentID=2, StudentName="Steve"},
            new Student(){ StudentID=3, StudentName="Ram"},
            new Student(){ StudentID=1, StudentName="Moin"}
        };
-------------------------
```

Dodavanje elemenata jedne liste u drugu

```
IList<int> intList1 = new List<int>();
intList1.Add(10);
intList1.Add(20);
intList1.Add(30);
intList1.Add(40);
List<int> intList2 = new List<int>();
intList2.AddRange(intList1);
-------------------------
```

Pristup elementima

```
List<int> intList = new List<int>() { 10, 20, 30 };
```

```
intList.ForEach(el => Console.WriteLine(el));
---------
IList<int> intList = new List<int>() { 10, 20, 30, 40 };
foreach (var el in intList)
        Console.WriteLine(el);
---------
IList<int> intList = new List<int>() { 10, 20, 30, 40 };
int elem = intList[1]; // returns 20
---------
IList<int> intList = new List<int>() { 10, 20, 30, 40 };
for (int i = 0; i < intList.Count; i++)
        Console.WriteLine(intList[i]);
---------
static void Print(IList<string> list)
{
   Console.WriteLine("Count: {0}", list.Count);
   foreach (string value in list)
   {
      Console.WriteLine(value);
   }
}
---------------
```

Dodavanje elementa u listu

Koristimo insert metodu koja dodaje u List<T> element na određenoj lokaciji označenoj indeksom.
Insert() signature: *void Insert(int index, T item);*
PRIMER:

```
IList<int> intList = new List<int>(){ 10, 20, 30, 40 };
intList.Insert(1, 11);// inserts 11 at 1st index: after 10.
---------------
```

Uklanjanje elementa liste

Koriste se Remove() ili RemoveAt() metode.
Remove() signature: *bool Remove(T item)*
RemoveAt() signature: *void RemoveAt(int index)*
PRIMER:

```
IList<int> intList = new List<int>(){ 10, 20, 30, 40 };
intList.Remove(10); // removes the 10 from a list
intList.RemoveAt(2); //removes the 3rd element (index starts from 0)
```

6.2. Rad sa fajlovima i folderima

Da bi se koristile naredbe za rad sa fajlovima i folderima, potrebno je uključiti u projekat biblioteku klasa System.IO. Koriste se klase File i directory.

PRIMERI:

Kopiranje fajlova

```
string fileName = "test.txt";
        string sourcePath = @"C:\Users\Public\TestFolder";
        string targetPath =  @"C:\Users\Public\TestFolder\SubDir";
```

```
        // Use Path class to manipulate file and directory paths.
        string sourceFile = System.IO.Path.Combine(sourcePath, fileName);
        string destFile = System.IO.Path.Combine(targetPath, fileName);

        // To copy a folder's contents to a new location:
        // Create a new target folder, if necessary.
        if (!System.IO.Directory.Exists(targetPath))
        {
            System.IO.Directory.CreateDirectory(targetPath);
        }

        // To copy a file to another location and
        // overwrite the destination file if it already exists.
        System.IO.File.Copy(sourceFile, destFile, true);

        // To copy all the files in one directory to another directory.
        // Get the files in the source folder. (To recursively iterate through
        // all subfolders under the current directory, see
        // "How to: Iterate Through a Directory Tree.")
        // Note: Check for target path was performed previously
        //       in this code example.
        if (System.IO.Directory.Exists(sourcePath))
        {
            string[] files = System.IO.Directory.GetFiles(sourcePath);

            // Copy the files and overwrite destination files if they already exist.
            foreach (string s in files)
            {
                // Use static Path methods to extract only the file name from the path.
                fileName = System.IO.Path.GetFileName(s);
                destFile = System.IO.Path.Combine(targetPath, fileName);
                System.IO.File.Copy(s, destFile, true);
            }
        }
        else
        {
            Console.WriteLine("Source path does not exist!");
        }
```

Premeštanje fajlova

```
string sourceFile = @"C:\Users\Public\public\test.txt";
        string destinationFile = @"C:\Users\Public\private\test.txt";

        // To move a file or folder to a new location:
        System.IO.File.Move(sourceFile, destinationFile);

        // To move an entire directory. To programmatically modify or combine
        // path strings, use the System.IO.Path class.
        System.IO.Directory.Move(@"C:\Users\Public\public\test\", @"C:\Users\Public\private");
```

Brisanje fajlova

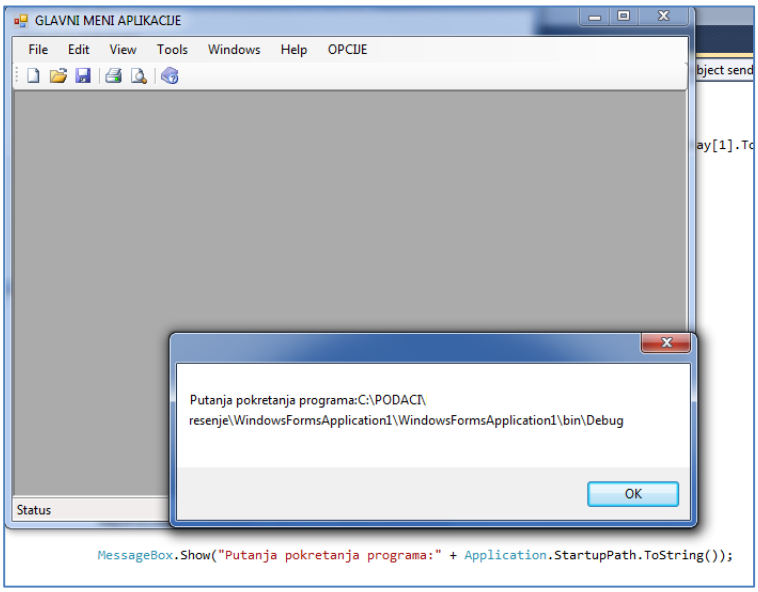
```
// Delete a file by using File class static method...
        if(System.IO.File.Exists(@"C:\Users\Public\DeleteTest\test.txt"))
        {
            // Use a try block to catch IOExceptions, to
            // handle the case of the file already being
            // opened by another process.
            try
            {
                System.IO.File.Delete(@"C:\Users\Public\DeleteTest\test.txt");
            }
            catch (System.IO.IOException e)
            {
                Console.WriteLine(e.Message);
                return;
            }
        }

        // ...or by using FileInfo instance method.
        System.IO.FileInfo fi = new System.IO.FileInfo(@"C:\Users\Public\DeleteTest\test2.txt");
        try
        {
            fi.Delete();
        }
        catch (System.IO.IOException e)
        {
            Console.WriteLine(e.Message);
        }

        // Delete a directory. Must be writable or empty.
        try
        {
            System.IO.Directory.Delete(@"C:\Users\Public\DeleteTest");
        }
        catch (System.IO.IOException e)
        {
            Console.WriteLine(e.Message);
        }
        // Delete a directory and all subdirectories with Directory static method...
        if(System.IO.Directory.Exists(@"C:\Users\Public\DeleteTest"))
        {
            try
            {
                System.IO.Directory.Delete(@"C:\Users\Public\DeleteTest", true);
            }

            catch (System.IO.IOException e)
            {
                Console.WriteLine(e.Message);
            }
        }
```

```
        // ...or with DirectoryInfo instance method.
        System.IO.DirectoryInfo di = new System.IO.DirectoryInfo(@"C:\Users\Public\public");
        // Delete this dir and all subdirs.
        try
        {
            di.Delete(true);
        }
        catch (System.IO.IOException e)
        {
            Console.WriteLine(e.Message);
        }
```

Osnovna putanja EXE fajla aplikacije

Ako ima potrebe da se očita putanja sa koje se pokreće EXE fajl aplikacije, može se koristiti:

```
Application.StartupPath
```



Slika 6.1. Testiranje očitavanje putanje programa prilikom pokretanja

6.3. Rad sa datotekama TXT i XML

Rad sa običnim tekstualnim datotekama zasniva se na primeni System.IO biblioteke klasa.

Čitanje TXT datoteke

```
String line;
try
{
//Pass the file path and file name to the StreamReader constructor
StreamReader sr = new StreamReader("C:\\Sample.txt");
```

```
//Read the first line of text
line = sr.ReadLine();

//Continue to read until you reach end of file
while (line != null)
{
//write the lie to console window
Console.WriteLine(line);
//Read the next line
line = sr.ReadLine();
}

//close the file
sr.Close();
Console.ReadLine();
}
catch(Exception e)
{
Console.WriteLine("Exception: " + e.Message);
}
  finally
{
Console.WriteLine("Executing finally block.");
}
```

Upis u TXT datoteku

```
try
{
//Pass the filepath and filename to the StreamWriter Constructor
StreamWriter sw = new StreamWriter("C:\\Test.txt");

//Write a line of text
sw.WriteLine("Hello World!!");

//Write a second line of text
sw.WriteLine("From the StreamWriter class");

//Close the file
sw.Close();
}
catch(Exception e)
{
Console.WriteLine("Exception: " + e.Message);
}
finally
{
Console.WriteLine("Executing finally block.");
}
```

Posebno se može naznačiti kodni raspored:

```
StreamWriter sw = new StreamWriter("C:\\Test1.txt", true, Encoding.ASCII);
```

Rad sa XML datotekama

Za rad sa XML možemo koristiti biblioteku klasa System.Xml, odnosno klasu XMLDocument i XMLNode.

Primer učitavanja XML fajla

Koristimo Load metodu za ucitavanje sadrzaja XML fajla na osnovu naziva fajla, odnosno ucitavanje direktno iz stringa samog XML-a putem LoadXML.

```
XmlDocument doc = new XmlDocument();
doc.PreserveWhitespace = true;
try { doc.Load("booksData.xml"); }
catch (System.IO.FileNotFoundException)
{
   doc.LoadXml("<?xml version=\"1.0\"?> \n" +
   "<books xmlns=\"http://www.contoso.com/books\"> \n" +
   "  <book genre=\"novel\" ISBN=\"1-861001-57-8\" publicationdate=\"1823-01-28\"> \n" +
   "    <title>Pride And Prejudice</title> \n" +
   "    <price>24.95</price> \n" +
   "  </book> \n" +
   "  <book genre=\"novel\" ISBN=\"1-861002-30-1\" publicationdate=\"1985-01-01\"> \n" +
   "    <title>The Handmaid's Tale</title> \n" +
   "    <price>29.95</price> \n" +
   "  </book> \n" +
   "</books>");
}
```

Primer čitanja iz XML čvorova, koji su osnovni elementi XML dokumenta:

```
using System;
using System.IO;
using System.Xml;

public class Sample {

  public static void Main() {

    XmlDocument doc = new XmlDocument();
    doc.LoadXml("<book ISBN='1-861001-57-5'>" +
              "<title>Pride And Prejudice</title>" +
              "<price>19.95</price>" +
              "</book>");

    XmlNode root = doc.FirstChild;

    //Display the contents of the child nodes.
    if (root.HasChildNodes)
    {
```

```
    for (int i=0; i<root.ChildNodes.Count; i++)
    {
      Console.WriteLine(root.ChildNodes[i].InnerText);
    }
   }
  }
}
```

Takođe, možemo koristiti i jednostavnije metode koje se nadovezuju na DataSet: ReadXML i WriteXML.

```
DataSet dsPodaci = new DataSet();

dsPodaci.ReadXML(PutanjaINazivFajlaXML);
```

6.4. Primeri sa strukturama podataka i datotekama

6.4.1. Zadaci

1. Realizovati da se putanja XML fajla za punjenje combo uzima relativno na osnovu putanje EXE fajla koja se ocitava iz Application.StartupPath.

Realizovati punjenje combo nakon preuzimanja vrednosti iz Zvanja.XML tako da se:

2. Vrednosti preuzmu iz XML u DataSet pa zatim u objekat tipa tipizirane liste List <string>.

3. Iz objekta tipa tipizirane liste List <string> napuniti combo nazivima zvanja koristeci foreach naredbu.

6.4.2. Rešenja

1. zadatak

Preuzimanje podataka iz XML, dodajemo odvajanje foldera i naziva fajla

```
    private DataSet PreuzmiPodatkeIzXML(string putanjaXMLFajla, string NazivXMLFajla)
      {
         DataSet dsPodaci = new DataSet() ;
         dsPodaci.ReadXml(putanjaXMLFajla + "\\" + NazivXMLFajla);
         return dsPodaci;
      }
```

Učitavanje forme, dodajemo Application.StartupPath kod poziva procedure za učitavanje XML

```
      private void fNovaForma_Load(object sender, EventArgs e)
      {
         // automatski postavljeno kada se postavi ReportViewer kontrola
         this.reportViewer1.RefreshReport();

         // 1. nacin - ucitavanje podataka u combo
        // NapuniCombo(PreuzmiPodatkeIzXML("", "Zvanja.XML"));

         // 2. nacin

         DataSet dsPodaci = PreuzmiPodatkeIzXML(Application.StartupPath, "Zvanja.XML");
         List<string> listaNazivaZvanja = new List<string>();
         listaNazivaZvanja = PreuzmiPodatkeIzDataSetuListu(dsPodaci);
         NapuniComboIzListeSaForEach(listaNazivaZvanja);

         MessageBox.Show("Putanja pokretanja programa:" +
Application.StartupPath.ToString());
      }
```

2. zadatak

Preuzimanje vrednosti iz XML u DataSet – ranije već korišćeno

```
private DataSet PreuzmiPodatkeIzXML(string putanjaXMLFajla, string NazivXMLFajla)
      {
         DataSet dsPodaci = new DataSet() ;
         dsPodaci.ReadXml(putanjaXMLFajla + NazivXMLFajla);
         return dsPodaci;
      }
```

Prebacivanje podataka iz DataSet u Listu

```
private List<string> PreuzmiPodatkeIzDataSetuListu(DataSet dsPodaci)
      {
         List<string> listaNazivaZvanja = new List<string>();

         for (int brojac = 0; brojac < dsPodaci.Tables[0].Rows.Count; brojac++)
            {

listaNazivaZvanja.Add(dsPodaci.Tables[0].Rows[brojac].ItemArray[1].ToString());
            }

         return listaNazivaZvanja;
      }
```

3. zadatak

Punjenje combo iz liste primenom for

```
private void NapuniComboIzListe(List<string> objListaNaziva)
      {
         int maxBroj = objListaNaziva.Count;

         for (int brojac = 0; brojac < maxBroj; brojac++)
         {
            cmbZvanje.Items.Add(objListaNaziva[brojac].ToString());
            cmbZvanjaFilter.Items.Add(objListaNaziva[brojac].ToString());
         };

      }
```

Punjenje combo iz liste sa foreach:

```
private void NapuniComboIzListeSaForEach(List<string> objListaNaziva)
      {
         foreach (string naziv in objListaNaziva)
         {
            cmbZvanje.Items.Add(naziv);
            cmbZvanjaFilter.Items.Add(naziv);
         };
      }
```

KORIŠĆENJE

```
private void fNovaForma_Load(object sender, EventArgs e)
{
   // automatski postavljeno kada se postavi ReportViewer kontrola
   this.reportViewer1.RefreshReport();

   // 1. nacin - ucitavanje podataka u combo
  // NapuniCombo(PreuzmiPodatkeIzXML("", "Zvanja.XML"));

   // 2. nacin

   DataSet dsPodaci = PreuzmiPodatkeIzXML("", "Zvanja.XML");
   List<string> listaNazivaZvanja = new List<string>();
   listaNazivaZvanja = PreuzmiPodatkeIzDataSetuListu(dsPodaci);
   NapuniComboIzListeSaForEach(listaNazivaZvanja);
}
```

7. OBJEKTNO-ORJENTISANO PROGRAMIRANJE u C#

7.1. Teorija

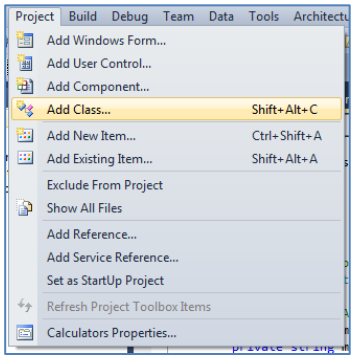
DEFINICIJA

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.



Slika 7.1. Opcija za dodavanje klase u projekat

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 dodati odgovarajuci DLL u References i pisati:

```
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.

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 KONCEPTI OBJEKTNO-ORJENTISANOG PROGRAMIRANJA

Osnovni koncepti 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.

```
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 varijante 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

Polimorfizam omogućuje da se objekti klasa 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 elementi 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.

7.2. Primeri primene elemenata objektno-orjentisanog programiranja

7.2.1. Zadaci

1. zadatak
   Kreirati klasu za obradu gresaka i primeniti na formi korisničkog interfejsa. Načini:
   a) standardna klasa
   b) klasa koja nasleđuje Exception i koja se instancira naredbom throw.

2. 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 i on 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.

3. zadatak
   Implementirati kalkulator primenom tipizirane liste karaktera.

7.2.2. Rešenja

1. Zadatak

PRVI NAČIN

Kreiramo u okviru Windows Forms projekta novi element tipa klase (Project – add class) za rad sa obradom gresaka.

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

```
namespace WindowsFormsApplication1
{
   public class clsExceptionIzvanDomena
   {
      // atributi
      private string pPorukaGreske = "";

     // public property

      public string PorukaGreske
      {
         get
         {
            return pPorukaGreske;
         }
         set
         {
            if (this.pPorukaGreske != value)
               this.pPorukaGreske = value;
         }
      }

      // konstruktor
      public clsExceptionIzvanDomena()
      {
         pPorukaGreske = "Podaci su izvan domena!";
      }

      // privatne metode

      // javne metode

   }
}
```

Napomena: U ovoj klasi nije potrebno da postoji set metoda, jer je tekst poruke definisan u konstruktoru i ne treba da dozvolimo da se menja prilikom korišćenja objekta ove klase. Ipak, ovde je data i set metoda radi ilustracije načina pisanja.

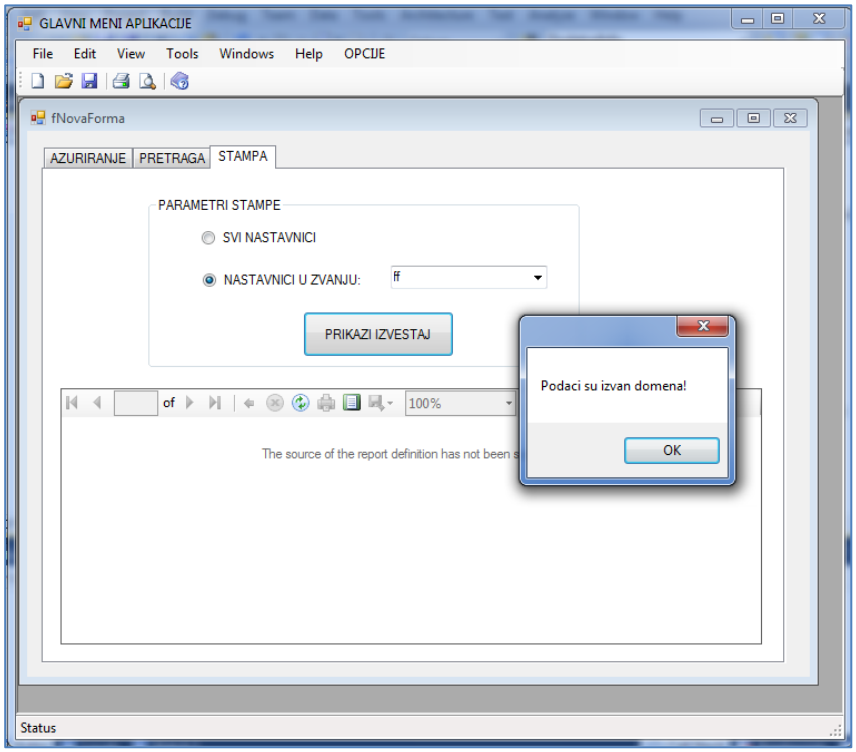
KORIŠĆENJE KLASE NA FORMI KORISNIČKOG INTERFEJSA

Menjamo programski kod u okviru tastera za prikaz izvestaja:

```
private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
      {

         // provera podataka parametara stampe
         if ((!rdbSviNastavnici.Checked) && (!rdbNastavniciUZvanju.Checked))
         {
            MessageBox.Show ("Niste izabrali tip izvestaja, tj. parametre stampe");
```

```
            return;
        }

        if ((rdbNastavniciUZvanju.Checked) & ((cmbZvanjaFilter.Text.Length == 0) ||
(cmbZvanjaFilter.Text.Equals("")) || (cmbZvanjaFilter.Text == null)))
        {
            MessageBox.Show("Niste izabrali zvanje kao kriterijum filtriranja izvestaja!");
            return;
        }

        if ((rdbNastavniciUZvanju.Checked) & (cmbZvanjaFilter.Text.Length > 0))
        {
            DataSet dsPodaciZaProveru = PreuzmiPodatkeIzXML("", "Zvanja.XML");
            bool podaciIzDomena = ProveraVrednostiZvanja(cmbZvanjaFilter.Text,
dsPodaciZaProveru);

            // bolje je u uslov staviti POZITIVNO PITANJE
            if (podaciIzDomena)
            {
                MessageBox.Show("Podatak za filtriranje JESTE iz dozvoljenog skupa
vrednosti!");

            }
            else // situacija kada su podaci van domena
            {
                clsExceptionIzvanDomena objGreskaIzvanDomena = new
clsExceptionIzvanDomena();
                MessageBox.Show(objGreskaIzvanDomena.PorukaGreske);
            }
        }

        // preuzimanje parametara stampe

        // selekcija tipa izvestaja

        // prikazivanje selektovanog izvestaja u report vieweru
    }
```

REZULTAT:



Slika 7.2. Rezultat izvršavanja 1. zadatka na prvi način

DRUGI NAČIN – PRIMENOM NASLEĐIVANJA STANDARDNE KLASE EXCEPTION

Preklapanje metoda (overloading)

```
        private DataSet PreuzmiPodatkeIzXML(string putanjaXMLFajla, string
NazivXMLFajla)
        {
            DataSet dsPodaci = new DataSet() ;
            dsPodaci.ReadXml(putanjaXMLFajla + "\\" + NazivXMLFajla);
            return dsPodaci;
        }

        private DataSet PreuzmiPodatkeIzXML(string NazivXMLFajla)
        {
            DataSet dsPodaci = new DataSet();
            dsPodaci.ReadXml(NazivXMLFajla);
            return dsPodaci;
        }
```

Metoda za proveru domena, ali koja radi throw exception

```
private void ProveraVrednostiZvanjaPrimenomException(string unetaVrednost, DataSet
dsPodaci)
        {
            // deklaracija i inicijalizacija lokalnih promenljivih
            bool podaciIzDomena = false;
```

```
            int maxBroj = dsPodaci.Tables[0].Rows.Count;

            // provera vrednosti
            for (int brojac = 0; brojac < maxBroj; brojac++)
            {
                podaciIzDomena =
unetaVrednost.Equals(dsPodaci.Tables[0].Rows[brojac].ItemArray[1].ToString());
                if (podaciIzDomena) break;
            };

            if (!podaciIzDomena)
            {
                throw new clsCustomException("PodaciIzvanDomena");
            }

        }
```

Klasa clsCustomException – kreirati je u okviru projekta WindowsFormsApplication1

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

namespace WindowsFormsApplication1
{
   public class clsCustomException: Exception
    {
        // ENKAPSULACIJA - SAKRIVANJE IMPLEMENTACIJE -  ATRIBUTI SU PRIVATNI
        private string mTipGreske;
        private string mTekstGreske;

        // NADJACAVANJE SVOJSTVA
        public override string Message
        {
            get
            {
                return mTekstGreske;

                // NE IDE PORUKA OD OPSTE KLASE EXCEPTION
                //return base.Message;
            }
        }

        // KONSTRUKTOR SA PARAMETROM
        public clsCustomException(string noviTipGreske)
        {
            mTipGreske = noviTipGreske;
            mTekstGreske = DajTekstGreske();

        }

        // ENKAPSULACIJA - SAKRIVANJE IMPLEMENTACIJE -  METODA JE PRIVATNA
        private string DajTekstGreske()
```

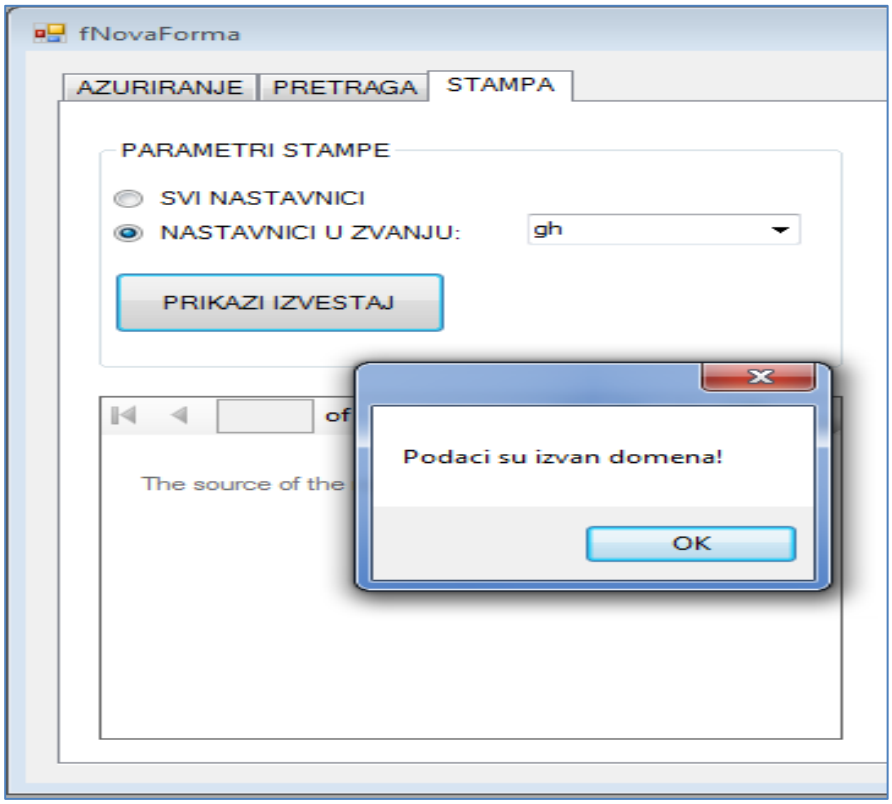
```
        {
            string tekstGreske = "";

            switch (mTipGreske)
            {
                case "OperacijaBezOperanda":
                    tekstGreske = "Operacija je bez operanda!";
                    break;
                case "PodaciIzvanDomena":
                    tekstGreske = "Podaci su izvan domena!";
                    break;
            }

            return tekstGreske;
        }

    } // klasa
} // namespace
```

Primena na tasteru „PrikaziIzvestaj"

```
private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
        {

            // provera podataka parametara stampe
            if ((!rdbSviNastavnici.Checked) && (!rdbNastavniciUZvanju.Checked))
            {
                MessageBox.Show ("Niste izabrali tip izvestaja, tj. parametre stampe");
                return;
            }

            if ((rdbNastavniciUZvanju.Checked) & ((cmbZvanjaFilter.Text.Length == 0) ||
(cmbZvanjaFilter.Text.Equals("")) || (cmbZvanjaFilter.Text == null)))
            {
                MessageBox.Show("Niste izabrali zvanje kao kriterijum filtriranja
izvestaja!");
                return;
            }

            if ((rdbNastavniciUZvanju.Checked) & (cmbZvanjaFilter.Text.Length > 0))
            {
                DataSet dsPodaciZaProveru = PreuzmiPodatkeIzXML("Zvanja.XML");

              // 2. NACIN - primena exception
                try
                {
                    // unutar ove procedure se uradi "throw exception:" ako ima uslova
za to
                    ProveraVrednostiZvanjaPrimenomException(cmbZvanjaFilter.Text,
dsPodaciZaProveru);
                }
                catch (clsCustomException greska) // ovde se sacekuje exception
                {
```

```
                // ovde se procesira odgovor na exception ako nastane
                MessageBox.Show(greska.Message);
            }
        }
        // preuzimanje parametara stampe
        // selekcija tipa izvestaja
        // prikazivanje selektovanog izvestaja u report vieweru
    }
```

REZULTAT:



Slika 7.3. Rezultat izvršavanja 1. zadatka na drugi način

2. zadatak

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.

REŠENJE:

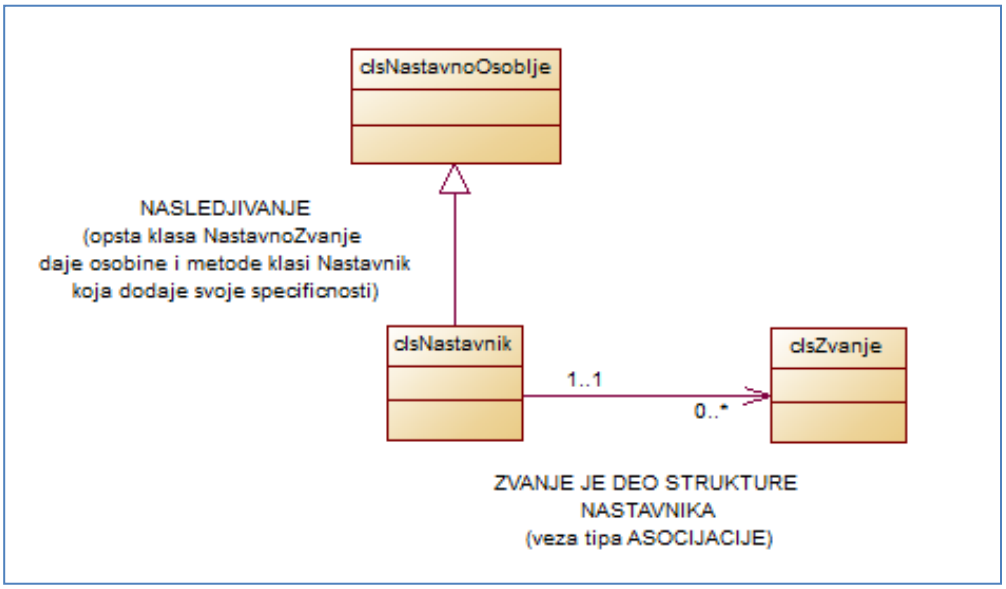
NastavnoOsoblje.XML

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:



Slika 7.4. Klase u realizaciji 2. zadatka

Klasa clsZvanje

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

namespace KlasePodataka
{
    public class clsZvanje
    {
        // atributi
        private int pSifra;
        private string pNaziv;

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

```
        }

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

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


        // privatne metode

        // javne metode
        public string DajPodatke()
        {
            return this.pSifra + " " + this.pNaziv;
        }


    }
}
```

Klasa clsNastavnoOsoblje

– sadrži objekat klase clsZvanje i DateTime atribut, omogućava redefinisanje metode DajPodatke prilikom nasleđivanja, jer sadrži reč virtual

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

namespace KlasePodataka
{
    public class clsNastavnoOsoblje
    {
        // atributi
        private string pPrezime;
        private string pIme;
        private DateTime pDatumVremeRodjenja;
```

```
        // public property

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

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


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

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

        // privatne metode

        private string DajDatum()
        {
```

```
            string Datum="";
            Datum = this.pDatumVremeRodjenja.Day.ToString() + "." +
this.pDatumVremeRodjenja.Month.ToString() + "." +
this.pDatumVremeRodjenja.Year.ToString() + ".";

            return Datum;
        }

        // public metode

        public virtual string DajPodatke()
        {
            return this.pPrezime + " " + this.pIme + " " + this.DajDatum();
        }


    }
}
```

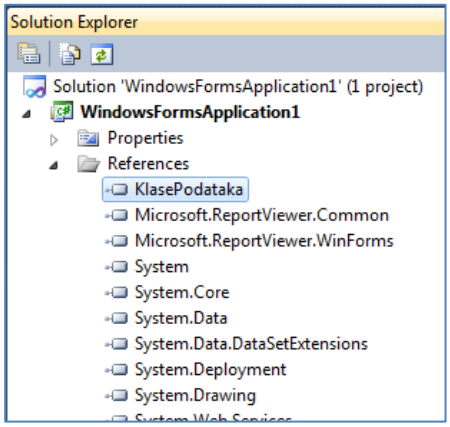
Klasa clsNastavnik

nasleđuje klasu clsNastavnoOsoblje i redefiniše metodu DajPodatke naredbom override.

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

namespace KlasePodataka
{
    public class clsNastavnik: clsNastavnoOsoblje
    {
        // atributi
        private clsZvanje pZvanje;

        // public property
        public clsZvanje 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()
        {
            return base.DajPodatke () + " " +  pZvanje.DajPodatke ();
        }
    }
}
```

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

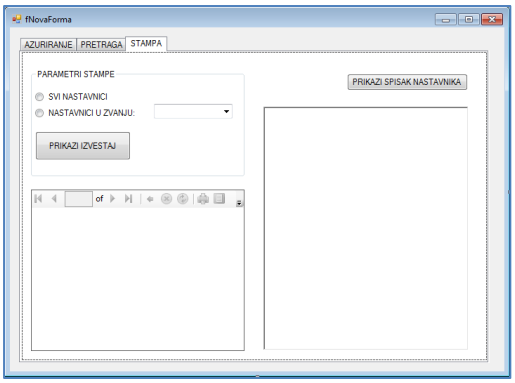


Slika 7.5. Povezana biblioteka klasa KlasePodataka.dll u okviru WindowsForms aplikacije

Ubacujemo using na početku forme:

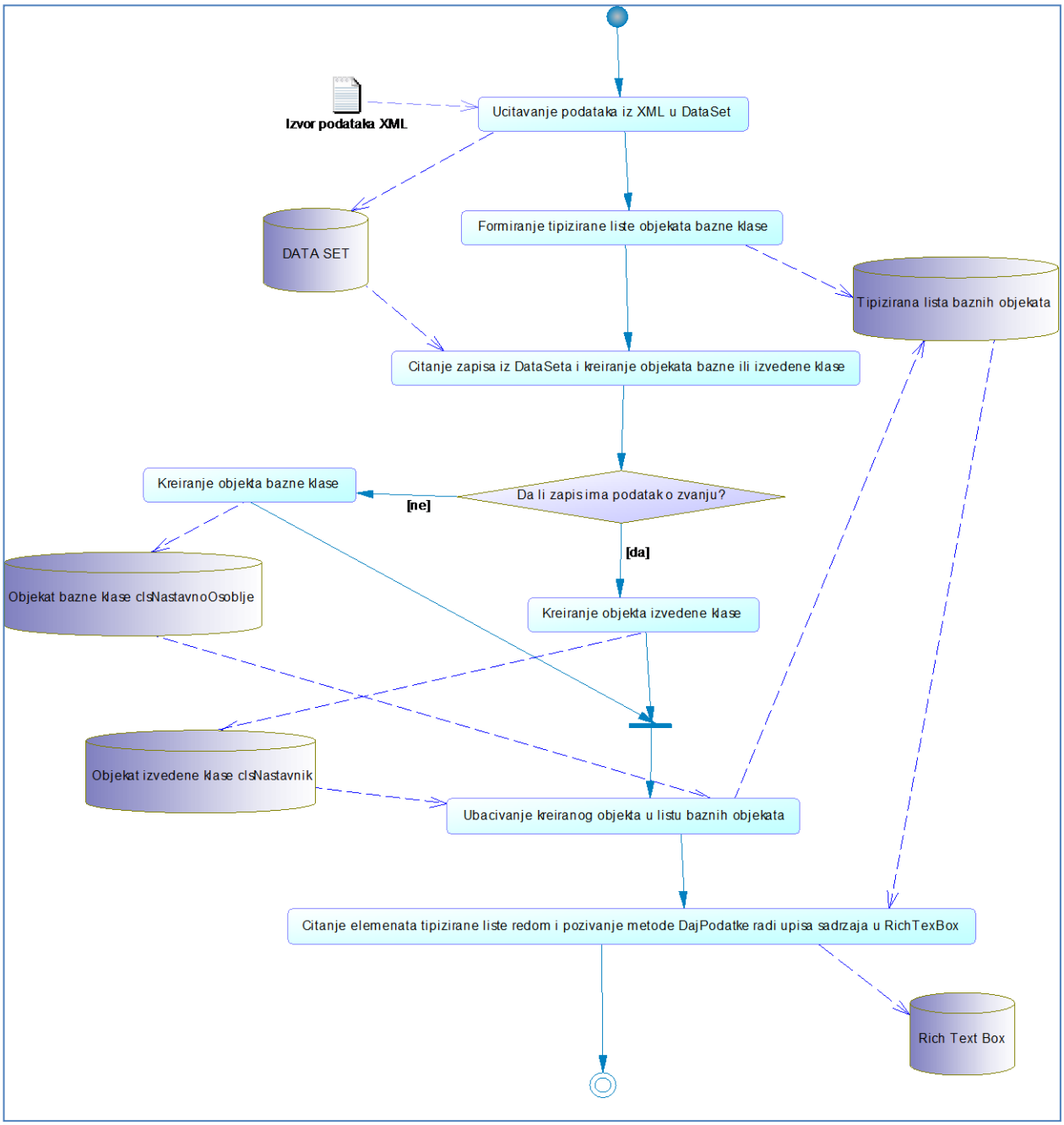
```
using KlasePodataka;
```

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



Slika 7.6. Postavljen Rich text box radi testiranja.

