Windows forme u C#-u

Lovrić, Valentina

Undergraduate thesis / Završni rad

2016

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:800058

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-20**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
ELEKTROTEHNIČKI FAKULTET

Stručni studij

WINDOWS FORME U C#-u

Završni rad

Valentina Lovrić

Osijek, 2016.

Sadržaj

1. UVOD 1

2. NASTANAK JEZIKA C# I .NET PLATFORME 2

2.1. Razvoj programskih jezika 2

2.2. Microsoft .NET 3

3. OSNOVNE KARAKTERISTIKE C# 5

3.1. Podatci i operatori 5

3.2. Tokovi programa 9

3.3. Klase i objekti 12

3.4. Osnove programiranja u C#-u 16

4. WINDOWS APLIKACIJE 21

4.1. Kreiranje i pokretanje Windows formi 21

4.2. Događaji 26

4.3. Osnovne kontrole Windows formi 26

4.4. Napredne mogućnosti Windows formi 32

5. PROJEKT VIDEOTEKA 39

5.1. Struktura aplikacije 39

5.2. Programiranje aplikacije 41

6. ZAKLJUČAK 48

LITERATURA 49

POPIS TABLICA I ILUSTRACIJA 51

SAŽETAK 53

APSTRACT 54

ŽIVOTOPIS 55

1. UVOD

U početcima, programiranje je bilo vrlo jednolično – nije postojala raznolikost načina kreiranja aplikacija niti grafička sučelja. No, u to vrijeme to nije bio problem. Svi programeri imali su jednaka ograničenja, tako da su više-manje sve aplikacije radile sa jednakim ograničenim skupom funkcionalnosti.

Do danas, isti principi programiranja su se zadržali no alati koji omogućuju pisanje računalnih programa znatno su napredovali s aspekta korisničke prihvatljivosti. Današnji moderni programeri su preplavljeni mogućnostima, ali jednako tako i korisničkim zahtjevima, među kojima je i kreiranje grafičkog sučelja.

U ovom radu omogućen je uvod u programiranje pomoću jednog od najpoznatijih alata za programiranje i jednostavno kreiranje grafičkog sučelja – Microsoft Windows Forms. Riječ je o alatu koji omogućuje kreiranje Windows desktop aplikacija korištenjem C# programskog jezika.

Unutar daljnjih poglavlja biti će objašnjene neke osnovne karakteristike C#-a, kao i kreiranje jednostavnih Windows formi za sve one koji se nikad nisu susreli sa ovakvim načinom programiranja. Ovaj rad predstavlja uvod svakom čitatelju u svijet Windows programiranja te mu daje potrebno osnovno znanje s kojim se može okušati u nekim naprednijim tehnikama i mogućnostima.

2. NASTANAK JEZIKA C# I .NET PLATFORME

2.1. Razvoj programskih jezika

Prva računala koja su se pojavila bila su veoma složena za korištenje stoga su ih koristili isključivo stručnjaci osposobljeni za komunikaciju s računalom. Njihova komunikacija sastojala se od samo dva koraka: davanje uputa računalu i čitanje dobivenih rezultata obrade. Unošenje uputa, tj. tadašnje programiranje sastojalo se od unošenja nizova nula i jedinica koji su predstavljali upute računalu, kao npr. „oduzmi dva broja“ ili „premjesti podatak iz jedne memorijske adrese na drugu“.

Pisanje nizova nula i jedinica zamijenilo je korištenje strojnog jezika gdje se od programera zahtijevalo prevođenje svakog algoritma na strojni jezik. Taj je pristup dao veću važnost izučavanju algoritama zbog toga što efikasniji algoritam smanjuje broj linija programskog koda i broj prepisivanja heksadekadskog koda sa kojima se opisivala naredba, a time i vjerojatnost pogreške. No, s obzirom da je i takve programe bilo veoma složeno za pisati, čitati i posebice ispravljati došlo je do pojave prvih programerskih alata nazvanih asembleri (eng. „*assemblers*“).

Asembler je prvi programski jezik koji je komunicirao sa strojem putem određenih naredbi umjesto niza brojeva (bili oni nule i jedinice ili heksadekadski zapis). Korištenjem mnemonika kao zamjenu za strojni kod znatno se povećala čitljivost programa. Nasuprot tomu pisanje i ispravljanje programa ostalo je i dalje veoma složeno zbog potrebe davanja računalu upute za svaku pojedinu operaciju. S vremenom, strojni jezik smatran je prvom generacijom strojnih jezika, a asembler drugom generacijom.

S vremenom, sve više se javljala potreba razvijanja programskih alata koji će osloboditi programera rutinskih poslova i dopustiti mu da se usredotoči na problem koji rješava. Počeli su se razvijati novi programski jezici prilagođeniji razvoju softvera od asemblerskih jezika. Tako su nastali jezici treće generacije čija je glavna prednost u odnosu na preteče bila neovisnost o procesoru. Najrazvijeniji primjeri takvih jezika su: FORTRAN (eng. „*FORmula TRANslator*“) koji se koristio za matematičke i znanstvene proračune, BASIC, Pascal namijenjen akademskoj upotrebi i COBOL (eng. „*Common Business Oriented Language*“) kojeg je razvila američka mornarica za svoje poslovne aplikacije, a bio je u pravilu namijenjen upravljanju bazama podataka.

Među gore navedene jezik trebamo svrstati i C programski jezik koji se pojavio u ranim 70-im godinama 20. stoljeća, a predstavlja preteču današnjeg C++ jezika. Autor C-a je Dennis Ritchie koji je stvorio ovaj programski jezik za rješavanje praktičnih problema kodiranja sistemskih programa i jezgre operacijskog sustava UNIX. C je bio prvi jezik opće namjene koji je postigao veliki uspjeh, a neki od razloga koji su tomu pridonijeli su: jednostavnost učenja, omogućavanje modularnog pisanja programa, sadržavao je samo naredbe koje se mogu jednostavno prevesti u strojni jezik, omogućavao je dobru kontrolu strojnih resursa, a samim time i optimizaciju koda.

Daljnjim razvojem programa i samih programera javlja se potreba za novim promjenama. Kompleksnost programa i velike duljine kodova (i do preko par tisuća linija) zahtijevaju uvođenje dodatnih mehanizama kako bi se omogućila veća jednostavnost pisanja i održavanja programa te iskoristivost jednog koda u više različitih programa. Zbog navedenih potreba došlo je do razvoja objektno orijentiranih programskih jezika, od kojih su najpoznatiji C# i C++.

Jezik C# razvio je tim na čelu sa dva istaknuta Microsoft inženjera Anders Hejlsberf i Scott Wiltamuth krajem 20.-og i početkom 21.-og stoljeća. Pisan je za .NET, novu razvojnu platformu koja se razvila u isto vrijeme. C# je jezik nastao na temeljima i iskustvima mnogih starijih programskih jezika: C (odlične performanse), C++ (objektno orijentirana struktura), Java™ (visoka razina sigurnosti) i Visual Basic (brz razvoj). Nastao je sa namjerom stvaranja novog jezika koji će u potpunosti odgovarati višeslojnim Web aplikacijama i sve do danas je jezik koji najbolje ispunjava mnoge programerske zahtjeve.

2.2. Microsoft .NET

.NET platforma je razvojni okvir koji omogućava novo sučelje za programiranje aplikacija. Ona ujedinjuje klasično sučelje Windows operacijskog sustava sa brojnim tehnologijama koje su proizašle iz Microsofta proteklih godina (npr. XML, ASP razvojni okvir za web, objektno-orijentirani dizajn i sl.).

.NET sastoji se od četiri grupe proizvoda:

- skup programskih jezika u okviru Visual Studio razvojne okoline – obuhvaća jezike: C#, Visual Basic .NET, Managed C++ i Jscript .NET

- skup .NET Enterprise servera (obuhvaća SQL server, Exchange Server i BizTalk server)
- komercijalne web usluge
- mobilni .NET uređaji kao npr. mobilni telefoni, uređaji za igre i slično[1]

Osnova arhitekture .NET- a je .NET okvir iliti *Framework*. To je Microsoftov okvir, tj. skup biblioteka koji povezuje programske jezike uključene u .NET platformu. .NET okvir omogućava korištenje istih klasa i objekata, njihovo nasljeđivanje i polimorfizam u različitim jezicima koje ta platforma podržava. Na taj način jezici u .NET platformi su u isto vrijeme i nezavisni i integrirani[2].

Prilikom kompilacije programa napisanih za .NET koristi se zajednička infrastruktura (eng. „*Common Language Infrastructure*“) za prijevod na nižu programsku razinu. Svi jezici .NET-a prevode se u zajednički posrednički jezik koji se naziva CIL (eng. „*Common Intermediate Language*“) koji je neovisan o platformi, a zatim se izvršavaju na CLR-u (eng. „*Common Language Runtime*“) koji je ovisan o platformi.

[1] http://www.efos.unios.hr/arhiva/dokumenti/RPA_P2_Kreiranje.pdf
[2] http://www.scribd.com/doc/220657903/Programsko-inzenjerstvo#scribd

3. OSNOVNE KARAKTERISTIKE C#

C# je dizajniran tako da omogućuje pet osnovnih karakteristika neophodnih za uspješno kreiranje svakog programa: jednostavnost, sigurnost, brzina, objektnu-orijentiranost i usmjerenost prema Internetu. To je jednostavan programski jezik koji u sebi sadrži oko 80 ključnih riječi i na desetke ugrađenih tipova podataka. Veoma je lagan za učenje i korištenje što je vjerojatno i razlog činjenici da ga korisnici više preferiraju od nekih drugih programskih jezika.

C# je u potpunosti objektno orijentirani jezik gdje je svaki podatak enkapsuliran u objekt neke klase. Sadrži sve dobre odlike takvog jezika koje većinom preuzima iz Jave i C++ jezika.

Iako je glavni zadatak jezika C# stvaranje i rad sa objektima najbolje je početi od osnovnih koraka, tj. od elemenata koji stvaraju objekte – varijabli i konstanti te njihovih tipova podataka.

3.1. Podatci i operatori

Kao i većina programskih jezika, C# upotrebljava varijable i konstante za rad s informacijama u memoriji. Glavna razlika između ova dva pojma je u tome da se vrijednosti varijable mogu mijenjati kroz program, dok se konstante koriste za memoriranje vrijednosti čija se vrijednost neće mijenjati kroz neku proceduru ili cijelu aplikaciju. Svaka varijabla i konstanta mora imati svoj jedinstveni identifikator. Identifikatori su nazivi koje programeri odaberu za svoje varijable, konstante, tipove, objekte i sl. Također bitno je napomenuti da se kod imena identifikatora mala i velika slova razlikuju te stoga C# nazive *mojaVar* i *MojaVar* tretira kao dva različita imena varijabli.

Kreiranje varijable izvodi se tako da deklariramo njezin tip te joj zatim dodijelimo naziv (identifikator). Moguće ju je inicijalizirati (dodijeliti joj neku vrijednost) prilikom deklariranja, a novu vrijednost može joj se dodijeliti u bilo kojem dijelu programa nakon njezine deklaracije.

Primjer 3.1. Deklaracija i inicijalizacija varijable

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            int var1; //deklaracija varijable
            var1 = 10; // inicijalizacija varijable
            var2 = 25; // promjena vrijednosti varijable var1
        }
    }
}
```

U primjeru 3.1. može se vidjeti postupak deklaracije varijable cjelobrojnog tipa (eng. „*integer*“) sa identifikatorom *var1*. Nakon toga varijabli *var1* dodijeljena je vrijednost 10 koja je zatim promijenjena u 25. Nakon provođenja ovog dijela koda vrijednost *var1* iznosi 25, sve dok se ona nanovo ne promijeni.

Kako bismo kreirali konstantu postupak deklaracije je jednak s tim da ispred tipa podataka potrebno je napisati ključnu riječ *const* kako bi dali do znanja procesoru da je riječ o konstanti. Ona se prilikom deklaracije mora i inicijalizirati, te se njezina vrijednost nakon toga više ne može promijeniti (ukoliko to pokušate procesor će izbaciti grešku).

Primjer 3.2. Deklaracija i inicijalizacija konstante