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



Slika 7.7. Algoritam izvršavanja programskog koda kojim se puni Rich Text Box podacima iz XML-a

DataSet nakon ucitavanja 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:

```
private void btnPrikaziSpisakNastavnika_Click(object sender, EventArgs e)
        {
            // PROMENLJIVE
            List<clsNastavnoOsoblje> objListaNastavnogOsoblja = new
List<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;
```

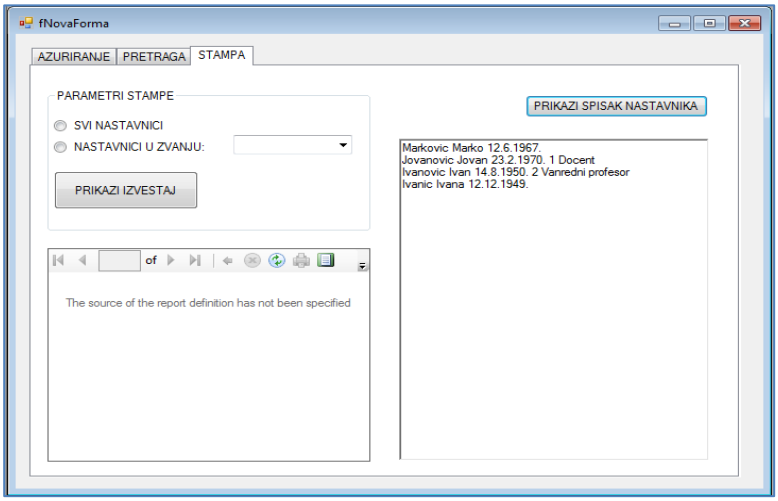
```
            for (int brojac = 0; brojac < maxBrojZapisa; brojac++)
            {
                // preuzimanje podataka iz DataSeta
                pomPrezime =
dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[0].ToString();
                pomIme =
dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[1].ToString();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

                if
(dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[3].ToString().Equals(""))
                {
                    objNastavnoOsoblje = new clsNastavnoOsoblje();
                    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].ToString());

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

                    // instanciramo objekat tipa nastavnika
                    objNastavnik = new clsNastavnik();
                    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(SifraZvanja))
            {
                pomNazivZvanja =
dsPodaciZvanja.Tables[0].Rows[brojac].ItemArray[1].ToString();
                break;
            }
        }


        return pomNazivZvanja;
    }
```

REZULTAT:



Slika 7.8. Rezultat izvršavanja zadatka kojim se ilustruje polimorfizam

8. HEURISTIČKA UPUTSTVA I KONVENCIJE ZA OBLIKOVANJE PROGRAMSKOG KODA

- Preporučljivo je kod uvođenja nove promenljive realizovati kompletnu definiciju, tj. deklaraciju i inicijalizaciju vrednosti.
- Uobičajeno je da kada promenljiva ima više reči, ne piše se odvojeno niti odvojeno crticama, već se piše spojeno, a svaka reč počinje velikim slovom, ostala su mala slova
- Razlikuje se samo situacija kada prva reč počinje malim slovom, a ostale reči počinju velikim slovom. To su 2 standardna načina pisanja:
    - 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….
- Privatni atributi klase, prema konvencijama, imaju prefix m_ ili p, a njegov par je public property bez prefiksa
- Mnemonički nazivi – prva tri slova odnose se na tip objekta, a nastavak je značenje primene objekta, npr. frmLogin = forma za Logovanje, txbPrezime = text box za unos prezimena, clsOsoba = klasa Osoba, objOsoba = objekat klase Osoba.
- Nazivi fajlova su takvi da se prvo navodi semantička grupa, a nakon toga konkretna funkcionalnost: frmMestoUnos, frmMestoTabelarni …zbog sortiranja da sve što se odnosi na mesto bude zajedno.
- Pisanje naredbi jedna ispod druge daje preglednost i čitljivost.
- Postavljanje komentara za pojedine blokove unapređuje čitljivost. Čak je preporučljivo pre pisanja koda planirati algoritam komentarima, pa tek onda postavljati kod.
- Preporučljivo je programski kod organizovati kroz procedure i funkcije koje realizuju atomarne operacije, čijim kombinovanjem u pozivu se dobijaju složenije funkcije. Npr. Treba odvojiti u posebne operacije učitavanje i prikazivanje.
- Uobičajeno je ugnježdeno pozivanje procedura, bez posebnih pomoćnih promenljivih, npr: NapuniCombo(UcitajPodatke(“Podaci.XML”));

- Preporučljivo je da u uslovu u okviru uslovnog grananja pitanje bude postavljeno kao pozitivno, tj. DA LI NEŠTO JESTE i da prvi odgovor bude rešen kroz blok naredbi za odgovor TRUE, a else da se odnosi na odgovor FALSE.
- Preporučljivo je da ako funkcija vraća bool bude nazvana kao pitanje, npr. DaLiJeSvePopunjeno, a ne ProveraPopunjenosti.
- Preporučljivo je da ukoliko funkcija vraća bool inicijalna vrednost interne promenljive bude false, pa zatim sledi blok naredbi za proveru ispunjenosti uslova i eventualne promene vrednosti te promenljive, a zatim naredba return nad tom promenljivom.
- Preporučljivo je da procedura dobija sve vrednosti putem parametara, osim kada su u pitanju vrednosti privatnih atributa klase, koje se mogu dobiti putem this.pNazivAtributa ili this.m_NazivAtributa. Nije dobro da procedura radi sa globalnim promenljivama koje su vidljive u proceduri (osim atributa klase), jer se može desiti nekontrolisana izmena zajedničkih podataka. Zato sve što ulazi u proceduru je preporučljivo da ulazi kroz parametre.

9. ELEMENTI VIŠESLOJNE SOFTVERSKE ARHITEKTURE

Savremeni razvoj softvera odlikuju karakteristike modularnosti koja omogućava:

- Razvoj složenih aplikacija u timovima
- Ponovnu iskoristivost programskog koda u sličnim softverskim rešenjima
- Povećanje brzine razvoja
- Povećanje kvaliteta realizovanog koda, prvenstveno u smanjenju grešaka i povećanju pouzdanosti rešenja.

Modularizacija omogućava korišćenje gotovih funkcija realizovanih od strane istog tima ili uključivanje modula realizovanih od strane drugih proizvođača. Tako se mogu koristiti biblioteke klasa ili konsultovati udaljeni web servisi koji daju realizaciju pojedinih softverskih funkcija, a kojima se pristupa kao servisima (uslugama) koje su stalno on-line dostupne.

Osnovna objektno-orjentisana softverska arhitektura obuhvata 4 sloja, sa odgovarajućim klasama u slojevima:

PREZENTACIONI SLOJ	Korisnički interfejs sa programskim kodom Klase prezentacione logike
SLOJ POSLOVNE LOGIKE	Klase sa poslovnim objektima Klase sa poslovnim pravilima Klase koje definišu radne tokove
SERVISNI SLOJ	Web servisi kao on-line biblioteke klasa Klase za mapiranje između slojeva
SLOJ ZA RAD SA PODACIMA	Klase za rad sa bazom podataka Klase za rad sa datotekama različitih formata Baze podataka, datoteke različitih formata

U okviru ovog praktikuma naglasak će biti na klasama za rad sa bazom podataka. Razlikujemo:

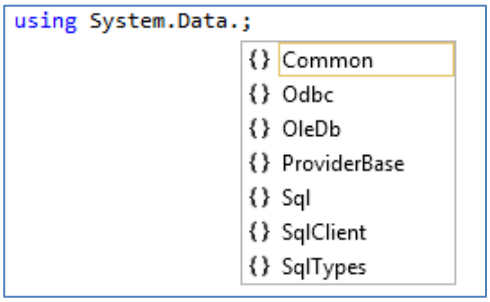
- Standardne biblioteke klasa za rad sa bazom podataka određenog tipa. Npr. za rad sa MS SQL Server bazom podataka imamo System.Data.SQLClient biblioteku.
- Sopstvene biblioteke klasa za rad sa bazom podataka, koje kreiramo kao novi projekat tipa ClassLibrary.
- Automatski generisane biblioteke klasa, npr. povezivanjem sa wizardom za kreiranje biblioteke klasa nad MS SQL Server bazom podataka možemo kreirati klase koje odgovaraju Entity Framework načinu kreiranja klasa.

Takođe, biće prikazan i najvažniji deo programskog koda implementacije osnovnih funkcija poslovne aplikacije u okviru korisničkog interfejsa.

10. PROGRAMSKI KOD IMPLEMENTACIJE WINDOWS APLIKACIJE

10.1. Memorijske kolekcije za rad sa podacima

Biblioteka klasa System.Data sadrži podbiblioteke za rad sa različitim klasama za rad sa podacima koji mogu biti memorijske strukture, a takođe sadrže i podršku za rad sa podacima u datotekama (npr. XML) i bazama podataka različitih DBMS (Sistema za upravljanje bazom podataka – npr. MS Access, MS SQL Server i druge – pristupom putem ODBC – Open Database Connectivity standarda).



System.Data kao biblioteka klasa sadrži univerzalne klase za rad sa podacima, nezavisne od izvora podataka koji mogu biti datoteke ili baze podataka različitih DBMS.

Najčešće korišćene univerzalne klase kao memorijske kolekcije podataka:

- DataTable - memorijska kolekcija podataka u formi tabele.
  - Sastoji se iz kolekcija Columns (elementi su DataColumn), Rows (elementi su DataRow)
  - Važne metode: Select, AcceptChanges
- DataSet – predstavlja memorijsku kolekciju podataka (kesirani podaci) koja se može sastojati iz više tabela.
  - Sastoji se od kolekcije Tables (element je DataTable)
  - Važne metode: ReadXML (učitava sadržaj XML u DataSet), Copy (kopira strukturu i podatke u drugi DataSet), Clone (pravi kopiju DataSeta), WriteXML (snima sadržaj DataSet-a u XML).
- Data Reader – slično kao DataSet.

10.2. Rad sa bazama podataka

10.2.1. Standardne klase za rad sa bazom podataka

U Visual Studio NET podržan je rad sa raznim sistemima za upravljanje bazom podataka. Za potrebe rada sa:

- Microsoft Access bazom podataka - koristi se System.Data.OleDB biblioteka sa odgovarajućim klasama: OleDBCommand, OleDBDataAdapter, OleDBConnection, OleDBDataReader, OleDBTransaction,
- Microsoft SQL Server bazom podataka – koristi se System.Data.SQLClient biblioteka sa odgovarajućim klasama: SqlConnection, SqlCommand, SqlDataAdapter, SqlDataReader, SqlTransaction,

- Ostali DBMS (npr. MySQL, ORACLE) – koristi se System.Data.ODBC - Open Database connectivity standard.

Bazi podataka se obraćamo:

- Konektovanjem na bazu podataka
- Upitima za izdvajanje podataka tipa select
- Upitima za ažuriranje podataka, tipa insert into, update, delete.

Da bismo se konektovali na bazu podataka, potrebno je podesiti parametre pristupa- naziv i vrstu sistema za upravljanje bazom podataka, naziv baze podataka, kao i opcione: korisnicko ime i sifru za pristup bazi podataka. Svi ovi podaci se u odredjenoj formi nalaze u tzv. stringu konekcije. Standardna klasa SQLConnection kao osnovni parameter prilikom instanciranja objekta ima upravo string konekcije, kako bi nakon toga, u okviru naredbe "Open" mogla da uputi bazi podataka sve potrebne parametre i konektuje se na bazu podataka.

Upiti za rad sa bazom podataka mogu biti napisani u programskom kodu u korisničkom interfejsu ili u klasama, ali je najbolji način da se pišu u okviru stored procedura ili pogleda (kada su select upiti bez parametara u pitanju) u okviru samih baza podataka, a da se iz programskog koda korisničkog interfejsa ili klasa pozivaju pozivima stored procedura ili pogleda.

10.2.2. Kreiranje sopstvenih klasa za rad sa bazom podataka

Možemo kreirati bibliteku klasa, kako bismo olakšali rad sa podacima iz baze podataka i pojednostavili programski kod. Na ovaj način se programski kod premešta iz korisničkog interfejsa u sloj za rad sa podacima. Biblioteka klasa za rad sa podacima može sadržati:

- Programski kod koji se odnosi na primenu standardnih klasa za rad sa određenim DBMS i sva podešavanja, SQL upite i slično. Tu je zajednički kod koji je tehnološki vezan i semantički određen.
- Programski kod klasa može biti univerzalan tako da se odvoji:
    - tehnološki vezan programski kod (odnosi se na primenu standardnih klasa za određeni DBMS) – TEHNOLOŠKA BIBLIOTEKA KLASA. Ova univerzalna biblioteka klasa može da se koristi u više semantički različitih projekata.
    - semantički deo koji se odnosi na konkretnu problematiku, tj. naziv baze podataka, tabele itd. To je BIBLIOTEKA KLASA KOJU BI MOGLI NAZVATI KLASE PODATAKA. Na ovaj način se u okviru semantičkih klasa pozivaju tehnološke klase radi konkretne implementacije. Preporučljivo je da semantičke klase sadrže pozive na stored procedure, a ne da sadrže SQL upite. Klase koje se nalaze u ovoj biblioteci klasa su slične po nazivu sa tabelama u bazi podataka. Za svaku tabelu imamo barem 3 klase – klasa pojedinac, klasa Lista pojedinaca i klasa DB koja sadrži CRUD metode (create – insert, read – select, update, delete), a koje se zasnivaju na pozivu stored procedura ili prosleđivanju SQL upita.

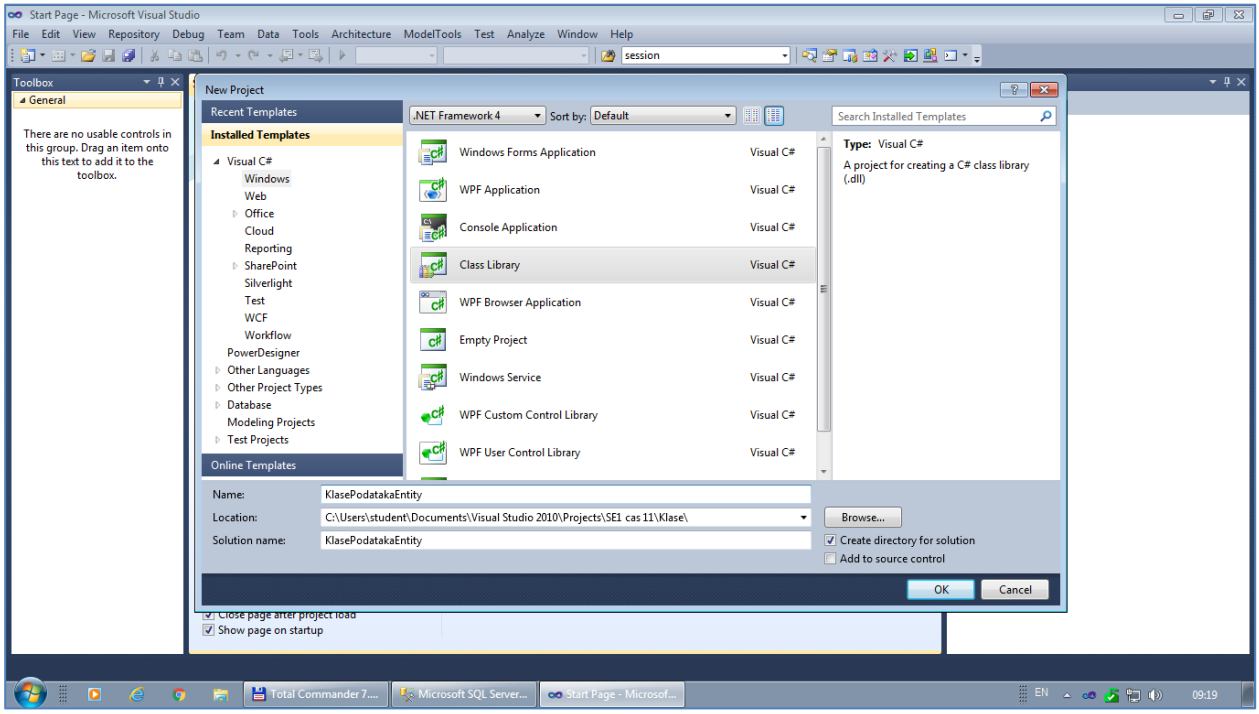
10.2.3. Generisanje i korišćenje Entity Framework klasa za rad sa bazom podataka

Biblioteka za rad sa bazom podataka može biti i generisana u određenom standardnom formatu i organizaciji. Pojedini proizvođači razvojnih okruženja ili softverske firme razvijaju svoje standarde i formulišu ih kao framework, pa se za konkretna rešenja mogu generisati elementi na osnovu tih opštih obrazaca i koristiti gotovi delovi, uz manje mogućnosti prilagođavanja konkretnim zahtevima.

U nastavku će biti dat opis postupka kreiranja biblioteke klasa po Microsoftovom standard nazvanom Entity Framework, koji kreira skupove klasa na osnovu tabela i ostalih elemenata baze podataka.

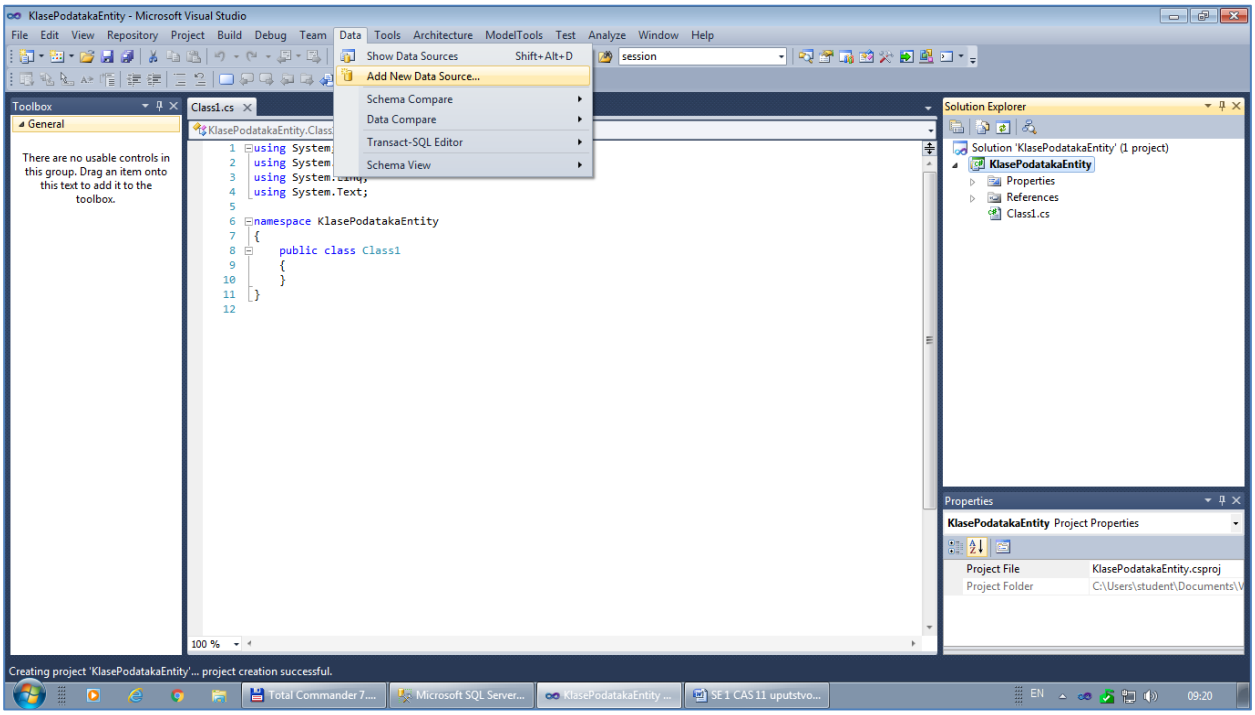
Postupak je opisan tako da se generišu klase u okviru posebne biblioteke klasa tj. Class Library, ali se može pokrenuti generisanje i u okviru projekta Windows Forms.

KREIRAMO NOVI PROJEKAT TIPA CLASS LIBRARY, NAZIV KlasePodatakaEntity.



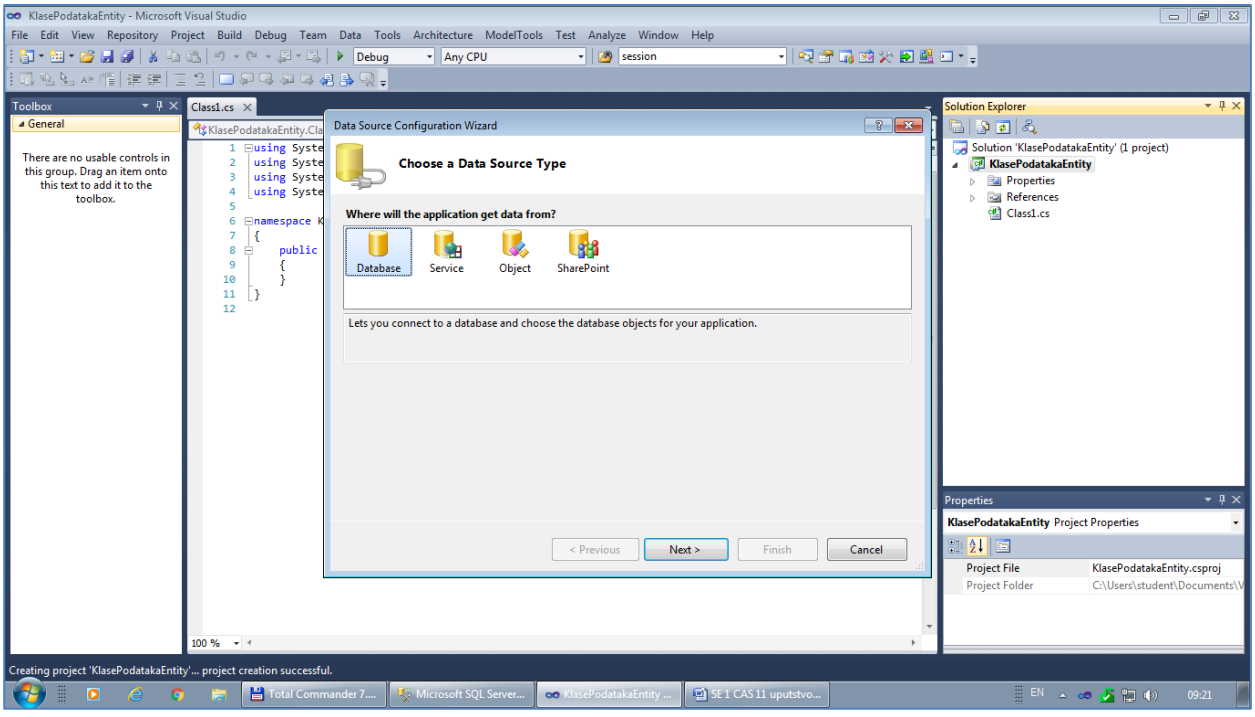
Slika 10.1. Kreiranje projekta tipa Class Library, naziv: KlasePodatakaEntity

Dodajemo novi datasource (Data -> Add new data source).



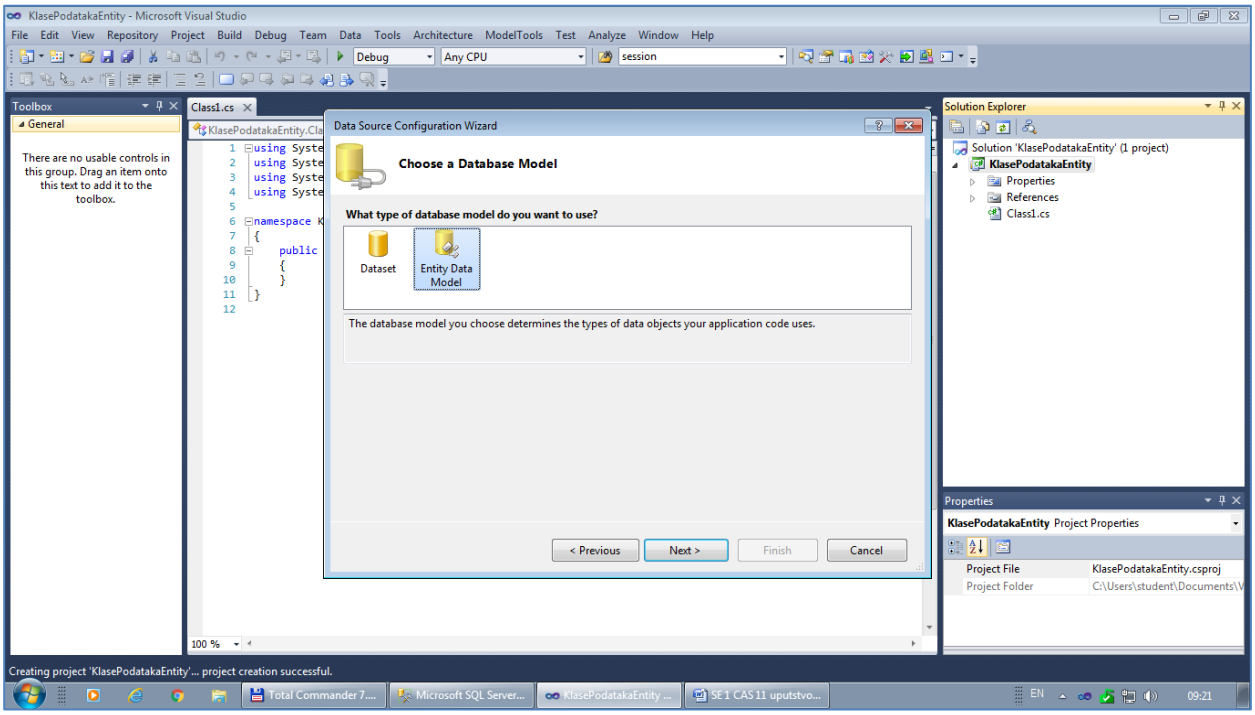
Slika 10.2. Dodavanje novog izvora podataka

Biramo izvor podataka DATABASE.



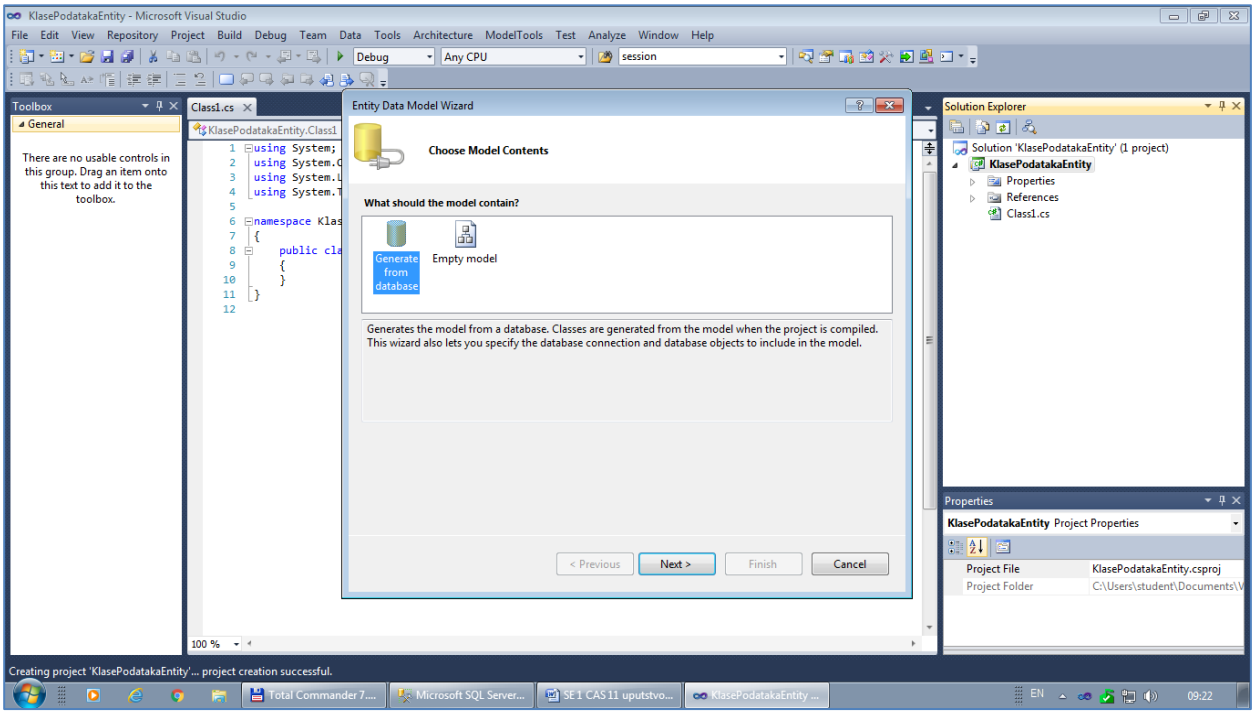
Slika 10.3. Izbor DATABASE za izvor podataka

Biramo entity data model.



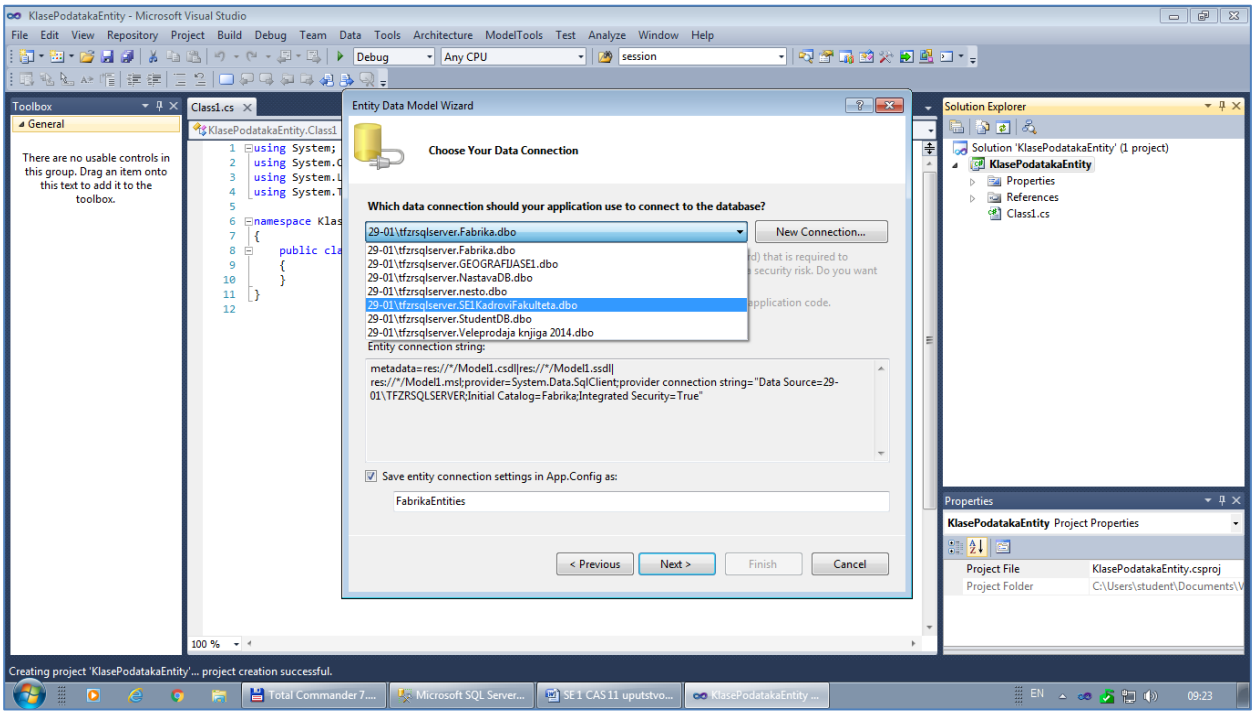
Slika 10.4. Izbor opcije "ENTITY DATA MODEL".

Biramo Generate from database.



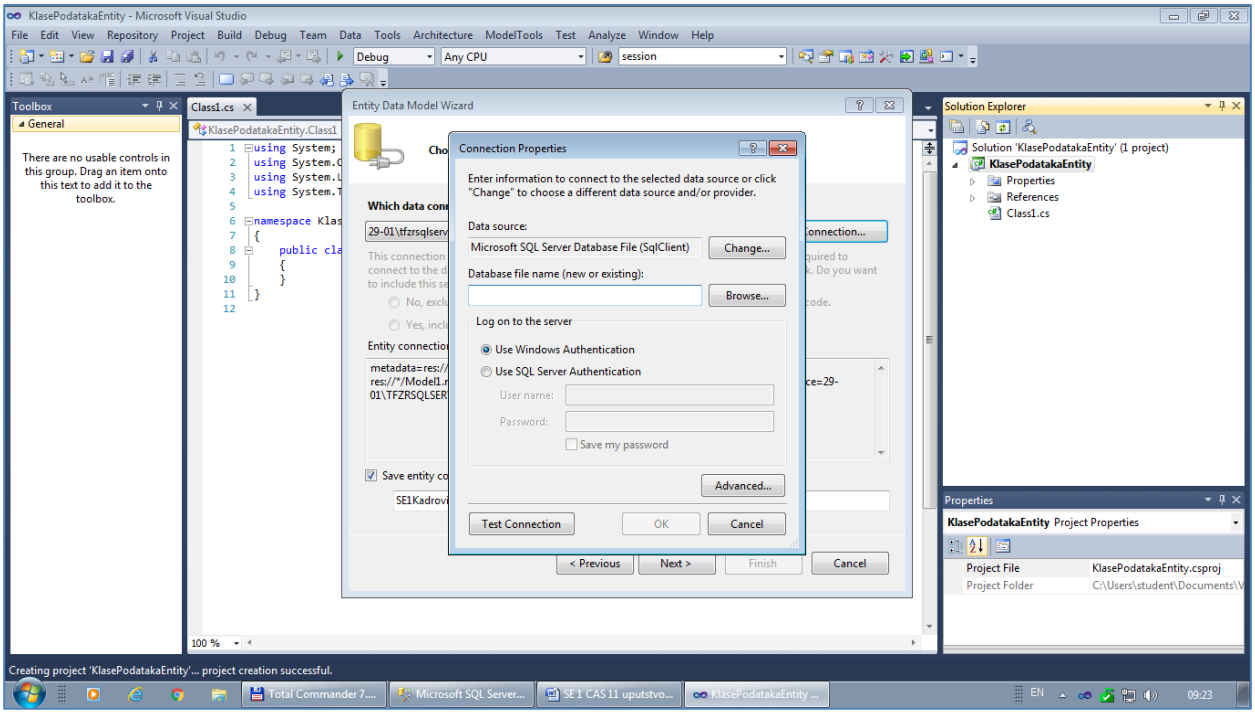
Slika 10.5. Izbor opcije "Generate from database".

Biramo konekciju do baze podataka, ako postoji.



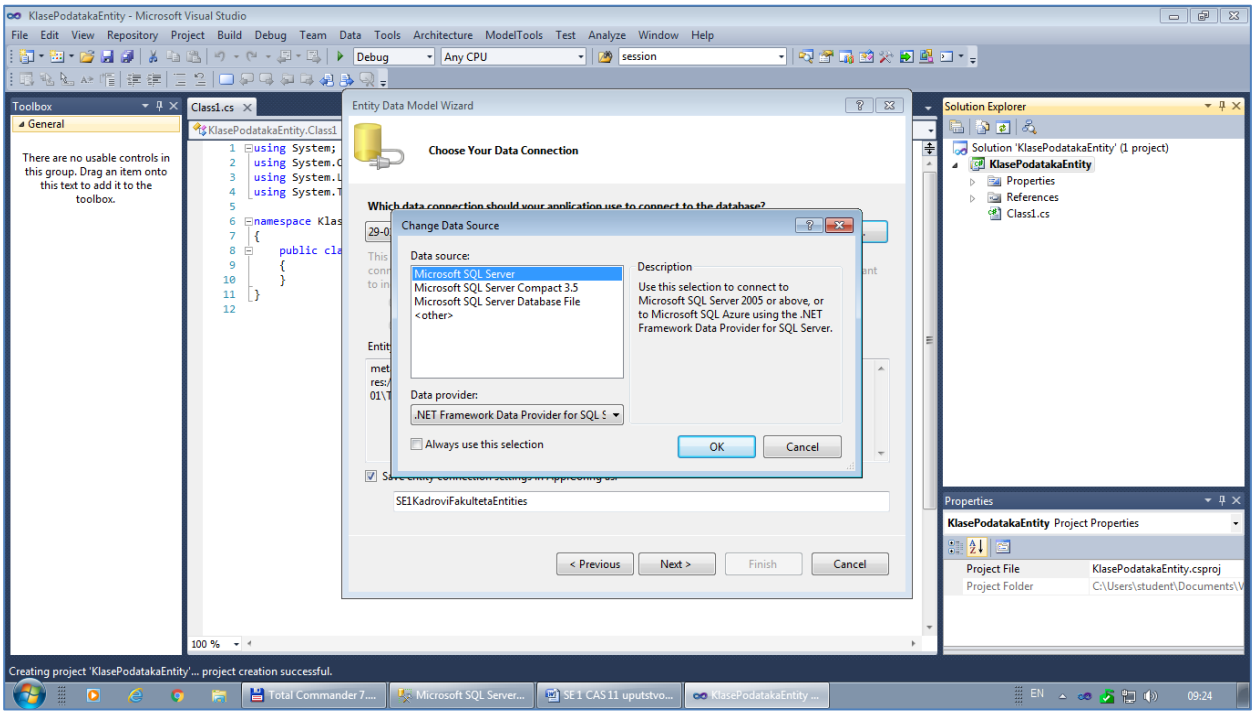
Slika 10.6. Izbor konekcije do baze podataka sa liste, ako postoji odgovarajuća

Ako ne postoji, kreiramo NEW CONNECTION.



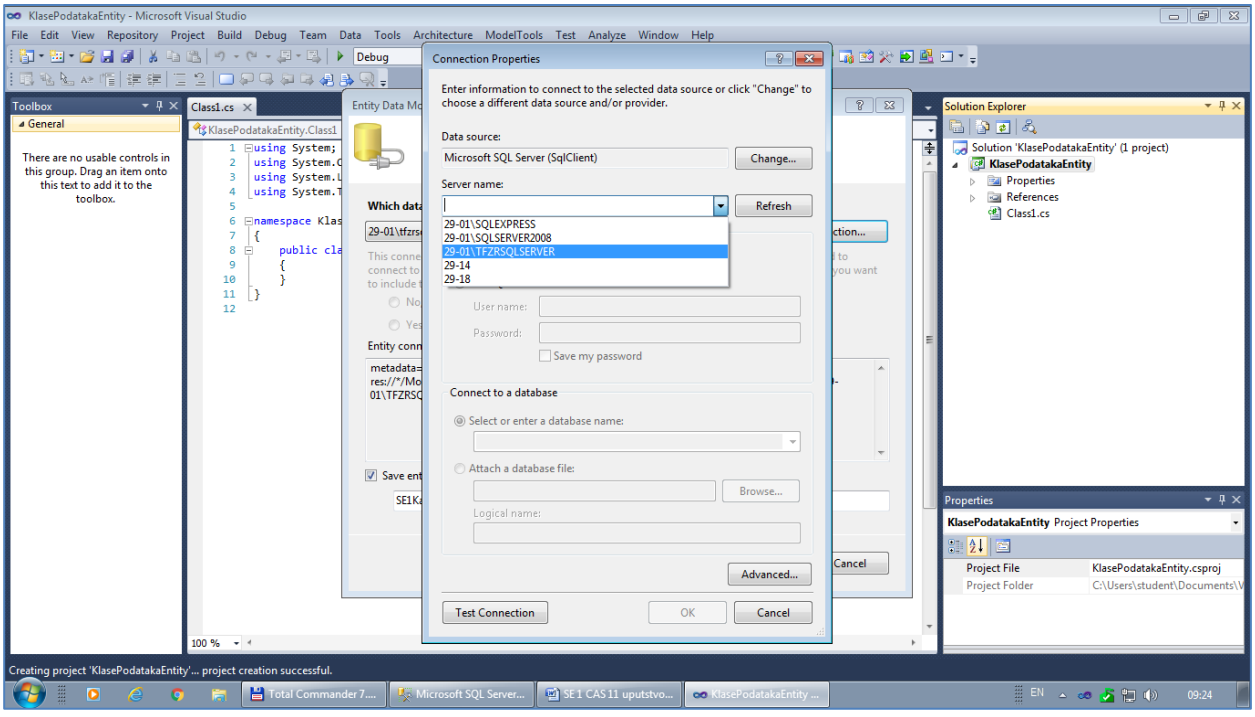
Slika 10.7. Kreiranje nove konekcije do baze podataka

Biramo change, da odaberemo data source – Microsoft SQL Server.



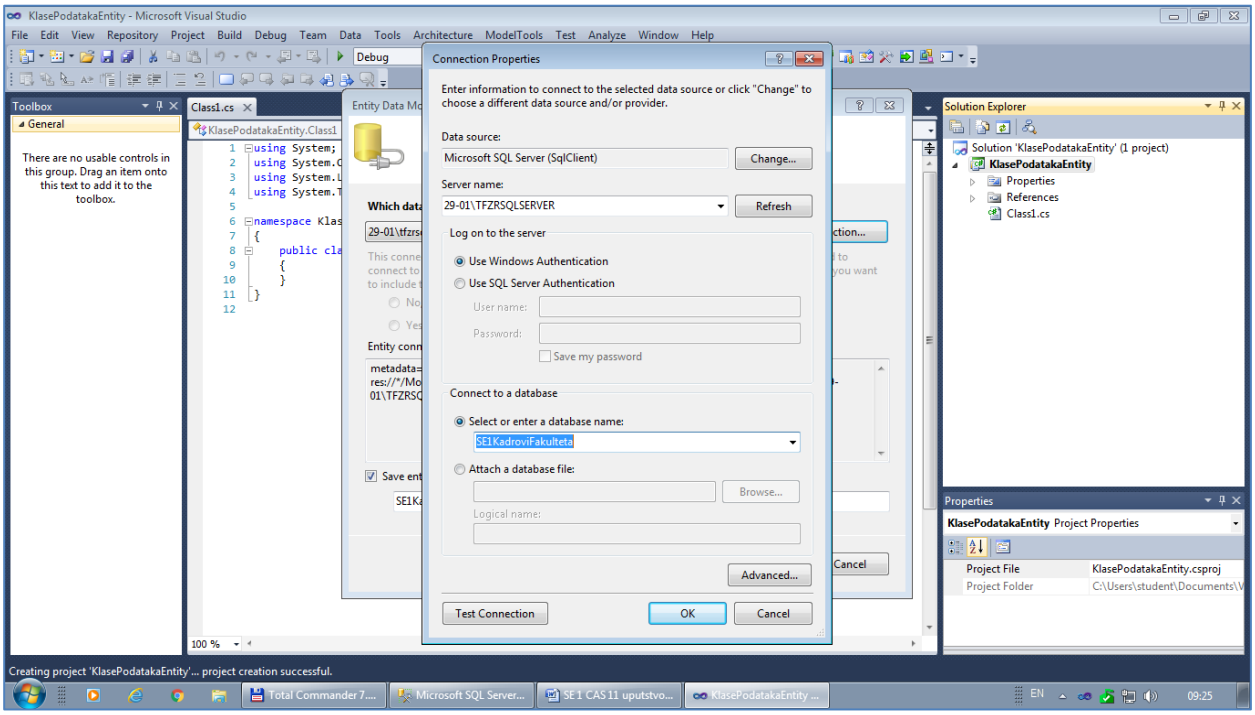
Slika 10.8. Biranje opcije Microsoft SQL Server

Biramo instancu sql server DBMS-a ili unosimo, ako nije na spisku.



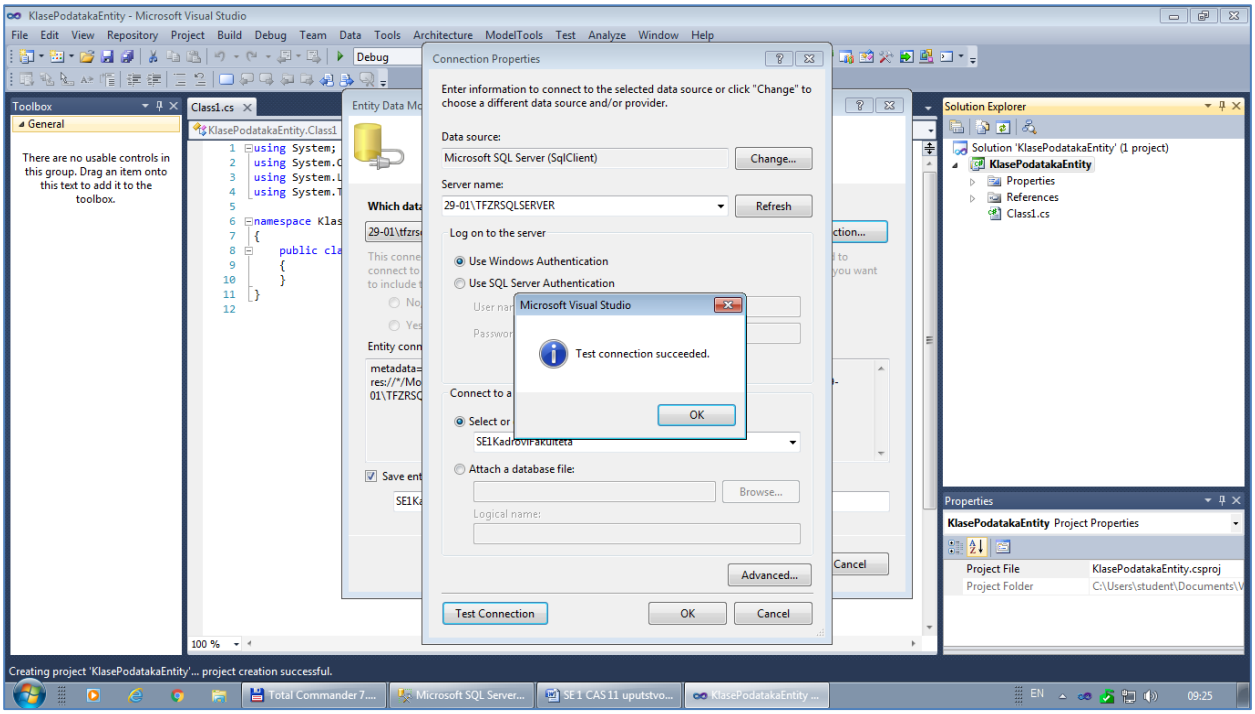
Slika 10.9. Izbor instance MS SQL Servera koja će biti korišćena

Biramo ili unosimo naziv baze podataka iz combo boxa “SELECT OR ENTER DATABASE NAME”.



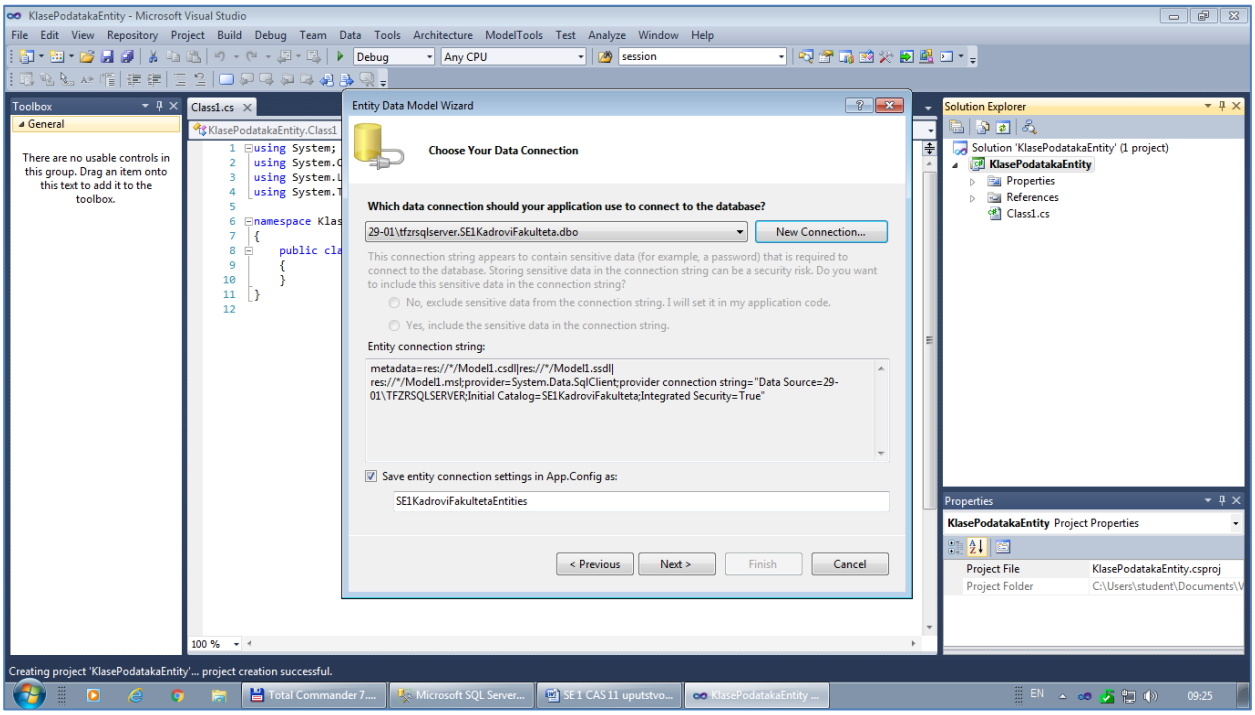
Slika 10.10. Izbor naziva baze podataka sa kojom će se raditi

Testiramo konekciju – opcija Test Connection.



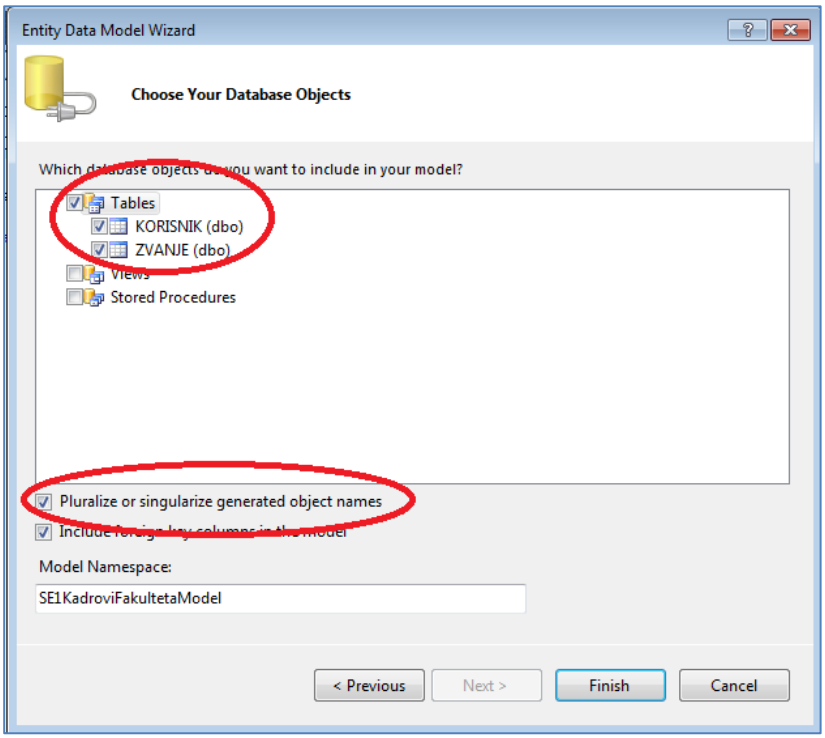
Slika 10.11. Rezultat testiranja konekcije do baze podataka

Dobijamo podatke o novoj konekciji, a posebno se može otvoriti i proveriti odeljak za string konekcije.



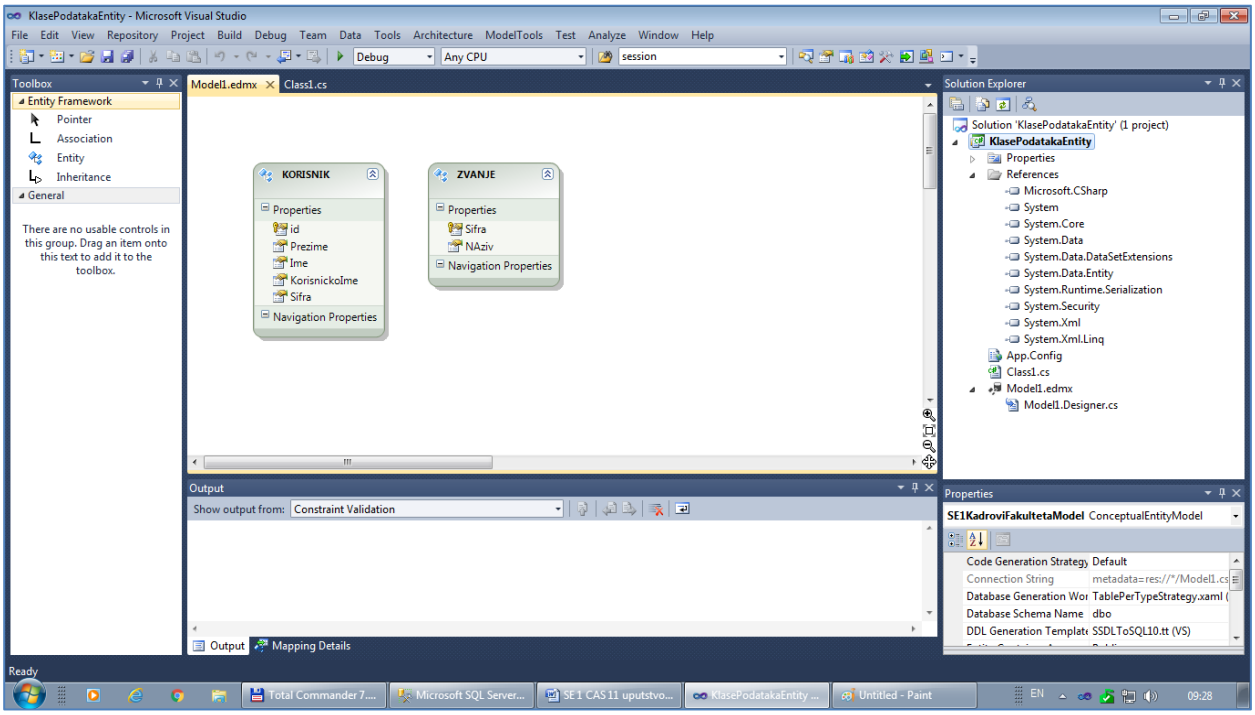
Slika 10.12. Prikaz podataka o novoj konekciji na bazu podataka

Biramo tabele iz baze podataka (ako u bazi podataka ima pogleda ili stored procedura, mogu se i one izabrati) i BIRAMO "Pluralize and singularize..." da bi prilikom kreiranja kolekcija objekata dodao s ili es sufiks na osnovni naziv tabele.



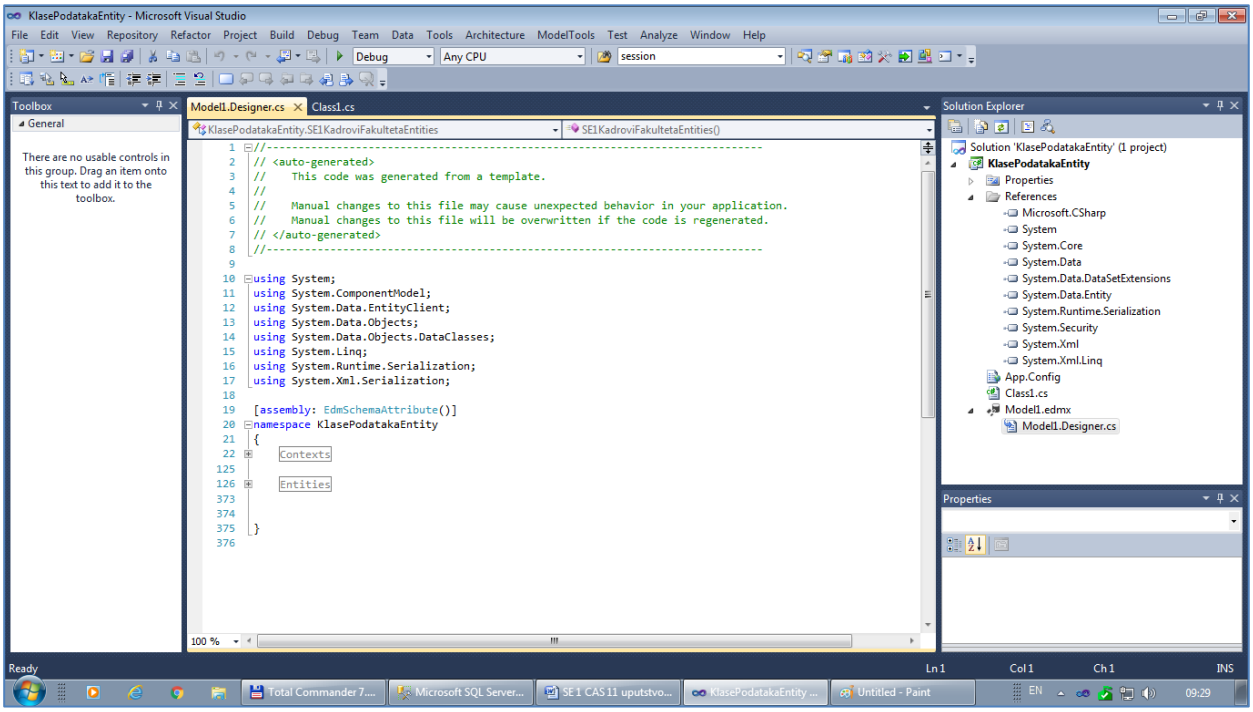
Slika 10.13. Izbor tabela iz baze podataka i podešavanje "Pluralize-singularize"

Biramo finish, dobijamo graficki prikaz Entity framework modela.



Slika 10.14. Grafički prikaz kreiranih klasa u okviru generisanog koda Entity framework klasa

Zatvorimo grafički deo (*.edmx fajl) i otvorimo Model1.Designer.cs fajl da vidimo programski kod klasa koje su generisane automatski.



Slika 10.15. Prikaz generisanog programskog koda Entity framework klasa

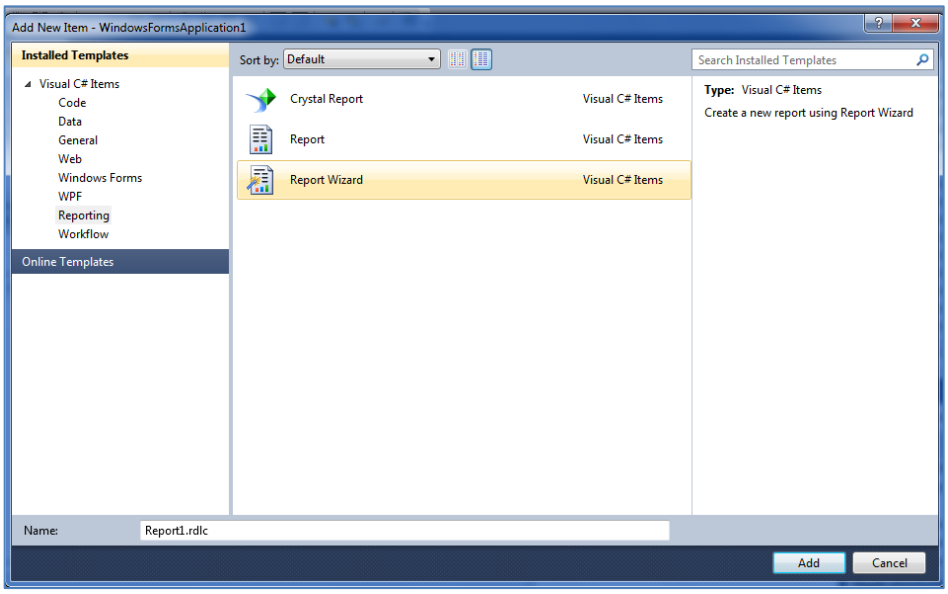
Generisani programski kod se može koristiti u okviru formi korisničkog interfejsa, instanciranjem i korišćenjem odgovarajućih klasa.

Generisanje programskog koda Entity Framework klasa se može realizovati u okviru posebne biblioteke klasa, ali može i u okviru projekta Windows forms aplikacije, kako je prikazano u prethodnom primeru.

10.3. Rad sa izveštajima

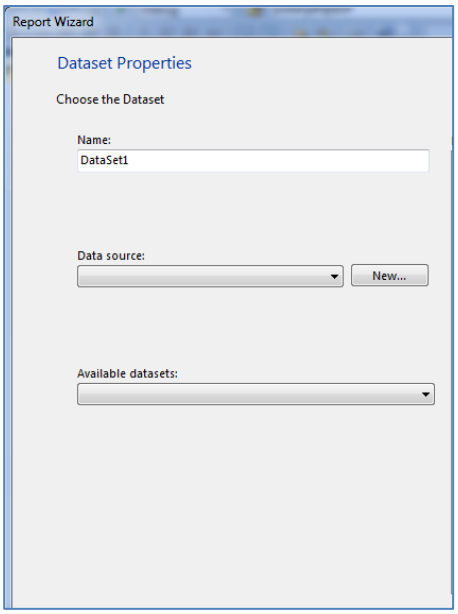
Novi izveštaj se može dizajnirati ručno ili korišćenjem wizarda. Nastaje fajl sa ekstenzijom rdlc.

Kreiramo izveštaj – project, add new item – Report Wizard. Biramo naziv fajla izveštaja.



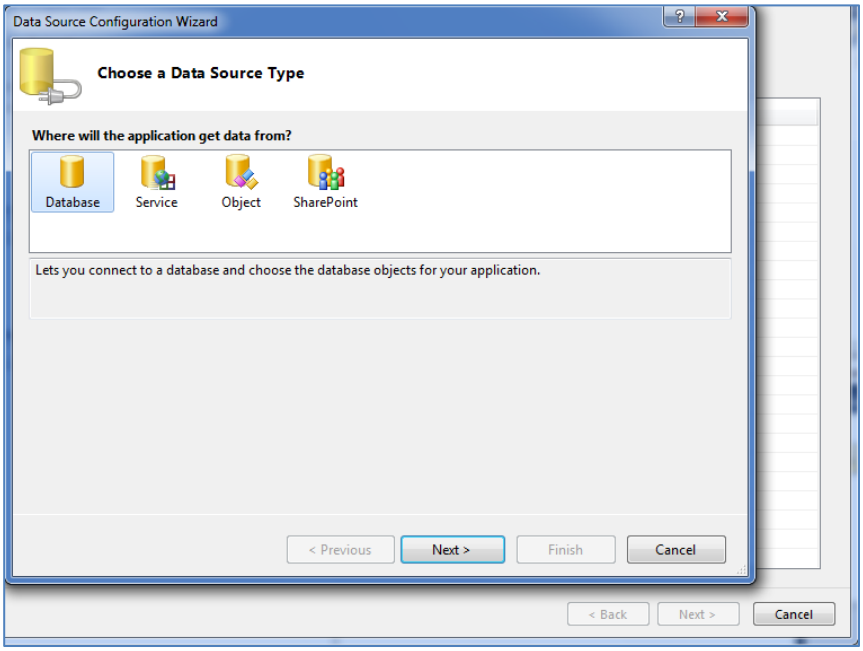
Slika 10.16. Izbor opcije za dodavanje novog izveštaja opcijom Report wizard

Biramo NEW za kreiranje izvora podataka Dataset.



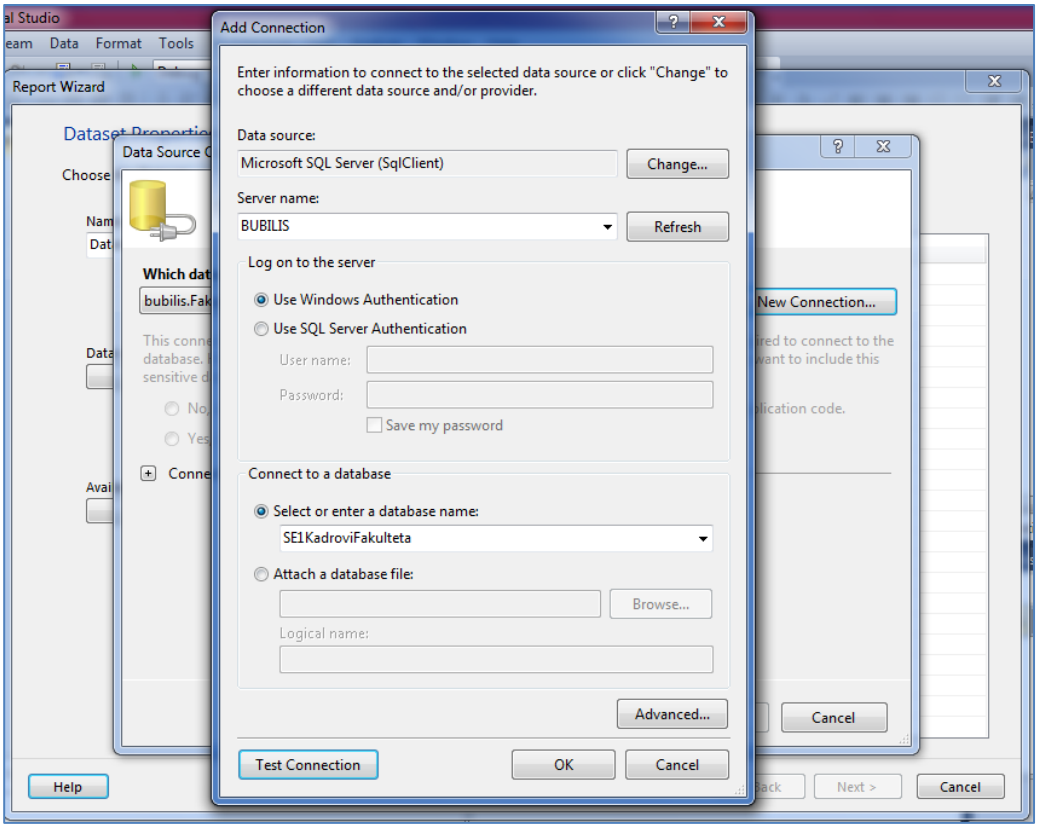
Slika 10.17. Kreiranje izvora podataka za izveštaj – novi DataSet

Biramo Database i Dataset.



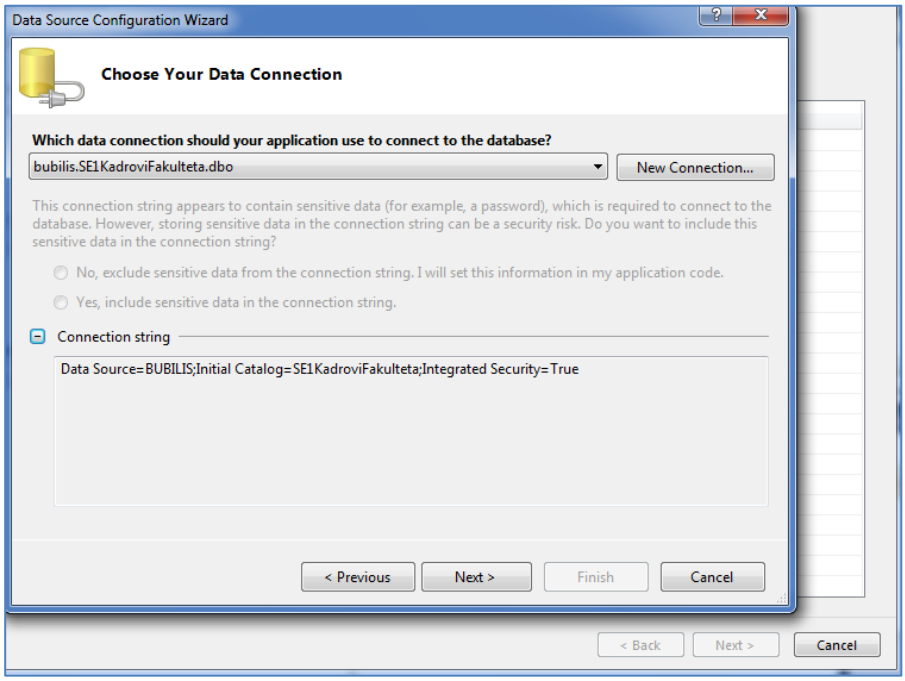
Slika 10.18. Izbor opcije Database u toku kreiranja novog DataSeta

Dobijamo dijalog prozor za kreiranje konekcije za bazu podataka. Biramo new connection.



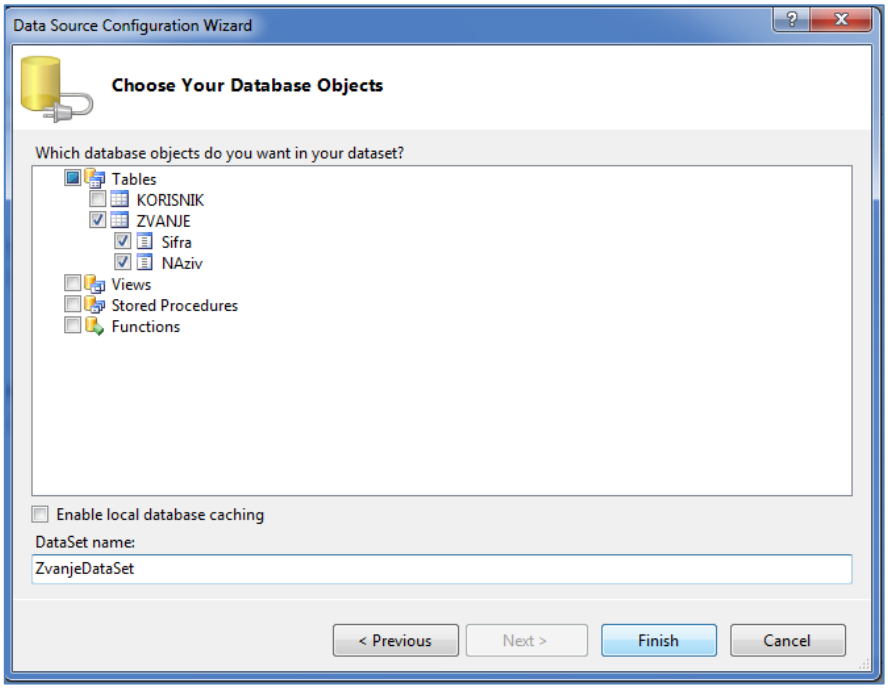
Slika 10.19. Kreiranje nove konekcije do baze podataka

Kreirali smo konekciju.



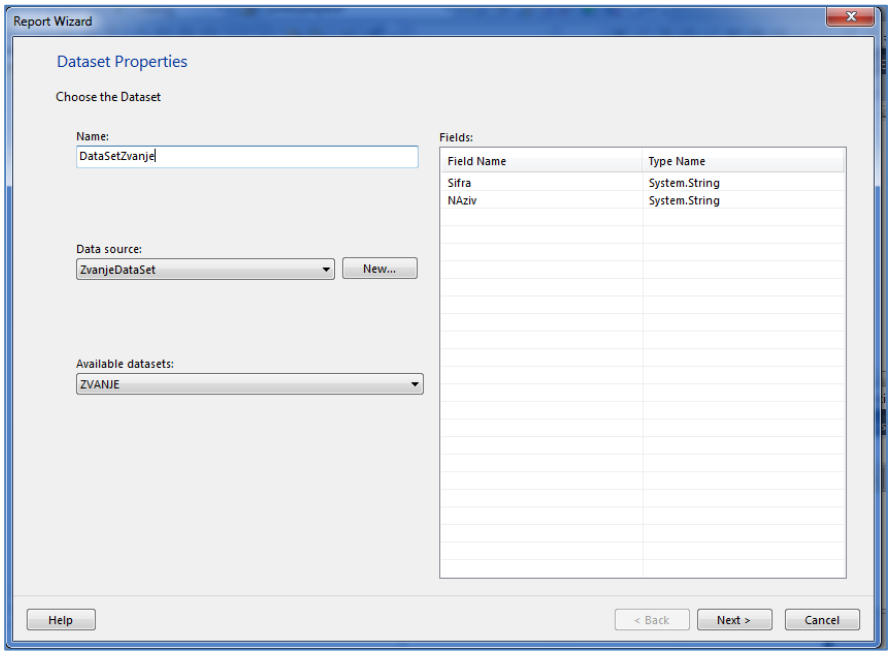
Slika 10.19. Izbor konekcije do baze podataka

Biramo Next…i dobijamo mogućnost izbora elemenata baze podataka za izvor podataka za Dataset. Unosimo naziv dataseta.