```
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            const int var2 = 7; // deklaracija i inicijalizacija konstante
            var2 = 10; // procesor javlja pogrešku
        }
    }
}
```

Tipovi podataka

C# je vrlo tipiziran jezik - pripada jezicima sa strogim određivanjem tipa[3] (eng. „*strongly typed*"), što znači da sve varijable i objekti moraju imati deklariran tip podatka. Također, prevoditelj će sprječavati pojavu pogrešaka tako što će inzistirati da objektima budu pridruženi samo podatci odgovarajućeg tipa. Tip objekta ili varijable govori prevoditelju koliko je on velik i koje su njegove mogućnosti (npr. gumbi se mogu povući, pritisnuti i sl.). Varijablama je moguće dodijeliti neki od ugrađenih tipova podataka koji su dio samog jezika (npr. *int* ili *char*) ili neki od korisnički-definiranih tipova podataka poput klase ili strukture.

C# podržava tri osnovna ugrađena tipa podataka[4]:

- numeričke – koriste se za predstavljanje cijelih i realnih brojeva. Najpoznatiji takvi tipovi te njihov raspon i zauzeće memorije navedeni su dolje u tablicama.

Tab. 3.1. Numerički tipovi podataka za prikaz cijelih brojeva

Tip	Opseg vrijednosti	Zauzeće memorije (bajtovi)
sbyte	-128 do 127	1
byte	0 do 255	1
short	-32 768 do 32 767	2
ushort	0 do 65 536	2
int	- 2 147 483 648 do 2 147 483 647	4
uint	0 do 4 294 967 295	4
long	-9 223 372 036 854 775 808 do 9 223 372 036 854 775 807	8
ulong	0 do 18 446 744 073 709 551 615	8

Tab. 3.2. Numerički tipovi podataka za prikaz realnih brojeva

Tip	Opseg vrijednosti
float	$\pm 1.5 \times 10^{-45}$ do $\pm 3.4 \times 10^{38}$
double	$\pm 5 \times 10^{-324}$ do $\pm 1.7 \times 10^{308}$
decimal	$\pm 1 \times 10^{-28}$ do $\pm 7.9 \times 10^{28}$

[3] https://www.fer.unizg.hr/_download/repository/11._Programski_jezici,_otvorenost.pdf
[4] R. Miles: C# Programming, str.26.-30.

- znakovne – tip *char* koji predstavlja samo jedan tekstualni znak. Zauzima 2 bajta u memoriji, a vrijednosti se pišu unutar jednostrukih navodnika, npr. '*a*', '*B*' i tip *string* koji predstavlja niz znakova i čije vrijednosti se pišu unutar dvostrukih navodnika, npr. „*Ovo je jedan string.*“.
- logičke – tip *bool* koji može poprimiti samo dvije vrijednosti: točno (eng. „*true*“) ili netočno (eng. „*false*“). Zauzima jedan bajt memorije.

Druga bitna podjela tipova podataka jest na vrijednosne (eng. „*value type*“) i referentne tipove (eng. „*reference type*“). Glavna razlika između ova dva tipa je njihov način pohranjivanja vrijednosti u memoriju. Varijable koje imaju vrijednosni tip direktno sadržavaju vrijednost, što znači da kada pridružujemo jednu varijablu vrijednosnog tipa drugoj varijabli, vrijednost se kopira. Nasuprot tomu, varijable referentnog tipa čuvaju samo reference na stvarne podatke tako da ako pridružujemo jednu varijablu referentnog tipa drugoj varijabli kopira se samo referenca, ali ne i sami objekt. Vrijednosnom tipu podataka pripadaju svi gore navedeni tipovi, a osnovni referentni tipovi su klasa i sučelje te ugrađeni referentni tipovi objekt i *string*.

Operatori

Postoje dvije glave podjele operatora: prema broju operatora i prema vrsti operacija koja se koristi.

S obzirom na broj operatora koje koriste dijelimo ih na:

- unarne – koriste samo jednu varijablu i rezultat je također samo jedna vrijednost
- binarne – vrše operaciju nad dvije varijable, a rezultat je jedna vrijednost
- ternarni – postoji samo jedan takav operator nazvan uvjetni operator „***?:***“ (npr. uvjet ? izraz1 : izraz2) koji radi na principu da prvo provjerava uvjet i kao rezultat toga vraća vrijednost tipa *bool*. Zatim, ako je uvjet istinit poziva *izraz1*, u suprotnom poziva *izraz2*. Ovaj operator koristi se kao zamjena za *if-else* petlju.

S obzirom na vrstu operacije koja se koristi, operatore dijelimo na aritmetičke, logičke, relacijske i operatore za rad sa bitovima te kao posebna vrsta postoje i operator pridruživanja, inkrement i dekrement. Naravno, postoji još mnogo naprednijih operatora, no ovdje su navedeni samo oni najbitniji i najčešće korišteni.

Operator pridruživanja uzrokuje promjenu vrijednosti operanda s lijeve strane operatora u vrijednost s desne strane. Inkrement koristimo za povećavanje vrijednosti varijable za 1, a dekrement za njezino smanjivanje za 1. Aritmetičkih operatora ima 5 i koristimo ih za osnovne aritmetičke operacije: zbrajanje, oduzimanje, množenje, dijeljenje i vraćanje ostatka. Relacijski operatori se koriste za usporedbu dvije vrijednosti i uvijek vraćaju logičku vrijednost 0 ili 1(*true* ili *false*), dok se logički operatori koriste uglavnom za povezivanje uvjeta u petljama. U tablici ispod nalazi se prikaz ovih operatora sa njihovim nazivima i simbolima[5].

Tab. 3.3. Osnovni operatori u C#

Aritmetički operatori	
Naziv	Operator
zbrajanje	+
oduzimanje	-
množenje	*
dijeljenje	/
ostatak dijeljenja	%
Logički operatori	
Naziv	Operator
I	&&
ILI	||
NE	!

Relacijski operatori	
Naziv	Operator
jednako	= =
nije jednako	!=
veće	>
veće ili jednako	>=
manje	<
manje ili jednako	<=
Posebni operatori	
Naziv	Operator
pridruživanje	=
inkrement	++
dekrement	--

3.2. Tokovi programa

Najjednostavniji programi imaju linearni tijek izvođenja programa, što znači da se naredbe izvode redom jedna po jedna, dok program ne dođe do kraja i onda stane. Vrlo često dolazi do pojava

[5] M.Michaelis, E.Lippert: Essential C# 5.0

situacija kada takav tijek izvođenja nije moguć, već se program mora mijenjati ovisno o primljenim podatcima.

Postoje tri osnovna tijeka programa:

- pravocrtno (linearno) izvođenje
- grananje – naredbe *if-else*, *switch-case* i uvjetni operator ?: objašnjen u prethodnom poglavlju
- petlje – *for*, *while* i *do-while*[6]

Naredba *if-else* služi za uvjetno grananje programa – ukoliko je točan postavljeni uvjet izvršava se dio koda iza *if* naredbe, a u suprotnom izvršava se dio koda iza *else* naredbe. Također, blokovi *if* naredbi mogu se ugnijezditi jedan unutar drugoga, tj. postoji dva ili više uvjeta i ovisno o onom koji se zadovolji izvodi se njemu dodijeljeni dio koda.

Primjer 3.3. Primjena *if-else* naredbe

Kreiran je jednostavan program koji provjerava da li je broj paran ili neparan i u ovisnosti o tome ispisuje poruku.

```
static void Main(string[] args)
{
   Console.WriteLine("Unesite pozitivan cijeli broj:");
   int a = Console.ReadLine();
   // učitava u a broj koji korisnik unese
   if (a % 2 == 0) Console.WriteLine("Broj je paran!");
   // ako kada a podijelimo sa 2 ostatak je 0 ispiši broj je neparan
   else Console.WriteLine("Broj je neparan!");
   // inače ispiči broj je paran
}
```

Switch-case naredba koristi se u slučajevima kada rezultat izračunavanja u nekom uvjetu daje više cjelobrojnih rezultata, a za svaki od njih treba provesti različite odsječke grananja. Princip rada jednak je kao kod ugniježđenih *if* naredbi, no ova naredba se većinom koristi zbog preglednosti[7].

[6] R. Miles: C# Programming, str.34.
[7] http://ss-tehnicka-ck.skole.hr/upload/ss-tehnicka-ck/newsattach/130/CGrananjaPetlje.pdf

Primjer 3.4. Primjena *switch-case* naredbe

Napravljen je program koji sadrži cjelobrojnu varijablu *dan* i u ovisnosti o unesenom broju ispisuje koji je to dan.

```
static void Main(string[] args)
{
	int dan;
	dan = Convert.ToInt32(Console.ReadLine());
	switch (dan){
	case (1): Console.WriteLine("Ponedjeljak"); break;
	// ako je unesen broj 1 izvršava se sve iza case (1)
	// break - zadovoljen je uvjet i izlazi se iz petlje
	case (2): Console.WriteLine("Utorak"); break;
	case (3): Console.WriteLine("Srijeda"); break;
	case (4): Console.WriteLine("Četvrtak"); break;
	case (5): Console.WriteLine("Petak"); break;
	case (6): Console.WriteLine("Subota"); break;
	case (7): Console.WriteLine("Nedjelja"); break;
	default: Console.WriteLine("Krivo uneseni broj!"); break;
	// ako nijedan uvjet nije zadovoljen izvršava se kod iza default
}
```

Petlje u programu se koriste ako jedan dio programa treba nekoliko puta ponoviti. Ukoliko nam je broj ponavljanja poznat unaprijed koristi se *for* petlja, ako je broj ponavljanja nepoznat koristi se *while* petlja. Za obje petlje karakteristično je da se uvjet provjerava prije izvođenja naredbi. Ukoliko je potrebno da se naredbe unutar petlje prvo jednom izvedu i onda u ovisnosti o njihovom ishodu ona eventualno ponavlja koristi se *do-while* petlja.

Primjer 3.5. Primjena *for* petlje

```
static void Main(string[] args)
{
	int a = 0, i;
	for (i = 1; i < 11; i++)
	{
		a = a + i;
	}
	Console.WriteLine(a);
}
```

Napisani program zbraja sve brojeve od 1 do 10 i sprema ih u varijablu a. Primjena petlje je vrlo jednostavna – varijabla a u početku je iznosila 0, zatim joj je unutar petlje dodan broj 1, pa 2 i tako redom sve dok joj nije dodan broj 10 što predstavlja izlaz iz petlje.

Primjer 3.6. Primjena while i do-while petlje

```
static void Main(string[] args)
{
	int broj;
	Console.WriteLine("Unesite broj između 1 i 100:");
	broj = Convert.ToInt32(Console.ReadLine());
	while (broj < 1 || broj > 100)
	{
		Console.WriteLine("Krivo unesen broj!");
		Console.WriteLine("Unesite broj između 1 i 100:");
		broj = Convert.ToInt32(Console.ReadLine());
	}
```

```
static void Main(string[] args)
{
	int broj;
	do
	{
		Console.WriteLine("Unesite broj između 1 i 100:");
		broj = Convert.ToInt32(Console.ReadLine());
	}
	while (broj < 1 || broj > 100);
}
```

U oba slučaja rađen je isti program koji traži unos broja između 1 i 100 sve dok korisnik ne unese točan broj. Oba koda su ispravna, no može se vidjeti da ako se koristi *while* petlja umjesto *do-while* dolazi do ponovljenih linija koda.

3.3. Klase i objekti

Klasa je najmoćniji podatkovni oblik u C#. Pripada korisnički definiranim tipovima podataka, a podatke koje definiramo ovakvim tipom nazivamo objektima. Jednostavno rečeno, klasa je nacrt iliti opis objekta, a objekt je konkretna realizacija klase. Klasa se sastoji od atributa koji definiraju podatke objekta i metoda koje definiraju njegovo ponašanje.

Prava pristupa podatcima

Prije nego krenemo na daljnje objašnjavanje klasa i objekata potrebno je razumjeti prava pristupa unutar C# koja su ključna za dobro rukovanje klasama. Postoje tri vrste prava pristupa:

- ***public*** – sve funkcije imaju pravo pristupa klasi

- ***private*** – pravo pristupa imaju metode iz klase i prijateljske funkcije[8] (funkcije deklarirane izvan klase koje mogu pristupiti svim podatcima klase)
- ***protected*** – koristi se kod nasljeđivanja; pravo pristupa imaju metode iz klase, prijateljske funkcije te metode i prijateljske funkcije izvedenih klasa

Definiranje klase

Svaku klasu definiramo izvan *main* funkcije pomoću ključne riječi *class* i njezinog identifikatora. Ostatak definicije klase nalazi se unutar vitičastih zagrada gdje se definiraju njezini atributi i metode. Ispred ključne riječi *class* nalazi se oznaka za pravo pristupa koju određuje sam programer ovisno o svojim željama i potrebama programa. Ukoliko pravo pristupa nije posebno naznačeno, neovisno da li je riječ o atributima, metodama ili cijeloj klasi, prevoditelj će takve podatke gledati kao *private.*

Primjer 3.7. Definiranje klase, njezinih atributa i metoda

```
public class Osoba
{
    public string ime; // definicija atributa ime, prezime i starost te
                       njihovih tipova
    public string prezime;
    public int starost;
    public string puno_ime() // metoda koja vraća ime i prezime osobe
    {
        return ime + " " + prezime;
    }
    public string svi_podatci() // metoda koja vraća sve podatke o osobi;
    {
        return ime + " " + prezime + " " + starost;
    }
}
```

Definiranje objekata i njihovih parametara

Realizaciju napravljene klase ostvarujemo kreiranjem objekta. Objekt možemo kreirati na dva načina:

- ime_klase + identifikator_objekta = *new* ime_klase();

[8] https://loomen.carnet.hr/pluginfile.php/281593/mod_resource/content/1/Predavanja/10._Enkapsulacija.pdf

- *var* + identifikator_objekta = *new* ime_klase();

Razlika je jedino u prvoj riječi naredbe – ukoliko ne napišemo ime klase, već ključnu riječ *var* kompajler sam određuje njezin povratni tip na temelju ostatka koda.

Dodjeljivanje vrijednosti atributima klase možemo također izvršiti na 2 načina:

- konstruiranjem metode za njihovo inicijaliziranje
- posebnom inicijalizacijom svakog atributa u glavnom programu (moguće je jedino ako su atributi postavljeni na *public*)

Svim atributima i metodama iz klase pristupamo tako da upišemo ime objekta i znak „**.**“ ukoliko se članovima klase pristupa preko imena objekta ili reference ili znakom „->“ ako im se pristupa preko pokazivača. Nakon toga program otvara padajući izbornik svih atributa i metoda kojima je moguće pristupiti.

Primjer 3.8. Definiranje objekta iz klase *Osoba* i njegovih parametara; pozivanje metoda klase *Osoba*

```
class Program
    {
        static void Main(string[] args)
        {
            Osoba osoba1 = new Osoba(); // kreiranje novog objeka klase osoba
            osoba1.ime = "Marko"; // definiranje parametara ime, prezime i
                                  starost
            osoba1.prezime = "Marić";
            osoba1.starost = 25;
            Console.WriteLine(osoba1.puno_ime()); // pozivanje metode
                                                     puno_ime
            Console.WriteLine(osoba1.svi_podatci()); // pozivanje metode
                                                        svi_podatci
        }
    }
```

Rezultat koda:

Konstruktor i destruktor

Konstruktor je posebna metoda koja se zove jednako kao i klasa i nema povratni tip, tj. konstruktor nam ne vraća ništa već samo obavlja neku radnju prilikom inicijalizacije objekta.

Postoje dvije vrste konstruktora: podrazumijevani (defaultni) i parametarski.

Podrazumijevani[9] je onaj konstruktor koji ne prima nikakve parametre. Većinom se koristi za postavljanje vrijednosti atributa na neku osnovnu vrijednost. Ukoliko u klasi programer taj konstruktor ne definira prevoditelj ga sam automatski stvara.

Nasuprot tomu, parametarski je onaj konstruktor kojem pri inicijalizaciji predajemo neke parametre. Vrijednost tih parametara potrebno je navesti pri deklaraciji objekta. U suprotnom pozvati će se podrazumijevani konstruktor.

Primjer 3.9. Definiranje podrazumijevanog i parametarskog konstruktora

```
namespace ConsoleApplication1
{
      public class Osoba
      {
         public string ime;
         public string prezime;
         public int starost;
         public Osoba() // podrazumijevani konstruktor
         {
             ime = "Davor";
             prezime = "Horvat";
             starost = 30;
         }
         public Osoba(string x, string y, int z) // parametarksi konstruktor
         {
             ime = x;
             prezime = y;
             starost = z;
         }
         public string puno_ime()
         {
             return ime + " " + prezime;
         }
         public string svi_podatci()
         {
             return ime + " " + prezime + " " + starost;
```

[9] https://loomen.carnet.hr/pluginfile.php/281593/mod_resource/content/1/Predavanja/10._Enkapsulacija.pdf

```
        }
    }
```

```
    class Program
    {
        static void Main(string[] args)
        {
            Osoba osoba1 = new Osoba(); // poziva se podrazumijevani
                                        konstruktor
            Osoba osoba2 = new Osoba("Marko", "Marić", 25); // poziva se
                                                parametarski konstruktor
            Console.WriteLine(osoba1.ime + osoba1.prezime + osoba1.starost);
            Console.WriteLine(osoba2.ime + osoba2.prezime + osoba2.starost);
        }
    }
}
```

Destruktor je također posebna vrsta metode koja se zove jednako kao klasa (s znakom ~ ispred imena). Destruktor ne može primati parametre, poziva se kada želimo uništiti objekt iz memorije, a definira na kraju klase.

Primjer 3.10. Definiranje destruktora

```
~Osoba() // destruktor
{
    Console.WriteLine("Pozvan je destruktor!");
}
```

3.4. Osnove programiranja u C#-u

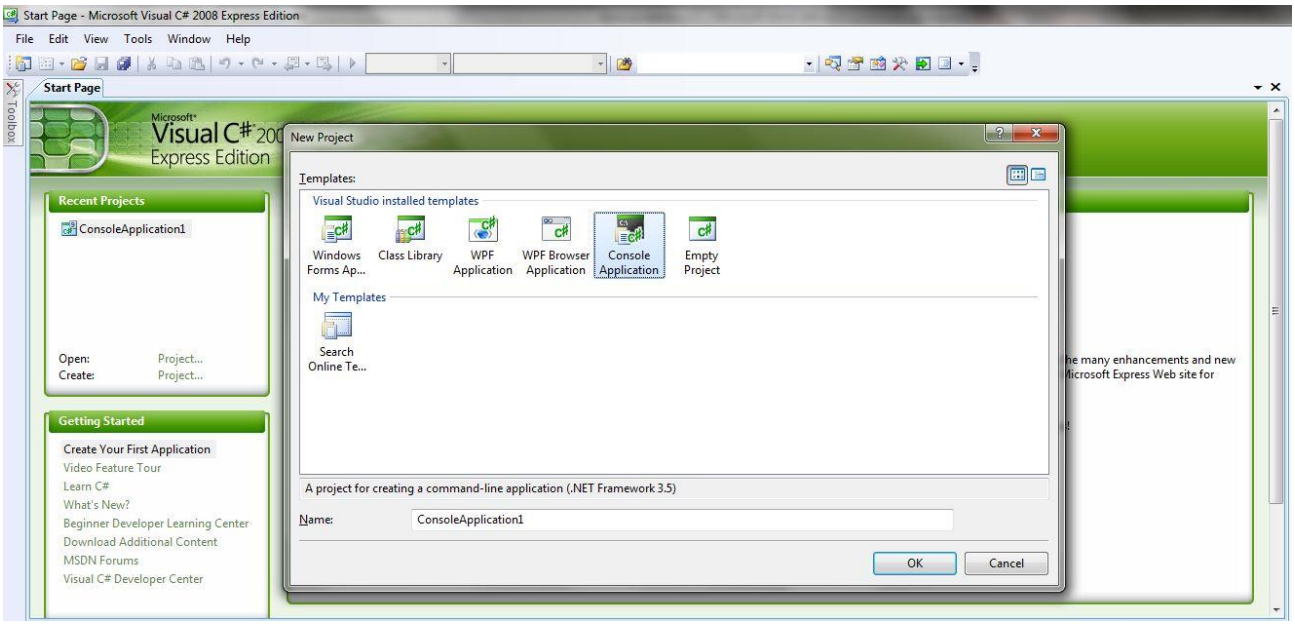
Unutar ovog poglavlja upoznati ćemo se sa procesom izrade osnovnog C# programa, njegovim bibliotekama, osnovama upisa i ispisa vrijednosti, načinima grananja, osnovnim svojstvima objektno orijentiranog programiranja te njihovom implementacijom u programe i slično.

Također, bitno je napomenuti da se svi navedeni primjeri rade pomoću programa Microsoft Visual C# 2008 Express Edition.

Kreiranje projekta

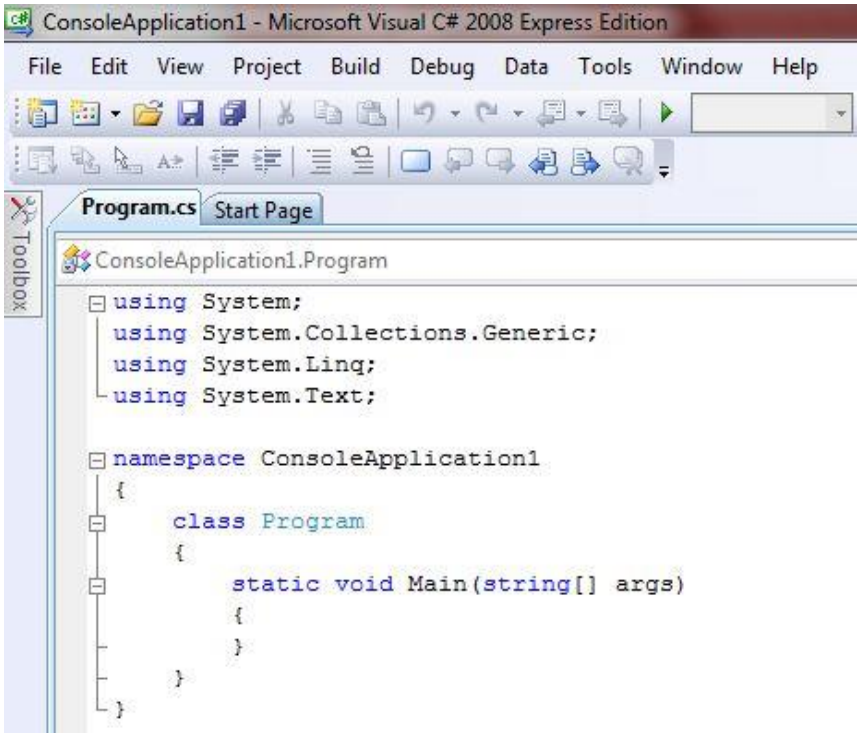
Kako bismo kreirali novi projekt koji ćemo raditi potrebno je slijediti naredne korake:

File → *New project* → u prozoru koji se otvori pod *Templates* odabrati *Console application* → upisati željeno ime projekta → *OK*.



Sl. 3.1. Kreiranje projekta

Nakon što je program kreiran otvara se novi prozor za pisanje našeg koda čiji izgled je prikazan sa slici 3.2.



Sl. 3.2. Izgled početnog programa

Na početku .cs datoteke nalaze se 4 *using* naredbe koje nam govore da C# automatski uključuje četiri imenika: *System*, *System.Collections.Generic*, *System.Linq* i *System.Text*. Unutar tih imenika nalazi se veliki broj već gotovih klasa koje su neophodne za pisanje svakog programa. Nakon uključenja navedenih imenika kreira se prostor imena sa nazivom naše aplikacije i unutar toga se generira nova klasa *Program* sa jednom statičnom metodom *void Main*. Ona predstavlja ulaznu točku kreiranog programa i izvršava se svaki put kada se program pokrene.

Unos i ispis podataka

Za ispis podataka na ekran postoji dvije moguće funkcije:

- *Console.Write()* – ispisuje tekst na ekran, a kursor ostavlja u istome redu
- *Console.WriteLine()* – prebacuje kursor u novi red

Unutar uglatih zagrada funkcije unutar navodnika upisujemo tekst koji želimo zatim ispisati na ekran.

Primjer 3.11. Ispis poruke na ekran

```
namespace ispis
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Ispis funkcije Console.Wrrite.");
            Console.WriteLine("Ispis funkcije Console.Writeline.");
        }
    }
}
```

Ako pokrenemo ovaj dio koda kao rezultat dobiti ćemo sliku 3.4.3., no ako obrnemo redoslijed naredbi rezultat poprima oblik slike 3.4.4. Promatrajući oba primjera veoma lako je uočiti razliku u pozicijama kursora nakon izvršenja svake naredbe.

Sl. 3.3. Ispis funkcije Console.Write pa Console.WriteLine

Sl. 3.4. Ispis funkcije Console.WriteLine pa Console.Write

Za učitavanje podataka sa tipkovnice koristimo funkciju *Console.ReadLine()*. Ukoliko želimo da taj podatak se ne samo pročita, već i da ga program upamti moramo ga spremiti u neku varijablu. Također, ova funkcija čita sve kao *string* tip podatka. Ako nam je potreban neki drugi tip podatka moramo pretvoriti ulaz u željeni tip.

Primjer 3.12. Unos vrijednosti varijabli *var1* i *var2*

```
namespace Ucitavanje
{
    class Program
    {
        static void Main(string[] args)
        {
            string var1 = Console.ReadLine();
```