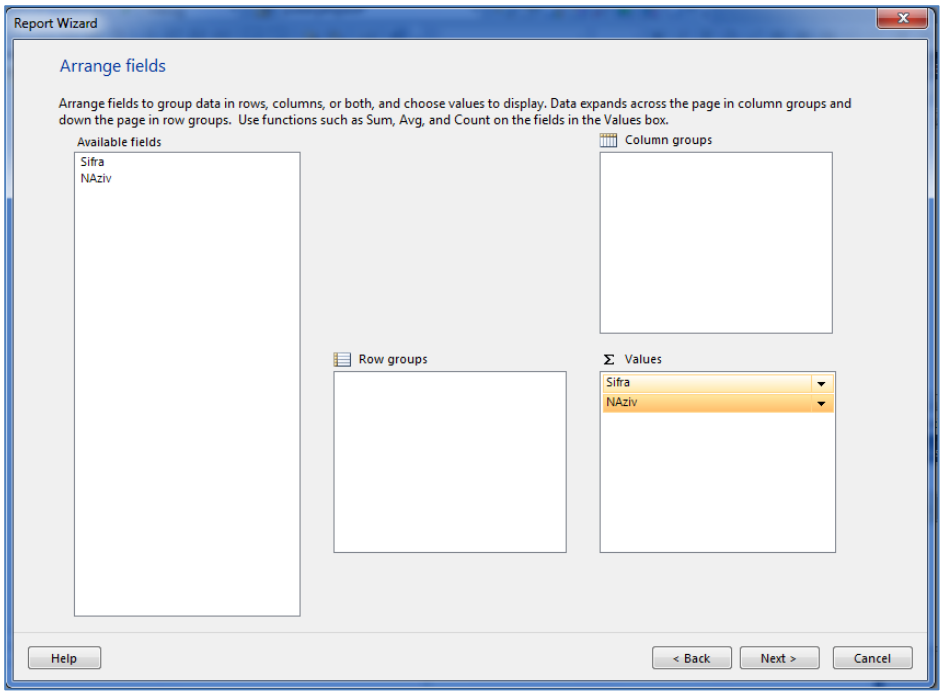
Slika 10.20. Izbor elemenata baze podataka koji će biti osnov za podatke na izveštaju

Dobijamo spisak polja koja čine dataset:



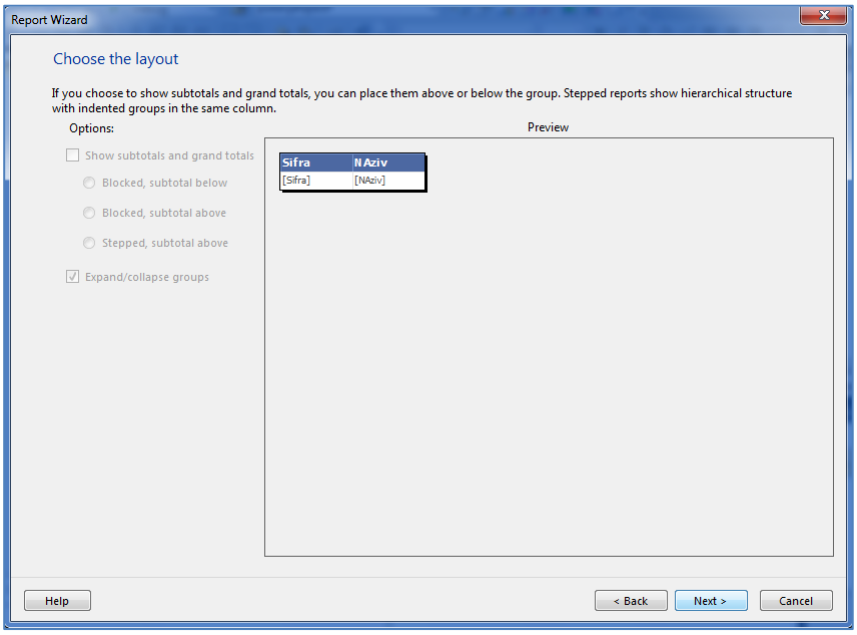
Slika 10.21. Spisak polja koja su uključena u novi DataSet

Biramo polja prevlačenjem iz Available fields - Values:



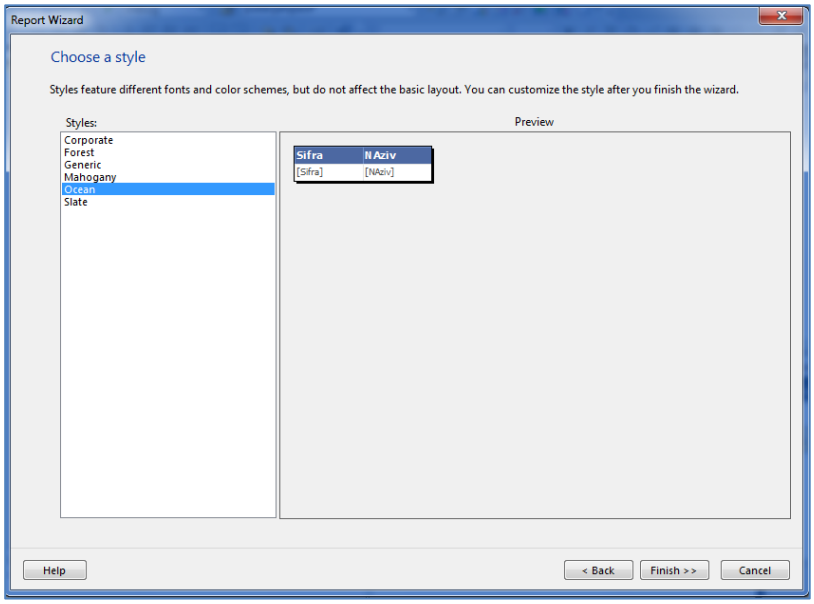
Slika 10.22. Prevlačenje polja radi kreiranja izveštaja

Početni izgled izveštaja predstavljen je sledećom slikom:



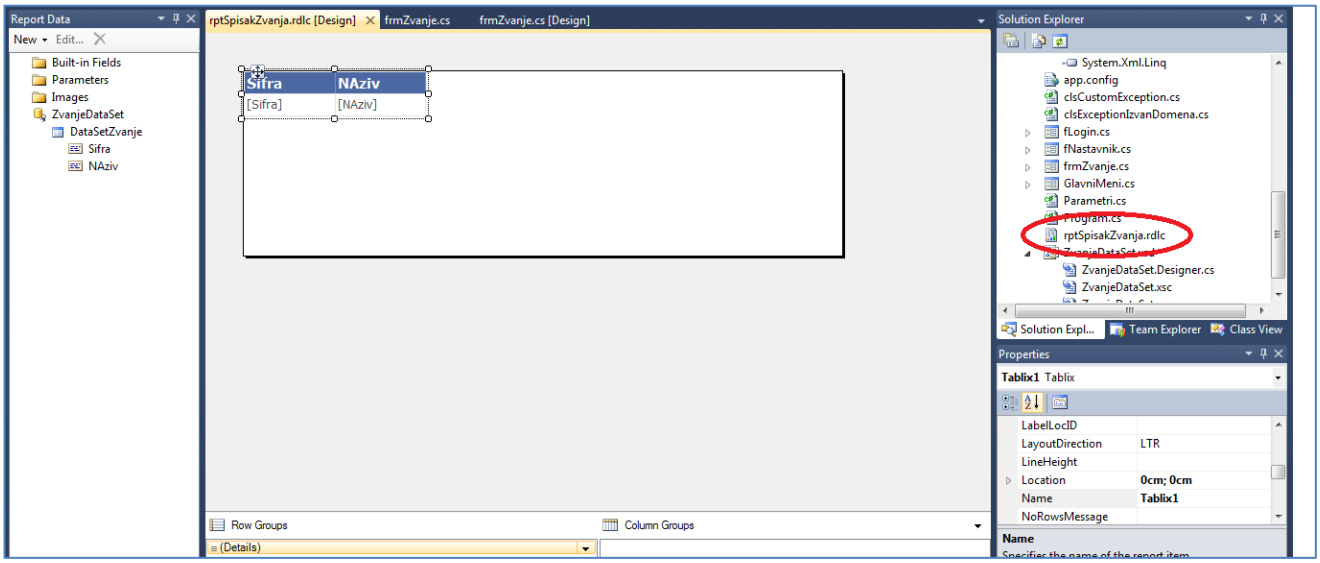
Slika 10.23. Početni izgled izveštaja

Biramo stil, tj. grafički deo izgleda izveštaja:



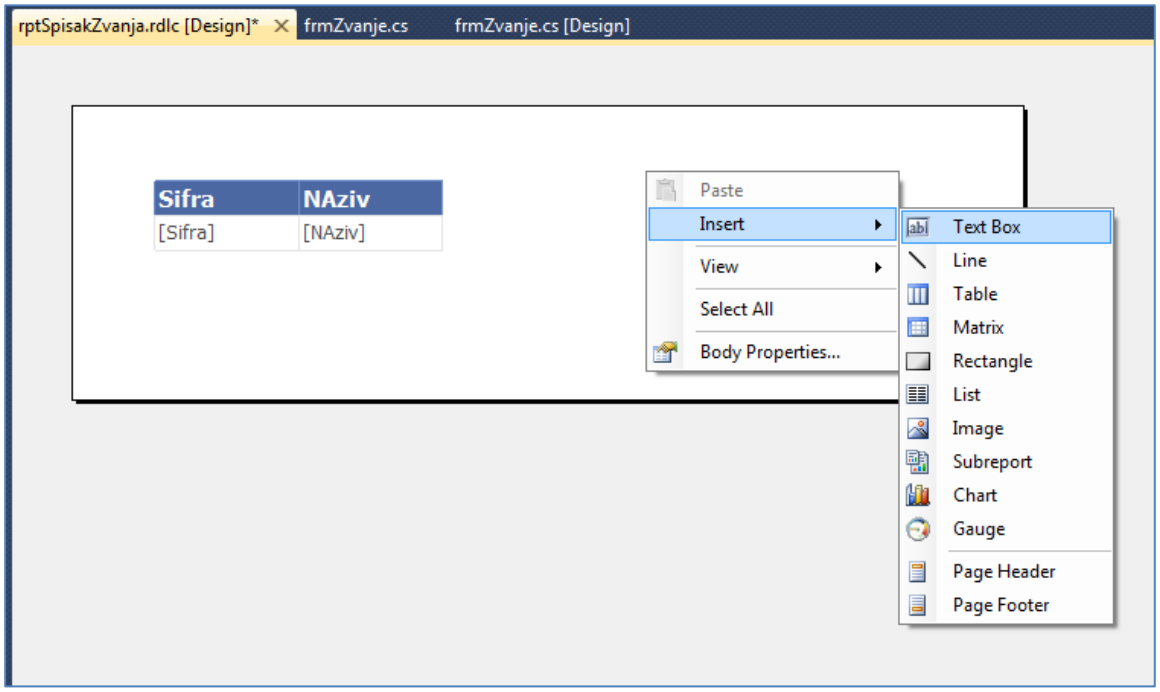
Slika 10.24. Izbor grafičkog dela izveštaja - stila

Izgled izveštaja u dizajn režimu, kao i njegova pojava na spisku fajlova solution explorera prikazan je narednom slikom:



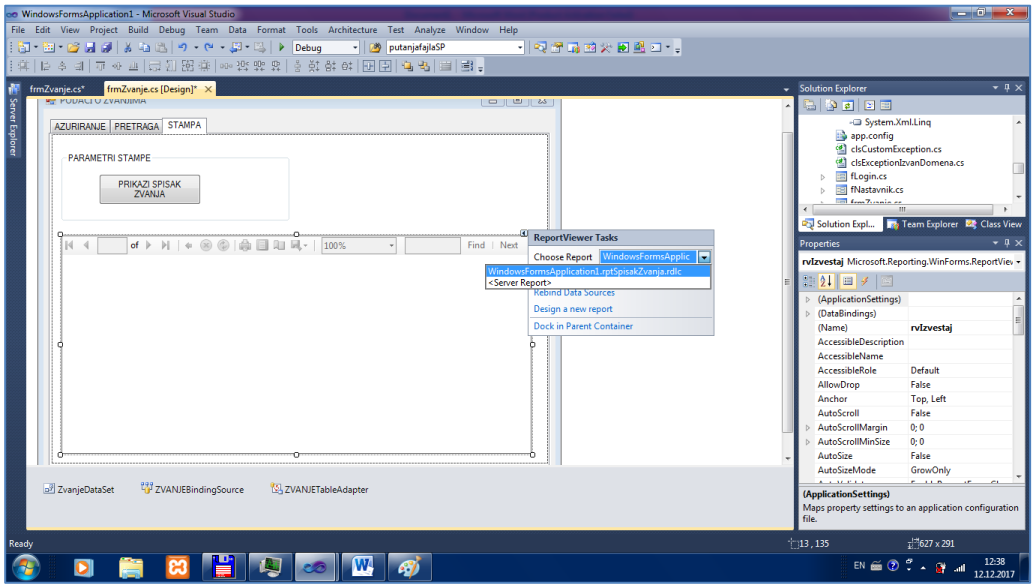
Slika 10.25. Prikaz izveštaja u dizajn režimu, sa mogućnošću izmene

Dodajemo naslov u text boxu (Desni taster, insert, text box) – SPISAK ZVANJA:



Slika 10.26. Dodavanje elemenata izveštaja radi finalnog uređivanja

Povezujemo grafičku kontrolu za rad sa izveštajima(ReportViewer) sa fajlom reporta:



Slika 10.27. Povezivanje fajla izveštaja sa grafičkom kontrolom za rad sa izveštajima

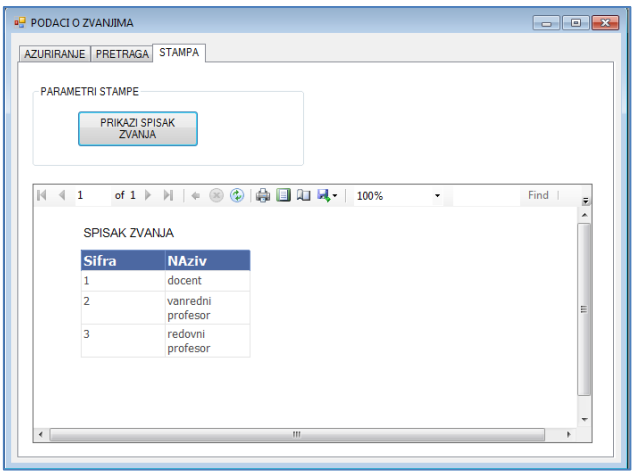
Čim smo povezali ReportViewer kontrolu sa fajlom reporta, dobijamo automatski na samoj formi kontrole za rad sa podacima koje obezbeđuju podatke za report (ZvanjeBindingSource, ZvanjeTableAdapter, ZvanjeDataSet), a u Form_load događaj se automatski ubacuje kod:

```
    private void frmZvanje_Load(object sender, EventArgs e)
    {
        // OVO JE ZA STAMPU, AUTOMATSKI SE UBACUJE KADA SE DODA IZVESTAJ
        this.ZVANJETableAdapter.Fill(this.ZvanjeDataSet.ZVANJE);
```

Dodajemo kod na tasteru PRIKAZI IZVESTAJ:

```
private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
    {

        rvIzvestaj.RefreshReport();
    }
```

REZULTAT:



Slika 10.28. Konačan izgled izveštaja u okviru report viewer kontrole

10.4. Primer programskog koda windows aplikacije

10.4.1. Zadaci

1. (SEMINARSKI RAD) Realizovati primenom standardne biblioteke klasa za rad sa MS SQL Server bazom podataka:

- Prijavu na sistem – Login forma

2. (KOLOKVIJUM) Realizovati unos i tabelarni prikaz za tabelu baze podataka Zvanje:

a) kreiranjem i korišćenjem sopstvenih biblioteka klasa koje koriste standardne klase i pozive stored procedura.

3. (KOLOKVIJUM) Realizovati sve osnovne funkcije programa za tabelu Zvanje.
   - Realizovati unos, brisanje, izmena, tabelarni prikaz, filtriranje uz uzimanje u obzir znaka, stampa), primenom samostalno kreirane biblioteke klasa KlasePodataka i klasa koje omogucavaju sve osnovne operacije sa podacima.
4. (KOLOKVIJUM) realizovati generisanje Entity Framework klasa u okviru posebne biblioteke klasa i povezati sa korisnickim interfejsom, realizovati unos i tabelarni prikaz.

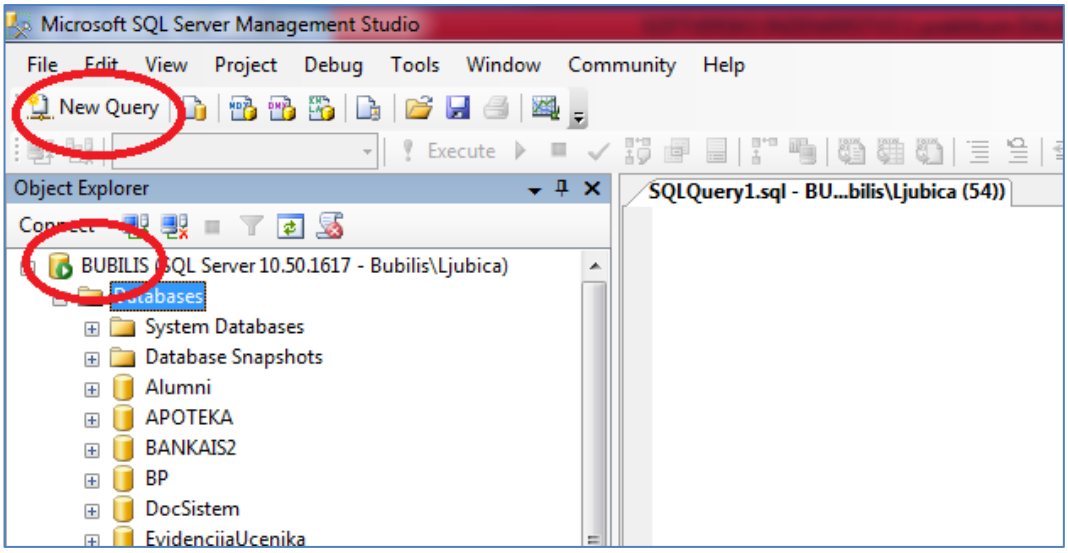
5. (SEMINARSKI RAD) Realizovati primenom sopstvenih klasa koje koriste standardne biblioteke klasa za rad sa MS SQL Server bazom podataka formu za evidenciju Nastavnika:

a) Tabelarni prikaz podataka o nastavniku (objedinjeni podaci nastavnika i zvanja, join u upitu)
b) Filtriranje podataka o nastavniku (filter prema prezimenu, zvanju i JMBG broju)
c) Stampanje podataka o nastavniku (na osnovu upita koji objedinjuje podatke nastavnika i zvanja) – izvestaj: SviNastavnici.rdlc
d) Parametarska stampa podataka o nastavniku – tip izvestaja NastavniciPremaZvanju.rdlc
e) Unos podataka o nastavniku (combo se puni podacima iz tabele baze podataka Zvanje)
f) Navigaciju podataka o nastavniku
g) Brisanje podataka o nastavniku
h) Izmenu podataka o nastavniku

10.4.2. Rešenja

1. zadatak

Pokrećemo grafički alat za rad sa bazom podataka MS SQL Server – Management Studio i startujemo opciju NEW QUERY.

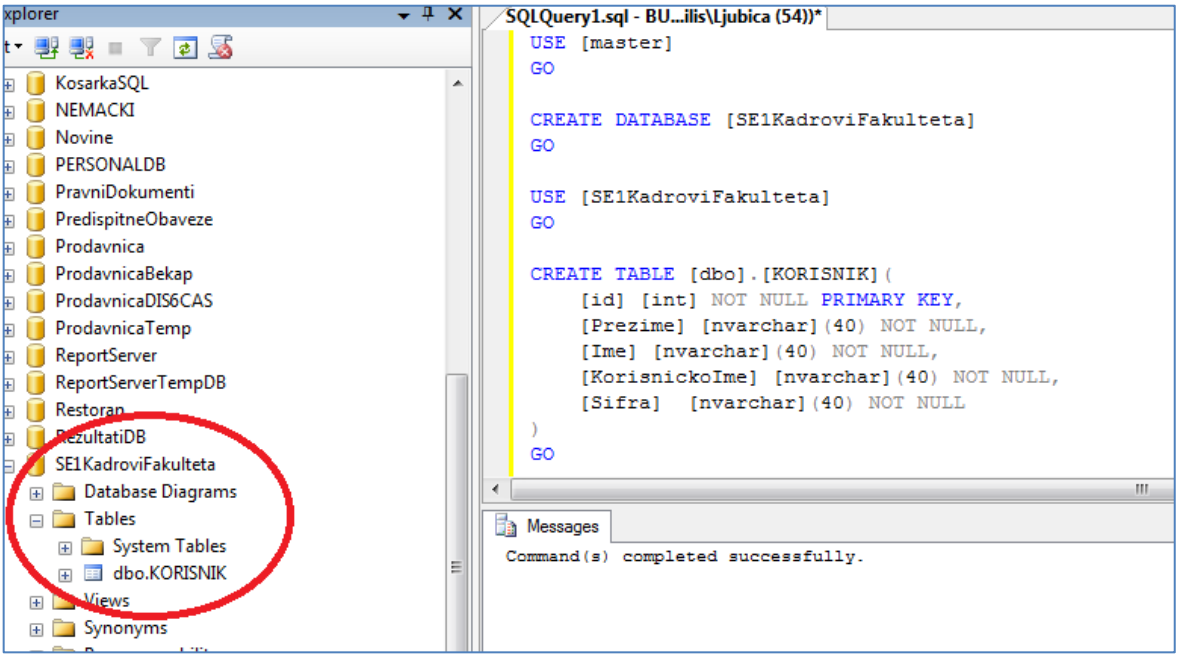


Slika 10.29. Pokretanje MS SQL Server Management Studio

Izvršavamo SQL skript za kreiranje baze podataka i tabele.

```
USE [master]
GO

CREATE DATABASE [SE1KadroviFakulteta]
GO

USE [SE1KadroviFakulteta]
GO

CREATE TABLE [dbo].[KORISNIK](
	[id] [int] NOT NULL PRIMARY KEY,
	[Prezime] [nvarchar](40) NOT NULL,
	[Ime] [nvarchar](40) NOT NULL,
	[KorisnickoIme] [nvarchar](40) NOT NULL,
	[Sifra]  [nvarchar](40) NOT NULL
)
GO
```

Nakon izvršavanja SQL skripta dobijamo bazu podataka:



Slika 10.30. Izvršavanje SQL skripta i kreiranje baze podataka

Unosimo korisnike (desni taster nad tabelom, edit top 200 rows).

Otvaramo korisnicki interfejs i dodajemo klasu Project- Add class. Ova klasa je staticka i sluzi za centralno cuvanje parametara applikacije, npr. String konekcije.

Slika 10.31. Public static klasa za čuvanje parametara konekcije, tj. stringa konekcije

```
public static class Parametri
    {
        public static string stringKonekcije = "Data Source=BUBILIS; Initial
Catalog=Se1KadroviFakulteta; Integrated Security=True;";
    }
```

Na formi za prijavljivanje treba postaviti programski kod za ucitavanje podataka iz baze podataka i proveru korisnika:

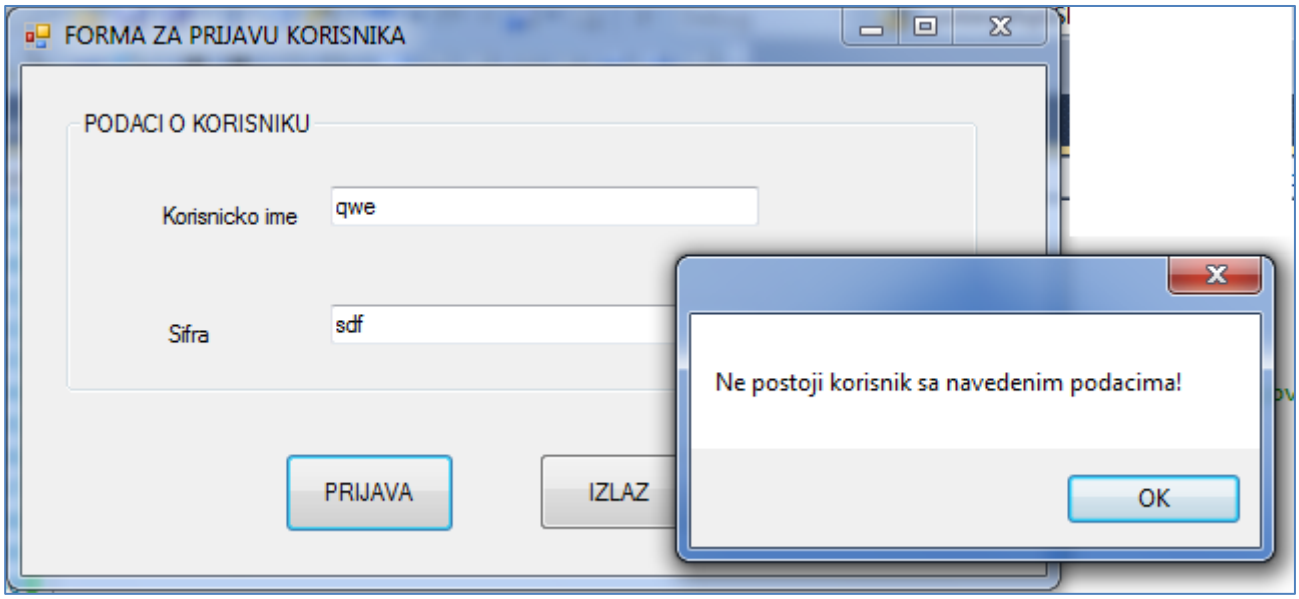
```
//
using System.Data;
using System.Data.SqlClient;
```

```
namespace WindowsFormsApplication1
{
    public partial class frmLogin : Form
    {

        // atributi

        // konstruktor
        public frmLogin()
        {
            InitializeComponent();
        }

        // nase metode

        private DataTable UcitajPodatke(string Upit)
        {
            DataTable TabelaPodataka = new DataTable();

            SqlConnection Veza = new SqlConnection(Parametri.stringKonekcije);
            Veza.Open();

            // preuzimanje podataka

            SqlCommand Komanda = new SqlCommand(Upit, Veza);
            SqlDataReader dr = Komanda.ExecuteReader();

            // transformacija datareader u data table
            // - ne mora, u web aplikaciji se moze povezati direktno
            // grid sa data readerom, ali postoji problem zbog zatvaranja konekcije
            // pa je bolje ipak da se podati postave u datatable, koji nije direktno povezan sa bazom

            if (dr.HasRows)
            {
                TabelaPodataka.Load(dr);
            }

            dr.Close();
            Veza.Close();
            Veza.Dispose();

            return TabelaPodataka;

        }

        private bool PostojiKorisnik()
        {
            bool pronadjen = false;
            DataTable dt = UcitajPodatke ("Select * from Korisnik where KorisnickoIme='" +
txbKorisnikoIme.Text + "'  and Sifra='" + txbSifra.Text + "'");
            if (dt.Rows.Count > 0)
            {
                pronadjen = true;
            }

            return pronadjen;
        }


        // dogadjaji
        private void frmLogin_Load(object sender, EventArgs e)
        {
```

```
        }

        private void btnPrijava_Click(object sender, EventArgs e)
        {
            if (PostojiKorisnik())
            {
                GlavniMeni objGlavniMeni = new GlavniMeni();
                objGlavniMeni.ShowDialog();
                this.Close();
            }
            else
            {
                MessageBox.Show("Ne postoji korisnik sa navedenim podacima!");
                txbKorisnikoIme.Focus();
            }
        }

        private void btnIzlaz_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
```

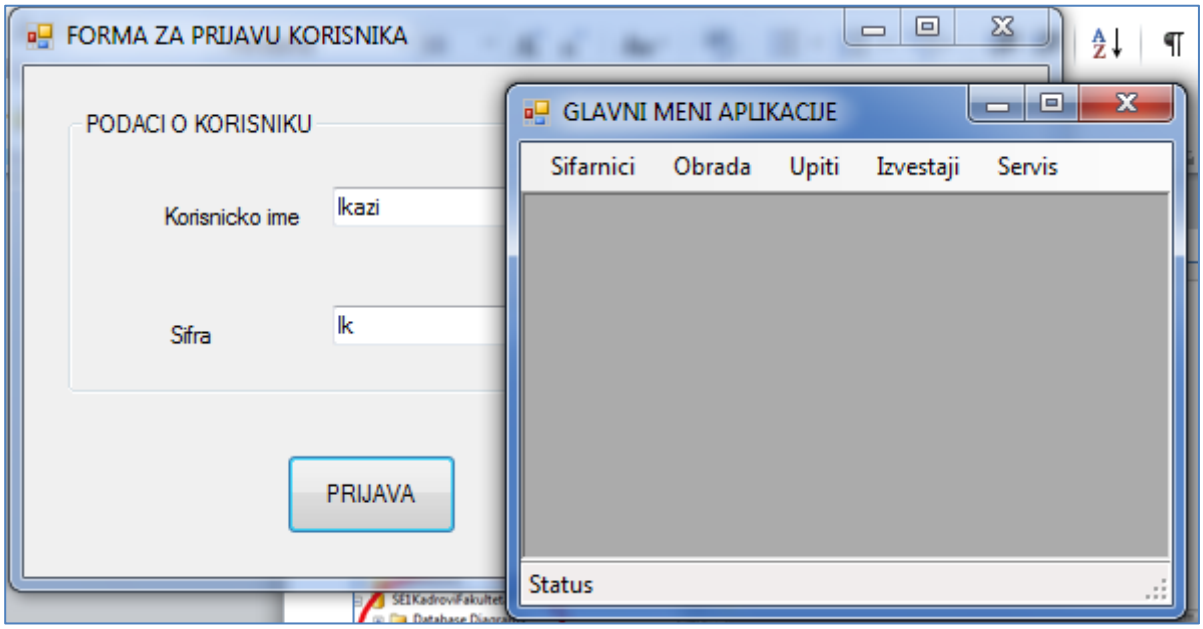
REZULTAT

Ako je neispravna sifra ili korisnicko ime:



Slika 10.32. Rezultat izvršavanja provere korisnika prilikom prijavljivanja nepostojećeg korisnika

Ako je ispravna, ucitava se glavni meni aplikacije:



Slika 10.32. Rezultat izvršavanja provere korisnika prilikom prijavljivanja postojećeg korisnika

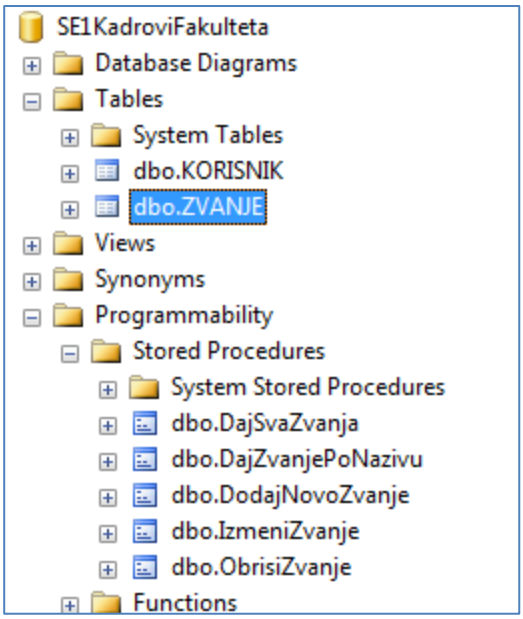
2. zadatak

U okviru ranije kreirane baze podataka kreiramo tabelu ZVANJE i sve pratece STORED PROCEDURE.

```
USE [SE1KadroviFakulteta]
GO

CREATE TABLE [dbo].[ZVANJE](
	[Sifra] [Char] (3) NOT NULL PRIMARY KEY,
	[NAziv] [nvarchar](40) NOT NULL,
)
GO

CREATE PROCEDURE [DajSvaZvanja]
AS
select * from Zvanje
GO

CREATE PROCEDURE [DajZvanjePoNazivu]
( @NazivZvanja nvarchar(40)
)
AS
select * from Zvanje where Zvanje.Naziv = @NazivZvanja
GO

CREATE PROCEDURE [DodajNovoZvanje]
(
@Sifra char(3),
@Naziv nvarchar(40)
)
AS
```

```
BEGIN
Insert into Zvanje(Sifra, Naziv) values (@Sifra, @Naziv)
END
GO

CREATE PROCEDURE [ObrisiZvanje](
@Sifra char(3))
AS
BEGIN
Delete from Zvanje where Sifra=@Sifra
END
GO

CREATE PROCEDURE [IzmeniZvanje](
@StaraSifra char(3),
@Sifra char(3),
@Naziv nvarchar(40)
)
AS
BEGIN
Update Zvanje set Sifra=@Sifra, Naziv=@Naziv where Sifra=@StaraSifra
END
GO
```

Rezultat:



Slika 10.33. Spisak stored procedura koje su kreirane u bazi podataka

Unosimo podatke o zvanju (tabela Zvanje, desni taster, Edit top 200 rows).

U biblioteci klasa KlasePodataka napravićemo za tabelu Zvanje 3 klase:

- klasa Pojedinac (ima samo set-get metode)
- Klasa Lista (sadrži listu objekata klase pojedinac)
- Klasa DB (rad sa bazom podataka i tabelom Zvanje)

KLASA POJEDINAC – cls Zvanje

Već ranije kreirana:

```
namespace KlasePodataka
{
    public class clsZvanje
    {
        // atributi
        private int pSifra;
        private string pNaziv;

        // konstruktor
        public clsZvanje()
        {
            Sifra = 0;
            Naziv = "";
        }

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

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


        // privatne metode

        // javne metode
        public string DajPodatke()
        {
            return this.pSifra + " " + this.pNaziv;
        }
```

```
    }
}
```

Klasa LISTA POJEDINACA – clsZvanjeLista

```
public class clsZvanjeLista
   {
      // atributi
      private List<clsZvanje> pListaZvanja;

      // property
      public List<clsZvanje> ListaZvanja
      {
         get
         {
            return pListaZvanja;
         }
         set
         {
            if (this.pListaZvanja != value)
               this.pListaZvanja = value;
         }
      }

      // konstruktor
      public clsZvanjeLista()
      {
         pListaZvanja = new List<clsZvanje>();

      }

      // privatne metode

      // javne metode
      public void DodajElementListe(clsZvanje objNovoZvanje)
      {
         pListaZvanja.Add(objNovoZvanje);
      }

      public void ObrisiElementListe(clsZvanje objZvanjeZaBrisanje)
      {
         pListaZvanja.Remove(objZvanjeZaBrisanje);
      }

      public void ObrisiElementNaPoziciji(int pozicija)
      {
         pListaZvanja.RemoveAt(pozicija);
      }

      public void IzmeniElementListe(clsZvanje objStaroZvanje, clsZvanje objNovoZvanje)
      {
```

```
        int indexStarogZvanja = 0;
        indexStarogZvanja = pListaZvanja.IndexOf(objStaroZvanje);
        pListaZvanja.RemoveAt(indexStarogZvanja);
        pListaZvanja.Insert(indexStarogZvanja, objNovoZvanje);
    }
```

Klasa tipa DB koja koristi standardne klase iz System.Data.SQLClient biblioteke – clsZvanjeDB

```
//
using System.Data;
using System.Data.SqlClient;

namespace KlasePodataka
{
    public class clsZvanjeDB
    {
        // atributi
        private string pStringKonekcije;

        // property
        // 1. nacin
        public string StringKonekcije
        {
            get
            {
                return pStringKonekcije;
            }
            set // OVO NIJE DOBRO, MOZE SE STRING KONEKCIJE STAVITI NA PRAZAN STRING
            {
                if (this.pStringKonekcije != value)
                    this.pStringKonekcije = value;
            }
        }
        // konstruktor
        // 2. nacin prijema vrednosti stringa konekcije spolja i dodele atributu
        public clsZvanjeDB(string NoviStringKonekcije)
        // OVO JE DOBRO JER OBAVEZUJE DA SE PRILIKOM INSTANCIRANJA OVE KLASE
        // MORA OBEZBEDITI STRING KONEKCIJE
        {
            pStringKonekcije = NoviStringKonekcije;
        }

        // privatne metode

        // javne metode
        public DataSet DajSvaZvanja()
        {
            DataSet dsPodaci = new DataSet();

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();
            SqlCommand Komanda = new SqlCommand("DajSvaZvanja", Veza);
```

```
            Komanda.CommandType = CommandType.StoredProcedure;
            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = Komanda;
            da.Fill(dsPodaci);
            Veza.Close();
            Veza.Dispose();

            return dsPodaci;
        }

        public DataSet DajZvanjaPoNazivu(string NazivZvanja)
        {
            // MOGU biti jos neke procedure, mogu SE VRATITI VREDNOSTI I U LISTU, DATA
TABLE...
            DataSet dsPodaci = new DataSet();

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();
            SqlCommand Komanda = new SqlCommand("DajZvanjePoNazivu", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@NazivZvanja", SqlDbType.NVarChar).Value =
NazivZvanja;
            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = Komanda;
            da.Fill(dsPodaci);
            Veza.Close();
            Veza.Dispose();

            return dsPodaci;
        }

        // overloading metoda - isto se zove, ima drugaciji parametar
        public DataSet DajZvanjaPoNazivu(clsZvanje objZvanjeZaFilter)
        {
            // MOGU biti jos neke procedure, mogu SE VRATITI VREDNOSTI I U LISTU, DATA
TABLE...
            DataSet dsPodaci = new DataSet();

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();
            SqlCommand Komanda = new SqlCommand("DajZvanjePoNazivu", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@NazivZvanja", SqlDbType.NVarChar).Value =
objZvanjeZaFilter.Naziv;
            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = Komanda;
            da.Fill(dsPodaci);
            Veza.Close();
            Veza.Dispose();

            return dsPodaci;
        }

        public bool SnimiNovoZvanje(clsZvanje objNovoZvanje)
```

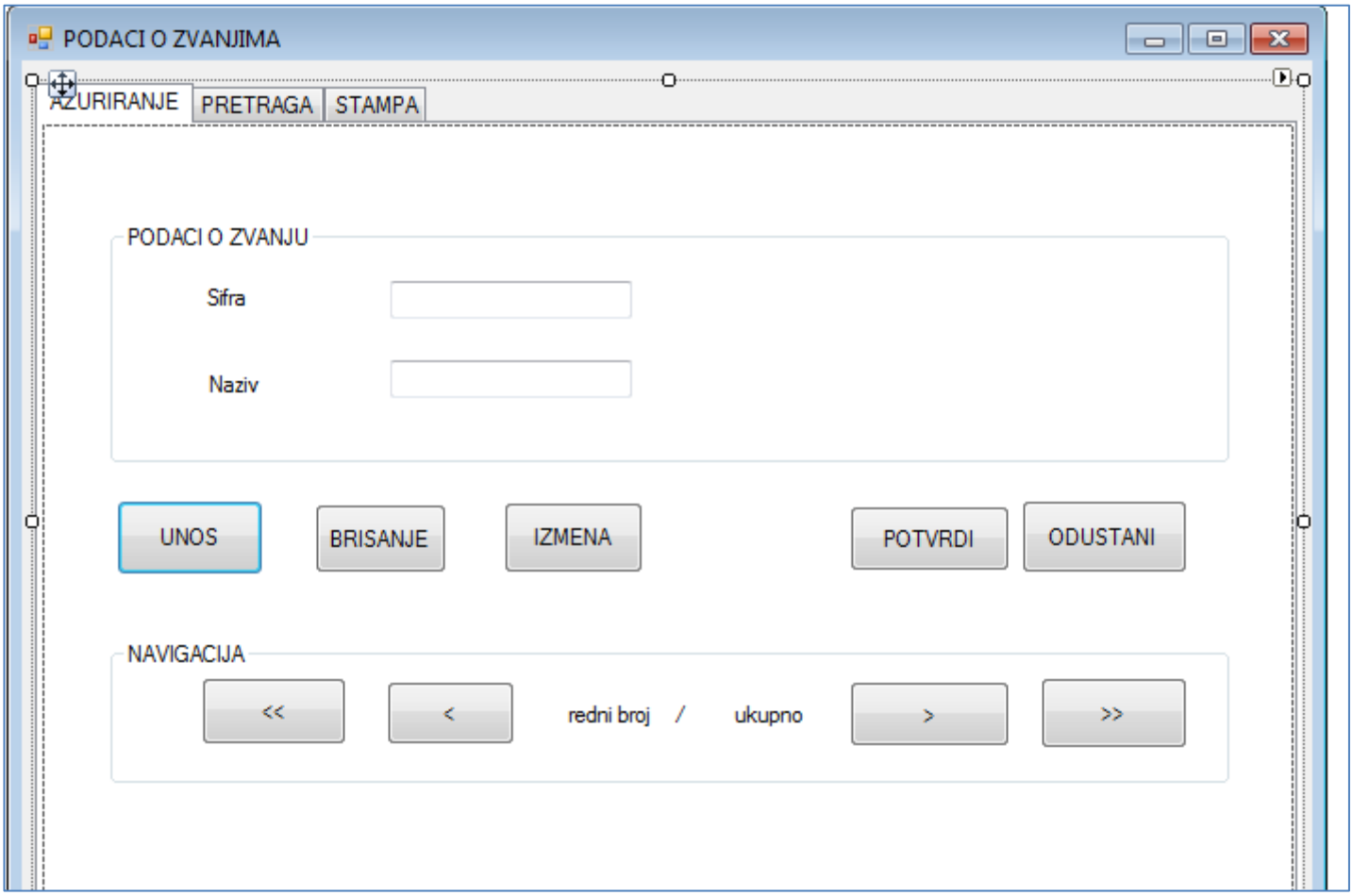
```
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("DodajNovoZvanje", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value =
objNovoZvanje.Sifra;
            Komanda.Parameters.Add("@Naziv", SqlDbType.NVarChar).Value =
objNovoZvanje.Naziv;

            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            // NEGATIVNO PITANJE - NIJE DOBRO if (brojSlogova ==0)
            /* 1. VARIJANTA - SKOLSKA
             * if (brojSlogova>0)
            {
                uspehSnimanja=true;
            }
            else
            {
                uspehSnimanja=false;
            }

            return uspehSnimanja;
             */

            // 2. varijanta
            return (brojSlogova > 0);

            //3. varijanta
            // NEMA SMISLA, ISTO KAO 2. VARIJANTA
            //return (brojSlogova > 0) ? true : false;

            //4. varijanta - NEGACIJA NEGACIJE, NIJE RAZUMLJIVO
            //return (brojSlogova == 0) ? false : true;

        }

        public bool ObrisiZvanje(clsZvanje objZvanjeZaBrisanje)
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
```

```
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("ObrisiZvanje", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value =
objZvanjeZaBrisanje.Sifra;
            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            return (brojSlogova > 0);

        }

        // method overloading - ista procedura sa razlicitim parametrom
        public bool ObrisiZvanje(string SifraZvanjaZaBrisanje)
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("ObrisiZvanje", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value =
SifraZvanjaZaBrisanje;
            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            return (brojSlogova > 0);

        }

        public bool IzmeniZvanje(clsZvanje objStaroZvanje, clsZvanje objNovoZvanje)
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("IzmeniZvanje", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@StaraSifra", SqlDbType.Char).Value =
objStaroZvanje.Sifra;
            Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value =
objNovoZvanje.Sifra;
```

```
            Komanda.Parameters.Add("@Naziv", SqlDbType.NVarChar).Value =
objNovoZvanje.Sifra;
            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            return (brojSlogova > 0);

        }

        // method overloading - ista metoda, samo drugaciji parametri
        public bool IzmeniZvanje(string SifraStarogZvanja, clsZvanje objNovoZvanje)
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("IzmeniZvanje", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@StaraSifra", SqlDbType.Char).Value =
SifraStarogZvanja;
            Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value =
objNovoZvanje.Sifra;
            Komanda.Parameters.Add("@Naziv", SqlDbType.NVarChar).Value =
objNovoZvanje.Sifra;
            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            return (brojSlogova > 0);

        }

    }
}
```

Otvaramo projekat tipa korisnički interfejs. Uklonimo staru verziju i dodajemo referencu na KlasePodataka.dll.

Dodajemo novu Windows Form frmZvanje. Kopiramo sve kontrole sa forme Nastavnik i izmenimo.



Slika 10.34. Grafički prikaz ekranske forme za rad sa zvanjima

Na formi za Glavni meni dodajemo kod za otvaranje ove forme:

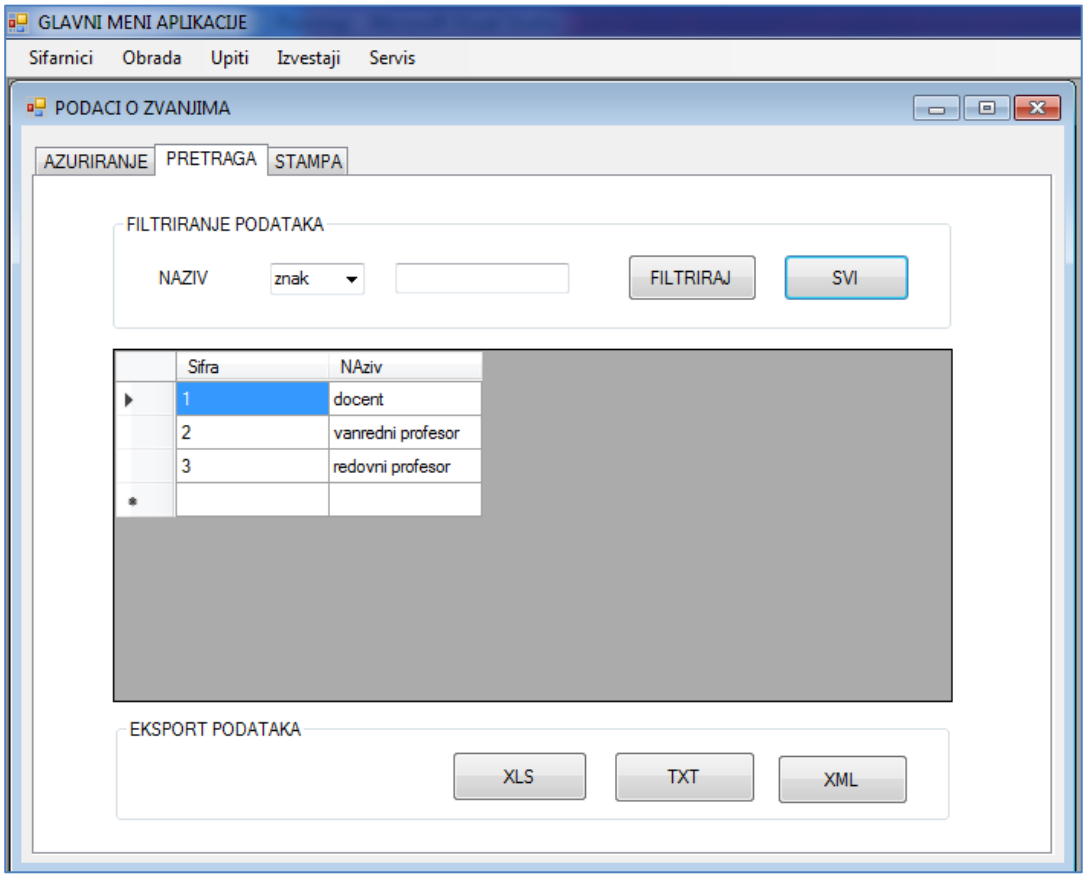
```
private void zvanjeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            frmZvanje objFormaZvanje = new frmZvanje();
            objFormaZvanje.MdiParent = this;
            objFormaZvanje.Show();
        }
```

Ubacujemo programski kod za realizaciju tabelarnog prikaza:

```
//
using KlasePodataka;

namespace WindowsFormsApplication1
{
    public partial class frmZvanje : Form
    {
       // atributi
        private clsZvanjeDB objZvanjeDB;

        // konstruktor
        public frmZvanje()
        {
            InitializeComponent();
        }
```

```
        // nase metode
        private void NapuniGrid(DataSet dsPodaci)
        {
            dgvSpisakZvanja.DataSource = dsPodaci.Tables[0];
            dgvSpisakZvanja.Refresh();
        }

        // dogadjaji
        private void frmZvanje_Load(object sender, EventArgs e)
        {
            objZvanjeDB = new clsZvanjeDB(Parametri.stringKonekcije);
        }

        private void btnSvi_Click(object sender, EventArgs e)
        {
            NapuniGrid(objZvanjeDB.DajSvaZvanja());
        }

        private void btnFiltriraj_Click(object sender, EventArgs e)
        {
            NapuniGrid(objZvanjeDB.DajZvanjaPoNazivu(txbFilter.Text));
        }
    }
}
```

REZULTAT:



Slika 10.35. Ekranska forma za rad sa zvanjima – izvršavanje tabelarnog prikaza podataka

3. zadatak

Koristimo biblioteku klasa KlasePodataka – klase: clsZvanje (za preuzimanje vrednosti sa korisničkog interfejsa i za čuvanje starih vrednosti kod izmene) i klasu clsZvanjeDB za rad sa bazom podataka. U klasi clsZvanjeDB dodata je još jedna metoda, koja omogućava filtriranje sa uzimanjem u obzir znaka (like ili =).

VERZIJA KLASE clsZvanjeDB koja se koristi u ovom zadatku:

```
//
using System.Data;
using System.Data.SqlClient;

namespace KlasePodataka
{
   public class clsZvanjeDB
   {
      // atributi
      private string pStringKonekcije;

      // property
      // 1. nacin
      public string StringKonekcije
      {
         get
         {
            return pStringKonekcije;
         }
         set // OVO NIJE DOBRO, MOZE SE STRING KONEKCIJE STAVITI NA PRAZAN STRING
         {
            if (this.pStringKonekcije != value)
               this.pStringKonekcije = value;
         }
      }
      // konstruktor
      // 2. nacin prijema vrednosti stringa konekcije spolja i dodele atributu
      public clsZvanjeDB(string NoviStringKonekcije)
      // OVO JE DOBRO JER OBAVEZUJE DA SE PRILIKOM INSTANCIRANJA OVE KLASE
      // MORA OBEZBEDITI STRING KONEKCIJE
      {
         pStringKonekcije = NoviStringKonekcije;
      }

      // privatne metode

      // javne metode
      public DataSet DajSvaZvanja()
      {
         DataSet dsPodaci = new DataSet();

         SqlConnection Veza = new SqlConnection(pStringKonekcije);
         Veza.Open();
```

```
        SqlCommand Komanda = new SqlCommand("DajSvaZvanja", Veza);
        Komanda.CommandType = CommandType.StoredProcedure;
        SqlDataAdapter da = new SqlDataAdapter();
        da.SelectCommand = Komanda;
        da.Fill(dsPodaci);
        Veza.Close();
        Veza.Dispose();

        return dsPodaci;
    }

    public DataSet DajZvanjaPoNazivu(string NazivZvanja)
    {
        // MOGU biti jos neke procedure, mogu SE VRATITI VREDNOSTI I U LISTU, DATA TABLE...
        DataSet dsPodaci = new DataSet();

        SqlConnection Veza = new SqlConnection(pStringKonekcije);
        Veza.Open();
        SqlCommand Komanda = new SqlCommand("DajZvanjePoNazivu", Veza);
        Komanda.CommandType = CommandType.StoredProcedure;
        Komanda.Parameters.Add("@NazivZvanja", SqlDbType.NVarChar).Value = NazivZvanja;
        SqlDataAdapter da = new SqlDataAdapter();
        da.SelectCommand = Komanda;
        da.Fill(dsPodaci);
        Veza.Close();
        Veza.Dispose();

        return dsPodaci;
    }

    // overloading metoda - isto se zove, ima drugaciji parametar
    public DataSet DajZvanjaPoNazivu(clsZvanje objZvanjeZaFilter)
    {
        // MOGU biti jos neke procedure, mogu SE VRATITI VREDNOSTI I U LISTU, DATA TABLE...
        DataSet dsPodaci = new DataSet();

        SqlConnection Veza = new SqlConnection(pStringKonekcije);
        Veza.Open();
        SqlCommand Komanda = new SqlCommand("DajZvanjePoNazivu", Veza);
        Komanda.CommandType = CommandType.StoredProcedure;
        Komanda.Parameters.Add("@NazivZvanja", SqlDbType.NVarChar).Value = objZvanjeZaFilter.Naziv;
        SqlDataAdapter da = new SqlDataAdapter();
        da.SelectCommand = Komanda;
        da.Fill(dsPodaci);
        Veza.Close();
        Veza.Dispose();

        return dsPodaci;
```

```
    }

    // overloading metoda - 2 parametra - dodat znak
    public DataSet DajZvanjaPoNazivu(string znak, string NazivZvanja)
    {
        DataSet dsPodaci = new DataSet();

        SqlConnection Veza = new SqlConnection(pStringKonekcije);
        Veza.Open();
        string uslov = "";
        if (znak.Equals ("="))
        {
            uslov = "Naziv='" + NazivZvanja + "'";
        }
        else
        {
            uslov = "Naziv like '%" + NazivZvanja + "%'";
        }

        SqlCommand Komanda = new SqlCommand("select * from Zvanje where " + uslov,
Veza);

        SqlDataAdapter da = new SqlDataAdapter();
        da.SelectCommand = Komanda;
        da.Fill(dsPodaci);
        Veza.Close();
        Veza.Dispose();

        return dsPodaci;
    }

    public bool SnimiNovoZvanje(clsZvanje objNovoZvanje)
    {
        // LOKALNE PROMENLJIVE UVEK NA VRHU
        int brojSlogova = 0;
        // 1. varijanta - skolska
        //bool uspehSnimanja= false;

        SqlConnection Veza = new SqlConnection(pStringKonekcije);
        Veza.Open();

        SqlCommand Komanda = new SqlCommand("DodajNovoZvanje", Veza);
        Komanda.CommandType = CommandType.StoredProcedure;
        Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value = objNovoZvanje.Sifra;
        Komanda.Parameters.Add("@Naziv", SqlDbType.NVarChar).Value =
objNovoZvanje.Naziv;

        brojSlogova = Komanda.ExecuteNonQuery();
        Veza.Close();
        Veza.Dispose();

        // NEGATIVNO PITANJE - NIJE DOBRO if (brojSlogova ==0)
```

```
         /* 1. VARIJANTA - SKOLSKA
          * if (brojSlogova>0)
         {
            uspehSnimanja=true;
         }
         else
         {
            uspehSnimanja=false;
         }

         return uspehSnimanja;
          */

         // 2. varijanta
         return (brojSlogova > 0);

         //3. varijanta
         // NEMA SMISLA, ISTO KAO 2. VARIJANTA
         //return (brojSlogova > 0) ? true : false;

         //4. varijanta - NEGACIJA NEGACIJE, NIJE RAZUMLJIVO
         //return (brojSlogova == 0) ? false : true;

      }

      public bool ObrisiZvanje(clsZvanje objZvanjeZaBrisanje)
      {
         // LOKALNE PROMENLJIVE UVEK NA VRHU
         int brojSlogova = 0;
         // 1. varijanta - skolska
         //bool uspehSnimanja= false;

         SqlConnection Veza = new SqlConnection(pStringKonekcije);
         Veza.Open();

         SqlCommand Komanda = new SqlCommand("ObrisiZvanje", Veza);
         Komanda.CommandType = CommandType.StoredProcedure;
         Komanda.Parameters.Add("@Sifra",                SqlDbType.Char).Value                =
objZvanjeZaBrisanje.Sifra;
         brojSlogova = Komanda.ExecuteNonQuery();
         Veza.Close();
         Veza.Dispose();

         return (brojSlogova > 0);

      }

      // method overloading - ista procedura sa razlicitim parametrom
      public bool ObrisiZvanje(string SifraZvanjaZaBrisanje)
      {
         // LOKALNE PROMENLJIVE UVEK NA VRHU
         int brojSlogova = 0;
```

```
         // 1. varijanta - skolska
         //bool uspehSnimanja= false;

         SqlConnection Veza = new SqlConnection(pStringKonekcije);
         Veza.Open();

         SqlCommand Komanda = new SqlCommand("ObrisiZvanje", Veza);
         Komanda.CommandType = CommandType.StoredProcedure;
         Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value = SifraZvanjaZaBrisanje;
         brojSlogova = Komanda.ExecuteNonQuery();
         Veza.Close();
         Veza.Dispose();

         return (brojSlogova > 0);

      }

      public bool IzmeniZvanje(clsZvanje objStaroZvanje, clsZvanje objNovoZvanje)
      {
         // LOKALNE PROMENLJIVE UVEK NA VRHU
         int brojSlogova = 0;
         // 1. varijanta - skolska
         //bool uspehSnimanja= false;

         SqlConnection Veza = new SqlConnection(pStringKonekcije);
         Veza.Open();

         SqlCommand Komanda = new SqlCommand("IzmeniZvanje", Veza);
         Komanda.CommandType = CommandType.StoredProcedure;
         Komanda.Parameters.Add("@StaraSifra",          SqlDbType.Char).Value          =
objStaroZvanje.Sifra;
         Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value = objNovoZvanje.Sifra;
         Komanda.Parameters.Add("@Naziv",          SqlDbType.NVarChar).Value          =
objNovoZvanje.Naziv;
         brojSlogova = Komanda.ExecuteNonQuery();
         Veza.Close();
         Veza.Dispose();

         return (brojSlogova > 0);

      }

      // method overloading - ista metoda, samo drugaciji parametri
      public bool IzmeniZvanje(string SifraStarogZvanja, clsZvanje objNovoZvanje)
      {
         // LOKALNE PROMENLJIVE UVEK NA VRHU
         int brojSlogova = 0;
         // 1. varijanta - skolska
         //bool uspehSnimanja= false;

         SqlConnection Veza = new SqlConnection(pStringKonekcije);
         Veza.Open();
```

```
        SqlCommand Komanda = new SqlCommand("IzmeniZvanje", Veza);
        Komanda.CommandType = CommandType.StoredProcedure;
        Komanda.Parameters.Add("@StaraSifra", SqlDbType.Char).Value = SifraStarogZvanja;
        Komanda.Parameters.Add("@Sifra", SqlDbType.Char).Value = objNovoZvanje.Sifra;
        Komanda.Parameters.Add("@Naziv", SqlDbType.NVarChar).Value =
objNovoZvanje.Sifra;
        brojSlogova = Komanda.ExecuteNonQuery();
        Veza.Close();
        Veza.Dispose();

        return (brojSlogova > 0);

    }

  }
```

Ceo programski kod korisničkog interfejsa, za realizaciju svih osnovnih funkcija dat je u nastavku. S obzirom da je realizovana i štampa, postupak za kreiranje report fajla koji se ovde koristi nije ovde dato, već je opisano u odeljku pod naslovom “Rad za izveštajima”.

```
//
using KlasePodataka;

namespace WindowsFormsApplication1
{
    public partial class frmZvanje : Form
    {
       // atributi
        private clsZvanjeDB objZvanjeDB;
        DataSet podaciGrid;

        string stanjePrograma="";
        //
        DataSet dsPodaciNavigacije;
        int pozicijaNavigacije;
        int UkupnoZapisaNavigacije = 0;
        //
        clsZvanje objStaroZvanje;


        // konstruktor
        public frmZvanje()
        {
            InitializeComponent();
        }

        // nase metode
        private void NapuniGrid(DataSet dsPodaci)
        {
            dgvSpisakZvanja.DataSource = dsPodaci.Tables[0];
            dgvSpisakZvanja.Refresh();
        }

        private void IsprazniKontrole()
        {
            txbSifra.Text = "";
```

```
            txbNaziv.Text = "";
        }

        private void PostaviStanjeKontrola(string stanje)
        {
            switch (stanje)
            {
            case "unos":
                    IsprazniKontrole();
                    txbSifra.Enabled =true;
                    txbNaziv.Enabled =true;
                    //
                    btnUnos.Enabled = false;
                    btnIzmena.Enabled = false;
                    btnBrisanje.Enabled = false;
                    btnPrvi.Enabled = false;
                    btnPrethodni.Enabled = false;
                    btnSledeci.Enabled = false;
                    btnPoslednji.Enabled = false;
                    btnPotvrdi.Enabled = true;
                    btnOdustani.Enabled = true;
                    //
                    txbSifra.Focus();
                    break;
            case "izmena":
                    txbSifra.Enabled = true;
                    txbNaziv.Enabled = true;
                    //
                    btnUnos.Enabled = false;
                    btnIzmena.Enabled = false;
                    btnBrisanje.Enabled = false;
                    btnPrvi.Enabled = false;
                    btnPrethodni.Enabled = false;
                    btnSledeci.Enabled = false;
                    btnPoslednji.Enabled = false;
                    btnPotvrdi.Enabled = true;
                    btnOdustani.Enabled = true;
                    //
                    txbSifra.Focus();
                    break;
            case "prikaz":
                    txbSifra.Enabled =false;
                    txbNaziv.Enabled =false;
                    //
                    btnUnos.Enabled = true;
                    btnIzmena.Enabled = true;
                    btnBrisanje.Enabled = true;
                    btnPrvi.Enabled = true;
                    btnPrethodni.Enabled = true;
                    btnSledeci.Enabled = true;
                    btnPoslednji.Enabled = true;
                    btnPotvrdi.Enabled = false;
                    btnOdustani.Enabled = false;
                    //
                    btnUnos.Focus();
                    break;
            case "brisanje":
                    txbSifra.Enabled = false;
                    txbNaziv.Enabled = false;
                    //
                    btnUnos.Enabled = false;
                    btnIzmena.Enabled = false;
```

```
                btnBrisanje.Enabled = false;
                btnPrvi.Enabled = false;
                btnPrethodni.Enabled = false;
                btnSledeci.Enabled = false;
                btnPoslednji.Enabled = false;
                btnPotvrdi.Enabled = true;
                btnOdustani.Enabled = true;
                //
                btnUnos.Focus();
                break;
            }
        }

        private void PrikaziPodatke(int pozicija)
        {
            txbSifra.Text = dsPodaciNavigacije.Tables[0].Rows[pozicija].ItemArray[0].ToString();
            txbNaziv.Text = dsPodaciNavigacije.Tables[0].Rows[pozicija].ItemArray[1].ToString();
        }




        private void UcitajPodatkeZaNavigaciju()
        {
            dsPodaciNavigacije = objZvanjeDB.DajSvaZvanja();
            pozicijaNavigacije = 0;
            UkupnoZapisaNavigacije = dsPodaciNavigacije.Tables[0].Rows.Count;
        }

        private void PrikaziSvePodatkeUGridu()
        {
            txbFilter.Text = "";
            cmbZnak.Text = "";
            podaciGrid = objZvanjeDB.DajSvaZvanja();
            NapuniGrid(podaciGrid);
        }

        // dogadjaji
        private void frmZvanje_Load(object sender, EventArgs e)
        {
            // OVO JE ZA STAMPU, AUTOMATSKI SE UBACUJE KADA SE DODA IZVESTAJ
            this.ZVANJETableAdapter.Fill(this.ZvanjeDataSet.ZVANJE);

            // OVO JE ZA ZAJEDNICKO KORISCENJE U SVIM PROCEDURAMA
            objZvanjeDB = new clsZvanjeDB(Parametri.stringKonekcije);

            podaciGrid = new DataSet();
            // podaci za prvu karticu, za navigaciju
            dsPodaciNavigacije = new DataSet();
            UcitajPodatkeZaNavigaciju();
            // za izmenu
            objStaroZvanje = new clsZvanje();
            stanjePrograma = "prikaz";
            PostaviStanjeKontrola(stanjePrograma);

        }

        private void btnSvi_Click(object sender, EventArgs e)
        {
            PrikaziSvePodatkeUGridu();
        }
```

```
        private void btnFiltriraj_Click(object sender, EventArgs e)
        {
            podaciGrid = objZvanjeDB.DajZvanjaPoNazivu(cmbZnak.Text, txbFilter.Text);
            NapuniGrid(podaciGrid);
        }

        private void btnExportXML_Click(object sender, EventArgs e)
        {
            podaciGrid.WriteXml("ExportZvanja.XML");
            MessageBox.Show("Uspesno eksportovani podaci u datoteku XML formata!");
        }

        private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
        {

            rvIzvestaj.RefreshReport();
        }

        private void btnUnos_Click(object sender, EventArgs e)
        {
            stanjePrograma = "unos";
            PostaviStanjeKontrola(stanjePrograma);

        }

        private void btnPotvrdi_Click(object sender, EventArgs e)
        {
            // sa korisnickog interfejsa uzimamo podatke i stavljamo u objekat Zvanje
            // potrebno je za sve 3 operacije sa podacima, jer je argument poziva metode
            clsZvanje objZvanje = new clsZvanje();
            objZvanje.Sifra = txbSifra.Text;
            objZvanje.Naziv = txbNaziv.Text;
            string poruka ="";
            try
            {
                switch (stanjePrograma)
                {
                    case "unos":
                        objZvanjeDB.SnimiNovoZvanje(objZvanje);
                        poruka = "Uspesno snimljeni podaci o novom zvanju!";
                        break;
                    case "izmena":
                        objZvanjeDB.IzmeniZvanje(objStaroZvanje, objZvanje);
                        poruka = "Uspesno izmenjeni podaci o zvanju!:";
                        break;
                    case "brisanje":
                        objZvanjeDB.ObrisiZvanje(objZvanje);
                        poruka = "Uspesno obrisani podaci o zvanju!";
                        break;
                } // kraj switch
            }
            catch (Exception greska)
            {
                MessageBox.Show("Greska:" + greska);
                txbSifra.Focus();
                return;
            }
            // ako je sve u redu, program dolazi do ove tacke:
            MessageBox.Show(poruka);
            PostaviStanjeKontrola("prikaz");

            // restartovanje - novo stanje podataka za navigaciju nakon svih promena
```

```
            UcitajPodatkeZaNavigaciju();
            pozicijaNavigacije = 0;
            PrikaziPodatke(pozicijaNavigacije);
            PrikaziSvePodatkeUGridu();
        }

        private void btnPrvi_Click(object sender, EventArgs e)
        {
            pozicijaNavigacije = 0;
            PrikaziPodatke(pozicijaNavigacije);
        }

        private void btnPrethodni_Click(object sender, EventArgs e)
        {
            if (pozicijaNavigacije > 0)
            {
                pozicijaNavigacije--;  //pozicijaNavigacije = pozicijaNavigacije - 1;
                PrikaziPodatke(pozicijaNavigacije);
            }
        }

        private void btnSledeci_Click(object sender, EventArgs e)
        {
            if (pozicijaNavigacije < (UkupnoZapisaNavigacije - 1))
            {
                pozicijaNavigacije++;  //pozicijaNavigacije = pozicijaNavigacije + 1;
                PrikaziPodatke(pozicijaNavigacije);
            }
        }

        private void btnPoslednji_Click(object sender, EventArgs e)
        {
            pozicijaNavigacije = UkupnoZapisaNavigacije - 1;
            PrikaziPodatke(pozicijaNavigacije);
        }

        private void btnBrisanje_Click(object sender, EventArgs e)
        {
            stanjePrograma = "brisanje";
            PostaviStanjeKontrola("brisanje");
        }

        private void btnOdustani_Click(object sender, EventArgs e)
        {
            IsprazniKontrole();
            PostaviStanjeKontrola("prikaz");
        }

        private void btnIzmena_Click(object sender, EventArgs e)
        {
            stanjePrograma = "izmena";

            // ocitavamo vrednost starog zvanja
            objStaroZvanje.Sifra = txbSifra.Text;
            objStaroZvanje.Naziv = txbNaziv.Text;

            PostaviStanjeKontrola(stanjePrograma);
        }
    }
}
```

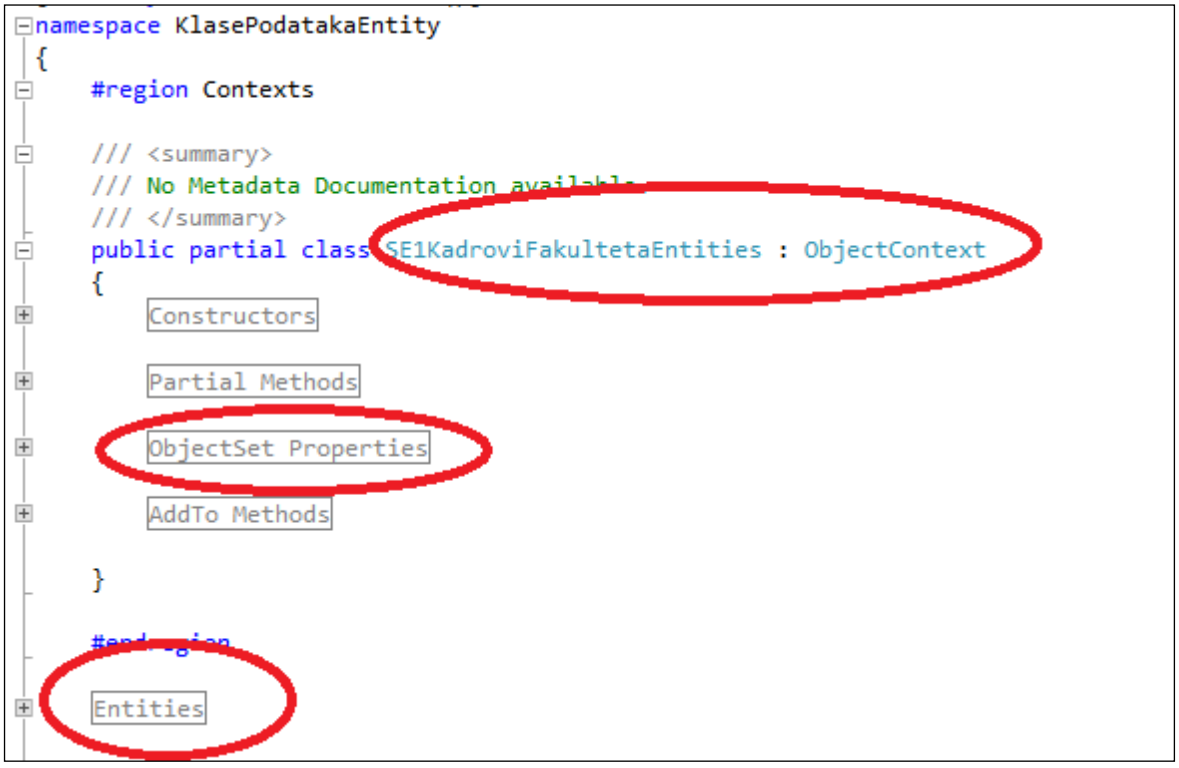
4. zadatak

Postupak kreiranja biblioteke klasa KlasePodatakaEntity koje sadrže generisane klase Entity Frameworka opisan je u odeljku koji se odnosi na rad sa entity framework klasama. U nastavku je data dopuna ove biblioteke klasa programskim kodom klase koja treba da olakša rad sa EntityFramework klasama, kao i programski kod korisničkog interfejsa koji koristi elemente biblioteke klasa KlasePodatakaEntity (generisane klase + dodatnu klasu).

Nastavljamo nakon generisanja Entity Framework klasa.

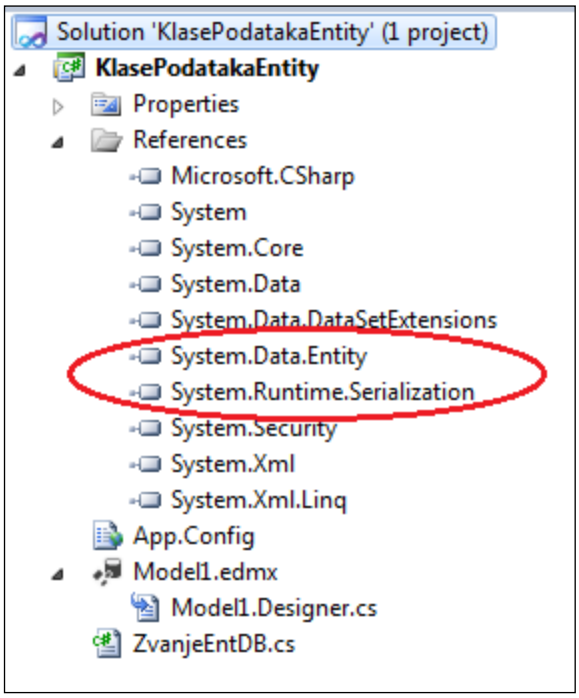
Otvorimo region CONTEXTS.