```
                // Console.Readline pročita uneseni niz sa tipkovnice i sprema ga
                // u varijablu var1 tipa string
                int var2 = Convert.ToInt32(Console.ReadLine());
                // unešeni niz za tipkovnice pretvara u 32-bitni int i zatim
                //sprema u var2 tipa int
            }
        }
    }
```

4. WINDOWS APLIKACIJE

Prilikom upoznavanja sa osnovama C# programskog jezika svi primjeri rađeni su preko takozvanih konzolnih aplikacija. Osnovna karakteristika konzolnih aplikacija jest da rade u *command prompt*-u i nemaju grafičko sučelje. Premda se one mogu vrlo jednostavno implementirati, postoje veće i bolje mogućnosti C# razvojne okoline. Glavni razlog velike korištenosti i popularnosti jezika C# je izgradnja Windows i Web aplikacija.

Ono što razlikuje Windows aplikacije od konzolnih aplikacija je grafičko sučelje. Alat za izradu Windows aplikacija naziva se *Windows Forms*. Forma predstavlja strukturirani prozor koji sadrži prezentacijske elemente za prikaz i unos podataka i one su najučinkovitiji i najjednostavniji način komunikacije korisnika sa programom[10]. Dvije osnovne vrste formi su:

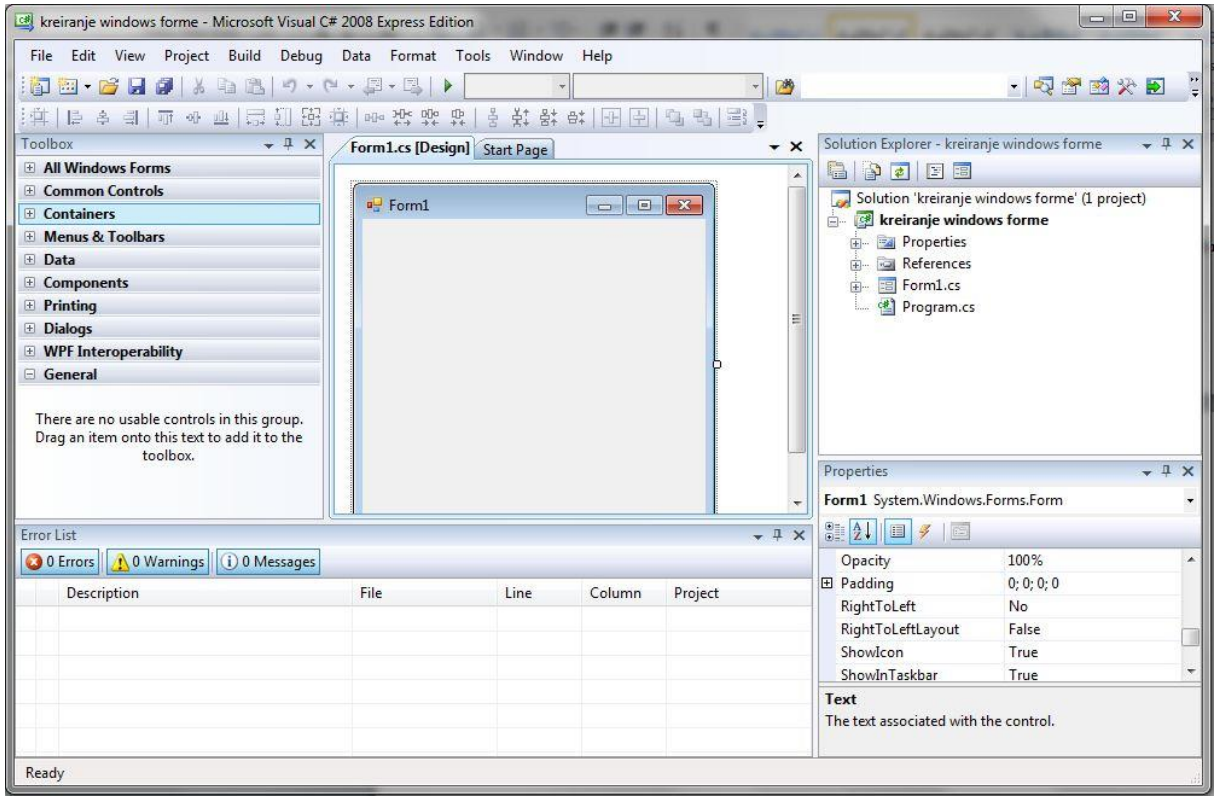
- SDI forme (eng. „*Single Documet Interace*“) – forme za sučelje jednog dokumenta. Za prikaz svakog dokumenta koristi se posebna forma, a svaka forma promatra se kao odvojeni prozor.
- MDI forme (eng. „*Multiple Document Interface*“) – forme za sučelje više dokumenata. Postoji jedan glavni prozor (forma), dok se svi ostali prozori (forme) se otvaraju preko njega. Glavna forma naziva se MDI roditelj, a ostale povezane forme nazivaju se MDI potomci[11].

4.1. Kreiranje i pokretanje Windows formi

Kako bismo kreirali novu formu, prvo je potrebno otvoriti Visual C# i odabrati: *File → New project*. Nakon ovog odabira otvara se prozor pod nazivom *New project*, prikazan prethodno na slici 3.4.1.. Unutar njega moguće je odabrati željenu vrstu aplikacije (u ovom slučaju pod dijelom *Templates* odabire se *Windows Forms Apllication*) i nadjenuti joj ime. Visual C# odgovara izradom nove Windows forme i postavlja vas u okoliš za razvoj (slika 4.1.) koji se sastoji od nekoliko osnovnih prozora:

[10]https://loomen.carnet.hr/pluginfile.php/373293/mod_resource/content/1/LV13%20-%20Rad%20s%20Windows%20formama.pdf
[11] http://www.efos.unios.hr/arhiva/dokumenti/RPA_P7_Forme_izbornici.pdf

- Dizajn (eng. „*Design*“) - prikazuje prazan obrazac kreirane forme. Kako bi se vidio njezin pozadinski kod koji obrađuje različite događaje potrebno je kliknuti desnom tipkom miša na samu formu i odabrati *View Code*.
- Istraživač rješenja (eng. „*Solution Explorer*“) – prikaz organizacije projekta (svih datoteka uključenih u projekt)
- Svojstva (eng. „*Properties*“) – prikazuje svojstva označenog elementa. U ovom slučaju kreirana forma nema niti jedan element u sebi, stoga je označena ona i prikazuju se njezina svojstva
- Alati (eng. „*Toolbox*“) – skup gotovih kontrola. Ovaj prozor može, ali i ne mora biti prikazan pri prvom pokretanju forme. Ukoliko nije prikazan potrebno je u izbornoj traci programa odabrati *View* → *Toolbox* kako bi se prikazala ili to učiniti putem prečaca *Ctrl-Alt-X* na tipkovnici.



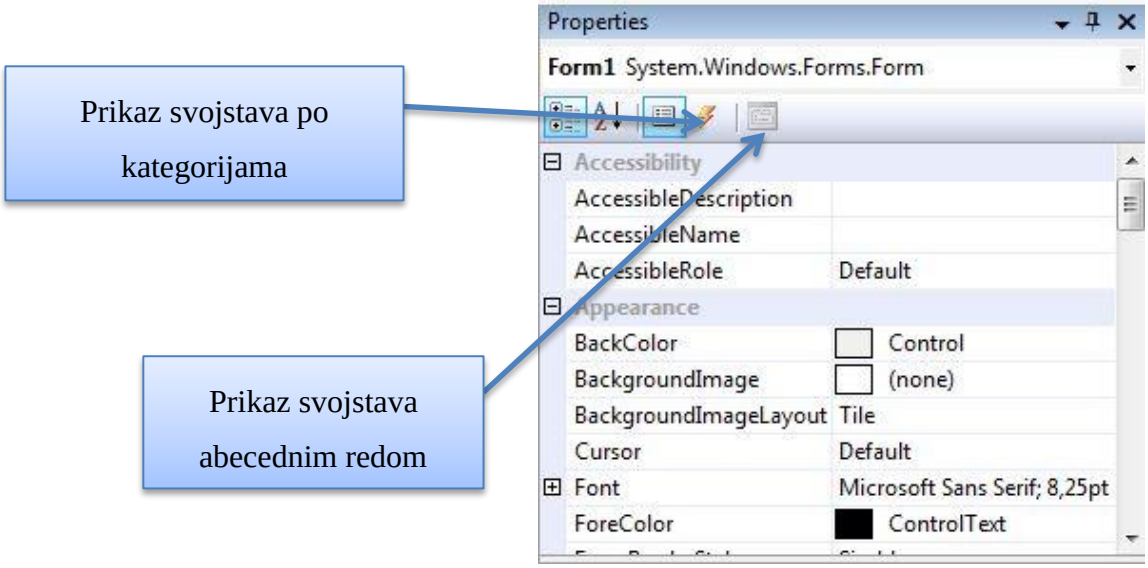
Sl. 4.1. Razvojni okoliš forme

Svojstva forme

Kao što je već rečeno u proteklom poglavlju, svojstva forme prikazana su u prozoru *Svojstva* kada se mišem označi forma. Moguće je odabrati način prikazivanja svojstava: abecednim redom ili prema kategorijama. Svojstva forme podijeljena su u devet kategorija:

- Pristupačnost (eng. „*Accessibility*“) – omogućava pravljenje aplikacije za širok izbor korisnika
- Izgled (eng. „*Apperance*“) – podešavanje boje ili slike pozadine, prikaz kursora, boje i tipa fonta..
- Ponašanje (eng. „*Behavior*“) – podešava ponašanje forme, npr. da li će forma biti omogućena
- Podatci (eng. „*Data*“) - najkorištenije svojstvo unutar ove kategorije je *Oznaka* (eng. „*Tag*“) koje omogućava programeru dodijeljivanje vlastitih oznaka. Većinom se koristi u kontrolama za vlastite potrebe i lakše snalaženje programera
- Dizajn (eng. „*Design*“) – određuje naziv forme, jezik, zaključavanje…
- Fokus (eng. „*Focus*“) - sadrži svojstvo *CauseValidation* koje ako je postavljeno pokreće validaciju kada se neka kontrola unutar forme promijeni[12]
- Predložak (eng. „*Layout*“) – određuje veličinu forme, njezinu poziciju pri pokretanju,
- Različito (eng. „*Misc*“) – podešavanje koje tipke će se koristiti kao gumbi za prihvaćanje i odustajanje (accept i cancel)
- Stil prozora (eng. „*Window Style*“) – određivanje da li je moguće povećati i smanjiti prozor (*minimize* i *maximize* gumbi), da li se pokazuje ikona za pomoć, određivanje ikone programa…

12 https://books.google.hr/books?id=RlM8tSROaKsC&pg=PA129&lpg=PA129&dq=c%23%20Accessibility%20Appearance%20Behavior%20Data%20Design%20Focus%20Layout%20Misc%20Window%20Style&source=bl&ots=_lyzwznKcr&sig=CcRwGGqE74ehxN0lC0hlEjAtnZ0&hl=hr&sa=X&ved=0ahUKEwis6ZndjJvPAhWI_ywKHdVGCrYQ6AEIKzAD#v=onepage&q=c%23%20Accessibility%20Appearance%20Behavior%20Data%20Design%20Focus%20Layout%20Misc%20Window%20Style&f=false



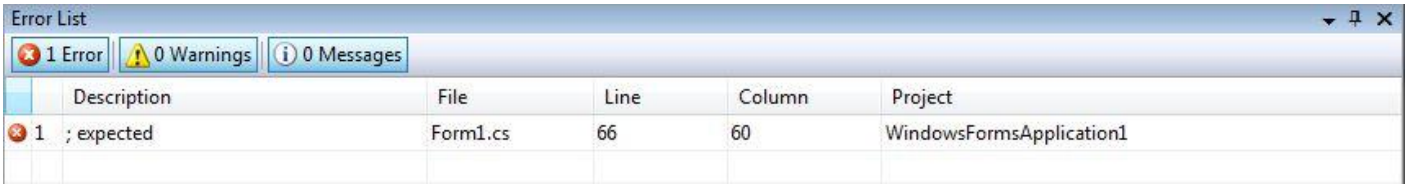
Sl. 4.2. Način prikaza svojstava forme

Provjera grešaka unutar forme

Prevođenje (eng. „compile“) Windows aplikacije izvršava se pomoću alata *Build Solution* koji se nalazi u izborniku *Build* ili jednostavnije, pritiskom na prečac *F6*. Proces prevođenja svodi se na prevođenje upisanih naredbi u strojni kod te detekcija mogućih grešaka. Ukoliko nema pogreški kada se završi prevođenje u prozoru *Error list* koji se nalazi na dnu programa pojaviti će se poruka da je operacija *Build* uspješno završena i da prilikom toga nisu pronađene pogreške kao što je prikazano na slici 4.1.3. Ako Build-er naiđe na neke pogreške, to će prikazati na način prikazan slikom 4.1.4. – prikazati će se pogreške, njihova vrsta te u kojem dokumentu i u kojoj liniji koda se nalaze. Također, dvostrukim klikom na određenu pogrešku, program sam odvodi korisnika na točno mjesto pogreške.

```
Compile complete -- 0 errors, 0 warnings
WindowsFormsApplication1 -> C:\Users\Valentina\AppData\Local\Temporary Projects\WindowsFormsApplication1\bin\Release\WindowsFormsApplication1.exe
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Sl. 4.3. Uspješno završena operacija *Build*

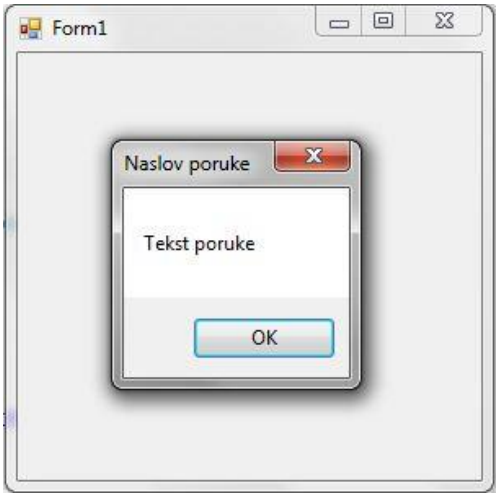


Sl. 4.4. Prikaz grešaka prilikom izvođenja operacije *Build*

Druga vrsta alata za prijevod i ispravak pogrešaka je *Debug* (izbornik *Debug* → *Start Debugging*) ili pritiskom na tipku F5. Glavna razlika između ova dva alata je to što *Debug* nakon prevođenja, ukoliko nije došlo do pogrešaka, pokreće program, dok *Build* to ne radi. Također, *Build* provjerava samo sintaktičke greške (krivo napisane naredbe), dok *Debug* provjerava još i semantičke greške (program ne radi za ono za što je namijenjen) i greške izuzetaka (pojavljuju se prilikom korištenja programa, neizbježne su, a uzroci mogu biti različiti).

Poruke

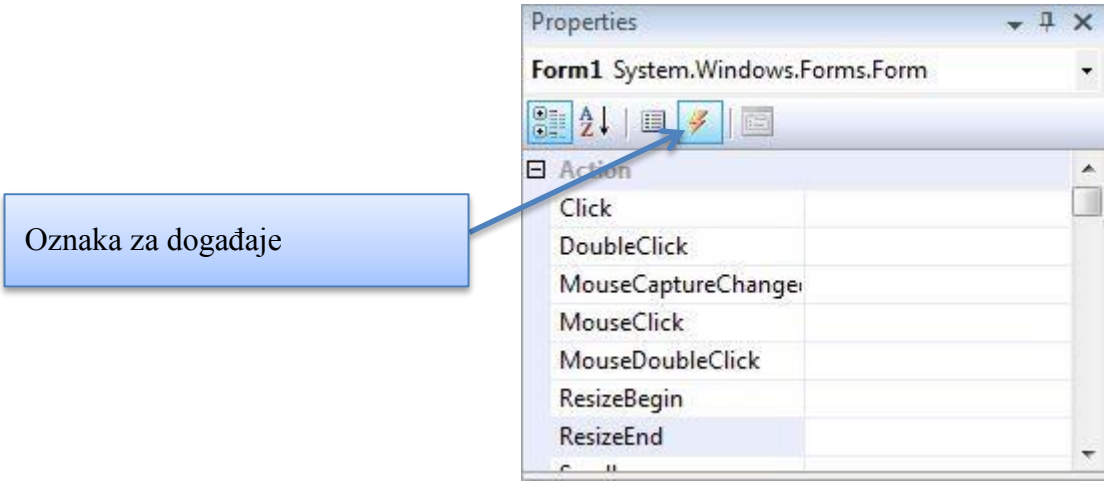
Poruke unutar Windows formi prikazuju se u obliku pop-up prozora. Sintaksa pri pisanju koda poruke je slijedeća: `MessageBox.Show("Tekst poruke", "Naslov poruke");`, gdje je *MessageBox* javni razred, a *Show* jedna od njegovih klasa i služi za prikaz.



Sl. 4.5. Izgled poruke

4.2. Događaji

Događaji (eng. „events“) su radnje koje se mogu izvršiti nad formom ili nekim njezinim elementom. Unutar prozora *Svojstva* nalazi se popis dostupnih događaja nad formom ili elementom koji je označen. Postoji jako velik broj događaja, no kod većine događaja njihovo ime opisuje na što se odnose.