Bitno je da uočimo kako se naziva Context klasa, jer ona upravlja celinom. Ovde je to: SE1KadroviFakultetaEntities (nasleđuje **ObjectContext**). Takođe bitno je obratiti pažnju na region **Entities**, gde se nalaze glavne klase tipa pojedinac (na osnovu tabela iz baze podataka, imaju samo atribute, tj. property) i na region **Objectset Properties**, gde se nalaze kolekcije objekata koje se koriste kod unosa podataka i slično.



Slika 10.36. Ključni elementi generisanih klasa putem Entity frameworka

Primetićemo u Solution exploreru da su automatski dodate biblioteke u References, koje omogućavaju rad sa Entity framework klasama – System.Data.Entity, System.Runtime.Serialization.



Slika 10.37. Ključne biblioteke klasa za rad sa generisanim Entity framework klasama

Umesto već ranije automatski kreirane prazne klase Class1 pravimo dodatnu klasu koja bi mogla da pojednostavi korišćenje generisanih klasa - nazvaćemo je ZvanjeEntDB. Dodali smo naziv konteksta kao privatni atribut. Osnovna generisana klasa iz regiona Entities se zove ZVANJE, a iz regiona sa kolekcijama objekata se zove ZVANJEs (kreirana s tim nazivom zbog podesavanja SINGULARIZE AND PLURALIZE u toku generisanja klasa. Konačno, kontekst snima sve promene, nakon dodavanja novog objekta.

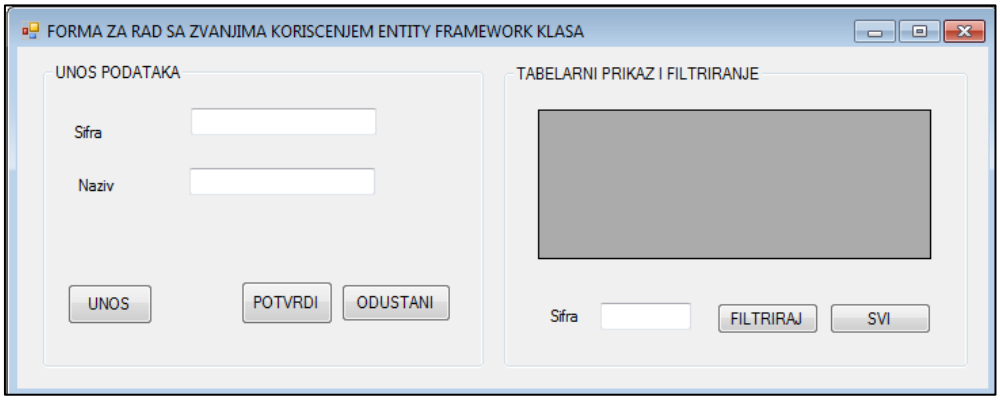
```
namespace KlasePodatakaEntity
{
    public class ZvanjeEntDB
    {
        private SE1KadroviFakultetaEntities pKontekst;

        // konstruktor
        public ZvanjeEntDB(SE1KadroviFakultetaEntities noviKontekst)
        {
            pKontekst = noviKontekst;
        }

        public void DodajNovoZvanje(ZVANJE novoZvanje)
        {
            using (pKontekst)
            {
                ZVANJE zvanjePodaci = new ZVANJE();
                zvanjePodaci.Sifra  = novoZvanje.Sifra;
                zvanjePodaci.NAziv = novoZvanje.NAziv;
                pKontekst.ZVANJEs.AddObject(zvanjePodaci);
                pKontekst.SaveChanges();
            } // kraj using

        } // kraj procedure
```

KOMPLETAN KOD KLASE ZvanjeEntDB:

```
namespace KlasePodatakaEntity
{
   public class ZvanjeEntDB
   {
      private SE1KadroviFakultetaEntities pKontekst;

      // konstruktor
      public ZvanjeEntDB(SE1KadroviFakultetaEntities noviKontekst)
      {
         pKontekst = noviKontekst;
      }

      public void DodajNovoZvanje(ZVANJE novoZvanje)
      {
         using (pKontekst)
         {
            ZVANJE zvanjePodaci = new ZVANJE();
            zvanjePodaci.Sifra  = novoZvanje.Sifra;
            zvanjePodaci.NAziv = novoZvanje.NAziv;
            pKontekst.ZVANJEs.AddObject(zvanjePodaci);
            pKontekst.SaveChanges();
         } // kraj using

      } // kraj procedure
      public List<ZVANJE> UcitajSvaZvanja()
      {
         List<ZVANJE> izlaznaListaZvanja = new List<ZVANJE>();

         using (pKontekst)
         {
            //1.  izdvajanje podataka iz baze podataka, koristimo nesto slicno SQL upitu tipa
select
            // ListaZvanjaEntBaza je netipizirana lista objekata
            var ListaZvanjaEntBaza = from skupZvanja in pKontekst.ZVANJEs select
skupZvanja;

            //2.  presipanje u tipiziranu listu ciji su clanovi objekti klase ZVANJE
            List<ZVANJE> tipiziranaListaZvanja = ListaZvanjaEntBaza.ToList<ZVANJE>();

            //3.  pristupamo svakom clanu tipizirane liste
            // citamo sadrzaj objekta i prebacujemo u nove objekte koji ulaze
            // u izlaznu tipiziranu listu zvanja
            foreach (ZVANJE objZvanje in tipiziranaListaZvanja)
            {
               ZVANJE novoZvanje = new ZVANJE();
               novoZvanje.Sifra = objZvanje.Sifra;
               novoZvanje.NAziv = objZvanje.NAziv;
               izlaznaListaZvanja.Add(novoZvanje);
```

```
            }
        }

        return izlaznaListaZvanja;
    } // kraj metode

    public List<ZVANJE> UcitajZvanjePremaSifri(string novaSifra)
    {
        // OVO JE FILTRIRANO IZDVAJANJE PODATAKA IZ BAZE PODATAKA
        // U TIPIZIRANU LISTU OBJEKATA

        List<ZVANJE> izlaznaListaZvanja = new List<ZVANJE>();

        using (pKontekst)
        {
            // izdvajanje podataka iz baze podataka
            var ListaZvanjaEntBaza = from skupZvanja in pKontekst.ZVANJEs where
skupZvanja.Sifra ==novaSifra select skupZvanja;
            // presipanje u listu
            List<ZVANJE> lista = ListaZvanjaEntBaza.ToList<ZVANJE>();

            foreach (ZVANJE objZvanje in lista)
            {
                ZVANJE novoZvanje = new ZVANJE();
                novoZvanje.Sifra = objZvanje.Sifra;
                novoZvanje.NAziv = objZvanje.NAziv;
                izlaznaListaZvanja.Add(novoZvanje);
            }
        }

        return izlaznaListaZvanja;
    } // kraj metode


} // kraj klase
```

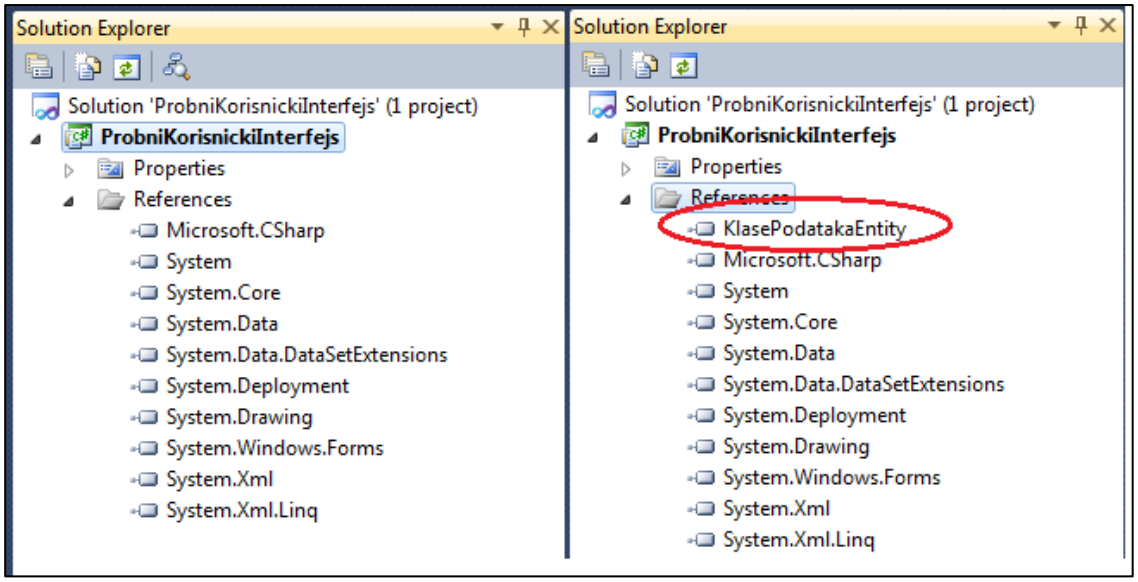
Generisemo DLL (Build). Na korisničkom interfejsu kreiramo jednostavnu formu - sa unosom, tabelarnim prikazom i filtriranjem podataka.



Slika 10.38. Grafički izgled ekranske forme za rad sa zvanjima

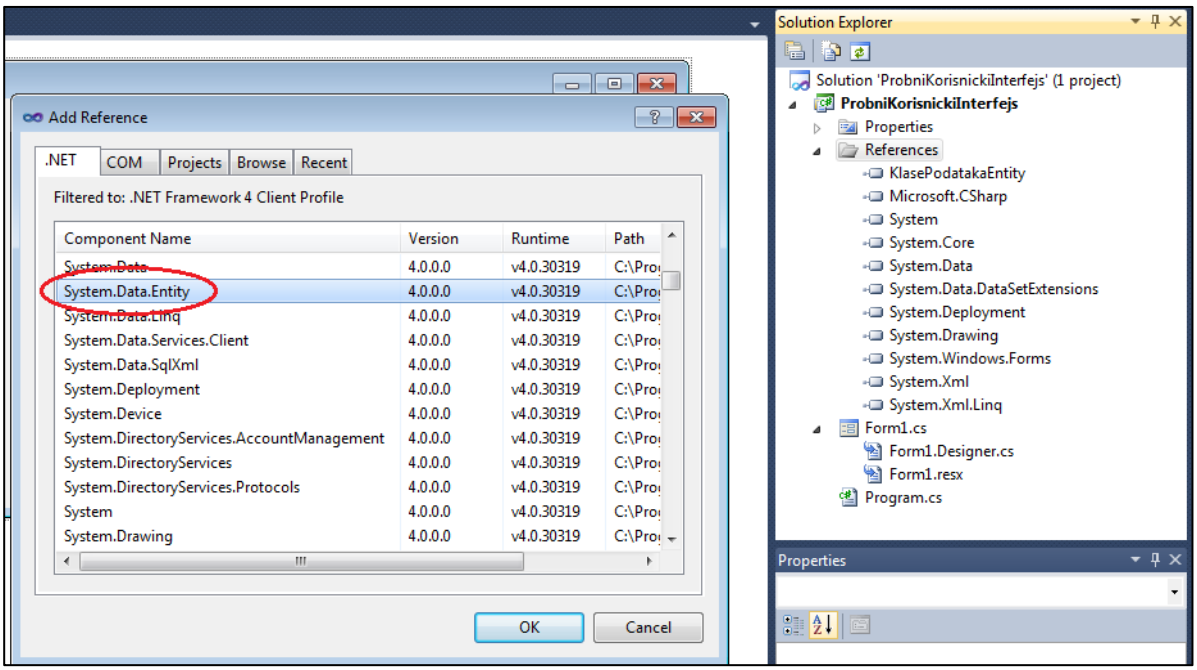
Dodajemo referencu na ranije kreirani KlasePodatakaEntity.dll.

Ako je to forma na novom projektu tipa Windows forms, onda je početna situacija sa skupom raspoloživih biblioteka incijalno prikazana levo, a nakon dodavanja KlasePodatakaEntity.dll biblliioteke desno.

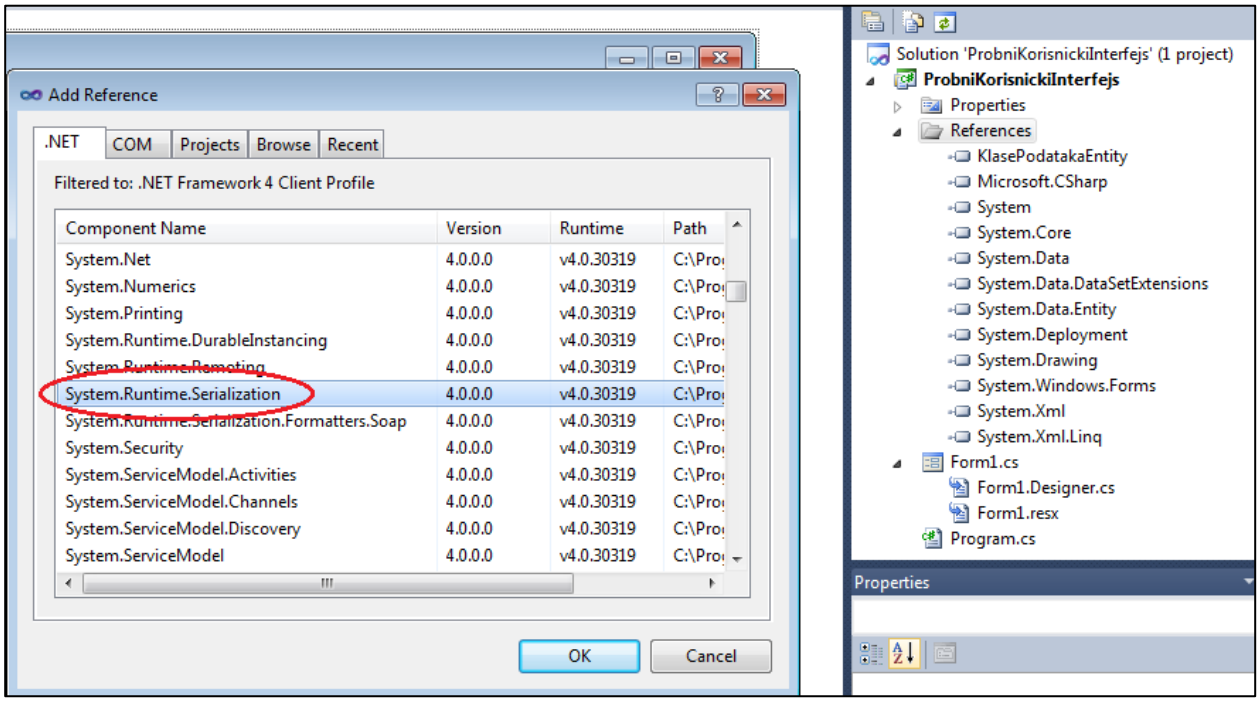


Slika 10.39. Početna situacija sa bibliotekama u references i situacija nakon dodavanja KlasePodatakaEntity.DLL biblioteke klasa

Da bi korisnicki interfejs programa mogao da radi, potrebno je da dodamo i standardne reference koje omogucavaju rad sa Entity framework klasama: System.Data.Entity i System.Runtime.Serialization.



Slika 10.40. Dodavanje System.Data.Entity za rad sa Entity framework klasama



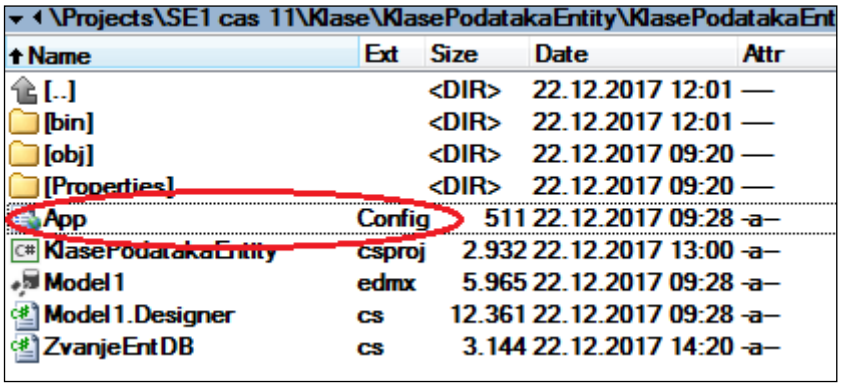
Slika 10.40. Dodavanje System.Runtime.Serialization za rad sa Entity framework klasama

Sad možemo koristiti klase iz KlasePodatakaEntity.dll. Dodajemo u using delu korišćenje te biblioteke klasa.

Najvažnija naredba je instanciranje objekta klase tipa kontekst, a tom prilikom možemo proslediti i string konekcije, kako bi kod mogao da se izvrsava na svakom računaru.

Slika 10.41. Instanciranje kontekst klase

Da bismo procitali string konekcije i koristili ga, pogledaćemo koji je string konekcije u biblioteci klasa KlasePodatakaEntity.dll, u fajlu App.Config.



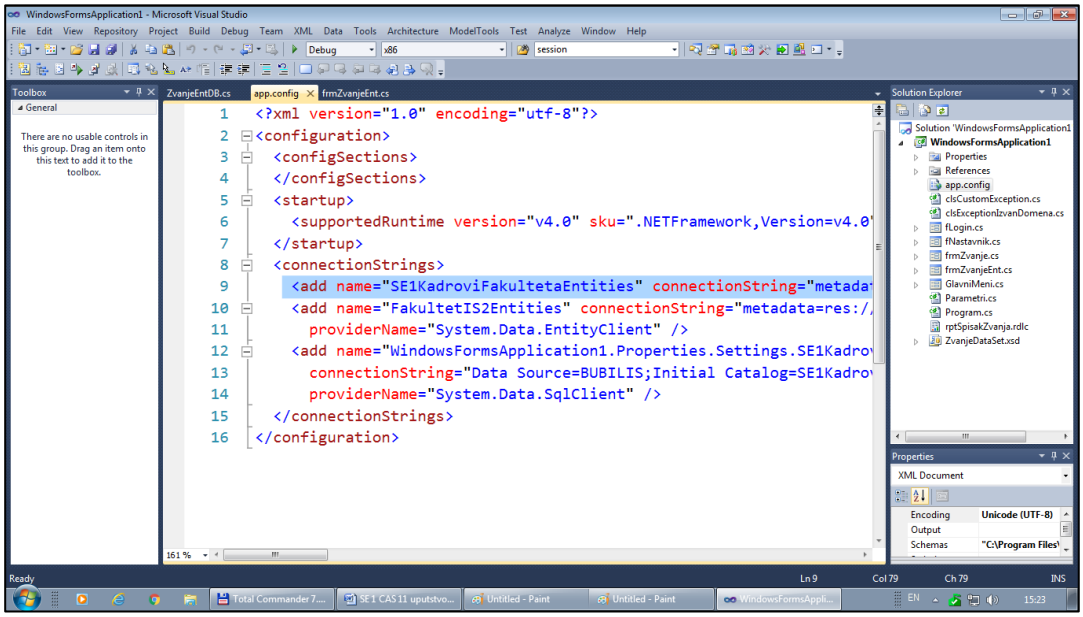
Slika 10.41. Fajl App.Config iz biblioteke klasa gde je kreiran Entity framework skup klasa

Otvorimo ovaj fajl iz Notepad-a.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="SE1KadroviFakultetaEntities"
connectionString="metadata=res://*/Model1.csdl|res://*/Model1.ssdl|
res://*/Model1.msl;provider=System.Data.SqlClient;provider connection
string=&quot;data source=29-01\TFZRSQLSERVER;initial
catalog=SE1KadroviFakulteta;integrated
security=True;multipleactiveresultsets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

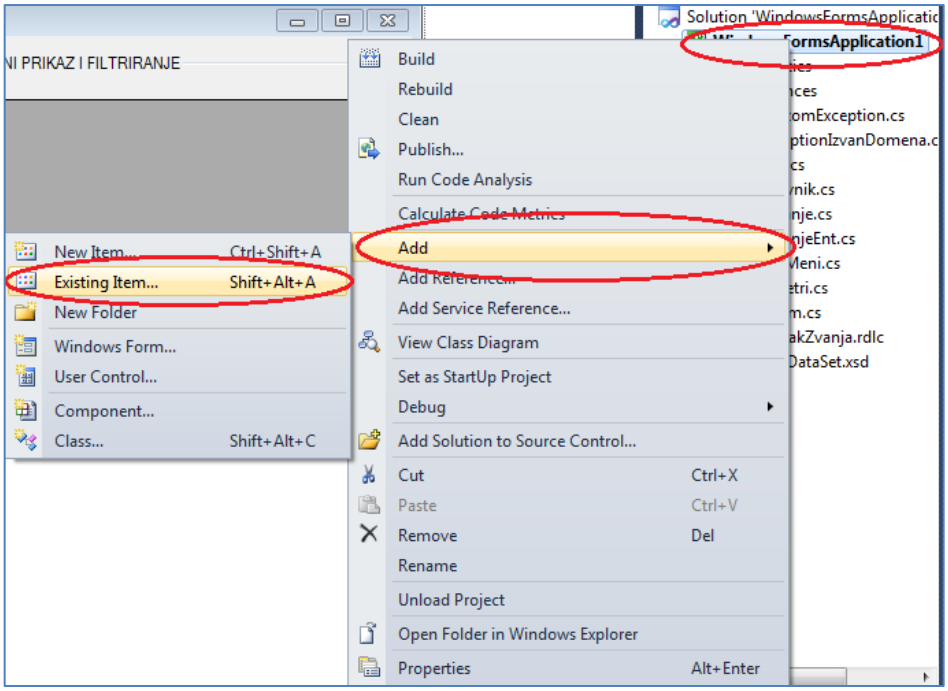
Slika 10.42. Sadržaj App.Config fajla

U app.config fajl korisnickog interfejsa prekopiramo ceo connection string.



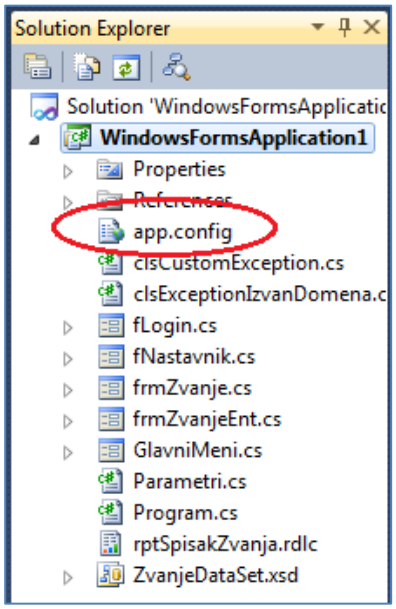
Slika 10.43. Kopiranje stringa konekcije u App.Config korisničkog interfejsa

Ukoliko u windows aplikaciji ne postoji fajl App.config, moze se dodati naknadno. Prekopiracemo App.Config fajl iz foldera biblioteke klasa gde je generisan kod Entity framework klasa u folder gde se nalazi Windows aplikacija (tj. gde se nalaze fajlovi formi i drugo). Koristicemo opciju NAZIVPROJEKTAKORISNICKOGINTERFEJSA (npr. WindowsFormsApplication - ADD - EXISTING ITEM.



Slika 10.44. Dodavanje App.Config fajla opcijom Add-Existing Item

Nakon ubacivanja:



Slika 10.45. Solution Explorer Windows Forms aplikacije nakon dodavanja App.Config fajla

Ubacujemo programski kod na korisnicki interfejs.

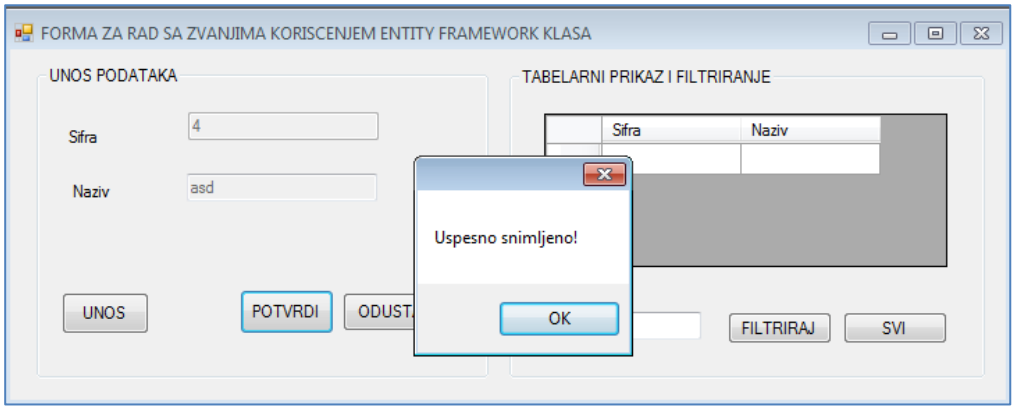
```
//
using KlasePodatakaEntity;

namespace WindowsFormsApplication1
{
   public partial class frmZvanjeEnt : Form
   {
      // atributi
      SE1KadroviFakultetaEntities objKontekst;
      ZvanjeEntDB objZvanjeEntDB;
      List<ZVANJE> listaZvanja = new List<ZVANJE>();

      // konstruktor
      public frmZvanjeEnt()
      {
         InitializeComponent();
         // RUCNO DEFINISEMO STRUKTURU DATA GRID VIEW KONTROLE
         dgvSpisakZvanja.Columns.Add("Sifra", "Sifra");
         dgvSpisakZvanja.Columns.Add("Naziv", "Naziv");

      }

      //nase metode
      private void IsprazniKontrole()
      {
         txbSifra.Text = "";
         txbNaziv.Text = "";
      }

      private void AktivirajKontrole()
      {
         txbSifra.Enabled = true;
         txbNaziv.Enabled = true;
      }

      private void DeaktivirajKontrole()
      {
         txbSifra.Enabled = false;
         txbNaziv.Enabled = false;
      }


      private void PrikaziTabeluPodataka(List<ZVANJE> novaListaZvanja)
      {
         dgvSpisakZvanja.Rows.Clear();

         for (int i = 0; i < novaListaZvanja.Count; i++)
         {
            dgvSpisakZvanja.Rows.Add();
            dgvSpisakZvanja.Rows[i].Cells["Sifra"].Value = novaListaZvanja[i].Sifra ;
```

```
            dgvSpisakZvanja.Rows[i].Cells["Naziv"].Value = novaListaZvanja[i].NAziv ;
        }
    }




    // dogadjaji
    private void btnUnos_Click(object sender, EventArgs e)
    {
        IsprazniKontrole();
        AktivirajKontrole();
        txbSifra.Focus();

    }

    private void btnPotvrdi_Click(object sender, EventArgs e)
    {
        string poruka = "";
        try
        {
            // preuzimanje vrednosti sa korisnickog interfejsa
            ZVANJE objNovoZvanje = new ZVANJE();
            objNovoZvanje.Sifra = txbSifra.Text;
            objNovoZvanje.NAziv  = txbNaziv.Text;

            // dodavanje novog zvanja u kolekciju i snimanje
            // 1. nacin - DIREKTNO KORISCENJE KLASA GENERISANIH U ENTITY MODELU
            //objKontekst.ZVANJAs.AddObject(objNovoZvanje);
            //objKontekst.SaveChanges();

            //2. nacin - POSREDNA KLASA DrzavaEntDB
            objZvanjeEntDB.DodajNovoZvanje (objNovoZvanje);

            poruka = "Uspesno snimljeno!";
        }
        catch (Exception ex)
        {
            poruka = "Nije uspesno snimljeno! Greska:" + ex.Message;
        }

        DeaktivirajKontrole();
        MessageBox.Show(poruka);

    }

    private void bnOdustani_Click(object sender, EventArgs e)
    {
        IsprazniKontrole();
        DeaktivirajKontrole();

    }
```

```
        private void btnFiltriraj_Click(object sender, EventArgs e)
        {
            // ucitavanje podataka iz baze preko entity modela u listu
            listaZvanja = objZvanjeEntDB.UcitajZvanjePremaSifri(txbSifra.Text) ;
            // prikaz u gridu
            PrikaziTabeluPodataka(listaZvanja);

        }

        private void btnSvi_Click(object sender, EventArgs e)
        {
            // ucitavanje podataka iz baze preko entity modela u listu
            listaZvanja = objZvanjeEntDB.UcitajSvaZvanja();
            // prikaz u gridu
            PrikaziTabeluPodataka(listaZvanja);

        }

        private void frmZvanjeEnt_Load(object sender, EventArgs e)
        {
            objKontekst = new SE1KadroviFakultetaEntities();
            objZvanjeEntDB = new ZvanjeEntDB(objKontekst);

        }
    }
}
```

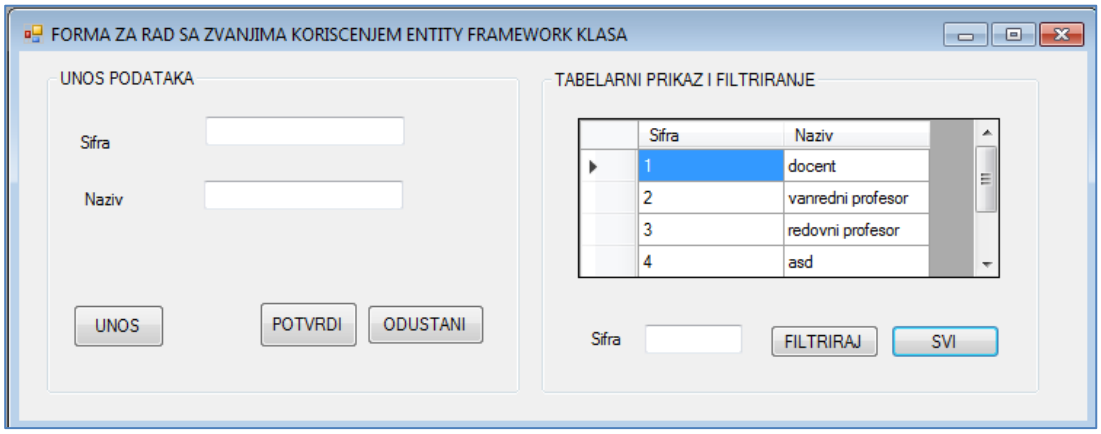
REZULTATI:

Testiramo unos.



Slika 10.46. Rezultat testiranja unosa u 4. zadatku

Testiramo tabelarni prikaz:



Slika 10.46. Rezultat testiranja tabelarnog prikaza u 4. zadatku

5. zadatak

1. BAZA PODATAKA - KREIRANJE TABELE NASTAVNIK I POVEZIVANJE SA TABELOM ZVANJE

```
USE [SE1KadroviFakulteta]
GO

CREATE TABLE [dbo].[NASTAVNIK](
    [JMBG] [int] NOT NULL,
    [Prezime] [nvarchar](40) NOT NULL,
    [Ime] [nvarchar](40) NOT NULL,
    [IDZvanja] [char](3) NOT NULL
)
GO

ALTER TABLE [dbo].[Nastavnik] ADD CONSTRAINT
[FK_Nastavnik_Zvanje] FOREIGN KEY([IDZvanja])
REFERENCES [dbo].[Zvanje] ([Sifra])
ON UPDATE CASCADE
GO
```

NAKNADNO DODAJEMO POLJE DATUM RODJENJA

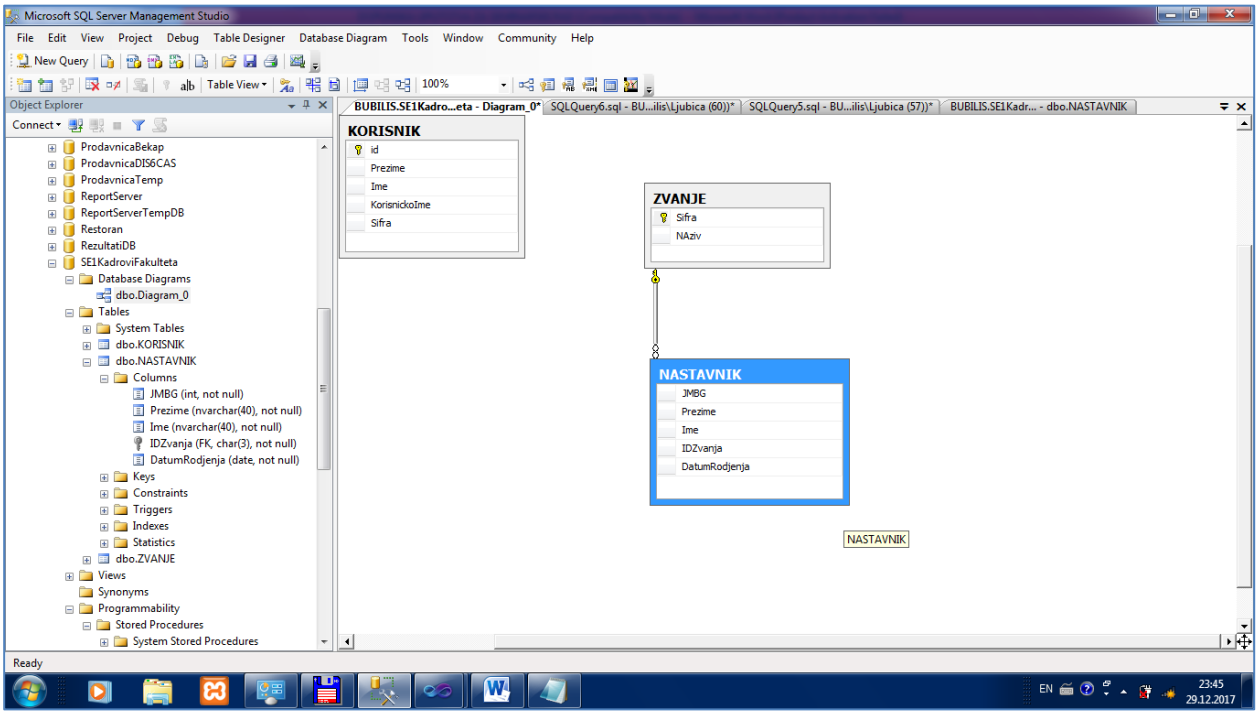
```
USE [SE1KadroviFakulteta]
GO

ALTER TABLE [dbo].[Nastavnik] ADD [DatumRodjenja] [Date] NOT NULL
```

2. BAZA PODATAKA - KREIRANJE STORED PROCEDURA

```
USE [SE1KadroviFakulteta]
GO

CREATE PROCEDURE [DajSveNastavnike]
AS
Select * from Nastavnik
GO

CREATE PROCEDURE [DajSveNastavnikeSaJoin]
AS
Select Nastavnik.JMBG, Nastavnik.Prezime, Nastavnik.Ime, Zvanje.Naziv as NazivZvanja,
Nastavnik.DatumRodjenja from Nastavnik inner join Zvanje on Nastavnik.IdZvanja = Zvanje.Sifra
```

```
GO

CREATE PROCEDURE [DajSveNastavnikeSaJoinSifromZvanja]
AS
Select Nastavnik.JMBG, Nastavnik.Prezime, Nastavnik.Ime, Zvanje.Naziv as NazivZvanja, Zvanje.Sifra as
SifraZvanja, Nastavnik.DatumRodjenja  from Nastavnik inner join Zvanje on Nastavnik.IdZvanja

= Zvanje.Sifra
GO

CREATE PROCEDURE [DajNastavnikaPoPrezimenu] @NastavnikPrezime nvarchar(30)
AS
select NASTAVNIK.JMBG, NASTAVNIK.Ime, NASTAVNIK.Prezime, ZVANJE.Naziv as NazivZvanja,
Nastavnik.DatumRodjenja from NASTAVNIK inner join ZVANJE on ZVANJE.Sifra = NASTAVNIK.IDZvanja

where NASTAVNIK.Prezime = @NastavnikPrezime
GO

CREATE PROCEDURE [DodajNovogNastavnika](
@JMBG int,
@Prezime nvarchar(40),
@Ime nvarchar(40),
@IDZvanja char(3),
@DatumRodjenja DateTime)
AS
BEGIN
Insert into Nastavnik(JMBG, Prezime, Ime, IDZvanja, DatumRodjenja) values (@JMBG, @Prezime, @Ime,
@IDZvanja, @DatumRodjenja)
END
GO

CREATE PROCEDURE [ObrisiNastavnika](
@JMBG int)
AS
BEGIN
Delete from Nastavnik where JMBG=@JMBG
END
GO

CREATE PROCEDURE [IzmeniNastavnika](
@StariJMBG int,
@JMBG int,
@Prezime nvarchar(40),
@Ime nvarchar(40),
@IDZvanja char(3),
@DatumRodjenja DateTime)
AS
BEGIN
Update Nastavnik set JMBG=@JMBG, Prezime=@Prezime, Ime=@Ime, IDZvanja=@IDZvanja,
DatumRodjenja=@DatumRodjenja where JMBG=@StariJMBG
END
GO
```

KREIRANA BAZA PODATAKA SA 2 TABELE POVEZANE RELACIJOM I STORED PROCEDURE



Slika 10.47. Shema kreirane baze podataka sa 2 tabele povezane relacijom i tabelom korisnik

3. DOPUNA BIBLIOTEKE KLASE PODATAKA – za tabelu NASTAVNIK kreiramo 3 klase

DOPUNA KLASE NASTAVNO OSOBLJE, DODAJEMO JMBG

```
namespace KlasePodataka
{
    public class clsNastavnoOsoblje
    {
        // atributi
        private string pJMBG;
        private string pPrezime;
        private string pIme;
        private DateTime pDatumVremeRodjenja;


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

        // public property

        public string Prezime
        {
            get
            {
                return pPrezime;
```

```
            }
            set
            {
                if (this.pPrezime != value)
                    this.pPrezime = value;
            }
        }

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


        public string JMBG
        {
            get
            {
                return pJMBG;
            }
            set
            {
                if (this.pJMBG != value)
                    this.pJMBG = value;
            }
        }


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

        // privatne metode

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

            return Datum;
        }

        // public metode
```

```
        public virtual string DajPodatke()
        {
            return this.pPrezime + " " + this.pIme + " " + this.pJMBG + " " + this.DajDatum();
        }


    }
```

KLASA NASTAVNIK

```
namespace KlasePodataka
{
    public class clsNastavnik: clsNastavnoOsoblje
    {
        // atributi
        private clsZvanje pZvanje;

        // konstruktor
        public clsNastavnik(): base ()
        {
            // ******** konstruktor inicijalizuje sve promenljive
            //
            //pozivom base () pokrece se konstruktor bazne klase
            // ovo je dodatak u odnosu na ono sto se desava u konstruktoru bazne klase
            //
            clsZvanje objZvanje = new clsZvanje();
            pZvanje = objZvanje;
        }

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

        // privatne metode

        // javne metode

        public override string DajPodatke()
        {
            return base.DajPodatke () + " " +  pZvanje.DajPodatke ();
        }

    }
}
```

KLASA NASTAVNIK LISTA

```
namespace KlasePodataka
{
    public class clsNastavnikLista
    {
        // atributi
        private List<clsNastavnik> pListaNastavnika;
```

```
        // property
        public List<clsNastavnik> ListaNastavnika
        {
            get
            {
                return pListaNastavnika;
            }
            set
            {
                if (this.pListaNastavnika != value)
                    this.pListaNastavnika = value;
            }
        }

        // konstruktor
        public clsNastavnikLista()
        {
            pListaNastavnika = new List<clsNastavnik>();

        }

        // privatne metode

        // javne metode
        public void DodajElementListe(clsNastavnik objNoviNastavnik)
        {
            pListaNastavnika.Add(objNoviNastavnik);
        }

        public void ObrisiElementListe(clsNastavnik objNastavnikZaBrisanje)
        {
            pListaNastavnika.Remove(objNastavnikZaBrisanje);
        }

        public void ObrisiElementNaPoziciji(int pozicija)
        {
            pListaNastavnika.RemoveAt(pozicija);
        }

        public void IzmeniElementListe(clsNastavnik objStariNastavnik, clsNastavnik objNoviNastavnik)
        {
            int indexStarogZvanja = 0;
            indexStarogZvanja = pListaNastavnika.IndexOf(objStariNastavnik);
            pListaNastavnika.RemoveAt(indexStarogZvanja);
            pListaNastavnika.Insert(indexStarogZvanja, objNoviNastavnik);
        }

    }
}
```

KLASA NASTAVNIK DB

```
//
using System.Data;
using System.Data.SqlClient;

namespace KlasePodataka
{
    public class clsNastavnikDB
    {
         // atributi
        private string pStringKonekcije;
```

```
        // property
        // 1. nacin
        public string StringKonekcije
        {
            get
            {
                return pStringKonekcije;
            }
            set // OVO NIJE DOBRO, MOZE SE STRING KONEKCIJE STAVITI NA PRAZAN STRING
            {
                if (this.pStringKonekcije != value)
                    this.pStringKonekcije = value;
            }
        }
        // konstruktor
        // 2. nacin prijema vrednosti stringa konekcije spolja i dodele atributu
        public clsNastavnikDB(string NoviStringKonekcije)
        // OVO JE DOBRO JER OBAVEZUJE DA SE PRILIKOM INSTANCIRANJA OVE KLASE
        // MORA OBEZBEDITI STRING KONEKCIJE
        {
            pStringKonekcije = NoviStringKonekcije;
        }

        // privatne metode

        // javne metode
        public DataSet DajSveNastavnike()
        {
            DataSet dsPodaci = new DataSet();

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();
            SqlCommand Komanda = new SqlCommand("DajSveNastavnike", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = Komanda;
            da.Fill(dsPodaci);
            Veza.Close();
            Veza.Dispose();

            return dsPodaci;
        }

        public DataSet DajNastavnikePoPrezimenu(string PrezimeNastavnika)
        {
            DataSet dsPodaci = new DataSet();

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();
            SqlCommand Komanda = new SqlCommand("DajNastavnikaPoPrezimenu", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@NastavnikPrezime", SqlDbType.NVarChar).Value =
PrezimeNastavnika;
            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = Komanda;
            da.Fill(dsPodaci);
            Veza.Close();
            Veza.Dispose();

            return dsPodaci;
        }
```

```
        // overloading metoda - isto se zove, ima drugaciji parametar
        public DataSet DajNastavikePoPrezimenu(clsNastavnik objNastavnikaZaFilter)
        {
            // MOGU biti jos neke procedure, mogu SE VRATITI VREDNOSTI I U LISTU, DATA TABLE...
            DataSet dsPodaci = new DataSet();

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();
            SqlCommand Komanda = new SqlCommand("DajNastavnikaPoPrezimenu", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@NastavnikPrezime", SqlDbType.NVarChar).Value =
objNastavnikaZaFilter.Prezime;
            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = Komanda;
            da.Fill(dsPodaci);
            Veza.Close();
            Veza.Dispose();

            return dsPodaci;
        }

        // overloading metoda - 2 parametra - dodat znak
        public DataSet DajNastavnikePoPrezimenu(string znak, string PrezimeNastavnika)
        {
            DataSet dsPodaci = new DataSet();

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();
            string uslov = "";
            if (znak.Equals ("="))
            {
                uslov = "Prezime='" + PrezimeNastavnika + "'";
            }
            else
            {
                uslov = "Prezime like '%" + PrezimeNastavnika + "%'";
            }

            SqlCommand Komanda = new SqlCommand("select * from Nastavnik where " + uslov, Veza);

            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = Komanda;
            da.Fill(dsPodaci);
            Veza.Close();
            Veza.Dispose();

            return dsPodaci;
        }

        public bool SnimiNovogNastavnika(clsNastavnik objNoviNastavnik)
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("DodajNovogNastavnika", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
```

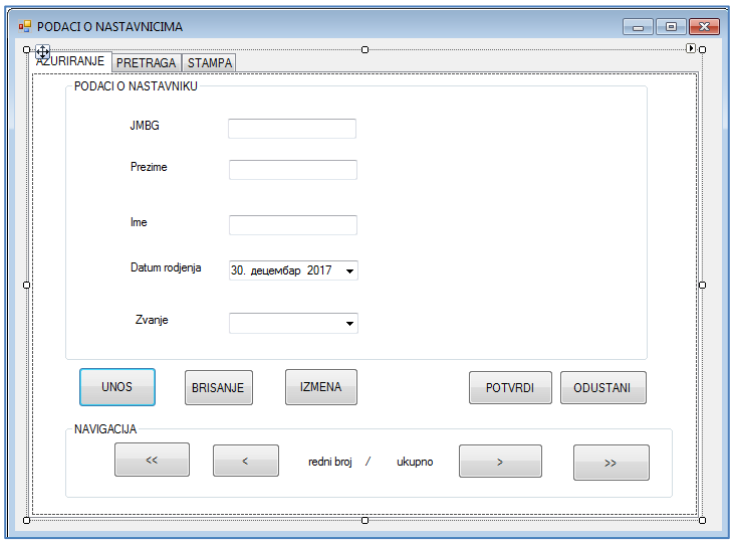
```
            Komanda.Parameters.Add("@JMBG", SqlDbType.Int).Value = int.Parse(objNoviNastavnik.JMBG);
            Komanda.Parameters.Add("@Ime", SqlDbType.NVarChar).Value = objNoviNastavnik.Ime;
            Komanda.Parameters.Add("@Prezime", SqlDbType.NVarChar).Value = objNoviNastavnik.Prezime;
            Komanda.Parameters.Add("@IDZvanja", SqlDbType.Char).Value =
objNoviNastavnik.Zvanje.Sifra;
            Komanda.Parameters.Add("@DatumRodjenja", SqlDbType.DateTime ).Value =
objNoviNastavnik.DatumVremeRodjenja ;

            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            // NEGATIVNO PITANJE - NIJE DOBRO if (brojSlogova ==0)
            /* 1. VARIJANTA - SKOLSKA
             * if (brojSlogova>0)
            {
                uspehSnimanja=true;
            }
            else
            {
                uspehSnimanja=false;
            }

            return uspehSnimanja;
             */

            // 2. varijanta
            return (brojSlogova > 0);

            //3. varijanta
            // NEMA SMISLA, ISTO KAO 2. VARIJANTA
            //return (brojSlogova > 0) ? true : false;

            //4. varijanta - NEGACIJA NEGACIJE, NIJE RAZUMLJIVO
            //return (brojSlogova == 0) ? false : true;

        }

        public bool ObrisiNastavnika(clsNastavnik objNastavnikZaBrisanje)
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("ObrisiNastavnika", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@JMBG", SqlDbType.Int).Value = objNastavnikZaBrisanje.JMBG;
            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            return (brojSlogova > 0);

        }

        // method overloading - ista procedura sa razlicitim parametrom
        public bool ObrisiNastavnika(string JMBGNastavnikaZaBrisanje)
        {
```

```
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("ObrisiNastavnika", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@JMBG", SqlDbType.Int).Value = JMBGNastavnikaZaBrisanje;
            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            return (brojSlogova > 0);

        }

        public bool IzmeniNastavnika(clsNastavnik objStariNastavnik, clsNastavnik objNoviNastavnik)
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("IzmeniNastavnika", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
            Komanda.Parameters.Add("@StariJMBG", SqlDbType.Int).Value =
int.Parse(objStariNastavnik.JMBG);
            Komanda.Parameters.Add("@JMBG", SqlDbType.Int).Value = int.Parse(objNoviNastavnik.JMBG);
            Komanda.Parameters.Add("@Prezime", SqlDbType.NVarChar).Value = objNoviNastavnik.Prezime;
            Komanda.Parameters.Add("@Ime", SqlDbType.NVarChar).Value = objNoviNastavnik.Ime;
            Komanda.Parameters.Add("@IDZvanja", SqlDbType.Char).Value =
objNoviNastavnik.Zvanje.Sifra;
            Komanda.Parameters.Add("@DatumRodjenja", SqlDbType.DateTime).Value =
objNoviNastavnik.DatumVremeRodjenja;

            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            return (brojSlogova > 0);

        }

        // method overloading - ista metoda, samo drugaciji parametri
        public bool IzmeniNastavnika(string JMBGStarogNastavnika, clsNastavnik objNoviNastavnik)
        {
            // LOKALNE PROMENLJIVE UVEK NA VRHU
            int brojSlogova = 0;
            // 1. varijanta - skolska
            //bool uspehSnimanja= false;

            SqlConnection Veza = new SqlConnection(pStringKonekcije);
            Veza.Open();

            SqlCommand Komanda = new SqlCommand("IzmeniNastavnika", Veza);
            Komanda.CommandType = CommandType.StoredProcedure;
```

```
            Komanda.Parameters.Add("@StariJMBG", SqlDbType.Int).Value =
int.Parse(JMBGStarogNastavnika);
            Komanda.Parameters.Add("@JMBG", SqlDbType.Int).Value = int.Parse(objNoviNastavnik.JMBG);
            Komanda.Parameters.Add("@Prezime", SqlDbType.NVarChar).Value = objNoviNastavnik.Prezime;
            Komanda.Parameters.Add("@Ime", SqlDbType.NVarChar).Value = objNoviNastavnik.Ime;
            Komanda.Parameters.Add("@IDZvanja", SqlDbType.Char).Value =
objNoviNastavnik.Zvanje.Sifra;
            Komanda.Parameters.Add("@DatumRodjenja", SqlDbType.DateTime).Value =
objNoviNastavnik.DatumVremeRodjenja;

            brojSlogova = Komanda.ExecuteNonQuery();
            Veza.Close();
            Veza.Dispose();

            return (brojSlogova > 0);

        }

    }
```

4. KORISNIČKI INTERFEJS

DORADA IZGLEDA KORISNIČKOG INTERFEJSA TAKO DA IMA SVA POTREBNA POLJA



Slika 10.48. Korisnički interfejs stranice za rad sa Nastavnicima

PROGRAMSKI KOD ZA REALIZACIJU 1. i 2. kartice

```
//
using System.IO;
using System.Xml;
using KlasePodataka;

namespace WindowsFormsApplication1
{
   public partial class fNastavnik : Form
   {
      // globalne promenljive - atributi klase
      clsZvanjeDB objZvanjeDB;
      clsNastavnikDB objNastavnikDB;
      DataSet dsPodaciGrid;

      string stanjePrograma = "";
      //
```

```
        DataSet dsPodaciNavigacije;
        int pozicijaNavigacije;
        int UkupnoZapisaNavigacije = 0;
        //
        clsNastavnik objStariNastavnik;

        // konstruktor
        public fNastavnik()
        {
            InitializeComponent();
            //System.Data.DataSet objDataSet = new System.Data.DataSet();
           // objDataSet.
        }

        // nase procedure
        private void IsprazniKontrole()
        {
            // brisanje sadrzaja kontrola
            txbJMBG.Text = "";
            txbPrezime.Text = "";
            txbIme.Text = "";
            cmbZvanje.Text = "";
            // s obzirom da ne moze da se isprazni, stavlja se na inicijalnu vrednost
            dtpDatumRodjenja.Value = DateTime.Now;
        }

        //-------------- RAD SA XML -------------------------------
        private DataSet PreuzmiPodatkeIzXML(string putanjaXMLFajla, string NazivXMLFajla)
        {
            DataSet dsPodaci = new DataSet() ;
            dsPodaci.ReadXml(putanjaXMLFajla + "\\" + NazivXMLFajla);
            return dsPodaci;
        }

        private DataSet PreuzmiPodatkeIzXML(string NazivXMLFajla)
        {
            DataSet dsPodaci = new DataSet();
            dsPodaci.ReadXml(NazivXMLFajla);
            return dsPodaci;
        }

        private XmlDocument PreuzmiPodatkeIzXMLuXMLDokument(string putanjaXMLFajla, string NazivXMLFajla)
        {
            XmlDocument doc = new XmlDocument();
            //doc.LoadXml(string XML sadrzaja);
            doc.Load(putanjaXMLFajla + NazivXMLFajla);
            return doc;
        }
        //-----------------RAD SA LISTAMA----------------------------

        private List<string> PreuzmiPodatkeIzDataSetUListuStringova(DataSet dsPodaci, int pozicijaPoljaUDataSetu)
        {
            List<string> listaNaziva = new List<string>();

            for (int brojac = 0; brojac < dsPodaci.Tables[0].Rows.Count; brojac++)
                {
                    listaNaziva.Add(dsPodaci.Tables[0].Rows[brojac].ItemArray[pozicijaPoljaUDataSetu].ToString());
//root.ChildNodes[i].InnerText);
                }

            return listaNaziva;
        }

        //-------------------RAD SA COMBO-------------------------
```

```
private void NapuniComboIzListe(List<string> objListaNaziva)
{
   int maxBroj = objListaNaziva.Count;

   for (int brojac = 0; brojac < maxBroj; brojac++)
   {
      cmbZvanje.Items.Add(objListaNaziva[brojac].ToString());
      cmbZvanjaFilter.Items.Add(objListaNaziva[brojac].ToString());
   };

}

private void NapuniComboIzListeSaForEach(List<string> objListaNaziva)
{
   int maxBroj = objListaNaziva.Count;

   foreach (string naziv in objListaNaziva)
   {
      cmbZvanje.Items.Add(naziv);
      cmbZvanjaFilter.Items.Add(naziv);
   };

}

private void NapuniCombo(DataSet dsPodaciZaCombo)
{
   int maxBroj = dsPodaciZaCombo.Tables[0].Rows.Count;

   for (int brojac=0; brojac<maxBroj; brojac++)
   {
      cmbZvanje.Items.Add(dsPodaciZaCombo.Tables[0].Rows[brojac].ItemArray[1].ToString());
      cmbZvanjaFilter.Items.Add(dsPodaciZaCombo.Tables[0].Rows[brojac].ItemArray[1].ToString());
   };

}

//-------------------PROVERE VREDNOSTI-------------------------
private bool ProveraVrednostiZvanja(string unetaVrednost, DataSet dsPodaci)
{
   // deklaracija i inicijalizacija lokalnih promenljivih
   bool podaciIzDomena = false;
   int maxBroj = dsPodaci.Tables[0].Rows.Count;

   // provera vrednosti
   for (int brojac = 0; brojac < maxBroj; brojac++)
   {
      podaciIzDomena=unetaVrednost.Equals (dsPodaci.Tables[0].Rows[brojac].ItemArray[1].ToString());
      if (podaciIzDomena) break;
   };

   return podaciIzDomena;
}

private void ProveraVrednostiZvanjaPrimenomException(string unetaVrednost, DataSet dsPodaci)
{
   // deklaracija i inicijalizacija lokalnih promenljivih
   bool podaciIzDomena = false;
   int maxBroj = dsPodaci.Tables[0].Rows.Count;

   // provera vrednosti
   for (int brojac = 0; brojac < maxBroj; brojac++)
   {
      podaciIzDomena = unetaVrednost.Equals(dsPodaci.Tables[0].Rows[brojac].ItemArray[1].ToString());
      if (podaciIzDomena) break;
```

```
        };

        if (!podaciIzDomena)
        {
            throw new clsCustomException("PodaciIzvanDomena");
        }

    }

    //---------------------PRERACUNAVANJE VREDNOSTI-------------------------------------------
    private string DajNazivZvanja(DataSet dsPodaciZvanja, string sifraZvanja)
    {
      string pomNazivZvanja="";
      DataRow[] RezultatFiltriranja = dsPodaciZvanja.Tables[0].Select("Sifra='" + sifraZvanja.ToString () + "'");
      pomNazivZvanja = RezultatFiltriranja[0].ItemArray[1].ToString();

      return pomNazivZvanja;

    }

    //---------------------PUNJENJE GRIDA------------------------------
    private void NapuniGrid(DataSet dsPodaci)
    {
        dgvSpisakNastavnika.DataSource = dsPodaci.Tables[0];
        dgvSpisakNastavnika.Refresh();
    }

    private void PrikaziSvePodatkeUGridu()
    {
        txbFilter.Text = "";
        cmbZnak.Text = "";
        // ovde namerno pravimo paralelno 2 promenljive da ih ne bismo povezali: podaciGrid i podaciNavigacija
        dsPodaciGrid = objNastavnikDB.DajSveNastavnike();
        NapuniGrid(dsPodaciGrid);
    }

    //---------------------POSTAVLJANJE STANJA PROGRAMA-----------------------------
    private void PostaviStanjeKontrola(string stanje)
    {
        switch (stanje)
        {
            case "unos":
                IsprazniKontrole();
                txbJMBG.Enabled = true;
                txbPrezime.Enabled = true;
                txbIme.Enabled = true;
                cmbZvanje.Enabled = true;
                dtpDatumRodjenja.Enabled = true;
                //
                btnUnos.Enabled = false;
                btnIzmena.Enabled = false;
                btnBrisanje.Enabled = false;
                btnPrvi.Enabled = false;
                btnPrethodni.Enabled = false;
                btnSledeci.Enabled = false;
                btnPoslednji.Enabled = false;
                btnPotvrdi.Enabled = true;
                btnOdustani.Enabled = true;
                //
                txbJMBG.Focus();
                break;
            case "izmena":
                txbJMBG.Enabled = true;
                txbPrezime.Enabled = true;
```

```
            txbIme.Enabled = true;
            cmbZvanje.Enabled = true;
            dtpDatumRodjenja.Enabled = true;
            //
            btnUnos.Enabled = false;
            btnIzmena.Enabled = false;
            btnBrisanje.Enabled = false;
            btnPrvi.Enabled = false;
            btnPrethodni.Enabled = false;
            btnSledeci.Enabled = false;
            btnPoslednji.Enabled = false;
            btnPotvrdi.Enabled = true;
            btnOdustani.Enabled = true;
            //
            txbJMBG.Focus();
            break;
        case "prikaz":
            txbJMBG.Enabled = false;
            txbPrezime.Enabled = false;
            txbIme.Enabled = false;
            cmbZvanje.Enabled = false;
            dtpDatumRodjenja.Enabled = false;
            //
            btnUnos.Enabled = true;
            btnIzmena.Enabled = true;
            btnBrisanje.Enabled = true;
            btnPrvi.Enabled = true;
            btnPrethodni.Enabled = true;
            btnSledeci.Enabled = true;
            btnPoslednji.Enabled = true;
            btnPotvrdi.Enabled = false;
            btnOdustani.Enabled = false;
            //
            btnUnos.Focus();
            break;
        case "brisanje":
            txbJMBG.Enabled = false;
            txbPrezime.Enabled = false;
            txbIme.Enabled = false;
            cmbZvanje.Enabled = false;
            dtpDatumRodjenja.Enabled = false;
            //
            btnUnos.Enabled = false;
            btnIzmena.Enabled = false;
            btnBrisanje.Enabled = false;
            btnPrvi.Enabled = false;
            btnPrethodni.Enabled = false;
            btnSledeci.Enabled = false;
            btnPoslednji.Enabled = false;
            btnPotvrdi.Enabled = true;
            btnOdustani.Enabled = true;
            //
            btnPotvrdi.Focus();
            break;
    }
}

//--------------------NAVIGACIJA------------------------------

private void UcitajPodatkeZaNavigaciju()
{
    dsPodaciNavigacije = objNastavnikDB.DajSveNastavnike();
    UkupnoZapisaNavigacije = dsPodaciNavigacije.Tables[0].Rows.Count;
}
```

```
        private void PrikaziAzurnePodatkeZaNavigaciju(int pozicijaNavigacije, int UkupnoZapisa)
        {
            // brojac na navigaciji
            lblUkupno.Text = (UkupnoZapisa).ToString();
            lblRedniBroj.Text = (pozicijaNavigacije + 1).ToString();
        }

        private void PrikaziPodatke(DataSet dsPodaci, int pozicija)
        {
            // realne pozicije idu od 0

            txbJMBG.Text = dsPodaci.Tables[0].Rows[pozicija].ItemArray[0].ToString();
            txbPrezime.Text = dsPodaci.Tables[0].Rows[pozicija].ItemArray[1].ToString();
            txbIme.Text = dsPodaci.Tables[0].Rows[pozicija].ItemArray[2].ToString();
            string idZvanja = dsPodaci.Tables[0].Rows[pozicija].ItemArray[3].ToString();
            cmbZvanje.Text = DajNazivZvanja(objZvanjeDB.DajSvaZvanja(), idZvanja);
            dtpDatumRodjenja.Value = DateTime.Parse(dsPodaci.Tables[0].Rows[pozicija].ItemArray[4].ToString());

            PrikaziAzurnePodatkeZaNavigaciju(pozicija, dsPodaci.Tables[0].Rows.Count);
        }

        //---------------------OCITAVANJE ZBOG UNOSA, BRISANJA I IZMENE------------------------------
        private clsNastavnik DajPodatkeIzKontrola()
        {
            clsNastavnik objNoviNastavnik = new clsNastavnik();
            objNoviNastavnik.JMBG = txbJMBG.Text;
            objNoviNastavnik.Prezime = txbPrezime.Text;
            objNoviNastavnik.Ime = txbIme.Text;
            objNoviNastavnik.DatumVremeRodjenja = dtpDatumRodjenja.Value;

            clsZvanje objZvanje = new clsZvanje();
            objZvanje.Naziv = cmbZvanje.Text;
            DataSet dsPodaciZvanja = objZvanjeDB.DajZvanjaPoNazivu(objZvanje.Naziv);
            objZvanje.Sifra = dsPodaciZvanja.Tables[0].Rows[0].ItemArray[0].ToString();

            objNoviNastavnik.Zvanje = objZvanje;

            return objNoviNastavnik;
        }

        //
##########################################################################
######
        //
##########################################################################
######
        // dogadjaji
        private void fNovaForma_Load(object sender, EventArgs e)
        {
            // --------------REPORT-------------------------
            // automatski postavljeno kada se postavi ReportViewer kontrola
            this.rvIzvestaj.RefreshReport();

            // --------------COMBO-------------------------
            // 1. nacin - ucitavanje podataka u combo iz XML
           // NapuniCombo(PreuzmiPodatkeIzXML("", "Zvanja.XML"));

            // 2. nacin - ucitavanje podataka u combo iz XML koristeci tipiziranu listu i foreach ciklus
            //DataSet dsPodaci = PreuzmiPodatkeIzXML(Application.StartupPath, "Zvanja.XML");
            //List<string> listaNazivaZvanja = new List<string>();
            //listaNazivaZvanja = PreuzmiPodatkeIzDataSetuListu(dsPodaci);
            //NapuniComboIzListeSaForEach(listaNazivaZvanja);
```

```
        //3. nacin - pozivanje metode klase clsZvanjeDB koja treba da obezbedi podatke
        DataSet dsPodaci = new DataSet();
        objZvanjeDB = new clsZvanjeDB(Parametri.stringKonekcije);
        dsPodaci = objZvanjeDB.DajSvaZvanja();

        List<string> listaNazivaZvanja = new List<string>();
        listaNazivaZvanja =PreuzmiPodatkeIzDataSetUListuStringova(dsPodaci, 1);
        NapuniComboIzListeSaForEach(listaNazivaZvanja);


        // --------------OPSTI ATRIBUTI ZA CELU STRANICU-------------------------
        objNastavnikDB = new clsNastavnikDB(Parametri.stringKonekcije);

        dsPodaciGrid = new DataSet();
        // podaci za prvu karticu, za navigaciju
        dsPodaciNavigacije = new DataSet();
        // za izmenu
        objStariNastavnik = new clsNastavnik();

        // --------------POSTAVLJANJE NA POCETNE VREDNOSTI-----------------------
        UcitajPodatkeZaNavigaciju();
        pozicijaNavigacije = 0;
        stanjePrograma = "prikaz";
        PostaviStanjeKontrola(stanjePrograma);
        PrikaziAzurnePodatkeZaNavigaciju(pozicijaNavigacije, UkupnoZapisaNavigacije);

    }

    private void btnPotvrdi_Click(object sender, EventArgs e)
    {
        // ----------------provera potpunosti podataka
        if ((txbPrezime.Text.Equals("")) || (txbPrezime.Text.Length == 0) || (txbPrezime.Text == null))
        {
            txbPrezime.Focus();
            return; // izlaz iz procedure
        }

        if ((txbIme.Text.Equals("")) || (txbIme.Text.Length == 0) || (txbIme.Text == null))
        {
            txbIme.Focus();
            return;
        }

        if ((cmbZvanje.Text.Equals("")) || (cmbZvanje.Text.Length == 0) || (cmbZvanje.Text == null))
        {
            cmbZvanje.Focus();
            return;
        }

        // ----------------preuzimanje vrednosti sa korisnickog interfejsa
        string JMBG = txbJMBG.Text;
        string prezime = txbPrezime.Text;
        string ime = txbIme.Text;
        DateTime datum = dtpDatumRodjenja.Value;
        string zvanje = cmbZvanje.Text;

        // provera jedinstvenosti podataka

        // provera ispravnosti podataka


        // ----------------snimanje podataka u bazu podataka------------------
        // sa korisnickog interfejsa uzimamo podatke i stavljamo u objekat Zvanje
        // potrebno je za sve 3 operacije sa podacima, jer je argument poziva metode
```

```
        clsNastavnik objNoviNastavnik = new clsNastavnik();
        objNoviNastavnik = DajPodatkeIzKontrola();

        string poruka = "";
        try
        {

            switch (stanjePrograma)
            {
                case "unos":
                    objNastavnikDB.SnimiNovogNastavnika(objNoviNastavnik);
                    poruka = "Uspesno snimljeni podaci o novom nastavniku!";
                    // dodajemo poziciju navigacije na kraj
                    pozicijaNavigacije++;
                    break;
                case "izmena":
                    objNastavnikDB.IzmeniNastavnika(objStariNastavnik, objNoviNastavnik);
                    poruka = "Uspesno izmenjeni podaci o nastavniku!:";
                    // pozicija navigacije se nije izmenila
                    break;
                case "brisanje":
                    objNastavnikDB.ObrisiNastavnika(objStariNastavnik);
                    poruka = "Uspesno obrisani podaci o nastavniku!";
                    // nakon brisanja vracamo na prvu poziciju
                    pozicijaNavigacije = 1;
                    break;
            } // kraj switch
        }
        catch (Exception greska)
        {
            MessageBox.Show("Greska:" + greska);
            txbJMBG.Focus();
            return;
        }

        //---------------- obavestavanje korisnika o uspesnosti snimanja----------------
        // ako je sve u redu, program dolazi do ove tacke:
        MessageBox.Show(poruka);
        PostaviStanjeKontrola("prikaz");

        // ----------------restartovanje - novo stanje podataka za navigaciju nakon svih promena

        UcitajPodatkeZaNavigaciju(); // ovde se puni dsPodaciNavigacije i UkupnoZapisaNavigacije
        PrikaziSvePodatkeUGridu();
        PrikaziPodatke(dsPodaciNavigacije, pozicijaNavigacije);
    }

    private void btnFiltriraj_Click(object sender, EventArgs e)
    {
        // preuzimanje vrednosti za filtriranje
        string znakFiltriranja = cmbZnak.Text;
        string vrednostFiltriranja = txbFilter.Text;

        // primena filtriranja nad preuzimanjem podataka
        dsPodaciGrid = objNastavnikDB.DajNastavnikePoPrezimenu(znakFiltriranja, vrednostFiltriranja);
        // prikaz filtriranih vrednosti u gridu

        NapuniGrid(dsPodaciGrid);
    }

    private void btnSvi_Click(object sender, EventArgs e)
    {
        // preuzimanje svih podataka iz baze podataka
```

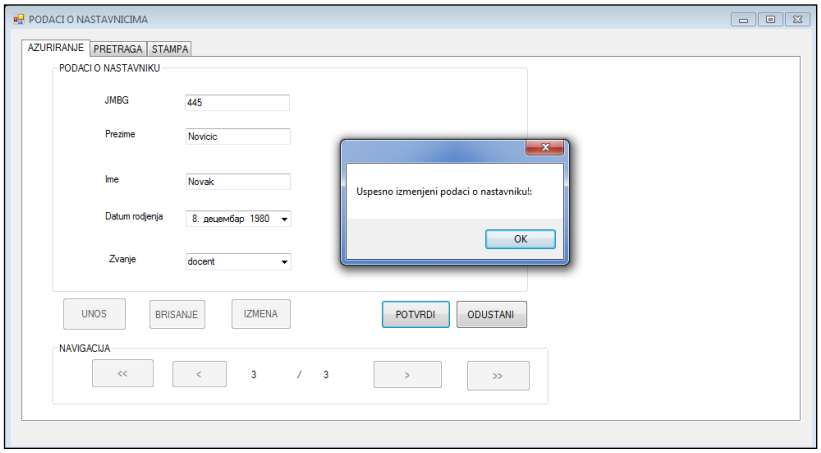
```
        // prikazivanje podataka u gridu
        PrikaziSvePodatkeUGridu();
    }

    private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
    {

        // provera podataka parametara stampe
        if ((!rdbSviNastavnici.Checked) && (!rdbNastavniciUZvanju.Checked))
        {
            MessageBox.Show ("Niste izabrali tip izvestaja, tj. parametre stampe");
            return;
        }

        if ((rdbNastavniciUZvanju.Checked) & ((cmbZvanjaFilter.Text.Length == 0) ||
(cmbZvanjaFilter.Text.Equals("")) || (cmbZvanjaFilter.Text == null)))
        {
            MessageBox.Show("Niste izabrali zvanje kao kriterijum filtriranja izvestaja!");
            return;
        }

        if ((rdbNastavniciUZvanju.Checked) & (cmbZvanjaFilter.Text.Length > 0))
        {
            DataSet dsPodaciZaProveru = PreuzmiPodatkeIzXML("Zvanja.XML");

            // 1. NACIN
            //bool podaciIzDomena = ProveraVrednostiZvanja(cmbZvanjaFilter.Text, dsPodaciZaProveru);
            // bolje je u uslov staviti POZITIVNO PITANJE
            //if (podaciIzDomena)
            //{
            //    MessageBox.Show("Podatak za filtriranje JESTE iz dozvoljenog skupa vrednosti!");

            //}
            //else // situacija kada su podaci van domena
            //{
                // 1. nacin - STANDARDNA KLASA
            //    clsExceptionIzvanDomena objGreskaIzvanDomena = new clsExceptionIzvanDomena();
            //    MessageBox.Show(objGreskaIzvanDomena.PorukaGreske);
            //}

            // 2. NACIN - primena exception
            try
            {
                // unutar ove procedure se uradi "throw exception:" ako ima uslova za to
                ProveraVrednostiZvanjaPrimenomException(cmbZvanjaFilter.Text, dsPodaciZaProveru);
            }
            catch (clsCustomException greska) // ovde se sacekuje exception
            {
                // ovde se procesira odgovor na exception ako nastane
                MessageBox.Show(greska.Message);
            }
        }

        //
##########################################################################
####################

        // preuzimanje parametara stampe

        // selekcija tipa izvestaja

        // prikazivanje selektovanog izvestaja u report vieweru
        rvIzvestaj.RefreshReport();
    }
```

```
        private void btnOdustani_Click(object sender, EventArgs e)
        {
            IsprazniKontrole();
            PostaviStanjeKontrola("prikaz");
        }

        private void btnPrikaziSpisakNastavnika_Click(object sender, EventArgs e)
        {
            // PROMENLJIVE
            List<clsNastavnoOsoblje> objListaNastavnogOsoblja = new List<clsNastavnoOsoblje>();
            clsNastavnoOsoblje objNastavnoOsoblje;
            clsNastavnik objNastavnik;
            clsZvanje objZvanje;

            string pomPrezime = "";
            string pomIme = "";
            DateTime pomDatumVremeRodjenja = DateTime.Now; // inicijalna vrednost
            string pomDatumRodjenjaString = "";
            string pomSifraZvanja = "";
            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().Equals(""))
                {
                    objNastavnoOsoblje = new clsNastavnoOsoblje();
                    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 = dsPodaciNastavnoOsoblje.Tables[0].Rows[brojac].ItemArray[3].ToString();

                    // konvertujemo sifru zvanja u naziv zvanja
                    pomNazivZvanja = DajNazivZvanja(dsPodaciZvanja, pomSifraZvanja);

                    // pripremamo objekat klase Zvanje
                    objZvanje = new clsZvanje();
                    objZvanje.Sifra = pomSifraZvanja;
                    objZvanje.Naziv = pomNazivZvanja;

                    // instanciramo objekat tipa nastavnika
                    objNastavnik = new clsNastavnik();
                    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);
        //}
    }
    private void btnUnos_Click(object sender, EventArgs e)
    {
        stanjePrograma = "unos";
        PostaviStanjeKontrola(stanjePrograma);
    }

    private void btnBrisanje_Click(object sender, EventArgs e)
    {
        stanjePrograma = "brisanje";
        // ocitavamo stare vrednosti za nastavnika
        objStariNastavnik = DajPodatkeIzKontrola();

        PostaviStanjeKontrola("brisanje");
    }

    private void btnIzmena_Click(object sender, EventArgs e)
    {
        stanjePrograma = "izmena";

        // ocitavamo stare vrednosti za nastavnika
        objStariNastavnik = DajPodatkeIzKontrola();


        PostaviStanjeKontrola(stanjePrograma);
    }

    private void btnPrvi_Click(object sender, EventArgs e)
    {
        pozicijaNavigacije = 0;
        PrikaziPodatke(dsPodaciNavigacije, pozicijaNavigacije);
    }

    private void btnPrethodni_Click(object sender, EventArgs e)
    {
        if (pozicijaNavigacije > 0)
        {
            pozicijaNavigacije--;  //pozicijaNavigacije = pozicijaNavigacije - 1;
            PrikaziPodatke(dsPodaciNavigacije, pozicijaNavigacije);
        }
    }

    private void btnSledeci_Click(object sender, EventArgs e)
```

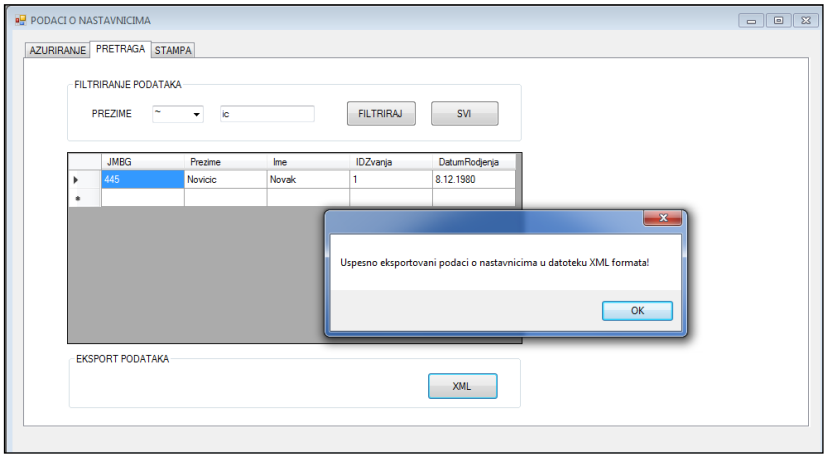
```
        {
            if (pozicijaNavigacije < (UkupnoZapisaNavigacije - 1))
            {
                pozicijaNavigacije++;  //pozicijaNavigacije = pozicijaNavigacije + 1;
                PrikaziPodatke(dsPodaciNavigacije, pozicijaNavigacije);
            }
        }

        private void btnPoslednji_Click(object sender, EventArgs e)
        {
            pozicijaNavigacije = UkupnoZapisaNavigacije - 1;
            PrikaziPodatke(dsPodaciNavigacije, pozicijaNavigacije);
        }

        private void btnExportXML_Click(object sender, EventArgs e)
        {
            dsPodaciGrid.WriteXml("ExportNastavnika.XML");
            MessageBox.Show("Uspesno eksportovani podaci o nastavnicima u datoteku XML formata!");
        }
    }
}
```

REZULTAT RADA 1. KARTICE



Slika 10.49. Rezultat unosa podataka o nastavniku

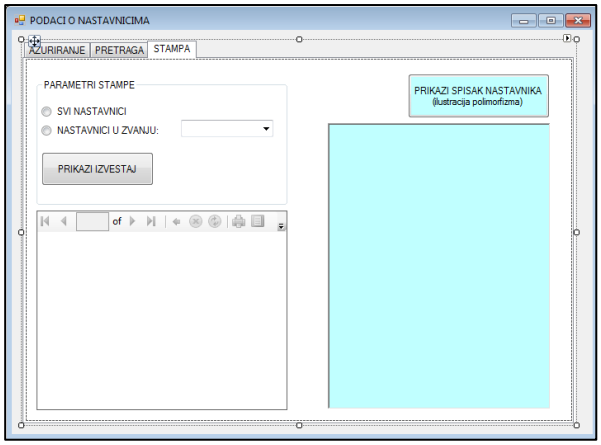
REZULTAT RADA 2. KARTICE



Slika 10.50. Rezultat tabelarnog prikaza podataka o nastavniku

REALIZACIJA ŠTAMPE PODATAKA O NASTAVNICIMA

Na kartici za štampu se nalazi deo koji je ostao od ranije, a koji smo koristili za ilustraciju polimorfizma. Taj deo smo označili posebnom (svetloplavom) bojom.