Sl. 4.6. Prikaz mogućih događaja i njihovo mjesto unutar programa

Rukovoditelj događaja (eng. „event handler“) je procedura u kodu koja određuje radnje koje se izvode kada se događaj dogodi (npr. kliknut je miš, otvorena je druga forma, označi se okvir za izbor…). Dvoklikom desno od imena događaja program sam generira proceduru, tj. rukovoditelja događaja, a na programeru je da unutar nje napiše kod koji će se izvršiti.

4.3. Osnovne kontrole Windows formi

Kontrole unutar Windows formi su najjednostavnije rečeno ono što postavimo na formu. Sve kontrole nalaze se unutar prozora *Alati*. To mogu biti razni gumbi, polja za unos teksta, kontrola za odabir datuma i mnoštvo drugih. Sve kontrole su zapravo instance različitih klasa i sukladno tome za njih vrijede pravila objektno orijentiranog programiranja.

Konzole u formu dovodimo jednim od dva jednostavna načina:

- dvoklikom na traženu kontrolu unutar prozora *Alati* nakon čega će se ona pojaviti na nekom slučajnom mjestu u formi (većinom oko lijevog gornjeg kuta forme)
- „*drag and drop*“ principom – željenu kontrolu dovučemo iz prozora *Alati* na željeno mjesto[13]

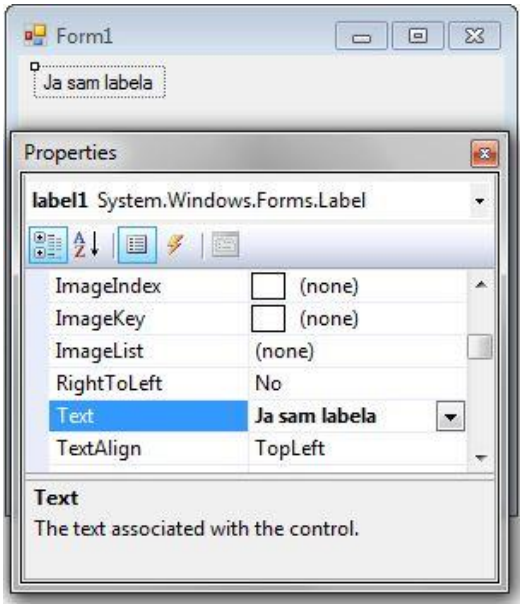
Svaku kontrolu moguće je naknadno premjestiti, promijeniti joj širinu ili visinu i mijenjati njezino poravnanje u odnosu na cijelu formu ili u odnosu na ostale objekte.

Oznake

Oznak (eng. „label“) je kontrola koja daje mogućnost prikaza teksta na formu. Može stajati kao samostalan tekst ili biti povezana sa nekom drugom kontrolom tako da opisuje njezino značenje. Također, oznaka nema mogućnost korisničkog unosa teksta već njezin sadržaj unosi sam programer prilikom pravljenja forme.

Nakon što postavimo oznaku unutar forme, klikom na nju unutar prozora *Svojstva* pojaviti će se sva njezina svojstva koja možemo promijeniti. Najvažnije svojstvo oznaka jest *Text* unutar kojeg se nalazi njezin sadržaj. Kao dokaz toga na slici 4.3.1. moguće je vidjeti kako su sadržaj polja *Text* i tekst oznake u formi jednaki.

[13] http://carpediem.hr/PublikacijeCarpeDiem/Publikacije/C%23%20programiranje.pdf



Sl. 4.7. Postavljanje sadržaja oznake

Gumbi

Gumb (eng. „button“) je najjednostavnija kontrola. Ono omogućava korisniku da klikom na njega pokrene odgovarajući dio programskog koda. Ako dvokliknemo na gumb unutar forme program će nas direktno odvesti na dio koda koji se odnosi na njega unutar kojeg se piše kod za ono što želimo da gumb uradi.

Najjednostavnija funkcija gumba jest da dvoklikom na njega se pojavi neka poruka:

```
private void button1_Click(object sender, EventArgs e)
{
      MessageBox.Show("Kliknuli ste gumb!");
}
```

Naravno, gumbe je moguće koristiti i uz drugačije događaje osim klasičnog dvoklika koje je moguće pronaći u prozoru *Svojstva.*

Tekstualno polje

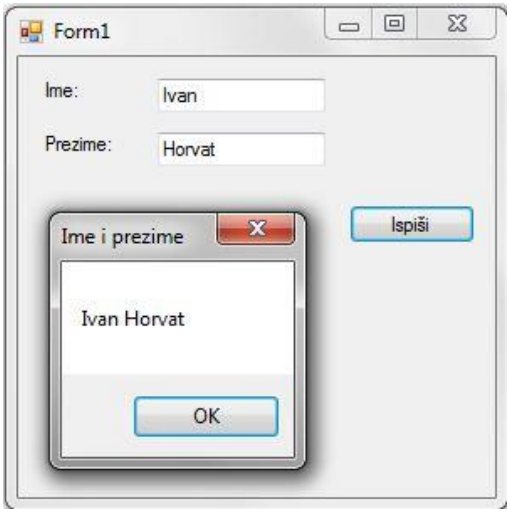
Tekstualno polje (eng. „*Textbox*“) ili linija za unos teksta je kontrola koja omogućava korisniku da unese podatke koji se dalje koriste u aplikaciji. Možemo joj promijeniti sva osnovna svojstva kao što

su veličina i boja fonta ili veličina. Posjeduje i dodatne mogućnosti ograničavanja broja unesenih karaktera (svojstvo *MaxLength*) te određivanja da li će se tekst unositi kroz jednu liniju teksta ili više (svojstvo *Multiline*). Od događaja, u kombinaciji sa tekstualnim poljem najviše se koristi *TextChanged* koji se pokreće svaki puta kada korisnik unese ili obriše karakter u kontroli[14].

Primjer 4.1. Korištenje tekstualnog polja za unos

U ovom primjeru napravljena je forma koja se sastoji od sve tri prethodne konzole: dvije oznake sadržaja *Ime* i *Prezime*, dva tekstualna polja za unos imena i prezimena, te gumb za ispis unesenih podataka. Postupak se sastoji od dovlačenja kontrola na željeno mjesto te programiranja gumba za jednostavnu funkciju ispisa:

```
private void b1_Click(object sender, EventArgs e)
{
      MessageBox.Show(textbox1.Text + " " + textbox2.Text,"Ime i prezime");
}
```



Sl. 4.8. Ispis podataka iz tekstualnog polja

[14] http://www.microsoftsrb.rs/download/obrazovanje/pil/Mala_skola_programiranja_Csharp.pdf

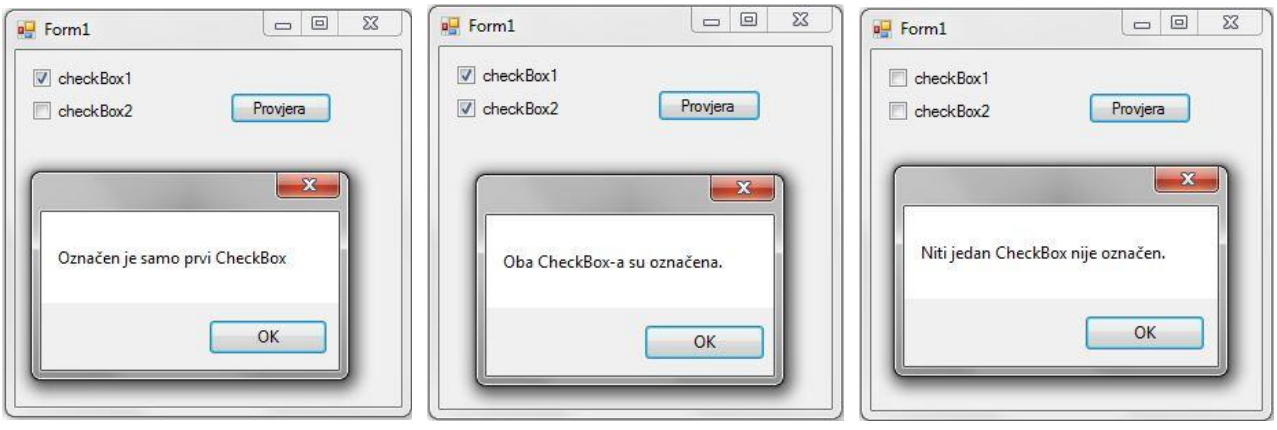
Okvir za izbor

Okvir za izbor (eng. „checkbox“) je kontrola koja omogućava korisniku da označi (ili ne označi) neku ponuđenu opciju. Ako na formi postoji više od jednog okvira za izbor, svi oni su međusobno nezavisni, tj. mogu se proizvoljno označavati. Osnovno svojstvo ove kontrole jest *označen* (eng. „*checked*“) koje može imati dvije vrijednosti: označeno (*true*) ili neoznačeno (*false*). Kod kodiranja većinom se koristi *if* uvjet kako bi se ispitalo da li je okvir za izbor označen ili ne i ovisno o tome se vrši daljnja izvedba koda[15].

Primjer 4.2. Ispitivanje vrijednosti okvira za izbor

U primjeru postavljena su dva nezavisna okvira za izbor te gumb Provjera koji nam klikom na njega govori koji okvir za izbor je označen. Kod za provjeru je:

```
private void button1_Click(object sender, EventArgs e)
{
    if (checkBox1.Checked == true && checkBox2.Checked == true)
        MessageBox.Show("Oba CheckBox-a su označena.");
    if (checkBox1.Checked == false && checkBox2.Checked == false)
        MessageBox.Show("Niti jedan CheckBox nije označen.");
    if (checkBox1.Checked == true && checkBox2.Checked == false)
        MessageBox.Show("Označen je samo prvi CheckBox");
    if (checkBox1.Checked == false && checkBox2.Checked == true)
        MessageBox.Show("Označen je samo prvi CheckBox"); }
```



Sl. 4.9. Provjera označenosti okvira za izbor

[15] http://www.microsoftsrb.rs/download/obrazovanje/pil/Mala_skola_programiranja_Csharp.pdf

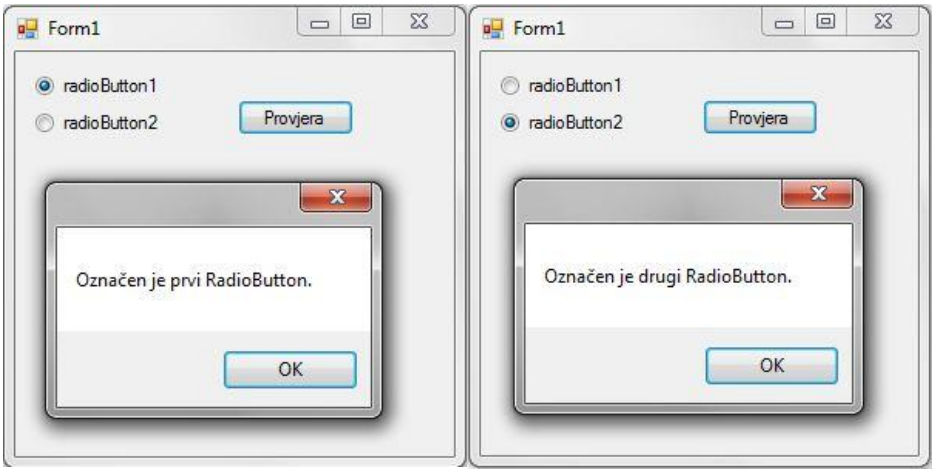
Opcijski gumbi

Opcijski gumb (eng. „*radio button*“) omogućuje korisniku da odabere samo jednu opciju iz skupine. Kada korisnik klikne na jednu opciju ona postane označena, a sve ostale opcije su automatski neoznačene. Pri pokretanju forme uvijek je označena prva opcija zato što opcijski gumb ne daje mogućnost ostavljanja svih opcija neoznačenih. Opcije mogu predstavljati tekst, sliku ili kombinaciju oboje. Osnovno svojstvo ove kontrole je *označen* (eng. „*checked*“) i ima jednaku funkciju kao kod okvira za izbor[16].

Primjer 4.3. Ispitivanje vrijednosti opcijskih gumba

Kod za provjeru vrlo je jednostavniji zbog ograničenog broja označenih opcija:

```
private void button1_Click(object sender, EventArgs e)
{
      if (radioButton1.Checked == true) MessageBox.Show("Označen je prvi
      RadioButton.");
      else MessageBox.Show("Označen je drugi RadioButton.");

}
```



Sl. 4.10. Provjera označenosti opcijskih gumba

[16] http://csharp.net-informations.com/gui/cs-radiobutton.htm

4.4. Napredne mogućnosti Windows formi

Kreiranje izbornika

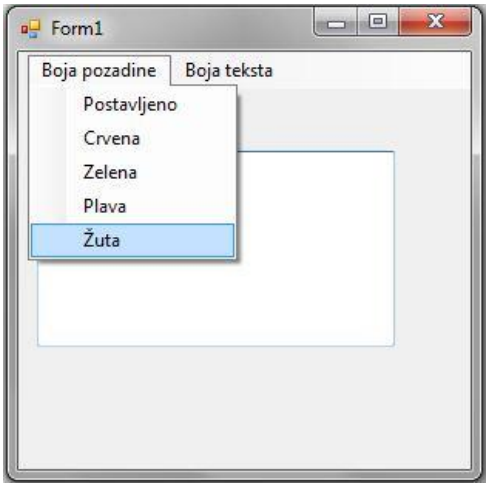
Gotovo svaka Windows aplikacija ima izbornik (eng. „*menu*“) kao jedan od osnovnih načina interakcije korisnika i programa. Većinom se nalazi na vrhu forme, iako ga je moguće staviti i na dno ili po vertikali. Sastoji se od dva dijela: horizontalni dijelovi predstavljaju skupine prema kojima se grupiraju ostale stavke izbornika, dok se vertikalni dijelovi prikazuju kada korisnik klikne na neki horizontalni dio. Nadalje, svaki vertikalni dio može imati još svoje dijelove. Takva hijerarhijska organizacija i malo zauzeće memorije su glavne prednosti korištenja izbornika u aplikacijama[17].