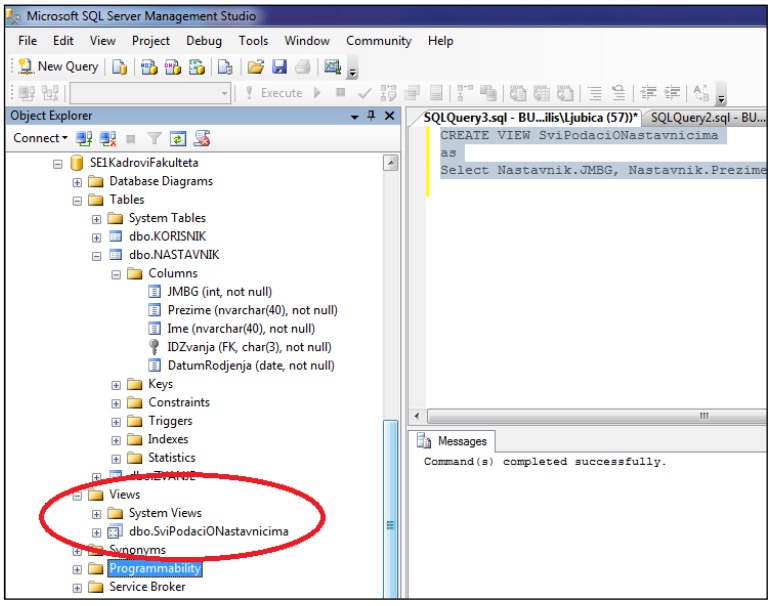


Slika 10.51. Izgleg kartice za štampu, sa delom za štampu i prethodnim delom ilustracije polimorfizma

U delu koji se odnosi na štampu (levi deo) izdvaja se deo za podešavanje parametara štampe – izbor izveštaja sa svim nastavnicima ili nastavnicima koji odgovaraju određenom zvanju (parametarski izveštaj).

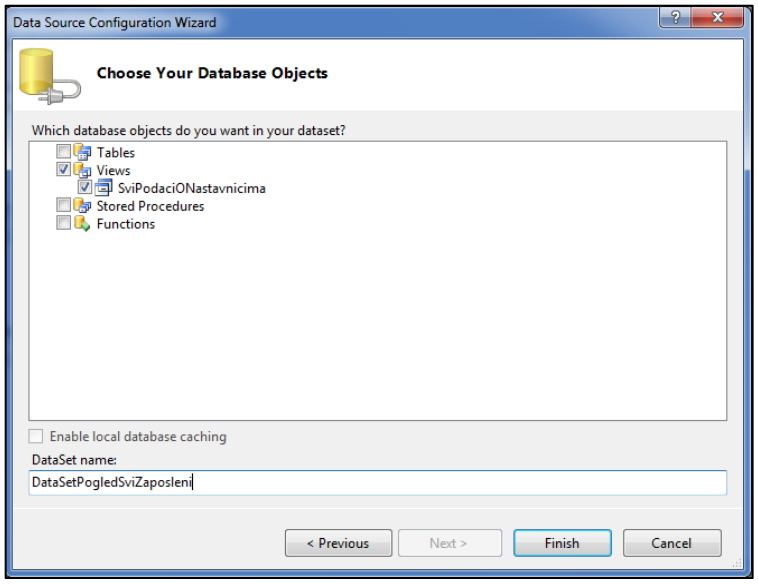
Za potrebe prikaza svih podataka o nastavniku, kreiraćemo pogled u bazi podataka.

```
CREATE VIEW SviPodaciONastavnicima
as
Select Nastavnik.JMBG, Nastavnik.Prezime, Nastavnik.Ime, Zvanje.Naziv as NazivZvanja,
Nastavnik.DatumRodjenja from Nastavnik inner join Zvanje on Nastavnik.IdZvanja =
Zvanje.Sifra
```



Slika 10.52. Kreiran pogled u bazi podataka, za potrebe izveštaja

U okviru report wizarda koristimo opciju za da izvor podataka bude pogled.



Slika 10.53. Izbor pogleda kao izvora podataka, u okviru Wizarda kreiranja fajla reporta

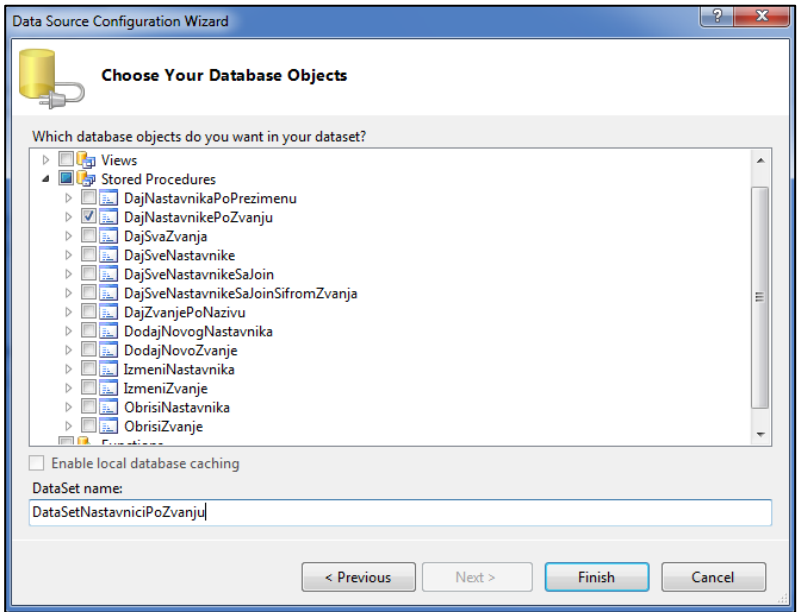
Kreiramo izveštaj sa spiskom svih nastavnika.

Slika 10.54. Izgled izveštaja sa spiskom nastavnika, u dizajn režimu

Da bismo kreirali izveštaj koji je filtriran (parametarska štampa), potrebno je kreirati kao izvor podataka stored proceduru koja spaja podatke iz 2 tabele, ali daje mogućnost filtriranja. Za potrebe kreiranja izvestaja sa spiskom nastavnika po zvanju, kreiramo stored proceduru koja kao parametar uzima naziv zvanja:

```
USE [SE1KadroviFakulteta]
GO

CREATE PROCEDURE [dbo].[DajNastavnikePoZvanju] @NazivZvanja nvarchar(40)
AS
select NASTAVNIK.JMBG, NASTAVNIK.Ime, NASTAVNIK.Prezime, ZVANJE.Naziv as
NazivZvanja, Nastavnik.DatumRodjenja from NASTAVNIK inner join ZVANJE on ZVANJE.Sifra =
NASTAVNIK.IDZvanja

where ZVANJE.NAziv = @NazivZvanja
```

Kreiramo izveštaj nad stored procedurom:

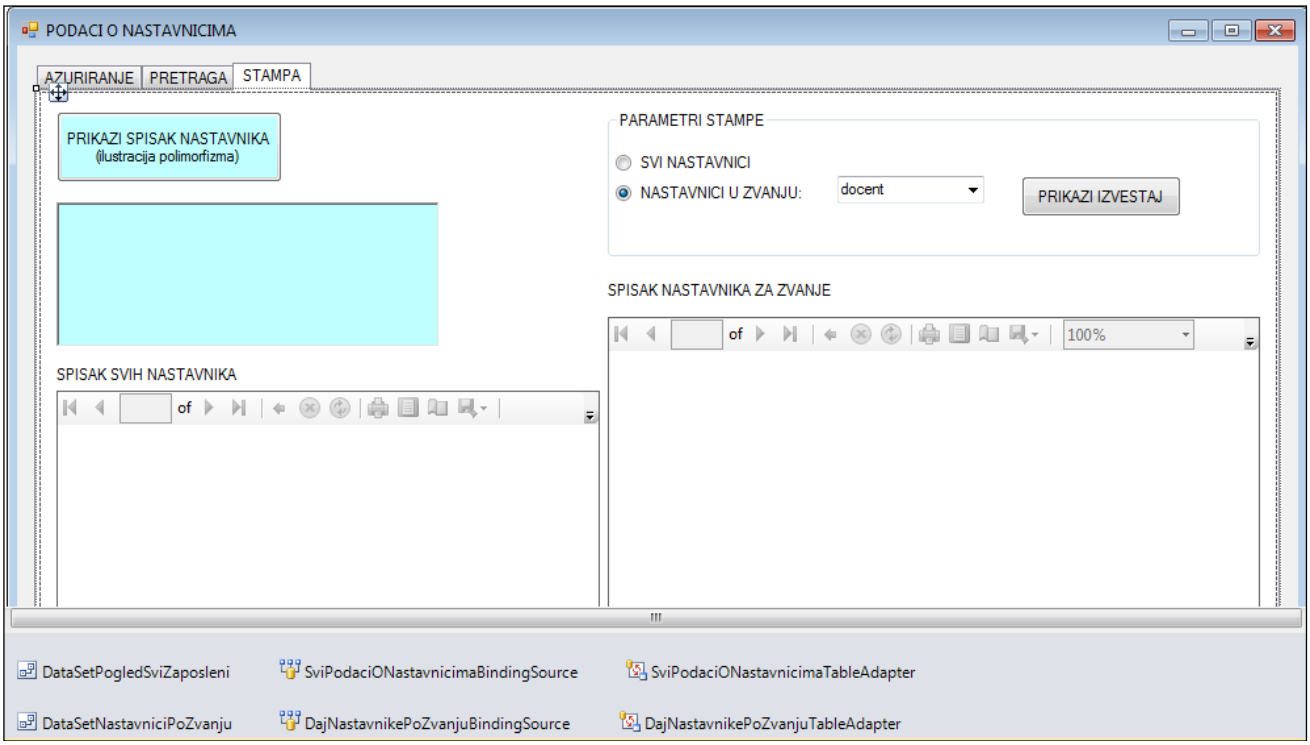


Slika 10.55. Izbor izvora podataka za izveštaj – stored procedura

Kreiramo izveštaj koji je namenjen prikazu filtriranih podataka – izdvojen spisak nastavnika po zvanju:

Slika 10.56. Izgled izveštaja za filtrirani prikaz nastavnika po zvanju

Izmenićemo izgled korisničkog interfejsa u kartici za štampu i povezati sa report viewer kontrolama oba izveštaja – spisak svih i filtrirani spisak. Postavićemo u properties za radio button da je selektovana opcija NASTAVNICI U ZVANJU i početna vrednost „docent" u combo boxu.



Slika 10.57. Izmenjena kartica za štampu sa dva report viewera

Nakon pridruživanja izveštaja report vieweru, automatski se u formload postavlja kod za punjenje (od strane data table adaptera) dataseta koji je potreban za report, za filtriranu varijantu dodajemo kriterijum filtriranja „docent" koji se prosledjuje stored proceduri koja obezbedjuje podatke za dataset i report.

```
private void fNovaForma_Load(object sender, EventArgs e)
        {
            // --------------REPORT-------------------------
            // automatski postavljeno kada se poveze izvestaj za ReportViewer kontrolu

            // TODO: This line of code loads data into the
'DataSetNastavniciPoZvanju.DajNastavnikePoZvanju' table. You can move, or remove it, as
needed.

this.DajNastavnikePoZvanjuTableAdapter.Fill(this.DataSetNastavniciPoZvanju.DajNastavnikePo
Zvanju, "docent" ); // cmbZvanjaFilter.Text );
            // TODO: This line of code loads data into the
'DataSetPogledSviZaposleni.SviPodaciONastavnicima' table. You can move, or remove it, as
needed.

this.SviPodaciONastavnicimaTableAdapter.Fill(this.DataSetPogledSviZaposleni.SviPodaciONasta
vnicima);
```

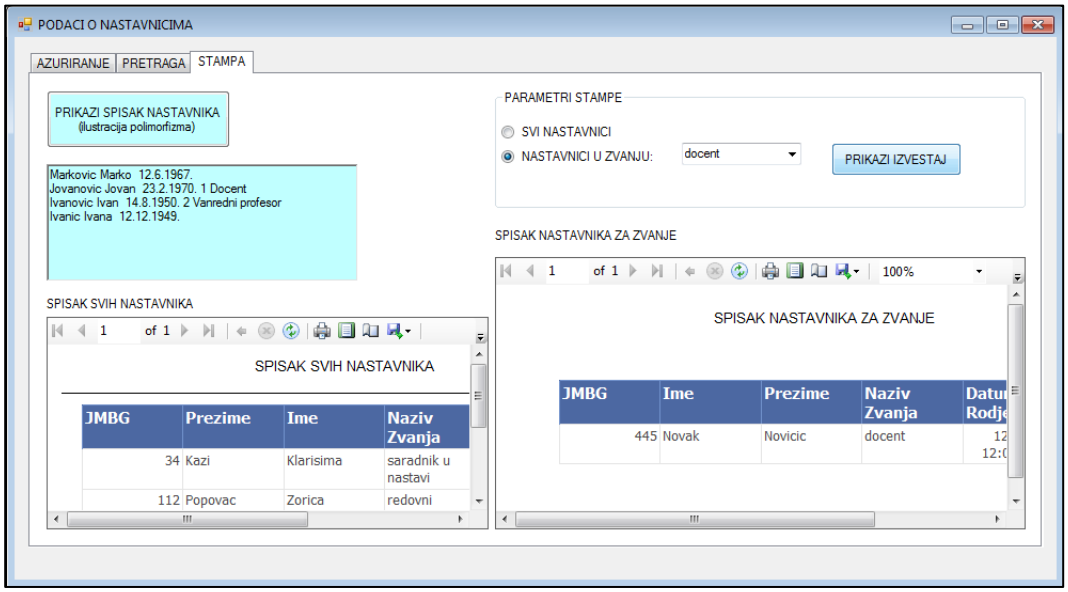
PROGRAMSKI KOD NA TASTERU ZA PRIKAZ IZVESTAJA:

```
 private void btnPrikaziIzvestaj_Click(object sender, EventArgs e)
        {

            // provera podataka parametara stampe
            if ((!rdbSviNastavnici.Checked) && (!rdbNastavniciUZvanju.Checked))
```

```
        {
            MessageBox.Show ("Niste izabrali tip izvestaja, tj. parametre stampe");
            return;
        }

        if ((rdbNastavniciUZvanju.Checked) & ((cmbZvanjaFilter.Text.Length == 0) ||
(cmbZvanjaFilter.Text.Equals("")) || (cmbZvanjaFilter.Text == null)))
        {
            MessageBox.Show("Niste izabrali zvanje kao kriterijum filtriranja izvestaja!");
            return;
        }

        if ((rdbNastavniciUZvanju.Checked) & (cmbZvanjaFilter.Text.Length > 0))
        {
            //1. NACIN - PODACI SE UZIMAJU IZ xml:
            //DataSet dsPodaciZaProveru = PreuzmiPodatkeIzXML("Zvanja.XML");

            DataSet dsPodaciZaProveru = new DataSet();
            dsPodaciZaProveru = objZvanjeDB.DajSvaZvanja();

            // 1. NACIN
            //bool podaciIzDomena = ProveraVrednostiZvanja(cmbZvanjaFilter.Text,
dsPodaciZaProveru);
            // bolje je u uslov staviti POZITIVNO PITANJE
            //if (podaciIzDomena)
            //{
            //    MessageBox.Show("Podatak za filtriranje JESTE iz dozvoljenog skupa
vrednosti!");

            //}
            //else // situacija kada su podaci van domena
            //{
               // 1. nacin - STANDARDNA KLASA
            //    clsExceptionIzvanDomena objGreskaIzvanDomena = new
clsExceptionIzvanDomena();
            //    MessageBox.Show(objGreskaIzvanDomena.PorukaGreske);
            //}

            // 2. NACIN - primena exception
            try
            {
               // unutar ove procedure se uradi "throw exception:" ako ima uslova za to
               ProveraVrednostiZvanjaPrimenomException(cmbZvanjaFilter.Text,
dsPodaciZaProveru);
            }
            catch (clsCustomException greska) // ovde se sacekuje exception
            {
               // ovde se procesira odgovor na exception ako nastane
               MessageBox.Show(greska.Message);
            }
        }
```

```
        //
##########################################################################
#################################

        // selekcija tipa izvestaja i preuzimanje parametara stampe
        if (rdbSviNastavnici.Checked)
        {
            // prikazivanje selektovanog izvestaja u report vieweru
            this.rvIzvestajSviZaposleni.RefreshReport();
        }
        if (rdbNastavniciUZvanju.Checked) // da li je combo napunjen i da li ima odgovarajuce
vrednosti se proverava ranije
        {

this.DajNastavnikePoZvanjuTableAdapter.Fill(this.DataSetNastavniciPoZvanju.DajNastavnikePo
Zvanju, cmbZvanjaFilter.Text );
            this.rvNastavniciUZvanju.RefreshReport();
        }


    }
```

REZULTAT KARTICA ŠTAMPA:



Slika 10.58. Rezultat izvršavanja štampe svih podataka, filtriranih podataka i polimorfnog prikaza

11. PITANJA ZA TEST

I DEO

1. UML MODELOVANJE SOFTVERA

- Šta je UML?
- Koje su vrste dijagrama uključene u UML 1.0?
- Namena i elementi use case dijagrama?
- Namena i elementi dijagrama komponenti?
- Namena i elementi dijagrama razmeštaja?
- Namena i elementi dijagrama klasa?
- Koje su vrste veza na dijagramu klasa?
- Koja je razlika između agregacije i kompozicije?
- Koja je razlika između asocijacije i kompozicije?
- Koji su elementi klase?
- Namena i elementi dijagrama sekvenci?
- Šta je poruka?

2. RAD U CASE ALATU POWER DESIGNER

- Koje su prednosti korišćenja CASE alata?
- Kako se generiše programski kod klasa na osnovu CDM modela?
- Šta je potrebno podesiti na relacijama u okviru dijagrama klasa?

2. RAZVOJNO OKRUŽENJE VISUAL STUDIO NET

- Tipovi projekata u VS NET?
- U čemu se ogleda razlika između različitih .NET frameworka?
- Šta je namespace?
- Kako se uključuje u projekat rad sa drugim bibliotekama klasa?
- Kako se uključuje na konkretnu stranicu rad sa drugim bibliotekama klasa?

3. GRAFIČKO OBLIKOVANJE WINDOWS APLIKACIJE I POVEZIVANJE FORMI

- Šta znači MDI forma?
- Šta znači modalna forma?
- Koja je razlika između Show i ShowDialog?
- Koje su osnovne kontrole u grafičkom dizajnu interaktivnog rada sa korisnikom?
- Koje se kontrole koriste za meni i statusnu liniju?
- Koja se kontrola koristi za tabelarni prikaz podataka?
- Koja se kontrola koristi za prikaz izveštaja?

4. OPŠTI POJMOVI U OBLASTI PROGRAMIRANJA

- Koja je razlika izmedju promenljive tipa vrednosti i promenljive tipa reference?
- Šta znači type casting?
- Koja 2 tipa transformacija tipova podataka (type castinga) postoje?
- Koja je razlika izmedju deklaracije, incijalizacije, definicije promenljive?
- Kada dodeljujemo promenljivu tipa reference drugoj promeljivoj istog tipa, sta ta druga promenljiva dobija?

5. SINTAKSA C# PROGRAMSKOG JEZIKA

- Koji su tipovi podataka u C# za celobrojne vrednosti i realne brojeve?
- Kako se spajaju stringovi u C#?
- Da li je obavezna inicijalizacija promenljivih u C#? Šta znači incijalizacija?
- Koja su 2 tipa eksplicitnog type castinga u C# - navesti primer?
- Koji simbol se koristi za dodelu vrednosti promenljive, a koji za proveru da li promenljiva ima neku vrednost?
- Sta znaci "PROMENLJIVA++"?
- Kako se oznacava NEGACIJA u C#? Sta znaci "!"?
- Kako označavamo kada procedura ne vraća vrednost?
- Kojom naredbom u kodu procedure (koja vraća vrednost) zapravo omogućavamo vraćanje vrednosti?
- Koji simboli se koristi u C# za logicko AND (2 vrste) iza logicko OR (2 vrste)? Objasniti razliku izmedju 2 vrste za svaku od ovih operacija.
- Kako se uslov u C# navodi u if delu kod uslovnog grananja? Navesti primer.
- Kako se pise u u C# case struktura – višestruko grananje? Navesti primer.
- Kako se piše ciklus sa preduslovom (while), a kako ciklus sa postuslovom (repeat until)? Navesti primer.
- Kako se piše for i koja je razlika između for i foreach? Navesti primer.

- Kako se piše blok za obradu grešaka? Navesti primer.
- Šta znači finally u try-catch bloku?
- Koja je razlika između niza, Array Liste i Tipizirane liste?

6. OPŠTI POJMOVI OBJEKTNO-ORJENTISANOG PROGRAMIRANJA

- Šta znači skraćenica dll?
- Koji su najčešće korišćeni modifikatori pristupa objasniti značenje?
- Šta je signatura metode?
- Koji je uobičajeni modifikator pristupa za polja (suštinske atribute) klase?
- Šta predstavlja i kako se realizuje svojstvo (property)? Koja je razlika u odnosu na polja klase?
- Šta je konstruktor? Koja je njegova uloga?
- Ako je konstruktor nema parametara i nijedna naredba ne piše u bloku naredbi u okviru konstruktora, da li konstruktor nešto izvršava? Šta?
- Da li konstruktor moze da ima vise varijanti u istoj klasi?
- Da li u istoj klasi moze biti vise metoda sa istim nazivom? Koji uslov mora biti ispunjen, ako moze?
- Koja je razlika atributa(polja) i svojstva? Kako se svojstvo implementira?
- Koja je razlika set i get metode? Čemu služe? Da li uvek moraju obe da se pišu u paru?
- Šta naslednik može da vidi i koristi u odnosu na baznu klasu?
- Šta znači nasleđivanje?
- Šta znači statička klasa? Cemu sluzi? Da li se moze instancirati?
- Šta je interfejs klasa? Koje su karakteristike?
- Razlika Overloading (preklapanje) i Overriding (redefinisanje)?
- Šta je apstraktna klasa? Da li se može nasleđivati? Da li se može instancirati? Koje su karakteristike?
- Koja je razlika apstraktne klase i interfejsa?

7. OBJEKTNO-ORJENTISANO PROGRAMIRANJE U C#

- Kako se realizuje u programskom kodu C# odnos asocijacije?
- Kako se realizuje u programskom kodu C# odnos kompozicije?
- Kako se implementira enkapsulacija?
- Kako se implementira polimorfizam?
- Kako se uništava objekat?
- Šta znači this?
- Šta znači base?
- Koja je bazna klasa za sve klase?
- Kako se realizuje u programskom kodu C# odnos nasleđivanja?
- Da li može pokazivač na nasledničku klasu da bude smešten u objektu bazne klase?

SMERNICE ZA KVALITET SOFTVERA I KONVENCIJE

- Koji načini imenovanja promenjivih i elemenata klase postoje i kada se koriste?
- Koja je razlika između camelCasing i PascalCasing?

II DEO

VIŠESLOJNA SOFTVERSKA ARHITEKTURA

- Koja su četiri osnovna sloja arhitekture poslovno-orjentisanog softvera?
- Koje su komponente prezentacionog sloja?
- Koja je razlika izmedju prezentacionog sloja i prezentacione logike?
- Koje su komponente sloja poslovne logike?
- Koje su komponente servisnog sloja?
- Koje su komponente sloja za rad sa podacima?
- Zašto je važno odvojiti delove aplikacije u slojeve?

PROGRAMIRANJE WINDOWS APLIKACIJE u .NET C#

- Koje su memorijske kolekcije podataka u C#?
- Sta je string konekcije?
- Koja se biblioteka klasa koristi za standardni rad sa MS SQL Server bazom podataka?
- Koji su načini rada sa bazom podataka kod objektno-orjentisanih .NET aplikacija?
- Kako se kreira sopstvena biblioteka klasa?
- Kako se kreira skup klasa koristeći Entity framework?
- Koja je glavna (objedinjujuća) klasa, koja je generisana u okviru Entity framework pristupa?

12. LITERATURA

Konsultovana literatura i web sajtovi

[1] Zvaničan web sajt UML: http://www.uml.org/

[2] Zvaničan web sajt Object-management group, specifikacija UML 2.5 verzije http://www.omg.org/spec/UML/2.5/

[3] Ivanković Zdravko, Radosav Dragica, Markoski Branko: Softversko inženjerstvo 2 – tutorijal za Microsoft visual studio 2010, Tehnički fakultet „Mihajlo Pupin" Zrenjanin, 2013.

[4] Grady Booch, James Rumbaugh, Ivar Jacobson: "The Unified Modeling Language User Guide",Addison-Wesley, 1999.

[5] Robin A. Reynolds – Haertle: "OOP sa Microsoft tehnologijama Visual Basic.NET i Visual C#. NET", CET, 2002.

[6] Jesse Liberty: "Programiranje na jeziku C#", Mikro knjiga, 2007.

[7] Microsoft: "Programming with C#", MSDN training workbook, Microsoft, 2002.

Predložena literatura za dalji rad

[8] Arthur J. Riel: "Heuristike objektno-orjentisanog dizajna", CET, 2003.

[9] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: "Design Patterns", Addison-Wesley, 1995.

[10] Martin Fowler: "Refactoring – Improving the Design of Existing Code", Addison-Wesley, 1999.

RECENZIJE

RECENZIJA PROF. DR BILJANE RADULOVIĆ

Autori su prezentovali nastavne sadržaje vežbi iz predmeta SOFTVERSKO INŽENJERSTVO 1 obrazovnog profila osnovnih akademskih studija INFORMACIONE TEHNOLOGIJE. Prema akreditovanom kurikulumu, sadržaj ovog nastavnog predmeta obuhvata ovladavanje korišćenjem objektno-orjentisane metode razvoja softvera i primena UML jezika objektno-orjentisanog modelovanja softvera. Sadržaj ovog praktikuma u potpunosti pokriva sve nastavne teme predviđene akreditovanim kurikulumom. Praktikum opisuje razvoj Windows aplikacija u Microsoft Visual Studio .NET 2010 okruženju, kao i UML modelovanje primenom alata Sybase Power Designer.

Praktikum se sastoji iz 12 celina, od kojih je 10 celina sa nastavnim sadržajima, 11. poglavlje se odnosi na pitanja za test i 12. poglavlje je literatura. Praktikum je obima 198 strana. Svako poglavlje sadrži teorijska i praktična opšta uputstva za rad u okviru odgovarajuće teme, ali i zadatke i rešenja zadataka za navedenu oblast.

Uvodni deo opisuje obuhvat tema, ali i širi kontekst predstavljenog gradiva, povezujući predstavljene sadržaje sa naprednijim tehnikama programiranja. Drugo poglavlje se odnosi na modelovanje softvera primenom UML jezika. Treće poglavlje opisuje upoznavanje sa razvojnim okruženjem Visual Studio .NET. Grafičko oblikovanje windows aplikacije i povezivanje formi opisano je u četvrtom poglavlju. Peto poglavlje daje pregled sintakse C# programskog jezika. Šesto poglavlje daje osnove rada sa strukturama podataka i datotekama. Osnove objektno-orjentisanog programiranja u C# opisano je u sedmom poglavlju. Osmo poglavlje daje heuristička uputstva i konvencije za oblikovanje programskog koda. Uvod u višeslojni razvoj softvera dat je u osmom poglavlju kroz opis elemenata višeslojne arhitekture. U desetom poglavlju dat je detaljan prikaz načina implementacije windows aplikacije kroz elemente programskog koda memorijskih kolekcija za rad sa podacima i bazama podataka, kao i rad sa izveštajima. Zadaci i rešenja koja su data u desetom poglavlju podesna su za korišćenje kao primeri zadataka za kolokvijume i seminarske radove, kao što je i posebno naznačeno. Jedanaesto poglavlje daje pregled pitanja koja se mogu koristiti za testiranje poznavanja osnovnih pojmova u praktičnom radu u okviru objektno-orjentisanog programiranja primenom jezika C#.

Smatram da će ovaj praktikum biti veoma koristan u okviru nastave iz predmeta SOFTVERSKO INŽENJERSTVO 1, kao i srodnih nastavnih predmeta u oblasti Informacionih tehnologija. Takođe, ovaj praktikum, zbog detaljnosti predstavljenih postupaka i rešenih primera, može biti koristan i u okviru stručnih kurseva ili samostalnog rada u izučavanju predstavljene problematike. Predloženi praktikum preporučujem da bude publikovan i korišćen u nastavnoj i stručnoj praksi.

Recenzent: Prof. Dr Biljana Radulović

RECENZIJA PROF. DR IVANE BERKOVIĆ

Knjiga „Osnove objektno-orjentisanog programiranja sa primerima u C#, praktikum za vežbe" nastala je kao sveobuhvatan materijal koji pokriva praktični deo nastave iz predmeta SOFTVERSKO INŽENJERSTVO 1 (osnovne akademske studije INFORMACIONE TEHNOLOGIJE). Praktikum obuhvata teorijske osnove i praktična uputstva za realizaciju objektno-orjentisanog modelovanja softvera i programiranja. Modelovanje je realizovano primenom UML jezika i alata Sybase Power Designer, a programiranje primenom C# programskog jezika i alata Visual Studio .NET 2010. Predstavljeni sadržaji u ovom praktikumu u potpunosti pokrivaju sve oblasti predviđene akreditovanim kurikulumom.

Knjiga se sastoji iz 198 strana organizovanih u 11 celina i sa literaturom. Sastoji se iz opisa teorijskih osnova, tutorijala za korišćenje alata, zadataka i rešenih zadataka, kao i testa pojmova. Rešenja zadataka su data u formi UML dijagrama, dizajna korisničkog interfejsa i programskog koda. Na ovaj način, praktikum daje potpun metodički oblikovan sadržaj za izvođenje nastave na nastavnom predmetu Softversko inženjerstvo 1, olakšavajući studentima da, korak po korak, realizuju osnovne elemente poslovne Windows aplikacije.

Uvodno poglavlje opisuje sadržaj i širi kontekst praktikuma i povezuje gradivo, izloženo u ovom rukopisu, sa mogućnostima daljeg rada u ovoj oblasti. Drugo poglavlje daje uvod u UML modelovanje softvera kroz teoriju o UML dijagramima, primere UML dijagrama u dizajnu poslovnog aplikativnog softvera, primer dokumentovanja softverskog rešenja UML dijagramima i generisanja programskog koda na osnovu modela u okviru CASE alata. Upoznavanje sa razvojnim okruženjem Visual studio .NET opisano je u trećem poglavlju kroz Windows forms i class library projekat. Grafičko oblikovanje windows aplikacije opisano je u četvrtom poglavlju opisom forme za prijavljivanje korisnika, glavnog menija i forme za rad sa podacima. Sintaksa C# programskog jezika opisana je u petom poglavlju osnovnim elementima sintakse, operatorima, programskim strukturama, obradom grešaka. Rad sa strukturama podataka i datotekama opisana je u šestom poglavlju kroz rad sa nizovima, listama, tipiziranim listama, fajlovima, folderima, datotekama TXT i XML. U sedmom poglavlju opisane su osnove objektno-orjentisanog programiranja u C# i primeri primene. Heuristička uputstva i konvencije za oblikovanje programskog koda data su u osmom poglavlju, a elementi višeslojne softverske arhitekture su opisani u devetom poglavlju. Deseto poglavlje daje sveobuhvatni prikaz opštih pristupa programiranju aplikativnog softvera za rad sa memorijskim kolekcijama i bazama podataka, ilustrovano zadacima i rešenim primerima koji mogu biti korišćeni na kolokvijumu i u okviru seminarskog rada, što je posebno naznačeno. Takođe, u ovom poglavlju je opisan i rad sa izveštajima. Rad sa bazom podataka opisan je na 3 načina – korišćenjem standardnih klasa, kreiranjem sopstvenih klasa i generisanjem i korišćenjem Entity Framework klasa.

Smatram da je ovaj praktikum dobro koncipiran i detaljan, tako da će biti veoma koristan studentima u savladavanju predviđenog gradiva. Primena ovog praktikuma će omogućiti kvalitetno izvođenje vežbi iz predmeta Softversko inženjerstvo 1. Preporučujem da dati rukopis bude publikovan.

Recenzent: Prof. Dr Ivana Berković