Izbornici također spadaju pod kontrole te ih stoga pronalazimo u prozoru *Alati* pod nazivom *MenuStrip*. Nakon što ga se dovuče u formu, program ga automatski pozicionira na vrh forme. Izbornik na vrhu forme se prikazuje pri samom umetanju u formu ili nakon klika na kontrolu *menuStrip1* koja se nalazi ispod forme. Na područja u izborniku na kojima piše „*Type here*“ unose se naslovi željenih stavki izbornika. Automatski kada se počne pisati ime jedne stavke ispod i desno od nje se otvaraju mogućnosti za unos novih horizontalnih i vertikalnih dijelova.

Svakom od dijelova izbornika moguće je dodati i skraćenicu (eng. „*hot key*“) kojom pomoću tipkovnice pristupamo dijelu kojeg predstavlja (npr. ako za stavku naziva *Otvori* želimo da skraćenica bude *Alt+O*, jednostavno u postavkama pod njezinim imenom upišemo *&Datoteka*). Još jedna karakteristična stvar za menije je horizontalni separator koji nastaje ako se umjesto imena stavke unese samo minus.

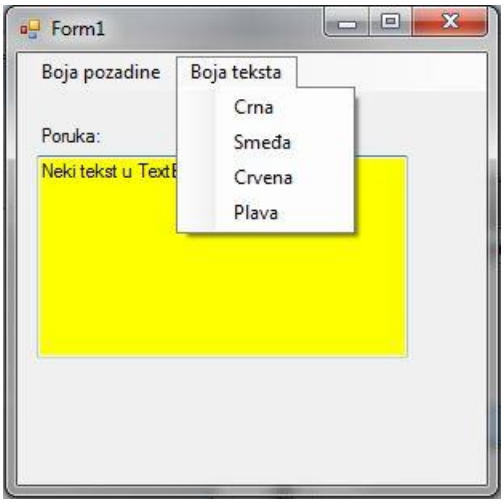
[17] http://www.microsoftsrb.rs/download/obrazovanje/pil/Mala_skola_programiranja_Csharp.pdf

Primjer 4.4. Kreiranje i programiranje jednostavnog izbornika

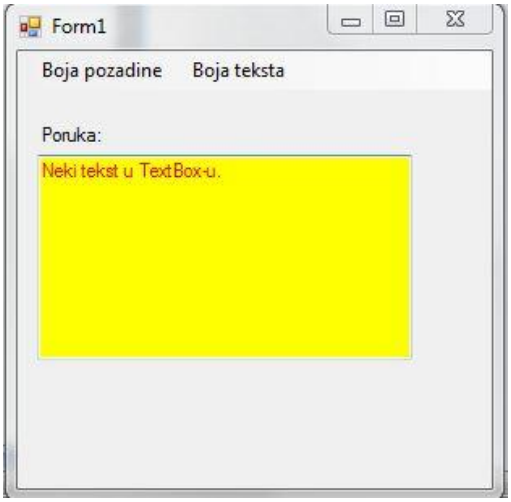


Sl. 4.11. Izgled stavke "*Boja pozadine*"

Sl. 4.12. Promjena pozadine u žuto



Sl. 4.13. Izgled stavke "*Boja teksta*"



Sl. 4.14. Promjena boje teksta u crveno

U primjeru vidimo najjednostavniji izbornik pomoću kojega se može mijenjati stil tekstualnog polja (njegovu boju pozadine i boju fonta). Programiranje svake stavke izvodi se dvoklikom na određenu stavku nakon čega nas program odvede u dio koda predviđen za određenu stavku. Kod koji se koristi za postavljanje boje pozadine:

```
private void crvenaToolStripMenuItem_Click(object sender, EventArgs e)
{
      textBox1.BackColor = System.Drawing.Color.Red;
}
```

Za postavljanje boje teksta tekstualnog polja korišten je kod:

```
private void crnaToolStripMenuItem_Click(object sender, EventArgs e)
{
      textBox1.ForeColor = System.Drawing.Color.Black;
}
```

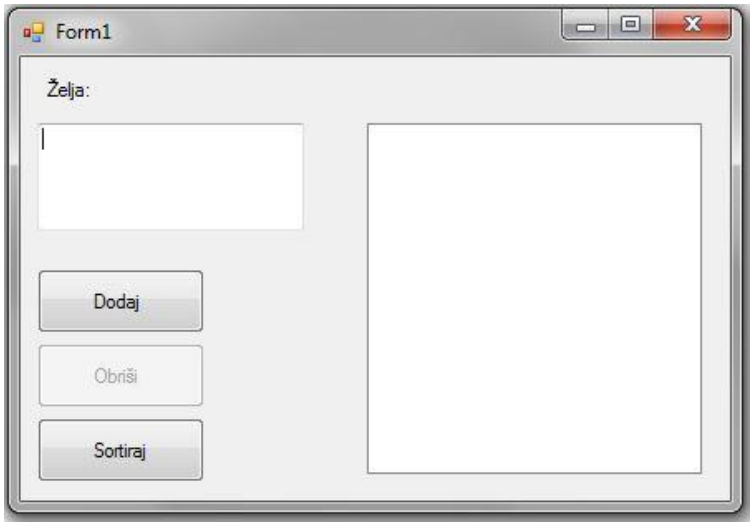
Kodovi za svaku boju razlikuju se jedino u nazivu boje. Boju teksta i pozadine moguće je promijeniti i u samim postavkama tekstualnog polja, no na taj način samo je moguće postaviti početnu boju. Ukoliko je boju ili bilo koju postavku potrebno promijeniti tijekom izvođenja aplikacije, to se mora učiniti preko samog koda.

Liste

Liste (eng „*list*“) su komade koje se koriste za pravljenje kolekcije sa mogućnošću dodavanja, brisanja i sortiranja elemenata unutar nje. Mogu se pronaći unutar prozora *Alati* pod nazivom *ListBox*. Imenik potreban za rad s listama je *System.Collection.Generics*. Ukoliko imenik nije dodan pri kreiranju aplikacije od stane programa, potrebno je to učiniti ručno.

Primjer 4.5. Izrada liste – dodavanje, brisanje i sortiranje elemenata liste

Unutar ove aplikacije biti će prikazano kreiranje tzv.“*bucket*“ liste iliti lista želja sa svim njezinim pripadajućim elementima. Prvi korak je kreirati novu formu i ako je potrebno uključiti potreban imenik. Zatim se kreira izgled forme (polje za unos podatka, polje za listu te gumbi za dodavanje, brisanje i sortiranje). Za kreiranje liste koristi se naredba `List <string> ime_liste = new List <string>();`. Također, potrebno je onemogućiti gumb za brisanje, pošto je na samom početku lista prazna – naredba: `button2.Enabled = true;`. Po završetku tog koraka aplikacija će izgledati približno kao na slici 4.4.5.



Sl. 4.15. Izgled aplikacije „*bucket list*“

Slijedeći korak je programiranje gumba za dodavanje novih stavki na listu, koje je dano kodom:

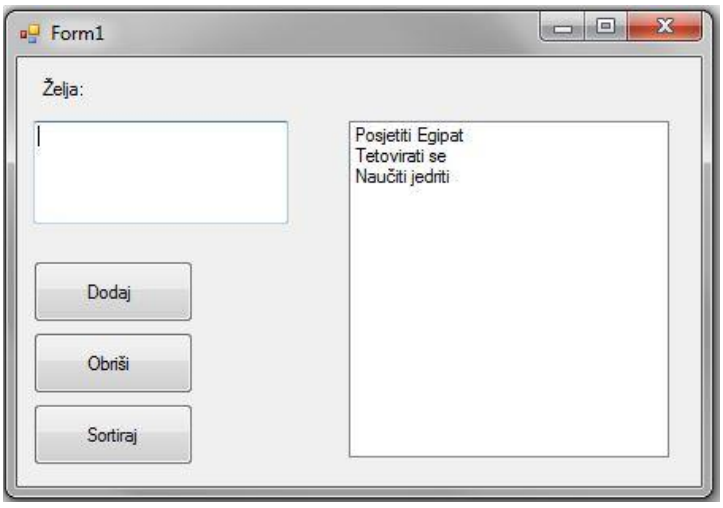
```
private void button1_Click(object sender, EventArgs e)
{
   bool prazno = false;
   if (textBox1.Text == "") // testiranje da li je unešena želja
   {
         MessageBox.Show("Niste unijeli nijedan podatak", "Pogreška");
         prazno = true;
   }

   if (!prazno) //ako lista nije prazna
   {
         popis.Add(textBox1.Text); // dodavanje nove stavke na listu
         listBox1.Items.Add(textBox1.Text); // prikaz nove stavke na listi
         button2.Enabled = true; // omogućavanje brisanja jer lista nije
                                 // prazna
         textBox1.Text = ""; // postavljanje praznog polja za unos
   }
```

Nakon što je gumb za brisanje omogućen potrebno je i njega kodirati. Cilj je da se mišem označeni element obriše pritiskom na taj gumb. To je ostvarenom metodom za brisanje *RemoveAt()* koja prima indeks elementa kojeg se želi obrisati. Indeks se dohvaća pomoću svojstva *Select Index*.

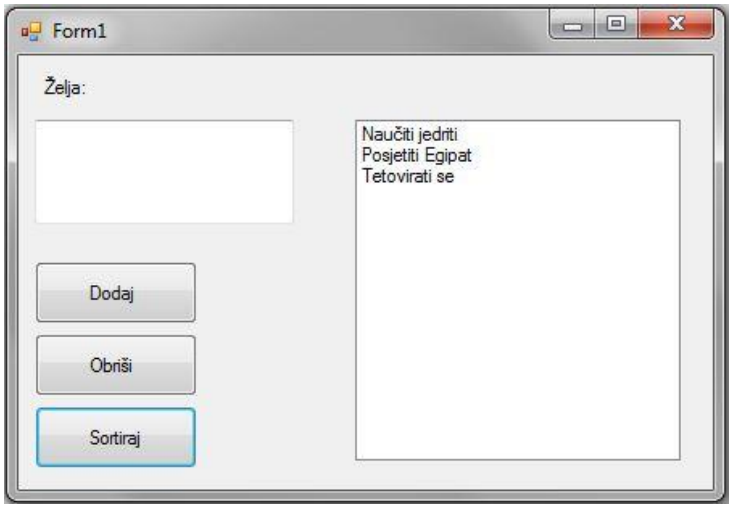
```
private void button2_Click(object sender, EventArgs e)
{
   popis.RemoveAt(listBox1.SelectedIndex); // brisanje odabrane stavke preko
                                           // indeksa
   listBox1.Items.Clear(); // brisanje svih stavki sa priazanog popisa
   foreach (string i in popis)
   {
         listBox1.Items.Add(i); // dodavanje svih preostalih stavki nazad na
                                // prikaz liste
         }
   if (listBox1.Items.Count == 0)
   {
   button2.Enabled = false; // ako su obrisane sve stavke sa liste onemogući
                            // gumb
   }

 }
```

Jedino što je još preostalo je gumb za sortiranje koji se veoma jednostavno programira pomoću ugrađene metode *Sort()* koja sortira zadane stavke abecednim redoslijedom.

```
private void button3_Click(object sender, EventArgs e)
{
   popis.Sort(); // pozivanje metode Sort()
   listBox1.Items.Clear(); // brisanje prikaza liste
   foreach (string i in popis) // dodavanje stavki na prikaz liste
   {                           // sortitanim redoslijedom
         listBox1.Items.Add(i);
   }
}
```



Sl. 4.16. Dodavanje stavki liste



Sl. 4.17. Sortiranje liste

Višestruke forme

Svi primjeri do sada rađeni preko jedne samostalne poruke. Kod većih i kompleksnijih aplikacija postojanje samo jedne forme nije dovoljno za njezinu realizaciju. Zbog toga uveden je pojam višestrukih formi gdje se unutar jedne aplikacije međusobno povezuje više formi.

Primjer 4.6. Aplikacija sa tri povezane forme

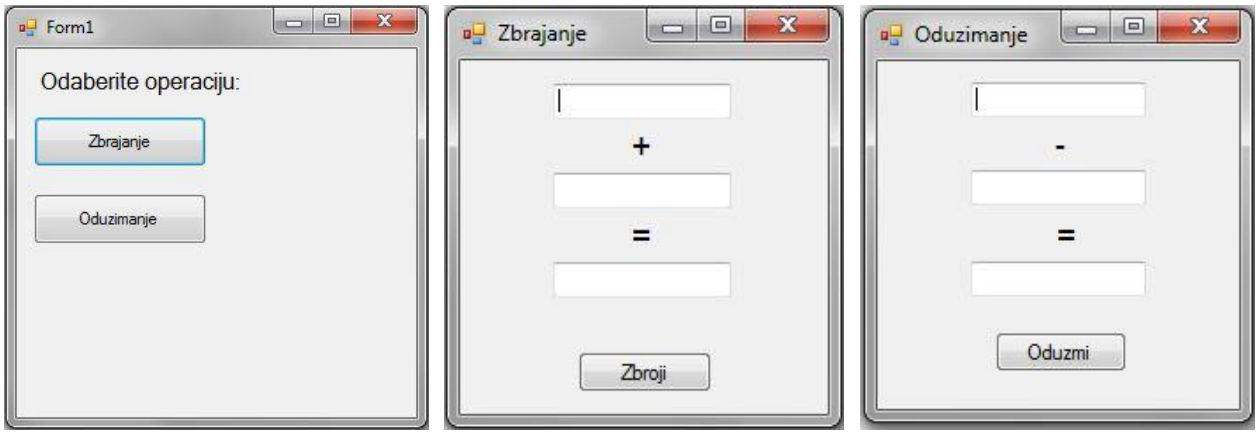
U primjeru prikazati će se izrada jednostavne aplikacije koja će se sastojati od jedne glavne forme na kojoj će korisnik moći odabrati jednu od dvije operacije: zbrajanje ili oduzimanje. Pritiskom na odgovarajući gumb, aplikacija ga odvodi na novu formu na kojoj se odvija odabrana operacija. Prvo je potrebno kreirati novi projekt i unutar njega još dvije dodatne forme. Nova forma dodaje se tako da se desnim gumbom miša klikne na ime projekta u prozoru *Solution Explorer* i odabere *Add → Windows form*.

Zatim se na početnu formu dodaju dva gumba: zbrajanje i oduzimanje. Pritiskom na određeni gumb treba se otvoriti odgovarajuća forma.

```
Form2 a = new Form2();// postavljanje forme a kao nove forme (Form2)
Form3 b = new Form3();// postavljanje forme b kao nove forme (Form3)
private void button1_Click(object sender, EventArgs e)
{
   a.Show();// prikazivanje forme a ako se pritisne button1
}
private void button2_Click(object sender, EventArgs e)
{
   b.Show();// prikazivanje forme b ako se pritisne button2

}
```

Jedino što je još potrebno je kreirati sučelje i kod za obje forme. Izgled formi prikazan je na kraju poglavlja, a kod za zbrajanje dan je ispod:

```
private void zbroji_Click(object sender, EventArgs e)
{
   int a, b, c; // deklaracija 3 broja (2 za zbrajanje i 1 za rezultat)
   a = Convert.ToInt32(textBox1.Text); //pretvaranja sadržaja TextBox-a koji
                                       je string
   b = Convert.ToInt32(textBox2.Text); // i dodjeljivanje vrijednosti u a i b
   c = a + b;
   textBox3.Text = Convert.ToString(c); // ispis rezultata u zadnji TextBox
}     // mora se konvertirati nazad u string da bi se moglo prikazati
```



Sl. 4.18. Izgled osnovne forme i formi za zbrajanje i oduzimanje

5. PROJEKT VIDEOTEKA

Kao praktični dio rada unutar kojeg je moguće vidjeti pravu primjernu formi izrađena je aplikacija za vođenje evidencije posudbe filmova jedne videoteke. Aplikacija treba omogućavati korisniku unos novih članova te ispis i brisanje postojećih članova. Potrebni podatci o korisniku su: njegovo ime i prezime, šifra člana, adresa i njegov broj telefona. Također, potrebno je omogućiti evidenciju posudbe filmova na način da je po članu maksimalno dozvoljeno posuditi četiri filma i za pojedinog člana prikazati filmove koje je posudio. Potrebni podatci o filmu su: ime filma, ime člana koji je posudio film (ukoliko je film posuđen) te datum posudbe.

5.1. Struktura aplikacije

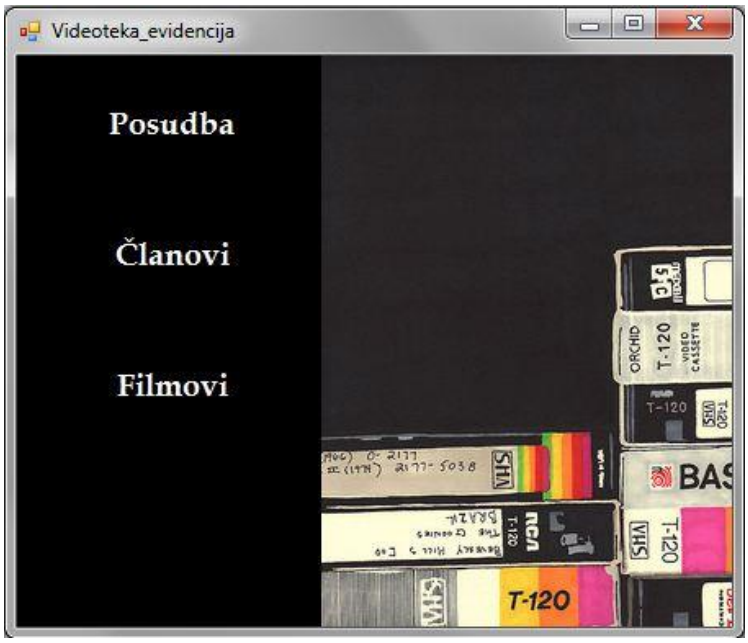
Aplikacija se sastoji od glavne forme Form1.cs i sedam sporednih formi koje su povezane sa glavnom. Sporednim formama pristupa se preko izbornika u glavnoj formi. Izbornik se sastoji od tri osnovna dijela:

- posudba – sadrži stavke za posuđivanje i vraćanje filmova
- članovi – sadrži stavke za dodavanje i brisanje članova kao i za njihov prikaz
- filmovi – sadrži stavke za dodavanje novog filma i prikaz postojećih filmova

Pritiskom na jednu od stavki izbornika otvara se odgovarajuća nova forma. Kreiranje svake forme obavlja se na jednak način: unutar prozora *Solution Explorer* gdje je prikazan sadržaj trenutno otvorenog projekta potrebno je desnim klikom kliknuti na ime projekta → *Add* →*Windows form*. U donjem dijelu novootvorenog prozora potrebno je upisat željeno ime forme i pritisnuti *OK*. Povezivanje svih formi obavlja se unutar koda glavne forme. Prvo je potrebno deklarirati nove forme te zatim definirati kada će se pojedina forma definirati i otvoriti. Pošto je za pristup ostalim formama korišten izbornik, dvoklikom na odgovarajuću stavku unutar njega otvara se generirani kod unutar kojeg se definira koja će se forma otvoriti.

```
noviClan noviC; // deklaracija novih formi
noviFilm noviF;
obrisiClana obrisiC;
```

```
private void dodajČlanaToolStripMenuItem1_Click(object sender, EventArgs e)
{
    noviC = new noviClan(); // definicija nove forme
    noviC.Show(); // pritiskom na stavku dodaj Člana u meniju otvara se forma
                  // s identifikatorom "novi"
}
```



Sl. 5.1. Izgled glavne frome „*Form1.cs*“

Podatci o svim članovima, filmovima i samoj posudbi pohranjuju se u tablice unutar baze podataka VideotekaDB.mdf. To je lokalna baza kreirana putem Microsoft SQL Express Servera i smještena je unutar same aplikacije. Postupak za kreiranje baze je slijedeći: unutar prozora *Server Explorer* desnim klikom miša ode se na *Data Connections* → *Add connection*. Ukoliko taj prozor nije vidljiv na početku moguće ga je uključiti u izborniku *View* → *Other Windows*. U prozoru koji se zatim otvori odabire se server, upisuje ime baze koju se želi kreirati i klikne se *OK*. Server zatim kreira bazu unutar koje je moguće dodavati tablice, funkcije, procedure i ostale elemene koje svaka baza podataka podržava. Baza za ovu aplikaciju sastoji se od tri tablice: *clanovi*, *filmovi* i *posudba* koje su potrebne za njezinu realizaciju (sl.5.1.2., sl.5.1.3. i sl.5.1.4.).

	Name	Data Type	Allow Nulls	Default
🔑	sifraFilma	int	☐	
	ime	varchar(100)	☐	
	posudjen	char(2)	☐	('ne')

Sl. 5.2. Struktura tablice "*filmovi*"

	Name	Data Type	Allow Nulls	Default
🔑	sifraClana	int	☐	
	ime	varchar(50)	☐	
	prezime	varchar(50)	☐	
	adresa	varchar(100)	☑	
	brojTelefona	varchar(20)	☑	
	brojPosudjenihFilmova	int	☐	((0))

Sl. 5.3. Struktura tablice "*clanovi*"

	Name	Data Type	Allow Nulls	Default
	sifraClana	int	☐	
	sifraFilma	int	☐	
	datumPosudbe	datetime	☐	
	datumVracanja	datetime	☑	
🔑	idPosudbe	int	☐	

Sl. 5.4. Struktura tablice "*posudba*"

5.2. Programiranje aplikacije

Spajanje na bazu podataka

Kako bi se omogućilo manipuliranje podatcima unutar baze potrebno je povezati samu aplikaciju i bazu. Svi objekti za spajanje sa SQL bazom nalaze unutar biblioteke *System.Data.SqlClient* koju je potrebno uključiti zasebno unutar svake forme koja obavlja neku operaciju nad podatcima u bazi. Za kreiranje konekcije koristi se objekt *SqlConnection* koji predstavlja vezu sa izvorom podataka. Sintaksa za njegovo definiranje je: S*qlConnection ime_indentifikatora = new SqlConnection (Connection_string)*. *String* za konekciju sa bazom (eng. „*Connection string*") generira sam program

pri kreiranju baze, no moguće ga je izmijeniti za vlastite potrebe. Potrebno je samo odabrati bazu podataka na koju se želi spojiti i u prozoru *Svojstva* nalazi se već definirani *string* za konekciju kojeg se samo kopira unutar koda.

U aplikaciji je korišten izvorni *string* pri konekciji sa bazom, tj. korištena je apsolutna putanja do baze podataka na računalu, a aplikacija je spremljena unutar C direktorija kako bi se olakšalo njezino pokretanje na nekom drugom računalu.

Drugi objekt potreban za manipulaciju podatcima je *SqlCommand* koji omogućuje slanje naredbi (npr. SQL naredbi ili spremljenih procedura) prema bazi podataka.

```
SqlConnection conn = new SqlConnection(@"Data    Source=(LocalDB)\v11.0;
AttachDbFilename=c:\Zavrsni rad Videoteka\Zavrsni rad Videoteka\
videotekaDB.mdf;Integrated Security=True"); // inicijalizacija identifikatora
                                            // konekcije

SqlCommand naredba = new SqlCommand("DELETE FROM clanovi WHERE sifraClana =
1;", conn); // primjer SqlCommand naredbe
```

Pošto se u cijeloj aplikaciji spaja uvijek na istu bazu podataka konekcija je inicijalizirana u tijelu forme. Nasuprot tomu, *SqlCommand* može sadržavati različite naredbe i pozivati se više puta tijekom izvođenja programa pa je on inicijaliziran tek pri neposredno prije izvršavanja naredbe (moguća je i inicijalizacija na samom početku na neku početnu vrijednost – naredbu, no način zapisivanja ovisi o samom programeru).

Dodavanje i brisanje podataka

Forma "*noviClan.cs*" služi za dodavanje novih članova. Sastoji se od tekstualnih polja u koja korisnik treba unijeti podatke i njima pripadajućih oznaka koje govore koji podatak korisnik mora gdje unijeti. Šifru člana, koja je ujedno i njegov identifikator nije potrebno unijeti zato što je prilikom kreiranja same tablice "clanovi" postavljeno da je to polje cijeli broj koji se automatski povećava za jedan prilikom svakog novog unosa. Nakon što su svi podatci unešeni pritiskom na gumb spremi izvršava se dodavanje novog člana i njegovih podataka u bazu pomoću SQL naredbe *Insert*.

Sl. 5.5. Izgled forme "*noviClan.cs*"

```
private void button1_Click(object sender, EventArgs e)
{
   if (textBox1.Text == "" || textBox2.Text == "" || textBox3.Text == "" ||
   textBox4.Text == "")
   {  // provjera da li je neko od polja ostalo neispunjeno
      MessageBox.Show("Nista unijeli sve podatke!");
   }
   else
   {
      conn.Open();
      SqlCommand naredba = new SqlCommand("INSERT INTO clanovi (ime, prezime,
      adresa, brojTelefona) VALUES('" + textBox1.Text + "','" + textBox2.Text
      + "','" + textBox3.Text + "','" + textBox4.Text + "');", conn);
      if (naredba.ExecuteNonQuery() > 0) // provjera da li je izvršena INSERT
                                         // naredba
      {
         MessageBox.Show("Uspješno uneseno");
      }
      conn.Close();
      textBox1.Text = ""; // pražnjenje TextBox-a
      textBox2.Text = "";
      textBox3.Text = "";
      textBox4.Text = "";
      }
}
```

Za dodavanje novoga filma postupak i kod su jednaki – jedina razlika jest naravno u podatcima koje korisnik upisuje. Potrebno je upisati samo naziv filma, dok se identifikator filma automatski unosi na jednakom principu kao i kod članova, a polje "*posudjen*" se pri unosu novog filma postavlja na početnu vrijednost "*ne*".

Za brisanje člana iz baze koristi se SQL naredba *Delete*. Prvo je potrebno odabrati člana kojeg se želi obrisati iz padajućeg izbornika, a zatim kliknuti gumb *OK*. Pritiskom na gumb *OK* uspostavlja

se konekcija prema bazi. Unutar baze traži se identifikator iliti šifra odabranog člana i nakon toga izvršava se njegovo brisanje iz tablice. Ukoliko nije došlo do nekakve pogreške aplikacija izbacuje poruku da je član obrisan i vraća se na početno stanje.

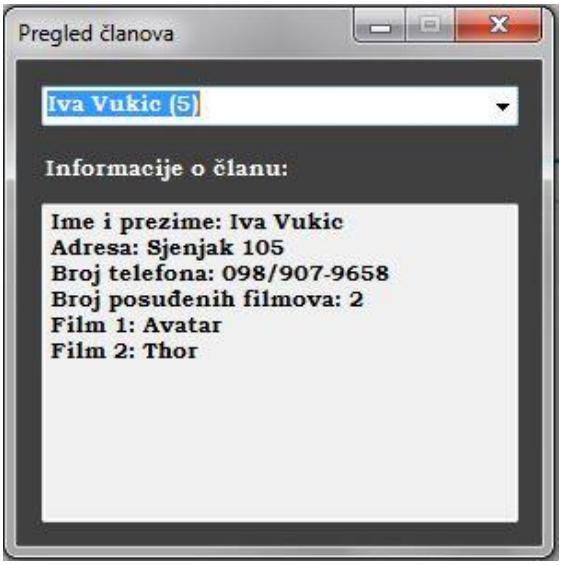
```
private void button1_Click(object sender, EventArgs e)
{
    conn.Open();
    string selectedClan = this.comboBoxClanovi.GetItemText
    (this.comboBoxClanovi.SelectedItem); // dohvaćanje označenog polja u
                                         // padajućem izborniku
    int sifraClana = 0;
    string[] words = selectedClan.Split(' ');
    SqlCommand getSifraClana = new SqlCommand("SELECT clanovi.sifraClana,
    clanovi.ime, clanovi.prezime FROM clanovi WHERE clanovi.ime = '" +
    words[0] + "' AND clanovi.prezime = '" + words[1] + "';", conn);
    try
    {
        SqlDataReader reader = getSifraClana.ExecuteReader();
        while (reader.Read())
        {
            sifraClana = Convert.ToInt32(reader["sifraClana"]);
            // dohvaćanje šifre odabranog člana
        }
        reader.Close();
        reader.Dispose();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
        SqlCommand naredba = new SqlCommand("DELETE FROM clanovi WHERE
        sifraClana=" + sifraClana + ";", conn);
        // brisanje člana
        naredba.ExecuteNonQuery();
        if (naredba.ExecuteNonQuery() == 0)
        {
            MessageBox.Show("Član obrisan");
        }
        conn.Close();
        this.comboBoxClanovi.Items.Clear(); // brisanje odabira padajućeg
                                            // izbornika
        this.comboBoxClanovi.SelectedIndex = -1;
        this.comboBoxClanovi.Text = "";
        selectClanova(); // pozivanje funkcije za ispis postojećih članova u
                         // padajući izbornik
}
```

Prikaz filmova i članova

Za prikaz podataka iz baze preko tablice koristi se kontrolu *DataGridView* u kombinaciji sa klasom *SqlDataAdapter* koja predstavlja kombinaciju SQL upita i konekcije na bazu, a služi kao posrednik za izvršavanje SQL naredbe i dohvaćanje podataka iz baze. Nakon što se podatci dohvate, potrebno ih je negdje pohraniti kako bi se mogli prikazati. Jedan od mogućih načina, koji je korišten i u ovoj aplikaciji je instanca klase *DataTable*. Potrebno je spremiti dohvaćene podatke u *DataTable* pomoću metode *Fill()* i postaviti ga kao izvor podataka za prikaz u *DataGridView*-u.

```
conn.Open();
SqlDataAdapter adapter = new SqlDataAdapter();
adapter.SelectCommand = new SqlCommand("SELECT filmovi.ime AS 'Film',
filmovi.posudjen as 'Posuđen', 'Posudio' = CASE WHEN filmovi.posudjen = 'ne'
THEN ' ' ELSE CONCAT (clanovi.ime, ' ', clanovi.prezime) END FROM filmovi
LEFT JOIN (SELECT * from posudba where posudba.datumVracanja IS NULL) as x
ON (filmovi.sifraFilma=x.sifraFilma) LEFT JOIN clanovi ON x.sifraClana =
clanovi.sifraClana ", conn);
DataTable dt = new DataTable();
adapter.Fill(dt);
dgvFilmovi.DataSource = dt;
conn.Close();
```

Za prikaz određenog člana i njegovih podataka korišten je ponovno padajući izbornik pomoću kojeg se odabire željeni član. Nakon što je član odabran u teksualnom polju ispod izbornika prikazuju se svi njegovi podatci unutar tablice "*clanovi*" kao i popis trenutno posuđenih filmova.



Sl. 5.6. Prikaz podataka o članu

Sl. 5.7. Prikaz podataka o filmovima

Posuđivanje i vraćanje filmova

Forme za posuđivanje i vraćanje filmova sastoje se od dva padajuća izbornika za odabir člana koji posuđuje/vraća film i odabir filma koji se posuđuje/vraća te od gumba za pokretanje potrebnih akcija. Pri učitavanju formi prikazuju se samo oni članovi koji su u mogućnosti posuditi ili vratiti film. Forma za posuđivanje prikazuje samo one članove koji imaju manje od četiri posuđena filma, a forma za vraćanje samo one koji imaju posuđen barem jedan film. Također, izbornik za filmove je onemogućen sve dok se ne odabere željeni član. Nakon šta je on odabran uključuje se izbornik za filmove koji sadrži samo one filmove koji nisu trenutno posuđeni ako je riječ o posudbi filmova ili one filmove koje taj član ima trenutno posuđene ako je riječ o vraćanju filma. Takva ograničenja služe kako bi se spriječile moguće pogreške unutar aplikacije.

Prilikom posuđivanja filma nakon pritiska gumba izvršavaju se SQL naredbe za dohvaćanje šifre odabranog člana i filma i spremaju se u neke varijable kako bi se olakšalo provođenje daljnjih naredbi. Izmjene do kojih dolazi u bazi podataka kod posudbe su: promjena vrijednosti "*posuđen*" u tablici filmova iz "*ne*" u "*da*", povećava se broj posuđenih filmova u tablici "*clanovi*" i dodaje se novi zapis o posudbi u tablici "posudba" sa trenutnim datumom i vremenom posudbe.

Kod vraćanja filma postupak obratan: vrijednost "*posuđen*" vraća se na "*ne*", smanjuje se broj posuđenih filmova za jedan kod člana koji vraća film, a zapis o posudbi se ažurira na način da se u polje "*datumVracanja*" koje je do tada imalo null vrijednost unosi trenutni datum i vrijeme.

```
SqlCommand naredba = new SqlCommand("SELECT brojPosudjenihFilmova FROM
   clanovi WHERE sifraClana = " + sifraClana + ";", conn);
   int broj = (int)naredba.ExecuteScalar();
   // unošenje novog zapisa u tablicu "posudba"
   SqlCommand naredba3 = new SqlCommand("INSERT INTO posudba
   (sifraClana,sifraFilma,datumPosudbe) VALUES (" + sifraClana + "," +
   sifraFilma + ",current_timestamp);", conn);
   // promjena vrijednosti "posuđen" iz "da" u "ne"
   SqlCommand naredba4 = new SqlCommand("UPDATE filmovi SET posudjen = 'da'
   WHERE sifraFilma = " + sifraFilma + ";", conn);
   // povećanje broja posuđenih filmova za 1
   SqlCommand naredba5 = new SqlCommand("UPDATE clanovi SET
   brojPosudjenihFilmova = " + (broj + 1) + " WHERE sifraClana = " + sifraClana
   + ";", conn);
   if (naredba3.ExecuteNonQuery() == 1 && naredba4.ExecuteNonQuery() == 1 &&
   naredba5.ExecuteNonQuery() == 1)
   {
      MessageBox.Show("Film je posuđen!");
   }
```

6. ZAKLJUČAK

Cilj ovog rada bio je upoznavanje s konceptom Windows formi i načinima realizacije i implementacije njezinih osnovnih komponenata. Polazilo se od pretpostavke da čitatelj poznaje samo teorijski pojam programiranja i temeljem toga teorija se razrađivala kroz same osnove C#-a do nastanka Windows formi i na kraju njezinih naprednijih mogućnosti. Rezultat koji se želio postići bio je isključivo dodjeljivanje osnovnih znanja o Windows formama čitatelju i postavljanje temelja za njegov daljnji razvoj.

C# u kombinaciji sa formama kao dio .NET tehnologije predstavlja moćno razvojno okruženje koje omogućava konstruiranje skoro svih aplikacija koje se traže na tržištu. No, iako je nužno, razumijevanje osnova programerskih ideja i principa rada predstavlja samo početni korak prema razvoju kvalitetnih i funkcionalnih aplikacija.

LITERATURA

[1] R.Miles: C# Programming, Department of Computer Science, University of Hull, 2012.

[2] M.Michaelis, E.Lippert: Essential C# 5.0, Pearson Education Inc., Michigan, 2012.

Linkovi:

[3] http://www.efos.unios.hr/arhiva/dokumenti/RPA_P2_Kreiranje.pdf (18.08.2015.)

[4] http://www.scribd.com/doc/220657903/Programsko-inzenjerstvo#scribd (18.08.2015.)

[5]https://www.fer.unizg.hr/_download/repository/11._Programski_jezici,_otvorenost.pdf (22.08.2015.)

[6]http://ss-tehnicka-ck.skole.hr/upload/ss-tehnicka-ck/newsattach/130/CGrananjaPetlje.pdf (22.08.2015.)

[7] https://loomen.carnet.hr/pluginfile.php/281593/mod_resource/content/1/Predavanja/10._Enkapsulacija.pdf (22.08.2015)

[8]https://loomen.carnet.hr/pluginfile.php/373293/mod_resource/content/1/LV13%20-%20Rad%20s20Windows%20formama.pdf (22.08.2015)

[9] http://www.efos.unios.hr/arhiva/dokumenti/RPA_P7_Forme_izbornici.pdf (22.08.2015.)

[10]https://books.google.hr/books?id=RlM8tSROaKsC&pg=PA129&lpg=PA129&dq=c%23%20Accessibility%20Appearance%20Behavior%20Data%20Design%20Focus%20Layout%20Misc%20Window%20Style&source=bl&ots=_lyzwznKcr&sig=CcRwGGqE74ehxN0lC0hlEjAtnZ0&hl=hr&sa=X&ved=0ahUKEwis6ZndjJvPAhWI_ywKHdVGCrYQ6AEIKzAD#v=onepage&q=c%23%20Accessibility%20Appearance%20Behavior%20Data%20Design%20Focus%20Layout%20Misc%20Window%20Style&f=false (10.09.2015.)

[11]http://carpediem.hr/PublikacijeCarpeDiem/Publikacije/C%23%20programiranje.pdf (22.08.2015.)

[12]http://www.microsoftsrb.rs/download/obrazovanje/pil/Mala_skola_programiranja_Csharp.pdf (10.09.2015.)

[13] http://csharp.net-informations.com/gui/cs-radiobutton.htm (10.09.2015.)

POPIS TABLICA I ILUSTRACIJA

Tablice:

3.1. Numerički tipovi podataka za prikaz cijelih brojeva 7

3.2. Numerički tipovi podataka za prikaz realnih brojeva 7

3.3. Osnovni operatori u C# 9

Slike:

3.1. Kreiranje projekta 17

3.2. Izgled početnog programa 18

3.3. Ispis funkcije Console.Write pa Console.WriteLine 19

3.4. Ispis funkcije Console.WriteLine pa Console.Write 19

4.1. Razvojni okoliš forme 22

4.2. Način prikaza svojstava forme 24

4.3. Uspješno završena operacija Build 24

4.4. Prikaz grešaka prilikom izvođenja operacije Build 25

4.5. Izgled poruke 25

4.6. Prikaz mogućih događaja i njihovo mjesto unutar programa 26

4.7. Postavljanje sadržaja oznake 28

4.8. Ispis podataka iz tekstualnog polja 29

4.9. Provjera označenosti okvira za izbor 30

4.10. Provjera označenosti opcijskih gumba 31

4.11. Izgled stavke "*Boja pozadine*" 33

4.12. Promjena pozadine u žuto 33

4.13. Izgled stavke "*Boja teksta*" 33

4.14. Promjena boje teksta u crveno 33

4.15. Izgled aplikacije „*bucket list*“ 35

4.16. Dodavanje stavki liste 36

4.17. Sortiranje liste 37

4.18. Izgled osnovne forme i formi za zbrajanje i oduzimanje 38

5.1. Izgled glavne forme „*Form1.cs*“ 40

5.2. Struktura tablice “*filmovi*” 41

5.3. Struktura tablice “*clanovi*” 41

5.4. Struktura tablice “*posudba*” 41

5.5. Izgled forme “*noviClan.cs*” 43

5.6. Prikaz podataka o članu 45

5.7. Prikaz podataka o filmovima 45

SAŽETAK

NASLOV: Windows forme u C#-u

Windows forme predstavljaju osnovni alat pri kreiranju desktop aplikacija za Windows operacijski sustav. Svaka aplikacija sastoji se od jedne ili više međusobno povezanih formi. Forma predstavlja prozor koji sadrži elemente za prikaz, unos i manipulaciju podatcima koji se nazivaju kontrole. Rad svake kontrole programira se putem događaja (engl. "events") pomoću C# programskog jezika.

Unutar rada detaljno su objašnjene osnove karakteristike jezika C#, kreiranje i primjena najčešćih kontrola u Windows formama. Također, cijeli teorijski dio potkrijepljen je slikama i primjerima koda programa.

Ključne riječi: desktop aplikacija, forma, c#, kontrole, događaji

APSTRACT

TITLE: Windows forms in C#

Windows forms represent the basic tool in creating a desktop application for Windows operating system. Each application consists of one or more interconnected forms. Form is a window that contains the elements to display, enter and manipulate data that are called control. Each control is programmed using events in C#.

In my thesis the basic features of the C# language, creation and implementation of the most common controls in Windows forms are explained in detail. Also, the entire theoretical part is supported by the figures and examples of program code.

Keywords: desktop applications, form, c #, control, events

ŽIVOTOPIS

Valentina Lovrić rođena je 15.11.1993. godine u Osijeku, Republika Hrvatska. Osnovnu školu pohađala je Osijeku od 2000. do 2008. godine. Nakon toga završila je Opću gimnaziju u Osijeku (2008. – 2012. godine). 2012. godine upisala je informatiku na Elektrotehničkom fakultetu u Osijeku. U svibnju 2016. završila je tečaj PHP web development-a na Otvorenom učilištu Algebra u Osijeku. Tijekom svojeg školovanja učila je dva strana jezika: engleski i njemački, a tijekom studiranja radila je na nekoliko blagajničkih i administrativnih poslova. Trenutno je zaposlena u „Siemens Convergence Creators“ u Osijeku kao „junior softver developer“. U slobodno vrijeme čita knjige i druži se s prijateljima.